Dieter Gollmann
Jan Meier
Andrei Sabelfeld (Eds.)

LNCS 4189

Computer Security – ESORICS 2006

11th European Symposium on Research in Computer Security
Hamburg, Germany, September 2006
Proceedings

Springer

Dieter Gollmann Jan Meier
Andrei Sabelfeld (Eds.)

Computer Security – ESORICS 2006

11th European Symposium on
Research in Computer Security
Hamburg, Germany, September 18-20, 2006
Proceedings

Springer

Volume Editors

Dieter Gollmann
Jan Meier
TU Hamburg-Harburg
Harburger Schlossstr. 20, 21079 Hamburg-Harburg, Germany
E-mail: diego@tu-harburg.de, j.meier@tuhh.de

Andrei Sabelfeld
Chalmers University of Technology
Dept. of Computer Science and Engineering
EDIT Bldg., Rännvägen 6B, 41296 Gothenborg, Sweden
E-mail: andrei@cs.chalmers.se

Library of Congress Control Number: 2006932266

CR Subject Classification (1998): E.3, D.4.6, C.2.0, H.2.0, K.6.5, K.4.4

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-540-44601-X Springer Berlin Heidelberg New York
ISBN-13 978-3-540-44601-9 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11863908 06/3142 5 4 3 2 1 0

Foreword

These proceedings contain the papers selected for presentation at the 11th European Symposium on Research in Computer Security – ESORICS, held in Hamburg, Germany, September 18-20, 2006.

The 160 papers submitted to ESORICS were each reviewed by at least three members of the program committee. A two-week discussion phase was then held electronically, where committee members could comment on all papers and all reviews. Finally, 32 papers were selected for presentation at ESORICS, giving an acceptance rate of about 21%.

In 2005, three specialized security workshops were organized in affiliation with ESORICS. This trend has continued. In addition to RAID, which is already a well established event in its own right, there were four more workshops this year, ESAS 2006, EuDiRights 06, STM 06, and FEE2, further strengthening the rôle of ESORICS as the major European conference on security research.

There were many volunteers who offered their time and energy to put together the symposium and who deserve our acknowledgment. We want to thank all the members of the program committee and the external reviewers for their hard work in evaluating and discussing the submissions. We are also very grateful to all those people whose work ensured a smooth organization: Joachim Posegga, who served as General Chair; Andreas Günter and his team at HITeC for taking on the conference management; Klaus-Peter Kossakowski for his efforts as Sponsorship Chair; Jan Meier for managing the ESORICS Web site, and Joachim Stehmann for the Web design; and Martin Johns for dealing with the growing number of affiliated workshops.

Last, but certainly not least, our thanks go to all the authors who submitted papers and all the attendees. We hope you found the program stimulating.

July 2006 Dieter Gollmann and Andrei Sabelfeld

Organization

General Chair

Joachim Posegga	University of Hamburg, Germany

Program Committee

Tuomas Aura	Microsoft Research, UK
Michael Backes	Saarland University, Germany
Gilles Barthe	INRIA Sophia-Antipolis, France
Lynn Batten	Deakin University, Australia
Giampaolo Bella	University of Catania, Italy
Joachim Biskup	University of Dortmund, Germany
Jan Camenisch	IBM Zurich Research Laboratory, Switzerland
Jason Crampton	Royal Holloway, UK
Frederic Cuppens	ENST Bretagne, France
Marc Dacier	Institut Eurécom, France
George Danezis	University of Leuven, Belgium
Sabrina De Capitani di Vimercati	University of Milan, Italy
Robert Deng	Singapore Management University, Singapore
Ulfar Erlingsson	Microsoft Research, USA
Simon Foley	University College Cork, Ireland
Philippe Golle	Palo Alto Research Center, USA
Dieter Gollmann (co-chair)	Hamburg University of Technology, Germany
Pieter Hartel	Twente University, Netherlands
Jaap-Henk Hoepman	Radboud University Nijmegen, Netherlands
Sushil Jajodia	George Mason University, USA
Alan Jeffrey	Bell Labs, USA
Audun Jøsang	QUT, Australia
Jan Jürjens	TU Munich, Germany
Markus Kuhn	University of Cambridge, UK
Xuejia Lai	Shanghai Jiaotong University, PR China
Kwok-Yan Lam	Tsinghua University, PR China
Volkmar Lotz	SAP, France
Heiko Mantel	RWTH Aachen, Germany
Vashek Matyas	Masaryk University Brno, Czech Republic
Flemming Nielson	DTU, Denmark

Peter Ryan	University of Newcastle upon Tyne, UK
Andrei Sabelfeld (co-chair)	Chalmers University, Sweden
Babak Sadighi	SICS, Sweden
Kazue Sako	NEC Corporation, Japan
Andre Scedrov	U. Pennsylvania, USA
Einar Snekkenes	Gjøvik University College, Norway
Eijiro Sumii	Tohoku University, Japan
Paul Syverson	Naval Research Laboratory, USA
Mariemma I. Yagüe	University of Malaga, Spain
Alf Zugenmaier	DoCoMo Labs Europe, Germany

Additional Reviewers

Imad Aad, Pedro Adão, Ana Almeida Matos, Toshinori Araki, Olav Bandmann, Vicente Benjumea, Gustavo Betarte, Stefano Bistarelli, Bruno Blanchet, Damiano Bolzoni, Ahmed Bouabdallah, Jeremy Bryans, Marzia Buscemi, Jan Cederquist, Shiping Chen, Lukasz Chmielewski, Daniel J.T. Chong, Morshed Chowdhury, Siu-Leung Chung, Jolyon Clulow, Céline Coma, Luca Compagna, Ricardo Corin, Veronique Cortier, Nora Cuppens-Boulahia, Marcin Czenko, Mads Dam, Marnix Dekker, Markus Dürmuth, Ulrich Flegel, Jun Furukawa, David Galindo, Han Gao, Flavio D. Garcia, Joaquin Garcia, Meng Ge, Pablo Giambiagi, Hidehito Gomi, Maria Isabel González Vasco, Thomas Gross, Toshiyuki Isshiki, Ana Jancic, Thomas Jensen, Florian Kerschbaum, Naoki Kobayashi, Steve Kremer, Jan Krhovjak, Marek Kumpost, Sven Lachmund, Julien Laganier, Yassine Lakhnech, Peeter Laud, Corrado Leita, Anyi Liu, Vaclav Lorenc, Henning Makholm, Matteo Maffei, Michael Meier, Dale Miller, Kengo Mori, Antonio Muñoz Gallego, Steven Murdoch, Thanh Son Nguyen, Christoffer Rosenkilde Nielsen, Satoshi Obana, Yutaka Oiwa, Joseph Pamula, Maura Paterson, Jean-Christophe Pazzaglia, Thea Peacock, Van Hau Pham, François Pottier, Christian W. Probst, Tony Ramard, Sankardas Roy, Pierangela Samarati, Thierry Sans, Andreas Schaad, Ludwig Seitz, Fred Spiessens, Henning Sudbrock, Hongwei Sun, Petr Svenda, Isamu Teranishi, Tachio Terauchi, Olivier Thonnard, Terkel K. Tolstrup, Laurent Vigneron, Jan Vitek, Lingyu Wang, Stephen D. Wolthusen, Konrad Wrona, Hirosuke Yamamoto, Yanjiang Yang, Chao Yao, Stefano Zanero, Ye Zhang, Xibin Zhao, Hongbin Zhou, Anna Zych

Local Organization

Wiebke Frauen, Andreas Günter, Martin Johns (workshop chair), Klaus-Peter Kossakowski (sponsorship chair), Jan Meier, Joachim Stehmann (Web design), Margit Wichmann

Table of Contents

Finding Peer-to-Peer File-Sharing Using Coarse Network Behaviors*

Michael P. Collins[1] and Michael K. Reiter[2]

[1] CERT/Network Situational Awareness, Software Engineering Institute, Carnegie Mellon University
mcollins@cert.org

[2] Electrical & Computer Engineering Department, Computer Science Department, and CyLab, Carnegie Mellon University
reiter@cmu.edu

Abstract. A user who wants to use a service forbidden by their site's usage policy can masquerade their packets in order to evade detection. One masquerade technique sends prohibited traffic on TCP ports commonly used by permitted services, such as port 80. Users who hide their traffic in this way pose a special challenge, since filtering by port number risks interfering with legitimate services using the same port. We propose a set of tests for identifying masqueraded peer-to-peer file-sharing based on traffic summaries (flows). Our approach is based on the hypothesis that these applications have observable behavior that can be differentiated without relying on deep packet examination. We develop tests for these behaviors that, when combined, provide an accurate method for identifying these masqueraded services without relying on payload or port number. We test this approach by demonstrating that our integrated detection mechanism can identify BitTorrent with a 72% true positive rate and virtually no observed false positives in control services (FTP-Data, HTTP, SMTP).

1 Introduction

Peer-to-peer file-sharing services are often constrained by organizations due to their widespread use for disseminating copyrighted content illegally, their significant bandwidth consumption for (typically) non-work-related uses, and/or the risk that they may introduce new security vulnerabilities to the organization. Karagiannis et al. [5] have shown that instead of obeying site bans on file-sharing, however, users hide their file-sharing activity. Moreover, file-sharing tools themselves are being updated to circumvent attempts to filter these services; e.g., BitTorrent developers now incorporate encryption into their products in order to evade traffic shaping.[1]

* This work was partially supported by NSF award CNS-0433540, and by KISA and MIC of Korea.

[1] "Encrypting BitTorrent to Take Out Traffic Shapers", TorrentFreak Weblog, http://torrentfreak.com/encrypting-BitTorrent-to-take-out-traffic-shapers/

While encryption makes filtering based on traffic content difficult, filtering packets by port number (as would typically be implemented in router ACLs, for example) remains an obstacle to peer-to-peer file-sharing. As such, common hiding methods also involve changing the port number used by the service to something that is not filtered. In networks that implement a "deny-than-allow" policy, the service traffic may be sent on a common service port, in particular 80/TCP (HTTP).

In such cases, ports do not reliably convey the services using them, while deep packet examination is viable only as long as packet payload is unencrypted. Analysts therefore need alternative methods to characterize and filter traffic. In this paper, we propose an alternative service detection and identification method that characterizes services behaviorally. We hypothesize that TCP services have quantifiable behaviors that can be used to identify them without relying on payload or port numbers. For example, we expect that the majority of HTTP sessions begin with a small initial request followed by a larger response, and then terminate. If a presumed HTTP client and HTTP server were communicating using symmetric short bursts of traffic in a single long-lived session, then we would have reason to consider an alternative hypothesis, such as a chat service.

Within this paper, we focus on a specific problem that motivated this line of research: demonstrating that a user who claims to be using a common service on its standard port (such as HTTP) is using another service, specifically BitTorrent. To do so, we implement tests that characterize traffic and show how they can be used together to effectively differentiate BitTorrent traffic from common services. The goal of our research is a collection of tests which can be used by analysts or automated systems to classify traffic. Given the increasing sophistication of evasion strategies, we seek to find behaviors that can be effective with as few assumptions as possible. For example, these tests do not use deep packet examination, and are therefore applicable to encrypted and unencrypted traffic.

We calibrate and validate our approach using logs of traffic crossing a large network. From these logs, we select traffic records describing BitTorrent and major services, specifically HTTP, FTP data channel and SMTP. The log data consists of NetFlow, a traffic summarization standard developed by CISCO systems[2]. Flow data is a compact representation of an approximate TCP session, but does not contain payload. In addition, we do not trust port numbers, making our tests port- and payload-agnostic. Despite this, we show that by classifying flows based on several behaviors we can effectively differentiate source-destination pairs engaged in BitTorrent communication from those involved in HTTP, FTP or SMTP exchanges. Specifically, our integrated test identifies BitTorrent with a 72% true positive rate and virtually no observed false positives in control services (FTP-Data, HTTP, SMTP).

The rest of this paper is structured as follows. Section 2 describes previous work done in service detection. Section 3 describes the behaviors with which we characterize flows, and that we use to distinguish certain file-sharing traffic from

[2] CISCO Corporation, *Netflow Services and Applications*, `http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/nappswp.htm`

other types of traffic. Section 4 describes our experiments using these classifications both individually and in aggregate, to identify BitTorrent activity. We conclude in Section 5.

2 Previous Work

Prior work in identifying file-sharing traffic varies primarily in the information used to do so. Several file-sharing detection tools have analyzed packet payload (e.g., [9,20]), a method which will not survive encryption of packet contents or might simply be infeasible due to performance or other limitations. Other approaches utilize aggregate packet attributes, such as interstitial arrival times or the presence of specific packet sequences (e.g., [22,8,2,24,3,10,11,12]). However, in sufficiently large and busy networks, even this degree of packet analysis can be problematic.

As a result, flows are increasingly used for various types of security analysis (e.g., [18,13]). Flows were originally specified by Partridge [15] for traffic summarization, and have since been adopted by CISCO for traffic reporting. NetFlow, the CISCO standard, uses timeouts to approximate TCP sessions, an approach originally developed by Claffy et al. [1]. Since flow records do not contain payload information, they are generally used for large-scale and statistical analysis. Notably, Soule et al. [21] developed a classification method to cluster flows, though they stopped short of mapping them to existing applications (or types of applications).

Since their development, peer-to-peer file-sharing systems have become targets of filtering and detection efforts. Karagiannis et al. [5] showed that peer-to-peer users increasingly evade detection by moving their traffic to alternate port numbers. Studies conducted on BitTorrent and other peer-to-peer file-sharing applications have examined the behavior of individual nodes (e.g., [4,23,7]) and application networks (e.g., [19,17]), but have not compared the behaviors observed to the behavior of more traditional services. Ohzahata et al. [14] developed a method for detecting hosts participating in the Winny file-sharing application by inserting monitoring hosts within the file-sharing network itself. Karagiannis et al. [6] developed a general method for identifying applications called Blinc, which uses various heuristics and interconnection patterns exhibited by groups of nodes to identify services. In contrast, we focus on the flow characteristics between a pair of nodes in isolation to identify the service in which they are participating, and as such our approach is complementary. Nevertheless, we believe there is potential in combining our point-to-point analyses with Blinc's on interconnection patterns, and hope to investigate this in future work.

3 Application Classification

In this section, we describe the behaviors used to differentiate BitTorrent traffic from other services. In Section 3.1 we describe a classification tree that we will use to classify flows into different types, and in Section 3.2 we describe the intuition and formulation of our tests.

3.1 Simple Taxonomy

Our analyses use flow records; a *flow* is a sequence of packets with the same addressing information (source and destination addresses, source and destination ports, and protocol) which occur within a short time of each other [1]. A *flow record* is a summary consisting of addressing, size and timing information about the flow, but no payload. We will refer to fields of a flow record f with "dot" notation (e.g., f.duration or f.bytes).

We restrict our data to TCP flows. Flow collection systems such as CISCO NetFlow record TCP flags by ORing the flags of every packet. As a result, flag distributions cannot be derived from multi-packet flow records, and certain behaviors—notably whether an endpoint is the initiator or responder of the TCP connection of which the flow represents one direction—are not discernible.

We divide flows into three categories: **Short Flows**, comprising three packets or less; **Messages**, which are 4–10 packets but less than 2 kB in size; and **File Transfers**, which are any flows longer than a Message. Figure 1 represents our taxonomy as a decision tree and the categories that this tree covers.

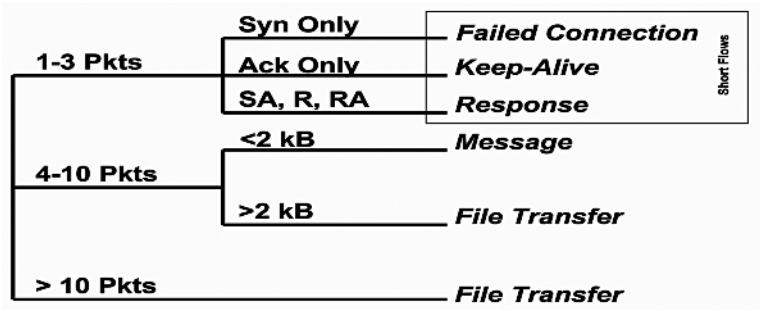


Fig. 1. Flow Classification Tree: The rules used on this tree assign each flow to a class

A **Short Flow** consists of three or fewer packets; since a complete TCP session will require at least three packets, Short Flows indicate some error in communication or anomaly in flow collection. Within Short Flows, we can acquire more information by examining the TCP flags of the flow; we use the flags to create three sub categories. A **Failed Connection** has a SYN flag and no ACKs. A **Keep-Alive** has ACK flags only. Since flow recording is timeout-based, Keep-Alives are recorded by the flow collector during long-lived sessions but currently do not have any substantial impact on analysis. A **Response** consists of any Short Flow whose flags imply a response from the TCP connection responder to the initiator: a SYN-ACK, a RST-ACK or a RST. We do not consider the other

TCP flags (e.g., PSH) significant in this analysis. As noted above, TCP flags are OR'ed in flow records, and as a result we only use flags in the Short Flow case, where the results are least ambiguous.

We define a **Message** as a flow consisting of 4–10 packets and with a total size less than 2 kB. We assume that Messages represent the structured exchange of service data between the source and destination. Example Messages include HTTP requests and the control messages sent by BitTorrent. We assume that Messages contain structured communication, as opposed to data intended for the application's users. Consequently, we expect that Messages will have fixed sizes and that certain Messages (with specific sizes) will appear more often than other ones.

We label any flow longer than 2 kB or 10 packets a **File Transfer**. We assume that a File Transfer is the exchange of non-service-specific information between two sites. We expect that certain services will tend to send shorter files than others. For example, we expect that HTTP servers will transfer less data than BitTorrent peers typically, since HTTP clients interact with users and therefore need a rapid response time.

Table 1 is a log showing BitTorrent flows; in this log, we have labeled each flow with its corresponding category. Of particular interest is the presence of repeated Failed Connections (the 144-byte SYN-only packets) and the 276-byte Message packets. Both of these behaviors will be used to construct tests in Section 3.2.

Table 1. Log of traffic and associated classification

Source Port	Destination Port	Packets	Bytes	Flags				Start Time	Class
				F	S	A	R		
3584	6881	1637	1270926		x	x		11/04/2005 21:09:33	File Transfer
3586	6881	5	276	x	x	x		11/04/2005 21:09:36	Message
3619	6881	5	276	x	x	x		11/04/2005 21:10:18	Message
3651	6881	5	276	x	x	x		11/04/2005 21:11:01	Message
3701	6881	5	276	x	x	x		11/04/2005 21:12:04	Message
1290	6881	3	144		x			11/04/2005 21:53:56	Failed Connection
2856	6881	5	636		x	x		11/04/2005 22:33:11	Message
3916	6881	5	276	x	x	x		11/04/2005 23:03:44	Message
4178	6881	5	636		x	x		11/04/2005 23:12:01	Message
4884	6881	3	144		x			11/04/2005 23:32:05	Failed Connection

3.2 Tests

In this section, we describe four tests for characterizing the flows generated by various services. Each test is performed on a log of flow records bearing the same source and destination, and hence are unidirectional. We rely on unidirectional flows for three reasons. First, CISCO NetFlow is reported unidirectionally; i.e., each direction of a connection is reported in a different flow. Second, on a network with multiple access points, there is no guarantee that entry and exit traffic passes through the same interface. As a result, determining the flows representing both directions of a connection on a large network can be prohibitively difficult. Finally, we wish to acquire results using as little information as possible.

Each test outlined in this section is characterized by function $\theta(x, F)$. This binary-valued function applies a particular threshold x to a measure calculated on a *flow log* F, where F consists of flows all bearing the same source and destination.

Failed Connections. We expect that, except for scanners, clients (respectively, peers) open connections only to servers (respectively, other peers) of which they have been informed by another party. For example, BitTorrent peers connect to peers that they have been informed of by their trackers. We further expect that when a client (peer) attempts to connect to a server (peer) that is not present, we will see multiple connection attempts.

We thus expect that Failed Connections occur more frequently in traffic from services with transient server/peer populations. We expect that BitTorrent peers regularly disconnect and that other peers will try to communicate with them after this. In contrast, we expect that the providers of HTTP and SMTP servers will implement policies to ensure maximum uptime. Therefore, under normal circumstances, we expect that the rate of Failed Connections for BitTorrent will be higher than it would be for SMTP or HTTP, for example.

Table 2 summarizes Failed Connections in an hour of observed data from four services: HTTP, FTP, SMTP and BitTorrent. From the observed flow records, we count the number of unique source-destination pairs and then count the number of pairs that have multiple Failed Connections. As the table shows, only SMTP and BitTorrent have a significant rate of Failed Connections, and BitTorrent has a somewhat higher rate of Failed Connections than SMTP does.

To evaluate connection failure as a detection method, we will use the following threshold test, $\theta_c(x, F)$:

$$\theta_c(x, F) = \begin{cases} 0 & \text{if the percentage of Failed Connections in } F \text{ is less than } x \\ 1 & \text{otherwise} \end{cases}$$

Bandwidth. In most client-server applications, a single server communicates with multiple clients; therefore, implementing faster transfers generally requires purchasing a higher-bandwidth connection for the server. BitTorrent increases

Table 2. Failed Connections per service for sample data

Service	Source-destination pairs	Pairs experiencing n Failed Connections			
		$n = 0$	$n = 1$	$n = 2$	$n > 2$
HTTP	3089	2956 (96%)	78 (3%)	22 (1%)	33 (1%)
FTP	431	431 (100%)	0 (0%)	0 (0%)	0 (0%)
SMTP	18829	15352 (82%)	1789 (10%)	619 (3%)	1069 (6%)
BitTorrent	49	37 (76%)	1 (2%)	0 (0%)	11 (22%)

the speed of a transfer by adding more peers to the BitTorrent "swarm": each peer transfers a fragment of the desired file, resulting in a high-bandwidth transfer comprised of multiple low-bandwidth connections. We expect that most dedicated servers will transfer data at higher speeds than a peer in a BitTorrent transfer will.

For flow logs of file transfer services (as opposed to a chat service, such as AIM), we of course expect that many of the corresponding connections are used for file transfers. The flow in the direction opposite the file transfer will consist almost exclusively of 40-byte zero-payload packets. Calculating bandwidth by counting the bytes in these packets will result in an artificially depressed value for the bandwidth consumed by the connection that gave rise to this flow. To compensate for this, we assume that the files transferred are considerably larger than the standard 1500-byte MTU for Ethernet, and consequently fragmented into MTU-length packets. We then estimate bandwidth for such flows by assuming that all of the ACK packets are in response to 1500-byte packets, resulting in the following formula:

$$b(f) = \frac{1500 \text{ bytes/packet} * f.\mathsf{packets}}{\max(f.\mathsf{duration}, 1 \text{ second})}$$

For bandwidth tests, our threshold function, θ_b, will be expressed as follows:

$$\theta_b(x, F) = \begin{cases} \perp & \text{if there are no File Transfers in } F \\ 0 & \text{if there is a File Transfer } f \in F \text{ such that } b(f) \geq x \\ 1 & \text{otherwise} \end{cases}$$

Note that $\theta_b(x, F)$ is undefined if there are no File Transfers in F. A bandwidth test cannot be meaningfully applied to a flow log with no File Transfers, since flow durations are recorded with second precision (while Messages and Short Flows typically take far less than a second).

Comparing Message Profiles. We expect that Messages will have fixed formats specified by the service. For example, the first action taken by a BitTorrent peer upon connecting to another peer is to send a 68-byte handshake containing a characteristic hash for its target file.[3] We therefore expect that a disproportionate number of BitTorrent Messages will have 68-byte payloads.

Comparative histograms for BitTorrent and HTTP Message sizes are shown in Figure 2. As this example shows, BitTorrent traffic spikes at 76 bytes per Message. This is most likely due to the 68 bytes necessary to send the BitTorrent handshake, plus a common 8-byte set of TCP options [16].

We test this hypothesis by generating histograms for the Messages from each major data set and then use the L_1 distance as an indicator of similarity. We define a histogram H on the Messages in a flow log F as a set of values $h_0(F) \ldots h_n(F)$, such that $h_i(F)$ is the observed probability that a Message in F

[3] See `http://wiki.theory.org/BitTorrentSpecification`

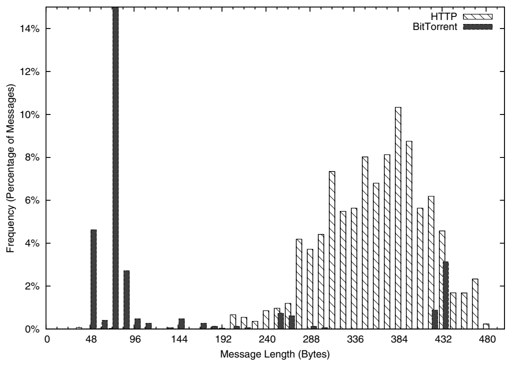


Fig. 2. Histogram comparing BitTorrent and HTTP Messages; the peak comprises 84% of BitTorrent Messages

has a payload between $[k(i-1), ki)$ bytes in size. In particular, $\sum_{i=1}^{n} h_i(F) = 1$. Given a reference flow log F_R and a test flow log F_T, the L_1 distance is defined as:

$$L_1(F_T, F_R) = \sum_{i=0}^{n} |h_i(F_T) - h_i(F_R)|$$

Histogram calculation is complicated by TCP headers and TCP options. TCP packets contain 40 bytes of header data above any payload. In addition, TCP sessions specify TCP payload options in the first packet; these options will be some multiple of 4 bytes and also affect the payload size. To compensate for the headers, we calculate the payload size for a flow f as:

$$p(f) = f.\mathsf{bytes} - (40 \text{ bytes/packet} \times f.\mathsf{packets})$$

We set k, the size of the bins used to calculate probabilities, to 12 bytes. This bin size is larger than the sizes of the most common TCP option combinations [16] and should therefore reduce profile mismatches caused by different option combinations.

The threshold test is then:

$$\theta_p(x, F) = \begin{cases} \perp & \text{if there are no Messages in } F \\ 0 & \text{if there exist Messages in } F \text{ and } L_1(F, F_R) \geq x \\ 1 & \text{otherwise} \end{cases}$$

It is understood that the L_1 distance is calculated only using the Messages in F and F_R. Note that $\theta_p(x, F)$ is undefined if there are no Messages in F: Short Flows should not have payload data, and we assume that File Transfers contain unstructured (and so arbitrary-length) data.

Volume. Because HTTP files are immediately viewed by a user, we expect that web pages are relatively small in order to ensure rapid transfer. Therefore, we

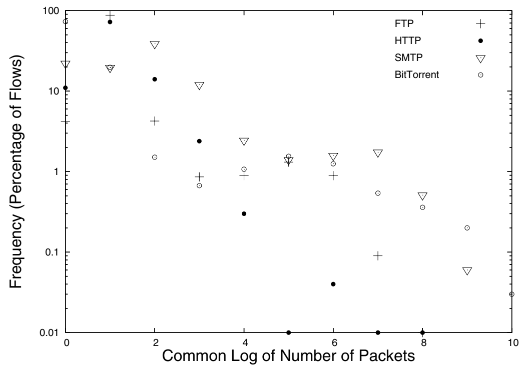


Fig. 3. Distribution of File Transfer volume for major sources

expect that if a BitTorrent user transfers a file, the resulting TCP session will transfer more bytes than a HTTP session would.

Figure 3 is a log plot of the frequency of flows; each flow is binned by the common logarithm of its packet count. The results in Figure 3 satisfy our intuition about file sizes: the majority of HTTP flows are relatively low-volume, with the majority consisting of 20 packets or less. We note that, in contrast to our original expectations, SMTP induces higher-volume flows than FTP.

For this behavior, the threshold function $\theta_t(x, F)$, is defined as:

$$\theta_t(x, F) = \begin{cases} 0 & \text{if } \forall f \in F : f.\mathsf{packets} \leq x \\ 1 & \text{otherwise} \end{cases}$$

Similar to the bandwidth test, the presence of even a single large-volume flow is an indicator.

4 Experiments

This section summarizes the tests conducted on the source data and their results. Section 4.1 describes the source data, and Section 4.2 describes how we calibrate the individual tests. Section 4.3 describes the integration and validation of these tests into a combined test for BitTorrent, and evaluates its accuracy. Section 4.4 discusses what countermeasures a user can take to evade the tests.

4.1 Calibration Data

Calibration data is selected from a log of traffic at the border of a large edge network. Since the flow logs do not contain payload information, we cannot determine what service any flow describes via signatures. In order to acquire calibration and validation data, we use a combination of approaches which are described below.

Each data set consists of FTP, SMTP, HTTP or BitTorrent flow records. SMTP and HTTP were chosen because they comprise the majority of bytes crossing the monitored network. The FTP data set is actually FTP-Data traffic (i.e., port 20). We chose to use FTP-Data because we expected that it would consist primarily of large File Transfers and thus be difficult to distinguish from BitTorrent.

In the case of FTP, SMTP and HTTP, server names are used to validate services. HTTP server names usually begin with a "www" prefix; similarly, SMTP server names often begin with "mx" or "mail". We restrict our flows to those with destination addresses to which a name containing a relevant string resolves. We assume that if a destination has an appropriate name and participates in the flow on the relevant port, then it is providing the suggested service.

In each case, the flows are outgoing flows collected from the observed network, i.e., flow sources are always inside the network and flow destinations are always outside. The calibration data are detailed below:

- HTTP: The HTTP data set consists of flow records where the destination port is port 80 and the source port is in the range 1024–5000, the ephemeral port range for Windows. In addition, every destination address must be that of an HTTP service (such as Akamai or Doubleclick) or have a name including "www" or "web".
- SMTP: The SMTP data set consists of flow records where the destination port is 25 and the source port is ephemeral. Destination addresses must have names containing the string "smtp", "mail" or "mx".
- FTP: The FTP data set consists of FTP-Data (port 20) flows; the corresponding FTP-Control (port 21) information is not used in this analysis. The FTP data set consists of flow records where the destination port is 20 and the source port is in the ephemeral range. Destination addresses must have names that begin with the string "ftp".
- BitTorrent: The BitTorrent data set consists of logs from hosts running BitTorrent in the observed network; these hosts were identified using a banner grabbing system configured to identify BitTorrent prefixes. While this system can identify hosts running BitTorrent, it does not identify when BitTorrent communication occurred. Therefore, when examining traffic logs to and from a suspected host we assume that other applications do not masquerade as BitTorrent, and so any interaction between two IP addresses (one of which is suspected) where the source port is ephemeral and the destination port is 6881 is assumed to be BitTorrent.

All source data comes from the first week of November 2005; due to the frequency with which particular services are used, each data set comprises a different period of time. In particular, BitTorrent traffic was collected over 2 days, while the other services were collected over several hours.

Table 3 summarizes the flows in the calibration data sets. Consider any individual "Service" column, e.g., the column "HTTP" that indicates the HTTP data set. For each flow classification, the row marked "flows" shows the number

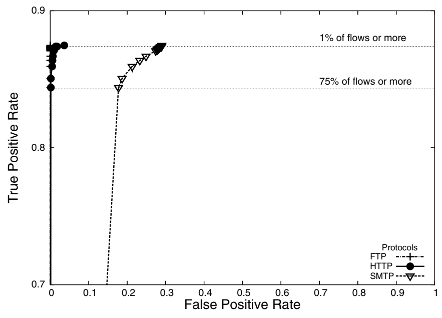
Table 3. Classification of flow records in calibration data

Flow Classification		Service			
		HTTP	FTP	SMTP	BitTorrent
All	flows	205702 (100%)	120144 (100%)	142643 (100%)	13617 (100%)
	src-dst pairs	3088 (100%)	746 (100%)	18829 (100%)	2275 (100%)
Failed Connection	flows	2427 (1%)	0 (0%)	13010 (9%)	9325 (68%)
	src-dst pairs	132 (4%)	0 (0%)	3476 (18%)	1992 (88%)
Keep-Alive	flows	5403 (3%)	1965 (2%)	31306 (22%)	1106 (8%)
	src-dst pairs	409 (13%)	504 (68%)	2566 (14%)	135 (6%)
Response	flows	18635 (9%)	23663 (20%)	12545 (9%)	199 (1%)
	src-dst pairs	730 (24%)	314 (42%)	2959 (16%)	53 (2%)
Message	flows	150937 (73%)	64558 (54%)	26271 (18%)	1615 (12%)
	src-dst pairs	2880 (93%)	558 (75%)	4704 (25%)	270 (12%)
File Transfer	flows	28300 (14%)	29958 (25%)	59511 (42%)	1372 (10%)
	src-dst pairs	1504 (49%)	504 (68%)	13771 (73%)	199 (9%)

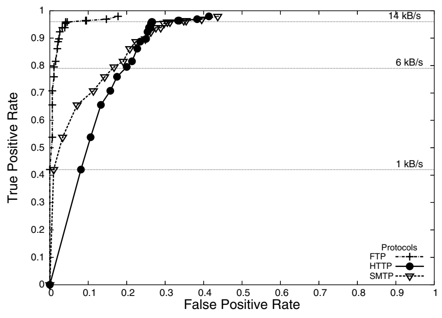
of flows of that type that appeared in that data set, and the approximate percentage of the total number of flows that these comprised. Ignoring the "All" classification, the rest of the classifications are mutually exclusive and exhaustive, and so the flows accounted for in these "flows" rows total to the number of flows in the data set (indicated under the "All" classification). Similarly, for each flow classification, the number of source-destination pairs for which at least one flow of that type appeared in the data set is shown in the "src-dst pairs" row. Since the same source-destination pair can be listed for multiple flow classifications—if flows of each of those classifications were observed between that source-destination pair—the sum of the source-destination pair counts for the various flow types (excluding "All") generally exceeds the number of source-destination pairs in the data set. Note that FTP does not show any failed connections, presumably because FTP-Data connections are opened after FTP-Control connections have succeeded.

4.2 Calibration

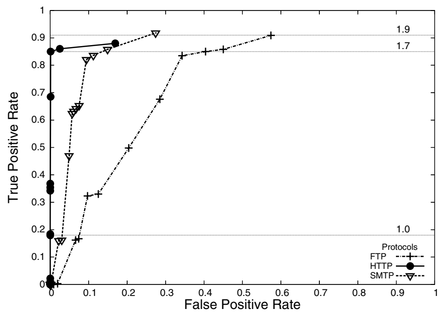
We have hypothesized that BitTorrent can be distinguished from other services by examining bandwidth, volume, the profile of Messages and the rate of Failed Connections. To evaluate this hypothesis, we now construct a series of tests, one for each behavior. In each test, we specify the behavior as a parameterized function and plot the results on a ROC (Receiver Operating Characteristic) curve. The ROC curve is generated by using the appropriate θ-test against flow logs selected from the data sets in Table 4. In each case, a control data set (either HTTP, SMTP or FTP) is partitioned into flow logs $C_1 \ldots C_\ell$, where ℓ is the total number of distinct source-destination pairs in the set and each C_i is log of all flows between one source-destination pair. Similarly, the BitTorrent data set is partitioned into flow logs $T_1 \ldots T_m$, each between one source-destination pair.



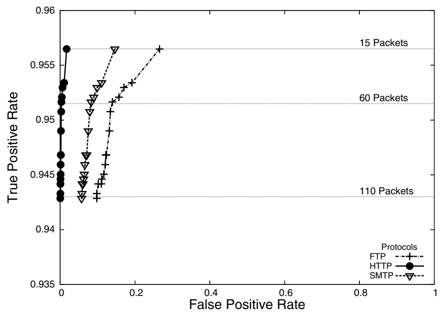
(i) Failed Connection ROC



(ii) Bandwidth ROC



(iii) Message profiling ROC



(iv) Volume ROC

Fig. 4. ROC curves generated from tests

The contents of C_i and T_j are then evaluated by the corresponding θ-function. The false and true positive rates are calculated as follows, given a test function $\theta(x, F)$.

$$FP(x) = \frac{\sum_{C \in \mathcal{C}} \theta(x, C)}{|\mathcal{C}|} \quad \text{where } \mathcal{C} = \{C_i : \theta(x, C_i) \neq \perp\}$$

$$TP(x) = \frac{\sum_{T \in \mathcal{T}} \theta(x, T)}{|\mathcal{T}|} \quad \text{where } \mathcal{T} = \{T_j : \theta(x, T_j) \neq \perp\}$$

Figure 4(i) shows how effective Failed Connections are for detecting BitTorrent versus control data. As described in Section 3.2, BitTorrent has an usually high number of Failed Connections: for 85% of the observed source-destination pairs, at least 75% of their flow logs consist of Failed Connections. SMTP traffic is the least distinguishable from BitTorrent; we believe that this is because both BitTorrent and SMTP connections are opened as the application requires, while HTTP and FTP connections are opened by user request. If an HTTP site is not available, a user is likely to give up; conversely, an SMTP client will retry quickly.

While the Failed Connection test is highly distinctive on these data sets, we note that our test data may be biased by the varying times used to collect data. While, as Table 1 shows, BitTorrent connections can fail over very short periods of time, a better statistical model may produce more precise results over homogeneous time periods.

Figure 4(ii) summarizes the results of our bandwidth tests. The operating characteristic is the maximum observed bandwidth, estimated between 1 kB/s and 20 kB/s using 1 kB/s increments. As this curve shows, the bandwidth cutoff for BitTorrent is approximately 14 kB/s, which allows a 90% correct identification for both HTTP and FTP traffic. SMTP traffic is less distinguishable, with higher than a 35% false positive rate at the same value.

We note that the bandwidth estimates are low; e.g., according to Figure 4(ii), roughly 30% of source-destination pairs for HTTP and SMTP had bandwidths less than 14 kB/s. We conjecture that this is primarily a result of limitations in using flow records to estimate bandwidth. For example, the available bandwidth of a persistent HTTP connection is not fully utilized for the duration of the connection; instead, content transfers on the connection are separated by intervening idle periods. Consequently, the flows recorded for such connections will have a deflated average bandwidth. Such effects could be rectified by using alternative flow-like metrics; several are proposed by Moore et al. [12], but would require modifying existing flow formats.

Figure 4(iii) summarizes the success rate for differentiating BitTorrent and other traffic using a profile of Messages. Each point in this graph is a mean of the results from 10 runs. In each such run, 20% of the source-destination pairs in the BitTorrent data set are selected uniformly at random and used to generate the profile. During the test, the remaining 80% are used to generate profiles for comparison. As this curve shows, Message profile comparison worked well for differentiating HTTP and SMTP traffic, but was considerably less accurate when comparing FTP traffic. We note that due to the random selection used during these tests, the resulting true positive rates vary for each data set; the thresholds shown in Figure 4(iii) are approximate across the three data sets.

Figure 4(iv) shows the efficacy of the volume test. In this curve, the operating characteristic ranges between 15 packets and 110 packets. We note that the true positive rate for the curve is very high, and that 94% of the source-destination flow logs observed for BitTorrent had one transfer of at least 110 packets, a number far higher than anything sent by our example HTTP flow logs.

4.3 Integration and Validation

We now combine the tests in order to identify BitTorrent. To do so, we use voting. For every source-destination pair in a data set consisting of mixed BitTorrent and control traffic, each test is applied to the corresponding flow log in order to determine whether the traffic is characteristic of BitTorrent. The result of the test is recorded as a vote, where an undefined result is a 0-vote. For each source-destination pair, the total number of votes is then used as the operating characteristic on a ROC curve. For this curve, a false positive is a flow log of control traffic that is identified as BitTorrent, and a false negative is a flow log of BitTorrent traffic identified as control traffic. This voting method is intended to be a simple integration attempt; in future work we may opt for more sophisticated methods.

The data sets used to validate these tests are summarized Table 4. These data sets are taken from different sampling periods than the calibration data sets and do not overlap them. Otherwise they are assembled using the same approach as the calibration data.

False positives and false negatives are weighed equally for each test: The upper left corner of a ROC curve represents perfect classification (i.e., no false positives and perfect true positives), and so we choose the point from each calibration curve that minimizes the distance from that corner. For the tests in Section 4.2, this results in a threshold x_c^{opt} for θ_c of 1%; for x_b^{opt} of 14 kB/s; for x_p^{opt} of 1.7; and for x_t^{opt} of 110 packets. Table 4 shows the results of each individual test applied against the validation data sets; we note that unlike calibration, the θ_p and θ_b tests are applied against all flow logs. As Table 4 shows, each behavior was demonstrated by at least 50% of the BitTorrent source-destination pairs. In comparison, none of the control services had a test which more than 30% of that service's flows passed.

Table 4. Validation data sets and individual test results

Data Set	Flows	Src-dst	Pairs for which $\theta(x^{\mathrm{opt}}, F) = 1$			
		pairs	θ_c	θ_b	θ_p	θ_t
BitTorrent	9911	70	36 (51%)	51 (73%)	63 (90%)	51 (73%)
HTTP	42391	1205	27 (2%)	156 (13%)	5 (0%)	64 (5%)
SMTP	46146	4832	729 (15%)	1350 (28%)	105 (2%)	1451 (30%)
FTP	47332	561	0 (0%)	15 (3%)	164 (29%)	158 (28%)

The integrated vote is plotted as a ROC curve in Figure 5. The integrated test results in a 72% true positive rate if 3 or more tests agree, and a corresponding false positive rate of 0% for all control services. None of the control source-destination pairs received four affirmative votes, and only source-destination pairs in the SMTP data set received three. HTTP was the most easily distinguished of the control services, with less than 1% of observed source-destination pairs having two affirmative votes.

4.4 Evasion

As noted in Section 1, users now actively evade detection methods. As a result, we must consider how the developers of a service can evade the tests discussed in this paper.

The BitTorrent specification can be changed to evade Message profiling by changing the sizes of the control Messages. If the Messages were randomly padded, a size-based profiling method would be less reliable. This approach is easy to implement.

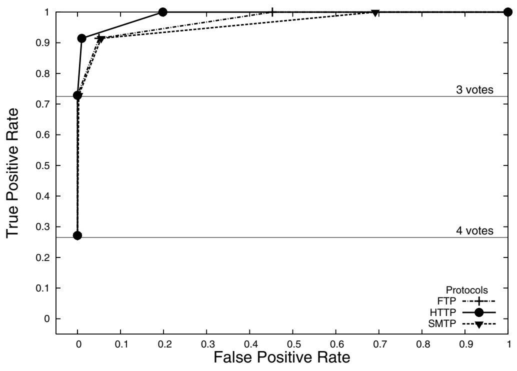


Fig. 5. Integrated ROC curve identifying BitTorrent

The volume test can be evaded by limiting the maximum amount of data received from a single host. However, this approach means that BitTorrent peers will have to communicate with a larger number of peers, at which point they are increasingly vulnerable to host-counting strategies such as Blinc's [6].

Tests based on Failed Connections can be evaded by increasing the time between connection attempts or by ensuring that a BitTorrent peer only communicates with peers that are known to be active, a function that is theoretically already supported by the tracker. However, either change will increase download time by increasing the time required to find an active host.

Evading bandwidth detection is more challenging: users must either purchase additional bandwidth or use all their bandwidth to transfer data.

5 Conclusions

In this paper, we have demonstrated that services have well-defined behaviors which can be used to identify masqueraded peer-to-peer file-sharing traffic without relying on payload or port numbers. To do so, we have developed a collection of tests based on service behavior and shown how BitTorrent differs from SMTP, HTTP and FTP traffic. These results are used to demonstrate that BitTorrent is characteristically different from these other services, and that these differences can be detected by looking at gross flow-level attributes, without relying on deeper packet examination than what is normally required to construct the flow.

References

1. K. Claffy, H. Braun, and G. Polyzos. A parameterizable methodology for internet traffic flow profiling. *IEEE Journal of Selected Areas in Communications*, 13(8):1481–1494, 1995.
2. J. Early, C. Brodley, and C. Rosenberg. Behavioral authentication of server flows. In *Proceedings of the 19th Annual Computer Security Applications Conference*, 2003.

3. F. Hernandez-Campos, A. Nobel, F. Smith, and K. Jeffay. Understanding patterns of TCP connection usage with statistical clustering. In *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2005.
4. M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garcés-Erice. Dissecting bittorrent: Five months in a torrent's lifetime. In *Proceedings of the 5th Annual Passive and Active Measurement Workshop*, 2004.
5. T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. Is p2p dying or just hiding? In *Proceedings of IEEE Globecom 2004 - Global Internet and Next Generation Networks*, 2004.
6. T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: multilevel traffic classification in the dark. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2005.
7. P. Karbhari, M. Ammar, A. Dhamdhere, H. Raj, G. Riley, and E. Zegura. Bootstrapping in gnutella: A measurement study. In *Proceedings of the 5th Annual Passive and Active Measurement Workshop*, 2004.
8. M. Kim, H. Kang, and J. Hong. Towards peer-to-peer traffic analysis using flows. In *Self-Managing Distributed Systems, 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, 2003.
9. C. Kruegel, T. Toth, and E. Kirda. Service specific anomaly detection for network intrusion detection. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, 2002.
10. A. McGregor, M. Hall, P. Lorier, and J. Brunskill. Flow clustering using machine learning techniques. In *Proceedings of the 5th International Workshop on Passive and Active Network Measurement*, 2004.
11. A. De Montigny-Leboeuf. Flow attributes for use in traffic characterization. Technical Report CRC-TN-2005-003, Communications Research Centre Canada, December 2005.
12. A. Moore, D. Zuev, and M. Crogan. Discriminators for use in flow-based classification. Technical Report RR-05-13, Department of Computer Science, Queen Mary, University of London, August 2005.
13. W. Nickless, J. Navarro, and L. Winkler. Combining CISCO netflow exports with relational database technology for usage statistics, intrusion detection, and network forensics. In *Proceedings of the 14th Annual Large Systems Administration (LISA) Conference*, 2000.
14. S. Ohzahata, Y. Hagiwara, M. Terada, and K. Kawashima. A traffic identification method and evaluations for a pure p2p application. In *Proceedings of the 6th Annual Passive and Active Measurement Workshop*, 2005.
15. C. Partridge. A Proposed Flow Specification. RFC 1363 (Informational), September 1992.
16. K. Pentikousis and H. Badr. Quantifying the deployment of TCP options, a comparative study. *IEEE Communications Letters*, 8(10):647–649, October 2004.
17. J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. A measurement study of the BitTorrent peer-to-peer file-sharing system. Technical Report PDS-2004-007, Delft University of Technology, April 2004.
18. S. Romig, M. Fullmer, and R. Luman. The OSU flow-tools package and CISCO netflow logs. In *Proceedings of the 14th Annual Large Systems Administration (LISA) Conference*, 2000.
19. S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking*, 2002.

20. S. Sen, O. Spatscheck, and D. Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *Proceedings of the 13th International Conference on World Wide Web*, 2004.
21. A. Soule, K. Salamatia, N. Taft, R. Emilion, and K. Papagiannaki. Flow classification by histograms: or how to go on safari in the internet. In *Proceedings of the 2004 Joint International Conference on Measurement and Modeling of Computer Systems*, 2004.
22. C. Taylor and J. Alves-Foss. An empirical analysis of NATE: network analysis of anomalous traffic events. In *Proceedings of the 10th New Security Paradigms Workshop*, 2002.
23. K. Tutschku. A measurement-based traffic profile of the eDonkey filesharing service. In *Proceedings of the 5th Annual Passive and Active Measurement Workshop*, 2004.
24. C. Wright, F. Monrose, and G. Masson. HMM profiles for network traffic classification. In *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, 2004.

Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses[⋆]

Vitaly Shmatikov and Ming-Hsiu Wang

The University of Texas at Austin

Abstract. Mix networks are a popular mechanism for anonymous Internet communications. By routing IP traffic through an overlay chain of mixes, they aim to hide the relationship between its origin and destination. Using a realistic model of interactive Internet traffic, we study the problem of defending low-latency mix networks against attacks based on correlating inter-packet intervals on two or more links of the mix chain. We investigate several attack models, including an active attack which involves adversarial modification of packet flows in order to "fingerprint" them, and analyze the tradeoffs between the amount of cover traffic, extra latency, and anonymity properties of the mix network. We demonstrate that previously proposed defenses are either ineffective, or impose a prohibitively large latency and/or bandwidth overhead on communicating applications. We propose a new defense based on adaptive padding.

1 Introduction

Mix networks are a practical way to enable anonymous communications on public networks. The goal is to hide the relationship between the origin of the traffic (*e.g.*, a Web browser) and the destination (*e.g.*, a website). A *mix*, first proposed by Chaum [6], can be thought of as a server that accepts incoming connections and forwards them in such a way that an eavesdropper cannot easily determine which outgoing connection corresponds to which incoming connection.

Because any given mix may be compromised, traffic is usually routed through a chain of mixes. Intuitively, even if some mixes in the chain are malicious, the other ones provide some anonymity for connections routed through them. Many different architectures for mix networks have been proposed in literature [3,13,9,18,11]. We focus on *low-latency* mix networks, whose main purpose is to protect privacy of *interactive* Internet communications, including popular applications such as Web browsing. Empirical evidence indicates that low-latency anonymity systems attract many more users than high-latency ones [12].

Like any anonymity system, a mix network can be attacked in a variety of ways. Some of the mix routers may be compromised by the attacker; endpoints of a repeatedly used chain may be linked by statistical analysis of message distribution within the network [7,10]; statistical properties of randomly constructed routes may be exploited to determine the likely route origin [32,26], and so on.

[⋆] This work was partially supported by NSF grants CNS-0509033 and IIS-0534198.

In this paper, we assume that the attacker has direct access to packet streams on some of the network links. Many mix networks are specifically intended to provide anonymity against attackers who control the communication medium. Traffic analysis is an especially serious threat for low-latency mix networks because it is very difficult to hide statistical characteristics of the packet stream *and* satisfy the stringent latency requirements imposed by interactive applications.

Details of the route establishment protocol are not important for our analysis. Once a route through the mix network has been constructed, all packets are usually encrypted and padded to hide payload size. Inter-packet time intervals are usually *not* hidden because low latency requires that each packet be dispatched as soon as it has been generated by the application. This can be exploited by the attacker. By correlating inter-packet intervals on two links, he may be able to determine with high probability that the links belong to the same route.

We will refer to this as the *timing analysis* attack. This attack is probabilistic, and may suffer from false positives and false negatives. The standard measure of success is the *crossover error rate*, at which the attacker's false positive rate is equal to his false negative rate. The lower the crossover error rate, the more successful the attack. The conventional defense is to send *cover traffic* on each link in order to hide actual packets in the stream of padding (dummy) packets.

Our contributions. We analyze resilience of low-latency mix networks to inter-packet interval correlation attacks using a *realistic* traffic model based on HTTP traces from National Laboratory for Applied Network Research (NLANR) [21].

We propose *adaptive padding*, a new defense against timing analysis. In our scheme, intermediate mixes inject dummy packets into statistically unlikely gaps in the packet flow, destroying timing "fingerprints" without adding any latency to application traffic. We also present a version of our scheme which defends against active attackers at the cost of relatively small extra latency.

The purpose of adaptive padding is to prevent the attacker from determining which of the multiple simultaneous connections is being carried on a given network link, not to hide whether a certain user or connection is currently active. Constant-rate padding may provide better protection for the latter, although variants such as defensive dropping [17] are trivially defeated by measuring packet density because real and dummy packets are dropped at different rates.

We focus on short-lived connections, and do *not* aim to provide a comprehensive defense against long-term statistical disclosure attacks. It is not clear whether this is at all achievable — in any real-world network, there will inevitably exist small statistical differences between packet flows which cannot be completely hidden unless dummy traffic generators are perfectly synchronized.

Using simulated traffic, we quantify the basic tradeoff between the padding ratio (average number of dummy packets per real packet) and protection against timing analysis. We show that adaptive padding can defeat timing analysis with relatively low padding ratios, *e.g.*, 0.4 attacker's crossover error rate can be achieved with the 2 : 3 average ratio between dummy and real packets. (Maximum crossover error rate is 0.5, which corresponds to random guessing.) We also investigate *active* attacks, in which artificial gaps and bursts are introduced into packet flows in order to "fingerprint" them. Adaptive padding provides significant protection against active attacks at a relatively low extra latency cost.

We compare adaptive padding with defenses based on sender-originated, constant-rate cover traffic and variants such as defensive dropping, showing that they are not feasible for traffic exhibiting realistic statistical characteristics. To defeat passive timing analysis with reasonable padding ratios, they require prohibitively high extra latency. They also fail completely against an active attack.

Organization of the paper. We survey related work in section 2. Our model and metrics are described in section 3, the adaptive padding scheme in section 4. Simulation methodology and results are presented in section 5. In section 6, we discuss active attacks; in section 7, we argue that constant-rate defenses are not feasible. Deployment issues are in section 8, future directions in section 9.

2 Related Work

Traffic analysis attacks based on packet flow correlation [28,2,1,25] and statistical characteristics of individual mixes [8] have been recognized as a serious threat to low-latency mix networks, but few defenses have been proposed to date.

Venkatraman and Newman-Wolfe [22,31] presented a mathematical model for passive traffic analysis attacks and proposed a defense that requires complete knowledge of traffic on all links by a trustworthy entity.

Timmerman [29,30] proposed an adaptive "traffic masking" technique for latency-insensitive applications such as email. Her S-DATM algorithm uses cover traffic and artificial delays to ensure that traffic emerging from each user conforms to a certain profile. By contrast, we move the responsibility for traffic shaping *inside* the mix network, and do not aim to precisely reproduce a particular traffic shape (as discussed below, this requires prohibitive latencies).

Berthold and Langos [4] also focus on high-latency networks, proposing that intermediate mixes inject dummy traffic to hide whether a connection is active or not. By contrast, our goal is to prevent the attacker from using fine-grained timing characteristics of the packet stream to determine which of several simultaneous connections is carried by a given network link.

Rennhard *et al.* [24] present an adaptive technique for artificially delaying packets from multiple connections at intermediate mixes in order to reduce the amount of cover traffic. A similar technique without any cover traffic was proposed by Zhu *et al.* [33]. With a realistic model of actual traffic, however, the timing "fingerprints" of two flows are likely to be sufficiently different so that the only effective defense is to actively reshape the flows at intermediate routers. Unfortunately, the cost of this defense is prohibitive latency.

Fu *et al.* [14,15], followed by Levine *et al.* [17], demonstrated that even packet flows protected by constant-rate cover traffic are vulnerable to statistical analysis of inter-packet intervals. Fu *et al.* propose to generate cover traffic with variable inter-packet intervals, which is achieved by our adaptive padding scheme.

In the "defensive dropping" scheme [17], route initiators generate dummy packets, marking each one so that it is dropped by one of the intermediate mixes. The initiator must send *all* traffic at the same constant rate, delaying real packets so as not to exceed the chosen rate. Bursty traffic generated by interactive applications suffers vast extra latency in this case (see section 7). The possibility of an active attack is mentioned in [17], but it is not included in the simulations, and no defenses are proposed.

Devastating timing attacks have been successfully demonstrated in real-world mix networks [19,23]. To prevent a particular type of timing analysis performed by a malicious client, Øverlier and Syverson recommend using a trusted entry node [23], which is complementary to the defenses proposed in this paper.

3 Model and Metrics

Network. We use a simplified model of a mix network. A single connection consists of an *initiator* (or *sender*), a sequence (*path*) of N mixes, and a destination server. In a "short-path" network, N is set randomly to 2 or 3; in a "long-path" one, N is selected randomly from between 5 and 8. We ignore the system-specific details of the path establishment protocol, and assume that all packets from the initiator to the server follow the same path through the (overlay) mix network.

We make the standard assumption that, following the path establishment protocol, all intermediate mixes share pairwise symmetric keys with the initiator, and that each consecutive pair of mixes on a path shares a pairwise symmetric key. We assume an end-to-end TCP connection between the initiator and the server, *i.e.*, there is no separate TCP connection between each consecutive pair of intermediate mixes. An example of such a system is Tarzan [13]. We further discuss feasibility of our techniques in various types of mix networks in section 8.

Timing analysis. We consider an attacker who measures *inter-packet intervals*, *i.e.*, time differences between observations of consecutive packets, on two network links in order to infer whether these links carry the same connection. Even when packets are padded and encrypted, inter-packet intervals tend to remain correlated within the same IP flow. Moreover, traffic associated with interactive applications such as Web browsing tends to occur in bursts. Sequences of inter-packet intervals vary widely between different packet flows, and can thus be used to "fingerprint" a connection. Following Levine *et al.* [17], we assume that the attacker divides time into fixed-size windows, counts the number of packets observed during each window, and correlates the sequences of packet counts.

An *active* attacker can also impose his own unique timing signature (by dropping packets or introducing artificial bursts) onto the flow he is interested in, and then attempt to identify this signature on other network links.

Our attack model is deliberately simple. We ignore the effects of packet drops and bursts on higher-level TCP behavior. In a real attack, the attacker may also look at timing characteristics other than inter-packet intervals, actively modify packet streams in order to cause observable changes in network behavior, corrupt packets to cause TCP retransmission, and so on. Our model is sufficient for demonstrating serious problems with previously proposed solutions, and enables us to directly compare adaptive padding with defensive dropping [17].

Defense metric. Our attacker correlates packet counts on two network links within a certain time window. If the correlation coefficient exceeds some threshold, the attacker decides that the links carry the same flow. This analysis can suffer from *false positives* (the attacker erroneously determines that unrelated links carry the same flow) and *false negatives* (the attacker erroneously determines that the links are unrelated even though they do carry the same flow).

High correlation thresholds increase the false negative rate and decrease the false positive rate, while low thresholds do the opposite. The standard metric in this situation is the *crossover error rate* (called *equal error rate* in [17]), at which the false positive rate is equal to the false negative rate. A low crossover error rate means that the attacker achieves *both* a low false positive rate and a low false negative rate, *i.e.*, the defense is ineffective. On the other hand, *high crossover rates mean that the defense is good.* The highest crossover rate is 0.5. If the error rate is greater than 0.5, the attacker can simply flip all the answers.

Negative impact on network performance. Defenses against timing analysis use dummy traffic to hide gaps between real packets and/or alter timing patterns of flows by delaying packets or dropping them completely. Both techniques have negative consequences for network performance as observed by the end users. Adding dummy traffic consumes bandwidth, while delaying or dropping packets increases latency. The ideal defense should minimize both effects.

Our metrics are the average *padding ratio* of dummy packets to real packets across all links of a single connection, and the maximum and average extra *delay* per real packet in the defended viz. undefended network.

4 Adaptive Padding

After a mix receives a packet, our adaptive padding algorithm samples from the statistical distribution of inter-packet intervals. If the next packet arrives before the chosen interval expires, it is forwarded and a new value is sampled. To avoid skewing resulting intervals towards short values, the distribution is modified slightly to increase the probability of drawing a longer interval next time. If the chosen interval expires before a packet arrives, the gap is “repaired” by sending a dummy packet. Each mix is assumed to have a store of properly encrypted dummy packets, ready for injection into the packet stream (see section 8).

We assume that each mix knows a rough statistical distribution of inter-packet intervals for a “normal” flow. This distribution can be pre-computed from traffic repositories such as NLANR [21], or from the mix’s own observations. Our defense is fundamentally probabilistic, and may provide relatively poor protection for flows whose distribution of inter-packet intervals is very different from the assumed distribution. Nevertheless, our method is a significant improvement over alternatives that simply ignore statistical characteristics of the protected flows by assuming that all senders emit traffic at the same constant rate.

Data structures. For each connection that passes through it, a mix maintains a data structure consisting of several *bins*. The bins are mutually exclusive and

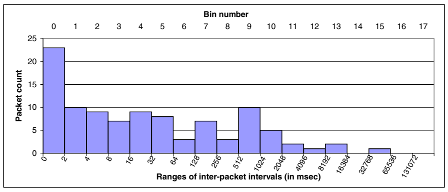


Fig. 1. Data structure representing distribution of inter-packet intervals at each mix

jointly cover all interval values from 0 to infinity. Each bin except the last corresponds to a finite range of inter-packet intervals; the last bin represents all intervals longer than a certain value. We will numbers bins as b_0 (corresponding to the shortest inter-packet intervals), b_1, and so on. For interactive Internet traffic, the distribution of inter-packet intervals tends to be wide but heavily biased to short values. Therefore, we found that increasing the range represented by each bin exponentially with the bin index works well. Intuitively, short inter-packet intervals, which occur frequently, are split into several bins, while long intervals, which are rare, are allocated to a single bin (see example in fig. 1).

Adaptive padding algorithm. We assume that a route through the network has been established prior to actual communication. Upon receiving the first packet of a connection, the mix forwards it and initializes the bins by drawing a sample of N values from the statistical distribution of inter-packet intervals, where N is a parameter of the system. Each sampled value falls into some range represented by exactly one bin, and a *token* is placed into that bin. For simplicity, a token counter is associated with each bin, and is incremented appropriately.

Upon receiving a packet, the mix randomly selects a token and removes it from its bin. An *expected inter-packet interval* (EIPI) is chosen randomly from the range represented by that bin. If another packet does *not* arrive before EIPI expires, the mix sends a dummy packet to the next mix, and selects a new token (the old token is *not* replaced). If a packet arrives before EIPI expires, this means that the mix has chosen a "wrong" interval. It places the token back into its bin, calculates the actual interval between the new packet and its predecessor, removes a token from the bin corresponding to that interval, and forwards the packet. This is done to avoid skewing the actual inter-packet interval distribution towards short values. A new token is then randomly chosen, and so on.

The number of tokens decreases as more packets arrive on the connection. Rarely, on an unusually long-lived connection, a packet may arrive after an interval that falls into an empty bin b_i. In this case, to avoid significantly affecting statistical characteristics of the resulting flow, the token is removed from the next non-empty bin b_j such that $i < j$. Alternatively, all bins can be re-filled from the distribution; the difference between the two methods is negligible in practice.

If all bins are empty, they are re-filled using a new sample from the distribution of inter-packet intervals, and the distribution itself updated, if necessary.

This adaptive padding algorithm repairs gaps that may have been caused by intentional or unintentional packet drops. The algorithm is probabilistic, and may randomly select a very short EIPI when there is a "legitimate" gap in the packet flow (*e.g.*, the sender's application is silent). This may result in tremendous amounts of cover traffic. One solution is to ignore the bins corresponding to low intervals in the token selection algorithm, *i.e.*, to only insert dummy packets into relatively large gaps. We investigate the tradeoff between padding ratios and effectiveness of adaptive padding against traffic analysis in section 5.2.

Destroying natural fingerprints. Real packet flows tend to be bursty, and each flow naturally has a unique pattern of inter-packet intervals which can be used as its "fingerprint" by the attacker in order to distinguish two or more flows. This fingerprint can be eliminated by dispatching all packets at the same constant rate, but this imposes prohibitive extra latency (see section 7).

The basic algorithm described above selects a new EIPI after receiving a packet or after the previous EIPI expired. To destroy natural patterns, however, gaps should be filled without adding packets where traffic is already dense. We thus propose a more sophisticated *dual-mode* algorithm.

(Burst) After a packet has been received, a new expected inter-packet interval is selected only from *higher* bins, *i.e.*, those associated with long intervals. This collection of bins will be referred to as the *High-Bins Set* (HBS).

(Gap) After the previously chosen expected inter-packet interval expired without receiving a packet and a dummy packet has been sent, the new interval is selected only from *lower* bins, *i.e.*, those associated with short intervals. This collection of bins will be referred to as the *Low-Bins Set* (LBS).

Intuitively, when the flow contains a natural burst, the next expected interval is chosen to be long to decrease the chance of introducing a dummy packet where packet density is already high. When the flow contains a natural gap, the next interval is short, to increase the chance of introducing a dummy packet. HBS and LBS can be configured based on observed traffic characteristics (see below).

Basic adaptive padding does not impose any extra latency on real packets. (A small delay is needed to defeat active attacks, as described in section 6.) Extra processing at each mix consists of generating a random number for each received packet, and, if necessary, drawing a dummy packet from its store. This is negligible compared to decryption and re-encryption that must be performed on each packet. The exact amount of dummy traffic is a parameter of the system, and depends on the desired effectiveness against timing analysis (see section 5.2).

5 Experimental Evaluation

We evaluate four schemes: undefended, defensive dropping [17], a variant of defensive dropping with constant-rate *cover* traffic (*i.e.*, real traffic is not delayed, only dummy packets are sent at constant rate), and adaptive padding. For each scheme, attacker's crossover error rate is computed on 3000 simulated flows.

To simplify comparisons with [17], we use the same model for network links. The packet drop rate is drawn from an exponential distribution with the mean of either 5% (between the sender and the first mix), or 1% (mix-to-mix links).

The average delay d is uniformly random between 0 and 100 ms. The actual delay for each packet is drawn from an exponential distribution with mean d.

The traffic model in our simulations is based on actual TCP traffic traces from Lawrence Berkeley Laboratory [16] and NLANR archives [21]. We found that our results are consistent across all traces. The data presented in the rest of the paper are based on the NLANR Auckland-VIII data set [20]. The distribution of inter-packet intervals within a single TCP connection, used by the adaptive padding algorithm in our simulations, was also extracted from these traces.

To simulate application traffic, we select an inter-packet interval at random from the distribution, generate a "real" packet after the interval expires, and repeat. For adaptive padding and undefended simulations, this is exactly the traffic emitted by the sender. For the variant of defensive dropping with constant-rate cover traffic, a real packet is sent as soon as it is generated, and a dummy packet is sent at some constant rate. For standard defensive dropping, both real and dummy packets are sent at the same constant rate, *i.e.*, real packets are placed in a queue until it's time to send them. We do *not* model TCP acknowledgements, re-transmissions, exponential backoff in response to dropped packets, and other TCP features that may be exploited by a sophisticated attacker. Our simple model is sufficient to demonstrate the power of inter-packet interval correlation.

In the undefended simulation, each mix simply forwards packets to the next mix. With defensive dropping, the first mix drops a dummy packet with probability 0.6. With adaptive padding, mixes inject dummy packets using the dual-mode algorithm of section 4 and the pre-set distribution of inter-packet intervals.

5.1 Attack Model

The attacker observes a set of *entry* links and a set of *exit* links with the goal to determine which exit link corresponds to which entry link.

Attacker's observation time is set to 60 seconds, enough to defeat previously proposed defenses. Increasing observation time reduces effectiveness of *any* defense, including ours. Even small statistical differences between packet flows can be detected if the observation time is long enough. We conjecture that long-term attacks cannot be prevented without assuming that some senders or mixes emit cover traffic in perfect synchrony, which cannot be achieved by any real system.

The attacker divides time into windows of W seconds [17]. Empirically, $W = 1$ gives the most accurate results. For each observed link, the attacker counts the number of packets x_k during the kth window, producing a sequence of packet counts for each link. For every possible entry-exit pair, he computes the cross-correlation of the two sequences as $r(d) = \frac{\sum_i ((x_i - \mu)(x'_{i+d} - \mu'))}{\sqrt{\sum_i (x_i - \mu)^2}\sqrt{\sum_i (x'_{i+d} - \mu')^2}}$, where delay $d = 0$ and μ, μ' are the means of the two sequences.

If the correlation $r(d)$ for a pair of links exceeds some threshold t, the attacker determines that the links carry the same flow. Otherwise, he determines that they carry different flows. For a given t, the false positive rate is the fraction of pairs that carry different flows but were erroneously determined to carry the same flow; the false negative rate is the fraction of same-flow pairs that were erroneously determined to carry different flows. The attacker chooses t so that the false positive and false negative rates are equal. This is the attacker's crossover error rate. High crossover rate means that the defense is effective.

5.2 Evaluation Results

An overview of our results is in fig. 2. The crossover rates for defensive dropping and adaptive padding are from configurations that produce, on average, one dummy packet for each real packet (1:1 padding ratio). The constant rate for both types of defensive dropping is 6.67 packets per second.

Undefended. Timing analysis is extremely effective against unmodified network flows. (Recall that only inter-packet intervals are unmodified; all other defenses, including encryption, are deployed, and no mixes are corrupted.) The crossover rate is close to 0, *i.e.*, there exists a correlation threshold that results in negligible false positive and false negative rates. Inter-packet intervals on two links of the same path have correlation close to 0.9 vs. less than 0.3 for unrelated links.

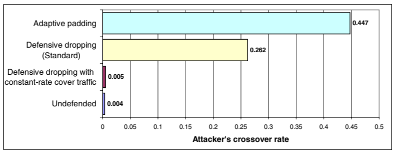


Fig. 2. Comparison of four defenses against timing analysis

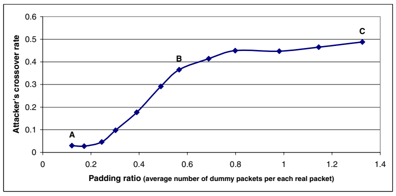


Fig. 3. Padding ratio vs. effectiveness against timing analysis

Defensive dropping with constant-rate cover. The variant of defensive dropping in which dummy packets are sent at constant rate while real packets are sent as soon as they are produced by the application does not provide much protection, with the crossover rate close to 0. Constant-rate cover traffic may hide periods of inactivity, but does not eliminate patterns of inter-packet intervals.

Adaptive padding and defensive dropping. Defensive dropping and adaptive padding are the two defenses that offer some protection against timing analysis, increasing the attacker's crossover rate to above 0.25 in our simulations.

The two defenses work for different reasons. Defensive dropping decreases correlations within the same flow to $0.4 - 0.6$, while raising (erroneous) correlations of different flows to $0.3 - 0.5$. This is due to constant-rate cover traffic, which causes all flows to look similar initially. For defensive dropping to work, other flows must be sent at the *same constant rate* as the protected flow.

Adaptive padding, on the other hand, raises correlation between different flows only to $0.1 - 0.3$, and most of the defensive effect is due to lowering correlation within the same flow to $0.2 - 0.4$. Adaptive padding thus provides *standalone* defense for a flow even if other flows do not use any defenses.

Fig. 3 displays the fundamental tradeoff of adaptive padding between the padding ratio and the attacker's crossover error rate. As the padding ratio increases, the attacker's error rate goes up at the cost of increased network congestion, as more dummy packets must be generated for each real packet.

At point A in fig. 3, LBS is bins b_{4-7}, and HBS is bins above b_{12}. The error rate is raised only to .03, but only 1 dummy packet is needed per 9 real packets. Defensive dropping achieves the same 1:9 padding ratio if the constant rate is set to 3 packets per second. The resulting error rate of 0.27 is better than adaptive padding, but the average *extra latency* per packet exceeds 3.5 seconds (see fig. 6).

At point B in fig. 3, LBS is set to bins b_{3-6}, and HBS to bins above b_{11}. On average, this requires 0.56 dummy packets per each real packet and achieves 0.37 error rate with zero extra latency. By contrast, the constant rate that achieves a comparable padding ratio for defensive dropping results in (significantly worse) 0.2 error rate, with average extra latency of 1.1 seconds per packet.

At point C, LBS is bins b_{0-8}, and HBS is bins above b_{10}. The resulting padding ratio is around 1.3:1, and the attacker's error rate is 0.48, close to theoretically optimal. In our simulations, defensive dropping was unable to achieve similar error rates with padding ratios under 50:1.

Short paths. When paths are short (2 or 3 mixes) and defensive dropping is used, attacker's error rates decrease slightly. Fewer variations due to natural network delays and drops are accumulated by the flows, and distinguishing features of entry links are still pronounced on exit links, leading to higher correlations.

With adaptive padding, crossover rates decrease, too. Padding ratios decrease as well, because fewer mixes inject dummy packets, *e.g.*, at point B in fig. 3, the error rate is 0.19 with the padding ratio of 1:3. Decreasing padding ratios tend to outweigh decreasing error rates, *i.e.*, for a given padding ratio, the error rate for shorter paths is comparable or better than that for longer paths.

6 Active Attacks

In addition to passively observing network links, an *active* attacker may impose his own timing signature onto the target flow and attempt to recognize this signature on other network links. We assume that he cannot create new packets (this requires knowledge of the symmetric keys of all subsequent mixes), nor replay packets (this is easy to prevent with caching and hash checking).

Artificial gaps. The attacker may drop several consecutive packets in the target flow to create a large gap. Fig. 4 shows the results of simulating 3000 normal flows and and 3000 target flows in which the attacker drops several consecutive packets on the entry link 1 second after the flow has started.

Dropping even a small number of consecutive packets drastically decreases the effectiveness of defensive dropping. With constant rate, all flows look very similar initially, so a noticeable change to one of the flows, such as introduction of a large gap, is almost guaranteed to create a recognizable feature.

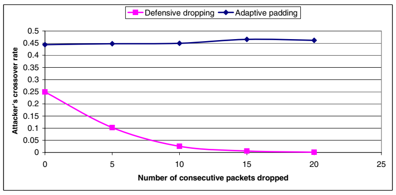


Fig. 4. Active dropping attack

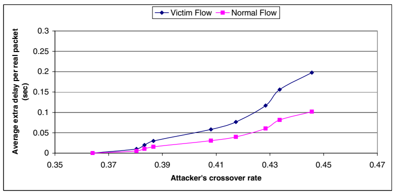


Fig. 5. Latency penalty vs. attacker's error rate for 5-sec artificial burst attack

With adaptive padding, artificial drops do not decrease the attacker's error rate. Intermediate mixes are likely to reduce the gap by injecting dummy packets. Moreover, other flows may have similar gaps due to natural variations.

Artificial bursts. The attacker can create a signature from artificial packets bursts by holding up packets on a link and then releasing all of them at once.

We simulated this attack with a 5-second and 15-second attacker's queue (while very effective, the latter is somewhat unrealistic, as it is likely to result in the higher-level TCP connection simply being dropped). Defensive dropping provides no defense: the crossover rate is 0 in both cases, *i.e.*, the attacker can perfectly identify the target flow. With adaptive padding, the crossover rate drops from .45 to .36 with a 5-second queue, and to .21 with a 15-second queue.

Our modified adaptive padding algorithm breaks up bursts by queueing all packets whose inter-arrival time is in the first bin (*i.e.*, very short). Each such packet is delayed for a short random interval. Fig. 5 shows the tradeoff between the crossover rate and the extra latency imposed on real packets. As expected, the longer the delay, the better the defense. This penalty is paid only by packets with extremely short inter-arrival times; the impact on normal flows is small.

7 Comparison with Constant-Rate Defenses

We further compare adaptive padding with constant-rate defenses, including variations such as the defensive dropping scheme of Levine *et al.* [17].

Latency is prohibitive. Constant-rate defenses fundamentally assume that senders emit traffic at a constant rate. Low-latency mix networks, however, are intended to provide anonymity for *interactive* applications such as Web browsing, and it is well-known (and borne out by real-world traces such as [21]) that Web traffic is bursty. Therefore, the sender must delay packets when the rate of actual traffic generated by the application exceeds the pre-set constant rate. Furthermore, this delay propagates to *all* subsequent packets. If the rate of real traffic exceeds the pre-set constant rate, packet delays increase to infinity.

Fig. 6 quantifies the latency penalty. For example, if the constant rate is 5 packets per second, real packets are delayed by 1.1 seconds on average. This delay may be acceptable for relatively one-directional and non-interactive applications.

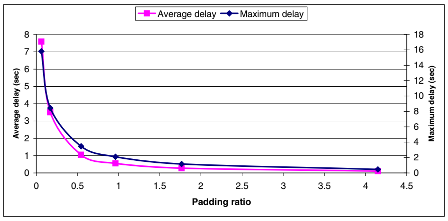


Fig. 6. Latency penalty of constant-rate defenses

Constant-rate defenses may thus be a good choice for users who are willing to tolerate increased latencies and are not worried about active attacks.

For interactive applications, however, extra latency is likely to be prohibitive. Moreover, there is evidence that low latency is essential for user adoption of anonymity systems [12]. A non-trivial increase in latency may cause fewer users to participate in the system, resulting in lower baseline anonymity.

The average *maximum* delay for each flow in our simulations is approximately 3.5 seconds, with many flows delayed by almost 10 seconds. Delays like these are likely to result in dropped TCP connections, disabling the network for all purposes (such effects are beyond the scope of our simplified model).

By contrast, adaptive padding does not impose any extra latency against a passive attacker, and only a small latency against an active attacker.

Everybody must send at the same constant rate. As observed in section 5.2, constant-rate defenses are effective only if *most* flows in the network are emitted at the same constant rate, which is clearly unrealistic. On the other hand, adaptive padding is effective in protecting a single flow even if other flows in the network do not use any defense against timing analysis.

There is no "right" constant rate. It may appear that the latency problem may be solved by setting a high constant rate which matches the shortest inter-packet interval(s) of actual traffic. Unfortunately, this is not feasible. Inter-packet intervals associated with traffic bursts are so short that the constant rate must be exceedingly high, resulting in vast amounts of dummy traffic when bursts are *not* occurring. From the network perspective, this "solution" is equivalent to taking the most congested time slice and expanding it to the entire connection.

Defense fails against active attacks. As shown in section 6, constant-rate defenses, including defensive dropping, do not provide any defense against artificial gaps or bursts introduced by an active attacker.

Defense fails at high traffic rates. With defensive dropping, only dummy packets may be dropped by intermediate mixes. If the application generates real packets at a higher rate than the chosen constant rate, most packets on each link are real and cannot be dropped. This becomes simple constant-rate traffic, which is vulnerable to basic timing analysis [14,17].

Defense reveals presence of real traffic. Even if real traffic is sparse (but bursty), the constant-rate blend of real and cover traffic produced by defensive dropping will likely consist of alternating sequences of real and dummy packets. Because only dummy packets may be dropped by intermediate mixes, the attacker can conclude that periods of sparse packets are padding and periods of dense packets are real traffic. If constant-rate cover traffic with defensive dropping is used to hide whether the connection is active or not, the attacker can break the defense simply by observing packet density.

8 Creation and Management of Dummy Packets

The main feature of our approach is that dummy packets are injected into the flow by intermediate mixes (as opposed to the route initiator), in order to "smooth out" statistical anomalies in inter-packet intervals. In the simplest case, the mix creates dummy packets itself, encrypting them with the next mix's key. The next mix decrypts the packet, recognizes it as a dummy, re-encrypts it and forwards it on. A passive attacker cannot feasibly distinguish encrypted dummy packets from encrypted real packets by observing the network. An active attacker who can compromise a mix, however, will be able to recognize dummy packets generated by the preceding mixes, negating the effects of cover traffic.

Pre-computation of dummy packets. For security in the presence of compromised mixes, injected dummy packets should be indistinguishable from real packets by all subsequent mixes. If layered encryption is used, dummy packets should be encrypted with the the same keys in the same order as real packets.

An intermediate mix does not know its successors in the mix chain, and thus cannot properly encrypt a dummy packet itself. One solution is to have the initiator *pre-compute* large batches of dummy packets for all mixes in the chain. This is done *offline*, *e.g.*, during route setup, and thus has no impact on the bandwidth and latency of actual communication. During route setup, each mix receives from the initiator a batch of dummy packets. The batch is encrypted with their shared pairwise key. Each dummy packet is properly encrypted with the keys of all successor mixes (this does not leak their identities). Whenever the mix needs to inject a dummy packet, it simply gets it from the batch. None of its successors on the route can tell a real packet from an injected dummy packet.

These batches can be replenished periodically, or, depending on the implementation, a mix may signal to the route initiator that it needs a new batch when the connection is quiet. This approach is secure against compromised mixes, and trades off storage at the intermediate mixes against online computation (since mixes no longer generate dummy packets on the fly), resulting in faster performance. It also prevents malicious mixes from flooding the connection with bogus dummy packets, because they will not decrypt properly at the successor mixes.

Injection of pre-computed packets into a stream of encrypted packets assumes that encryption is block cipher-based, as in, *e.g.*, the Freedom system [5]. If a stream cipher is used, as in onion routing [27,11], the state of the cipher used for the ith layer of encryption must be synchronized between the sender and the *i*th mix in the path. Because the sender cannot predict when an intermediate mix may need to inject a dummy packet, pre-computation is infeasible, and an alternative mechanism such as reverse paths (see below) must be used.

Malicious clients and servers. Vulnerability of real-world mix networks to timing analysis performed by malicious clients and servers has been shown by, respectively, Øverlier and Syverson [23], and Murdoch and Danezis [19]. In our case, we assume that dummy packets are sent beyond the last mix to the destination server, and that the latter can recognize and discard them.

This can sometimes be achieved without server cooperation. For example, if the sender knows that the server discards all packets with an invalid message authentication code (MAC), he can append invalid MACs to all dummy packets. Even if the attacker compromises the last mix, he does not learn the key shared between the sender and the destination, and thus cannot check MAC validity.

In general, adaptive padding requires server cooperation, and thus does not protect from malicious servers. This may appear to be a serious disadvantage viz. sender-originated cover traffic, but, in reality, many low-latency applications such as Web browsing are bidirectional, and thus also require server cooperation. For instance, if defensive dropping is used to protect HTTP connections, a malicious server can easily track propagation of its responses back to the client.

Protection offered by adaptive padding (or, in general, by any mix-injected cover traffic) is a mirror image of protection offered by sender-originated cover traffic. With the former, the last link of a mix chain is padded, but the first link is not. With the latter, the first link is padded, but the last link is not. Therefore, the former requires server cooperation, while the latter requires client cooperation (and is insecure against a malicious client).

Reverse routes. Another solution is to reverse the conventional route setup process so that the server (rather than the initiator) ends up sharing a key with each mix in the chain. The sender encrypts packets only with the server's key and sends them to the first mix on the path. Each succeeding mix encrypts the packet with the key it shares with the server. The server unwraps all onion layers of encryption. (A similar mechanism is used by location-hidden servers in Tor.) With reverse routes, intermediate mixes can easily inject dummy packets into the flow — all they need to do is simply encrypt them with the key they share with the next mix, and send them with the proper route identification.

The Øverlier-Syverson attack on hidden servers demonstrates the importance of protecting reverse paths from malicious clients. Since this cannot be done with sender-based defenses, mix-injected cover traffic is a better solution in this case.

9 Challenges and Future Directions

Attacks considered in this paper by no means exhaust the list of possible threats against low-latency mix networks. We made the standard, unrealistic assumption that all connections start at the same time. In reality, attacks based on correlating start and end times of connections may prove very successful. In general, it is very difficult to hide connection start and end times using dummy traffic because the mix network handles dummy and real packets differently (*e.g.*, dummies can or even should be dropped, while real packets are never dropped).

With adaptive padding, any intermediate mix may inject a dummy packet. The more mixes a flow has traversed, the denser it tends to become. The attacker may be able to estimate the hop count of a flow by measuring its density. This cannot always be used to attack anonymity, however. Furthermore, padding ratios are higher on the links closer to the destination. Even when the average padding ratio is low, links toward the end of the path may experience more congestion. On the flip side, the client-side links, which tend to be slower and less reliable, are free of padding. In two-way communication, where the same route with adaptive padding is used in both directions, the padding ratios balance out.

References

1. A. Back, I. Goldberg, and A. Shostack. Freedom Systems 2.1 security issues and analysis. `http://www.freehaven.net/anonbib/cache/freedom21-security.pdf`, May 2001.
2. A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Proc. 4th International Workshop on Information Hiding*, volume 2137 of *LNCS*, pages 245–257, 2001.
3. O. Berthold, H. Federrath, and S. Köpsell. Web MIXes: a system for anonymous and unobservable Internet access. In *Proc. Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *LNCS*, pages 115–129, 2000.
4. O. Berthold and H. Langos. Dummy traffic against long-term intersection attacks. In *Proc. 2nd International Workshop on Privacy-Enhancing Technologies*, volume 2482 of *LNCS*, pages 110–128, 2002.
5. P. Boucher, A. Shostack, and I. Goldberg. Freedom Systems 2.0 architecture. `http://www.freehaven.net/anonbib/cache/freedom2-arch.pdf`, December 2000.
6. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.
7. G. Danezis. Statistical disclosure attacks. In *Proc. Security and Privacy in the Age of Uncertainty*, volume 250 of *IFIP Conference Proceedings*, pages 421–426, 2003.
8. G. Danezis. The traffic analysis of continuous-time mixes. In *Proc. 4th International Workshop on Privacy-Enhancing Technologies*, volume 3424 of *LNCS*, pages 35–50, 2004.
9. G. Danezis, R. Dingledine, and N. Mathewson. Mixmixion: Design of a type III anonymous remailer protocol. In *Proc. IEEE Symposium on Security and Privacy*, pages 2–15, 2003.
10. G. Danezis and A. Serjantov. Statistical disclosure or intersection attacks on anonymity systems. In *Proc. 6th International Workshop on Information Hiding*, volume 3200 of *LNCS*, pages 293–308, 2004.
11. R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proc. 13th USENIX Security Symposium*, pages 303–320, 2004.
12. R. Dingledine, N. Mathewson, and P. Syverson. Challenges in deploying low-latency anonymity. NRL CHACS Report 5540-265, 2005.
13. M. Freedman and R. Morris. Tarzan: a peer-to-peer anonymizing network layer. In *Proc. 9th ACM Conference on Computer and Communications Security*, pages 193–206, 2002.

14. X. Fu, B. Graham, R. Bettati, and W. Zhao. On effectiveness of link padding for statistical traffic analysis attacks. In *Proc. 23rd IEEE Conference on Distributed Computing Systems*, pages 340–349, 2003.
15. X. Fu, B. Graham, R. Bettati, W. Zhao, and D. Xuan. Analytical and empirical analysis of countermeasures to traffic analysis attacks. In *Proc. 32nd International Conference on Parallel Processing*, pages 483–492, 2003.
16. Internet Traffic Archive. Two hours of wide-area TCP traffic. `http://ita.ee.lbl.gov/html/contrib/LBL-TCP-3.html`, January 1994.
17. B. Levine, M. Reiter, C. Wang, and M. Wright. Timing attacks in low-latency mix systems. In *Proc. 8th International Conference on Financial Cryptography*, volume 3110 of *LNCS*, pages 251–265, 2004.
18. U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster protocol version 2. `http://www.abditum.com/mixmaster-spec.txt`, July 2003.
19. S. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *Proc. IEEE Symposium on Security and Privacy*, pages 183–195, 2005.
20. National Laboratory for Applied Network Research. Auckland-VIII data set. `http://pma.nlanr.net/Special/auck8.html`, December 2003.
21. National Laboratory for Applied Network Research. PMA special traces archive. `http://pma.nlanr.net/Special/`, 2005.
22. R. Newman-Wolfe and B. Venkatraman. High level prevention of traffic analysis. In *Proc. IEEE/ACM 7th Annual Computer Security Applications Conference*, pages 102–109, 1991.
23. L. Øverlier and P. Syverson. Locating hidden servers. In *Proc. IEEE Symposium on Security and Privacy*, 2006.
24. M. Rennhard, S. Rafaeli, L. Mathy, B. Plattner, and D. Hutchison. Analysis of an anonymity network for Web browsing. In *Proc. IEEE WETICE*, pages 49–54, 2002.
25. A. Serjantov and P. Sewell. Passive attack analysis for connection-based anonymity systems. In *Proc. 8th European Symposium on Research in Computer Security*, volume 2808 of *LNCS*, pages 116–131, 2003.
26. V. Shmatikov. Probabilistic analysis of an anonymity system. *J. Computer Security*, 12(3-4):355–377, 2004.
27. P. Syverson, D. Goldschlag, and M. Reed. Anonymous connections and onion routing. In *Proc. IEEE Symposium on Security and Privacy*, pages 44–54, 1997.
28. P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an analysis of onion routing security. In *Proc. Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *LNCS*, pages 96–114, 2000.
29. B. Timmerman. A security model for dynamic adaptable traffic masking. In *Proc. New Security Paradigms Workshop*, pages 107–116, 1997.
30. B. Timmerman. Secure adaptive traffic masking. In *Proc. New Security Paradigms Workshop*, pages 13–24, 1999.
31. B. Venkatraman and R. Newman-Wolfe. Performance analysis of a method for high level prevention of traffic analysis using measurements from a campus network. In *Proc. IEEE/ACM 10th Annual Computer Security Applications Conference*, pages 288–297, 1994.
32. M. Wright, M. Adler, B. Levine, and C. Shields. An analysis of the degradation of anonymous protocols. In *Proc. ISOC Network and Distributed System Security Symposium*, 2002.
33. Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao. On flow correlation attacks and countermeasures in mix networks. In *Proc. 4th International Workshop on Privacy-Enhancing Technologies*, volume 3424 of *LNCS*, pages 207–225, 2004.

TrustedPals: Secure Multiparty Computation Implemented with Smart Cards

Milan Fort[2], Felix Freiling[3], Lucia Draque Penso[3]
Zinaida Benenson[1], and Dogan Kesdogan[2]

[1] Department of Information Technology, Uppsala University, SE-751 05 Uppsala, Sweden
[2] Computer Science Department, RWTH Aachen University, D-52056 Aachen, Germany
[3] Computer Science Department, University of Mannheim, D-68131 Mannheim, Germany

Abstract. We study the problem of *Secure Multi-party Computation* (*SMC*) in a model where individual processes contain a tamper-proof security module, and introduce the *TrustedPals* framework, an efficient smart card based implementation of *SMC* for any number of participating entities in such a model. Security modules can be trusted by other processes and can establish secure channels between each other. However, their availability is restricted by their host, that is, a corrupted party can stop the computation of its own security module as well as drop any message sent by or to its security module. We show that in this model SMC can be implemented by reducing it to a fault-tolerance problem at the level of security modules. Since the critical part of the computation can be executed locally on the smart card, we can compute any function securely with a protocol complexity which is polynomial only in the number of processes (that is, the complexity does not depend on the function which is computed), in contrast to previous approaches.

1 Introduction

Motivation. The problem of *Secure Multi-party Computation* (*SMC*, sometimes also referred to as *Secure Function Evaluation*), is one of the most fundamental problems in security. The setting is as follows: a set of n parties jointly wants to compute the result of an n-ary function F. Every party provides its own (private) input to this function but the inputs should remain secret to the other parties, except for what can be derived from the result of F. The problem is easy to solve if you assume the existence of a trusted third party (TTP) which collects the inputs, computes F and distributes the result to everyone. However, the problem is very challenging if you assume that there is no TTP available and parties can misbehave arbitrarily, that, they can send wrong messages or fail to send messages at all. Still, the protocol must correctly and securely compute F as if a TTP were available.

Having been initially proposed by Yao in 1982 [28], it got its first solution only in 1987, when Goldreich, Micali and Wigderson [15] showed that in a synchronous system with cryptography a majority of honest processes can simulate a centralized trusted third party. This was done by transforming the function F into a computation over a finite field and then showing that addition and multiplication in this finite field could be implemented securely using secret sharing and agreement protocols. It was also shown that a majority of honest processes was necessary for SMC.

All existing general solutions to SMC are based on the original idea of Goldreich, Micali and Wigderson [15]. Hence, the message complexity always depends on the function that is computed. For example, the most efficient solution to SMC we are aware of [17] requires communicating $O(m \cdot n^3)$ field elements (m is the number of multiplication gates in F) and at least $O(n^2)$ rounds of communication (in fact, the round complexity also depends on F). Thus, despite solutions, many practictioners have been prevented to attempting to implement general SMC due to lack of efficiency.

Recently, there has been an increasing interest in SMC which probably stems from the growing importance and the difficulty to implement fault-tolerance in combination with security in today's networks. In fact, in the concluding remarks on the COCA project, Zhou, Schneider and van Renesse [29] call to investigate practical secure multi-party computation.

Related Work. In 2003, MacKenzie, Oprea and Reiter [20] presented a tool which could securely compute a two-party function over a finite field of a specific form. Later, Malkhi *et al.* [21] presented *Fairplay*, a general solution of two-party secure computation. Both papers follow the initial approach proposed by Goldreich, Micali and Wigderson [15], that is, they make extensive use of compilers that translate the function F into one-pass boolean circuits. Iliev and Smith [18] report in yet unpublished work on performance improvements using trusted hardware. In this paper we revisit SMC in a similar model but using a different approach.

The approach we use in this paper was pioneered by Avoine and Vaudenay [4]. It assumes a synchronous model with no centralized TTP, but the task of jointly simulating a TTP is alleviated by assuming that parties have access to a local *security module* (Avoine and Vaudenay [4] call this a *guardian angel*). Recently, manufacturers have begun to equip hardware with such modules: these include for instance smart cards or special microprocessors. These are assumed to be tamper proof and run a certified piece of software. Examples include the Embedded Security Subsystem within the recent IBM Thinkpad or the IBM 4758 secure co-processor board [10]. A large body of computer and device manufacturers has founded the Trusted Computing Group (TCG) [27] to promote this idea. Security modules contain cryptographic keys so that they can set up secure channels with each other. However, they are dependant on their hosts to be able to communicate with each other.

Later, Avoine *et al.* [3] showed that, in a model with security modules, the fair exchange problem, an instance of SMC, can be reduced to an agreement problem amoung security modules, which can itself be transformed to the consensus problem, a classical problem of fault-tolerant distributed computing. The reduction allows modular solutions to fair exchange, as the agreement abstraction can be implemented in different ways [9,13]. The problem of SMC has not yet been investigated in this model.

Contributions. In this paper, we investigate the resilience and efficiency of SMC in the model of untrusted hosts and security modules. In this model the security modules and their communication network form a subnetwork with a more benign fault assumption, namely that of general omission [24]. In the general omission failure model processes may simply stop executing steps or fail to send or receive messages sent to them.

We extend the work by Avoine *et al.* [3] and derive a novel solution to SMC in a modular way: We show that SMC is solvable if and only if the problem of Uniform Interactive Consistency (UIC) is solvable in the network of security modules. UIC is closely related to the problem of Interactive Consistency [23], a classic fault-tolerance problem. From this equivalence we are able to derive a basic impossibility result for SMC in the new model: We show that UIC requires a majority of correct processes and from this can conclude that SMC is impossible in the presence of a dishonest majority. This shows that, rather surprisingly, adding security modules cannot improve the resilience of SMC. However, we prove that adding security modules can considerably improve the efficiency of SMC protocols. This is because the computation of F can be done locally within the security modules and does not affect the communication complexity of the SMC protocol. Therefore our solution to SMC which uses security modules requires only $O(n)$ rounds of communication and $O(n^3)$ messages. To the best of our knowledge, this is the first solution for which the message and round complexity do not depend on the function which is computed.

Furthermore, we give an overview of *TrustedPals*, a peer-to-peer implementation of the security modules framework using Java Card Technology enabled smart cards [8]. Roughly speaking, in the TrustedPals framework, F is coded as a Java function and is distributed within the network in an initial setup phase. After deployment, the framework manages secure distribution of the input values and evaluates F on the result of an agreement protocol between the set of security modules. To show the applicability of the framework, we implemented the approach of Avoine *et al.* [3] for fair exchange. To our knowledge, TrustedPals is the first practical implementation of SMC (1) for *any* number of processes and (2) which does not require a transformation into and a subsequent computation in a finite field. While still experimental, the TrustedPals framework is available for download [1].

Roadmap. We first present the model in Section 2. We then define the *security* problem of secure multi-party computation (SMC) and the *fault-tolerance problem* of uniform interactive consistency (UIC) in Sections 3 and 4. The security problems arising when using UIC within a SMC protocol are discussed in Section 5. We present the equivalence of SMC and UIC, in Section 6, and finally describe the TrustedPals efficient framework in Section 7.

2 Model

2.1 Processes and Channels

The system consists of a set of processes interconnected by a synchronous communication network with reliable secure bidirectional channels. Two processes connected by a channel are said to be adjacent.

A reliable secure channel connecting processes P and Q satisfies the following properties:

- (No Loss) No messages are lost during the transmission over the channel.
- (No Duplication) All messages are delivered at most once.
- (Authenticity) If a message is delivered at Q, then it was previously sent by P.
- (Integrity) Message contents are not tampered with during transmission, i.e., any change during transmission will be detected and the message will be discarded.
- (Confidentiality) Message contents remain secret from unauthorized entities.

In a synchronous network communication proceeds in rounds. In each round, a party first receives inputs from the user and all messages sent to it in the previous round (if any), processes them and may finally send some messages to other parties or give outputs to the user.

2.2 Untrusted Hosts and Security Modules

The set of processes is divided into two disjoint classes: *untrusted hosts* (or simply *hosts*) and *security modules*. We assume that there exists a fully connected communication topology between the hosts, i.e., any two hosts are adjacent. We denote by n the number of hosts in the system. Furthermore, we assume that every host process H_A is adjacent to exactly one security module process M_A (there is a bijective mapping between security modules and hosts). In this case we say that H_A is *associated with* M_A (M_A is H_A's associated security module). We call the part of the system consisting only of security modules and the communication links between them the *trusted system*.

We call the part of the system consisting only of hosts and the communication links between them the *untrusted system*. The notion of association can be extended to systems, meaning that for a given untrusted system, the *associated trusted system* is the system consisting of all security modules associated to any host in that untrusted system (see Figure 1).

In some definitions we use the term process to refer to both a security module and a host. We do this deliberately to make the definitions applicable both in the trusted and the untrusted system.

2.3 Relation to Systems with Trusted Hardware

The model sketched above can be related to the setup in practice as follows: untrusted hosts model Internet hosts and their users, whereas security modules

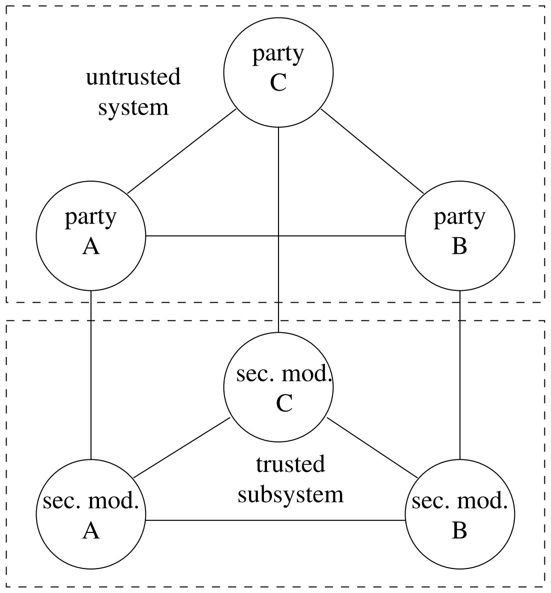


Fig. 1. Hosts (parties) and security modules

abstract tamper proof components of user systems (like smart cards). Intuitively, security modules can be trusted by other security modules or hosts, and hosts cannot be trusted by anybody. Hosts may be malicious, i.e., they may actively try to fool a protocol by not sending any message, sending wrong messages, or even sending the right messages at the wrong time.

Security modules are supposed to be cheap devices without their own source of power. They rely on power supply from their hosts. In principle, a host may cut off the power supply to its security module whenever he chooses, thereby preventing the security module from continuing to execute steps. Instead of doing this, a host may inhibit some or even *all* communication between its associated security module and the outside world.

2.4 Trust and Adversary Model

The setting described above is formalized using distinct failure models for different parts of the system. We assume that nodes in the untrusted system can act arbitrarily, i.e., they follow the Byzantine failure model [19]. In particular, the incorrect processes can act together according to some sophisticated strategy, and they can pool all information they possess about the protocol execution. We assume however that hosts are computationally bounded, i.e., brute force attacks on secure channels are not possible.

For the trusted system we assume the failure model of *general omission* [24], i.e., processes can crash or fail by not sending messages or not receiving messages.

A process is *faulty* if it does not correctly follow the prescribed protocol. In particular, a security module is faulty if it crashes or commits send or receive omissions. Otherwise a process is said to be *correct.* In a system with n processes, we use t to denote a bound on the number of hosts which are allowed to be faulty.

3 Secure Multi-party Computation

In *secure multi-party computation* (SMC), a set of processes $p_1, \ldots, p_n$, each starting with an input value x_i, wants to compute the result of a deterministic function F, i.e., $r = F(x_1, \ldots, x_n)$. Result r should be computed reliably and securely, i.e., as if they were using a trusted third party (TTP). This means that the individual inputs remain secret to other processes (apart from what is given away by r) and that malicious processes can neither prevent the computation from taking place nor influence r in favorable ways.

We assume that F is a well-known deterministic function with input domain X and output domain Y upon which all processes have agreed upon beforehand and that all correct processes jointly begin the protocol. We say that r is an F*-result* if r was computed using F. Since faulty processes cannot be forced to submit their input value, F may be computed using a special value $\perp \notin X$ instead of the input value of a faulty process.

Instead of defining SMC using a TTP [14], we now define SMC using a set of abstract properties.

Definition 1 (secure multi-party computation). *A protocol solves* secure multi-party computation (SMC) *if it satisfies the following properties:*

- *(SMC-Validity) If a process receives an F-result, then F was computed with at least the inputs of all correct processes.*
- *(SMC-Agreement) If some process p_i receives F-result r_i and some process p_j receives F-result r_j then $r_i = r_j$.*
- *(SMC-Termination) Every correct process eventually receives an F-result.*
- *(SMC-Privacy) Faulty processes learn nothing about the input values of correct processes (apart from what is given away by the result r and the input values of all faulty processes).*

From a security protocols perspective, the above definition can be considered slightly stronger than the usual (cryptographic) definitions of SMC since it demands that SMC-properties hold without any restriction. In the literature it is often stated that the probability of a violation of SMC-properties can be made arbitrarily small. We have chosen this stronger definition to simplify the presentation. We believe that definitions, theorems and proofs can be transferred into a probabilistic model with moderate effort.

The properties of SMC are best understood by comparing them to a solution based on a TTP. There, the TTP waits for the inputs of all n processes and computes the value of F on all those inputs which it received. Since all correct processes send their input value to the TTP, F is computed on at least those values, which motivates SMC-Validity. After computing F, the TTP sends the result back to all processes. Hence, all correct processes eventually receive that result (SMC-Termination). Additionally, if a process receives a result from the TTP, then it will be the same result which any other process (whether correct or faulty) will receive. This motivates SMC-Agreement. SMC-Privacy is motivated by the fact that the TTP does all the processing and channels to the TTP are confidential: no information about other processes' input values leaks from this idealized entity, apart of what the result of F gives away when it is finally received by the processes.

4 Uniform Interactive Consistency

The problem of *Interactive Consistency* (IC) was introduced by Pease, Shostak and Lamport in 1980 [23]. It is one of the classical problems of reliable distributed computing since solutions to this problem can be used to implement almost any type of fault-tolerant service [26]. In this problem, every process starts with an initial value v_i. To solve the problem, the set of processes needs to agree on a vector D of values, one per process (Agreement property). Once vector D is output by process p, we say that p *decides* D. The i-th component of this vector should be v_i if p_i does not fail, and can be $\perp$ otherwise. IC is equivalent to the (also classic) *Byzantine Generals Problem* [19].

Definition 2 considers a version of IC with a stronger agreement property called *Uniform Agreement*. Uniform Agreement demands that *all* processes should decide the same (if they decide) — it does not matter whether they are correct or faulty.

Definition 2 (uniform interactive consistency). *A protocol solves* uniform interactive consistency (UIC) *if it satisfies the following properties:*

- *(UIC-Termination) Every correct process eventually decides.*
- *(UIC-Validity) The decided vector D is such that $D[i] \in \{v_i, \perp\}$, and is v_i if p_i is not faulty.*
- *(UIC-Uniform Agreement) No two different vectors are decided.*

Parvédy and Raynal [22] studied the problem of UIC in the context of general omission failures. They give an algorithm that solves UIC in such systems provided a majority of processes is correct. Since their system model is the same as the one used for trusted systems in this paper, we conclude:

Theorem 1 ([22]). *If $t < n/2$ then UIC is solvable in the trusted system.*

Parvédy and Raynal also show that Uniform Consensus (UC), a problem closely related to UIC, can be solved in the omission failure model only if $t < n/2$. In Uniform Consensus, instead of agreeing on a vector of input values like in UIC, all processes have to decide on a single value which must be input value of some process. Given a solution to UIC, the solution to UC can be constructed in the following way: the processes first solve UIC on their input values and then output the first non-$\perp$ element of the decided vector as the result of UC. Thus, we conclude:

Corollary 1. *UIC is solvable in the trusted system only if $t < n/2$.*

5 Maintaining Secrecy in Trusted Systems

The problem of UIC, which was introduced in the previous section, will be used as a building block in our solution to SMC. The idea is that a protocol for SMC will delegate certain security-critical actions to the trusted system in which the UIC protocol runs. In this section we argue that we have to carefully analyze the security properties of the protocols which run within the trusted system in order to maintain confidentiality and be able to implement SMC-Privacy.

5.1 Example of Unauthorized Information Flow

Assume that a host H_A hands its input value v_A of SMC to its associated security module M_A and that M_A sends v_A over a secure channel to the security module M_B of host H_B. Since all communication travels through H_B, H_B can derive some information about v_A even if all information is encrypted. For example, if no special care is taken, H_B could deduce the size (number of bits) of v_A by observing the size of the ciphertext of v_A. This may be helpful to exclude certain choices of v_A and narrow down the possibilities in order to make a brute-force attack feasible.

As another example, suppose M_A only sends v_A to M_B if v_A (interpreted as a binary number) is even. Since we must assume that H_B knows the protocol which is executed on M_A and M_B, observing (or not observing) a message on the channel at the right time is enough for M_B to deduce the lowerest order bit of v_A. In this example, the control flow of the algorithm (exhibited by the message pattern) unintentionally leaks information about secrets.

5.2 Security Properties of Protocols in the Trusted System

A protocol running in the trusted system needs to satisfy two properties to be of use as a building block in SMC:

- (Content Secrecy) Hosts cannot learn any useful information about other hosts' inputs from observing the *messages* in transit.
- (Control Flow Secrecy) Hosts cannot learn any useful information about other host's inputs from observing the *message pattern.*

To provide these two secrecy properties in general, it is sufficient to use a communication protocol between the processes that ensures *unobservability.* Unobservability refers to the situation when an adversary cannot distinguish meaningful protocol actions from "random noise" [25]. In particular, unobservability assumes that an adversary knows the protocol which is running in the underlying network. It demands that despite this knowledge and despite observing the messages and the message pattern on the network it is impossible for the adversary to figure out in what state the protocol is. The term “state” refers to all protocol variables including the program counter, e.g., the mere fact whether the protocol has started or has terminated must remain secret.

Definition 3 (unobservability). *A protocol satisfies* unobservability *if an unauthorized entity which knows the protocol cannot learn any information about the state of the protocol.*

Obviously, if unobservability is fulfilled during the application of UIC then an adversary cannot obtain any information which may be derived from the control flow of the algorithm and therefore Content Secrecy and Control Flow Secrecy are fulfilled.

There are known techniques in the area of unobservable communication that guarantee perfect unobservability [25]. It goes without saying that unobservability techniques are not for free. However, the cost does not depend on the function F and depends only polynomially on the number of processes (i.e. number of real messages).

Unobservability as well as Content and Control Flow Secrecy are sufficient to maintain secrecy in the trusted subsystem. However, Control Flow Secrecy is sometimes not necessary. In the fair exchange implementation of Avoine *et al.* [3] it was shown that, to ensure security, it is sufficient that the adversary does not know when the agreement protocol is in its *final* round. The adversary may know whether the protocol is running or not.

This (weaker) form of Control Flow Secrecy was implemented using the idea of *fake rounds*. In the first round of the protocol, a random number ρ is distributed amoung the security modules. Before actually executing the agreement protocol, ρ fake rounds are run. Using encryption and padding techniques, it is impossible for the adversary to distinguish a fake round from an actual round of the agreement protocol.

6 Solving SMC with Security Modules

6.1 Relating SMC to UIC

The following theorem shows that SMC and UIC are “equivalent” in their respective worlds. The idea of the proof is to distribute the input values to the function F within the trusted subsystem using UIC and then evaluate F on the resulting vector. The functional properties of SMC correspond to those of UIC while the security properties of SMC are taken care of the properties of the security modules and the fact that the UIC protocol can be made to operate in an unobservable way.

Theorem 2. *SMC is solvable for any deterministic F in the untrusted system if and only if UIC is solvable in the associated trusted system.*

```
**SMC**(*input* $x_i$)
    $D := secureUIC(x_i)$
    **return** $F(D)$
```

Fig. 2. Implementing SMC using UIC on security modules. Code for the security module of host H_i. The term "secure UIC" refers to a UIC protocol that satisfies Content Secrecy and Control Flow Secrecy.

```
**UIC**(*input* $v_i$)
    $D := SMC_F(v_i)$
    **return** $D$
```

Fig. 3. Implementing UIC on security modules using SMC for the function $F(v_1, \ldots, v_n) = (v_1, \ldots, v_n)$ in the untrusted system. Code for the security module of host H_i.

Proof. ($\Leftarrow$) We first prove that the solvability of UIC in the trusted system implies the solvability of SMC in the untrusted system. Fig. 2 shows the transformation protocol which is executed within the security module. The hosts first give their inputs for SMC to their security modules. Security modules run "secure UIC" (i.e., UIC which satisfies Message Secrecy and Control Flow Secrecy) on these inputs, compute F on the decided vector and give the result to their hosts. We prove that the properties of SMC are achieved.

First consider *SMC-Validity.* UIC-Termination guarantees that all correct processes eventually decide on some vector D. UIC-Validity guarantees that D contains the inputs of all correct processes, and hence, SMC-Validity holds for the output of the transformation algorithm.

Consider *SMC-Agreement.* From UIC-Uniform Agreement it follows that all processes decide on the same vector. As F is deterministic, all processes compute the same F-result if they compute such a result.

SMC-Termination follows immediately from UIC-Termination.

Now consider *SMC-Privacy.* Since secure UIC is executed (i.e., the construction and proof techniques presented in Section 5 have been applied to ensure Content Secrecy and Control Flow Secrecy) and because security modules are tamper-proof, we conclude that there is no unauthorized information flow from within the trusted system to the outside (i.e., to the untrusted system). The only (authorized) flow of information occurs at the interface of the security modules when they output the result of computing F. SMC-Privacy easily follows from this observation.

($\Rightarrow$) We now prove that if SMC is solvable in the untrusted system, then UIC is solvable in the trusted system.

First note that if SMC is solvable in the untrusted system, then SMC is trivially also solvable in the trusted system. This is because the assumptions available to the protocol are much stronger (general omission failures instead of Byzantine).

To solve UIC, we let the processes compute the function $F(v_1, \ldots, v_n) = (v_1, \ldots, v_n)$ (see Fig. 3). We now show that the properties of UIC follow from the properties of SMC.

UIC-Termination follows immediately from SMC-Termination.

To see *UIC-Validity*, consider the decided vector $D = (d_1, \ldots, d_n)$. SMC-Validity and the construction of Fig. 3 guarantee that D contains the inputs of all correct processes. Consider a faulty process p_j with input value v_j. Then either F was computed using its input, and then $d_j = v_j$, or, according to our definition of SMC, function F was computed using a special input value $\perp$ instead of v_j, and then, $d_j = \perp$.

To see *UIC-Uniform Agreement* follows directly from SMC-Agreement.

This concludes the proof. □

Theorem 2 allows us to derive a lower bound on the resilience of SMC in the given system model using Theorem 1.

Corollary 2. *There is no solution to SMC in the untrusted system if* $t \geq n/2$.

6.2 Analysis

Theorem 2 and Corollary 2 show that adding security modules cannot improve the resilience of SMC compared to the standard model without trusted hardware [15]. For the model of perfect security (i.e., systems without cryptography) our secure hardware has a potential to improve the resilience from a two-thirds majority [5,7] to a simple majority. However, this would rely on the assumption that security modules can withstand *any* side-channel attack, an assumption which can hardly be made in practice.

Since F is computed locally, the main efficiency metric for our solution is message and round complexity of the underlying UIC protocol. For example, the worst case message complexity of the protocol of Parvédy and Raynal [22] is $O(n^3)$ and the worst case round complexity if $O(n)$ even if modifications for unobservable communication are added [6]. This is in contrast to the most efficient solution to SMC without secure hardware which requires at least $O(n^2)$ rounds and $O(mn^3)$ messages where m is the number of multiplications in F [17].

7 TrustedPals Implementation

We now report on the implementation of the trusted subsystem using smart cards [1]. Our implementation is fundamentally a peer-to-peer distributed system. Each peer consists of two parts, the security module and its (untrusted) host application. It leverages the power of different programming platforms and software technologies.

7.1 Programming Platforms and Technologies

The security module is realized using a Java Card Technology enabled smart card, whereas its (untrusted) host application is implemented as a normal Java desktop application.

The Java Card Technology defines a subset of the Java platform for smart cards and allows application developers to create and install smart card applications on their own, even after the smart card was manufactured. Multiple Java Card applications (so-called *Applets*) from different vendors can run on the same smart card, without compromising each other's security. The communication between the host computer and the smart card is a half-duplex, master-slave model. In Java Card Technology, there are two programming models used for communication with the smart card, the *APDU* message-passing model and the Java Card Remote Method Invocation (*JCRMI*), a subset of Java SE RMI distributed-object model. Though our current implementation uses the APDU model, we plan to migrate to JCRMI. For more information about the Java Card Technology, we refer the interested readers to Chen [8].

The host part of the application is implemented in Java, using the *Spring Framework* [12]. Spring is a lightweight, dependency injection inversion of control container that allows us to easily configure and assemble the application components. For more on dependency injection and inversion of control containers, please refer to Fowler [11].

The *OpenCard Framework* [16] is used by the host part of the application for communication with the smart card, whereas the *JMS* (Java Message Service) is used for communication with the other hosts. JMS is a standard Java API, part of Java Platform, Enterprise Edition, for accessing enterprise messaging systems. It allows the applications to communicate in a loosely coupled, asynchronous way. Besides, the provided infrastructure supports different quality of service, fault tolerance, reliability, and security requirements. As a JMS provider, we have used *ActiveMQ* [2]. It is an open source, 100% compliant JMS 1.1 implementation, written in Java. ActiveMQ can be seamlessly integrated and configured through Spring. What is more, its support for *message-driven POJOs* concept and the Spring's JMS abstraction layer makes much easier the development of message applications. Another interesting feature of ActiveMQ is its concept called *Networks of Brokers*. It allows us to start a broker for each host; these brokers then interconnect with each other and form a cluster of brokers. Thus, a failure of any particular host does not affect the other hosts.

7.2 Architecture

The overall (simplified) architecture of our project is depicted in Figure 4, where relations between components are depicted. The components comprise the Protocol on the smart card, the Controller, the Clock, the Message Communication Unit and the Adversary which operate on the host.

The Protocol on the smart card in fact consists of several parts: (1) a Java implementation of the function F which is to be evaluated, and (2) an implementation of UIC which satisfies sufficient secrecy properties as explained above.

All messages generated during the execution of the protocol are encrypted and padded to a standard length and sent over the APDU interface to the host. The Communication Unit takes care of distributing these messages to other hosts using standard Internet technology. The Controller and the Clock are under total

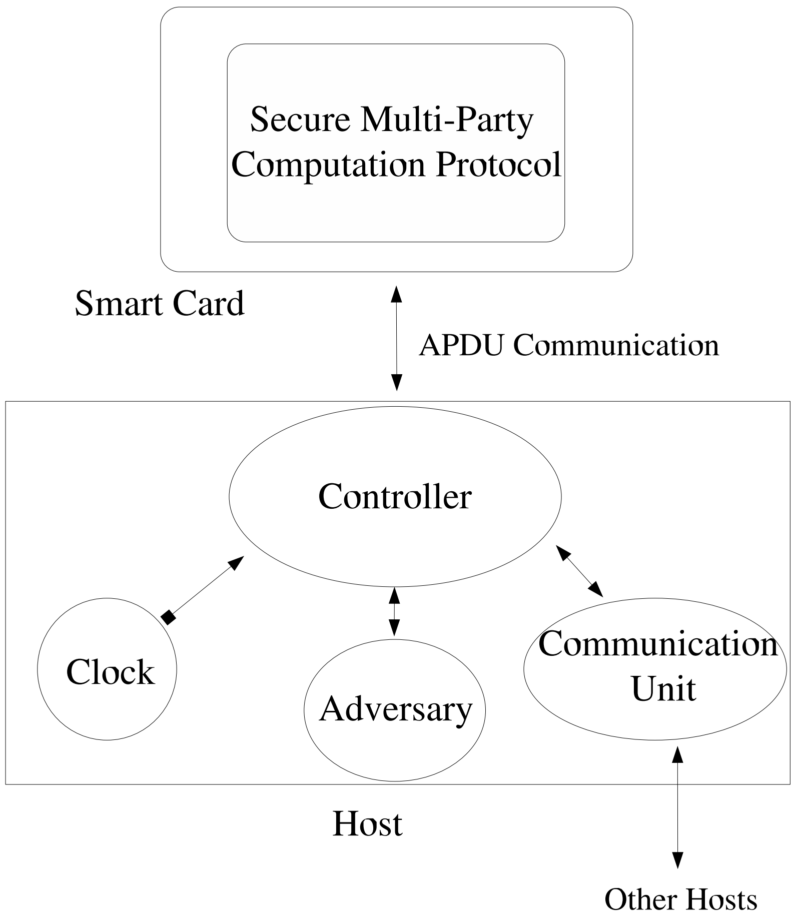


Fig. 4. Relations between architecture components

control of the host. This is acceptable as tampering with them would just result in message deletion or in disturbing the synchronous time intervals, which might cause message losses as well. That would be no problem, since the adversary has power to destroy messages stored in the Message Communication Unit, due to the general omission model nature, where process crashes and message omissions may occur - note that a process crash may be simulated by permanent message omissions, that is, all messages are omitted. In order to perform fault-injection experminents, the Adversary unit may simulate transient or permanent message omissions.

In the host memory, the Clock triggers individual rounds in the Controller, which then triggers the individual rounds in the protocol on the smart card. To run a different protocol, it is sufficient to add a new (so called applet) protocol to the (multiple applet) smart card.

7.3 Experiences

We implemented the fair exchange protocol of Avoine *et al.* [3] within TrustedPals and tested it with four participating parties. The code of one party was executed on a smart card while the code of the other parties was simulated on a single PC.

We did some initial measurements on the speed of the protocol. We observed that the communication between the host PC and the smart card is the bottleneck and dominates the time needed to execute a synchronous round. Our implementation needed roughly 600 ms to perform the necessary communication and so in our setup we set the round length to one second. Depending in the necessary level of security the protocol needs between 4 and 10 rounds (i.e., between 4 and 10 seconds) to complete. Since our implementation is not optimzed for speed and future technologies promise to increase the communication bandwith on between host and smart card, we believe that this performance can be improved considerably.

We did some fault-injection experiments and tested the implementation using the random adversary. Within more than 10.000 runs the protocol did not yield any single successful security violation.

References

1. Trustedpals source code. Downloadable from `http://pi1.informatik.uni-mannheim.de`, April 2006.
2. ActiveMQ. http://activemq.codehaus.org.
3. Gildas Avoine, Felix Gärtner, Rachid Guerraoui, and Marko Vukolic. Gracefully degrading fair exchange with security modules. In *Proceedings of the Fifth European Dependable Computing Conference*, pages 55–71. Springer-Verlag, April 2005.
4. Gildas Avoine and Serge Vaudenay. Fair exchange with guardian angels. In *The 4th International Workshop on Information Security Applications – WISA 2003*, Jeju Island, Korea, August 2003.
5. Michael Ben-Or, Shafi Goldwasser, and Ari Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th Annual Symposium on Theory of Computing (STOC)*, pages 1–10, Chicago, IL USA, May 1988. ACM Press.
6. Zinaida Benenson, Felix C. Gärtner, and Dogan Kesdogan. Secure multi-party computation with security modules. Technical Report AIB-10-2004, RWTH Aachen, December 2004.
7. David Chaum, Claude Crepeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In Richard Cole, editor, *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pages 11–19, Chicago, IL, May 1988. ACM Press.
8. Z. Chen. *Java Card Technology for Smart Cards - 1st Edition.* Addison-Wesley Professional, 2000.
9. Carole Delporte-Gallet, Hugues Fauconnier, and Felix C. Freiling. Revisiting failure detection and consensus in omission failure environments. In *Proceedings of the International Colloquium on Theoretical Aspects of Computing (ICTAC05)*, Hanoi, Vietnam, October 2005.

10. Joan G. Dyer, Mark Lindemann, Ronald Perez, Reiner Sailer, Leendert van Doorn, Sean W. Smith, and Steve Weingart. Building the IBM 4758 secure coprocessor. *IEEE Computer*, 34(10):57–66, October 2001.
11. M. Fowler. *Inversion of Control Containers and the Dependency Injection Pattern.* http://martinfowler.com/articles/injection.html.
12. Spring Framework. http://www.springframework.org.
13. F. Freiling, M. Herlihy, and L. Penso. Optimal randomized fair exchange with secret shared coins. In *Proceedings of the Ninth International Conference on Principles of Distributed Systems.* Springer, December 2005.
14. Oded Goldreich. Secure multi-party computation. Internet: `http://www.wisdom.weizmann.ac.il/~oded/pp.html`, 2002.
15. Oded Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proceedings of the 19th ACM Symposium on the Theory of Computing (STOC)*, pages 218–229, 1987.
16. U. Hansmann, M. Nicklous, T. Schäck, A. Schneider, and F. Seliger. *Smart Card Application Development Using Java - 2nd Edition.* Springer, 2002.
17. Martin Hirt, Ueli Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In *Proceedings of Asiacrypt*, 2000.
18. Alexander Iliev and Sean Smith. More efficient secure function evaluation using tiny trusted third parties. Technical Report TR2005-551, Dartmouth College, Computer Science, Hanover, NH, July 2005.
19. L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.
20. P. MacKenzie, A. Oprea, and M.K. Reiter. Automatic generation of two-party computations. In *SIGSAC: 10th ACM Conference on Computer and Communications Security.* ACM SIGSAC, 2003.
21. Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay — A secure two-party computation system. In *Proceedings of the 13th USENIX Security Symposium.* USENIX, August 2004.
22. Philippe Raïpin Parvédy and Michel Raynal. Uniform agreement despite process omission failures. In *17th International Parallel and Distributed Processing Symposium (IPDPS 2003).* IEEE Computer Society, April 2003. Appears also as IRISA Technical Report Number PI-1490, November 2002.
23. M. Pease, R. Shostak, and L. Lamport. Reaching agreements in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.
24. Kenneth J. Perry and Sam Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering*, 12(3):477–482, March 1986.
25. Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity — A proposal for terminology. In H. Federrath, editor, *Anonymity 2000*, number 2009 in Lecture Notes in Computer Science, pages 1–9, 2001.
26. Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.
27. Trusted Computing Group. Trusted computing group homepage. Internet: `https://www.trustedcomputinggroup.org/`, 2003.
28. A. C. Yao. Protocols for secure computations (extended abstract). In *23th Annual Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164, Los Alamitos, Ca., USA, November 1982. IEEE Computer Society Press.
29. Lidong Zhou, Fred B. Schneider, and Robbert van Renesse. COCA: A secure distributed on-line certification authority. *TOCS*, 20(4):329–368, November 2002.

Private Information Retrieval Using Trusted Hardware

Shuhong Wang[1], Xuhua Ding[1], Robert H. Deng[1], and Feng Bao[2]

[1] School of Information Systems, SMU
godintears@gmail.com,
{xhding, robertdeng}@smu.edu.sg
[2] Institute for Infocomm Research, Singapore
baofeng@i2r.a-star.edu.sg

Abstract. Many theoretical PIR (Private Information Retrieval) constructions have been proposed in the past years. Though information theoretically secure, most of them are impractical to deploy due to the prohibitively high communication and computation complexity. The recent trend in outsourcing databases fuels the research on practical PIR schemes. In this paper, we propose a new PIR system by making use of trusted hardware. Our system is proven to be information theoretically secure. Furthermore, we derive the computation complexity lower bound for hardware-based PIR schemes and show that our construction meets the lower bounds for both the communication and computation costs, respectively.

1 Introduction

Retrieval of sensitive data from databases or web services, such as patent databases, medical databases, and stock quotes, invokes concerns on user privacy exposure. A database server or web server may be interested in garner information about user profiles by examining users' database access activities. For example, a company's query on a patent from a patent database may imply that it is pursuing a related idea; an investor's query on a stock quote may indicate that he is planning to buy or sell the stock. In such cases, the server's ability of performing information inference is unfavorable to the users. Ideally, users' database retrieval patterns are not leaked to any other parties, including the servers.

A PIR (Private Information Retrieval) scheme allows a user to retrieve a data item from a database without revealing information about the data item. The earliest references of "query privacy" date back to Rivest et al [18] and Feigenbaum [9]. The first formal notion of PIR was defined by Chor et al [6]. In their formalization, a database is modelled as a n-bit string $x = x_1 x_2 \cdots x_n$, and a user is interested in retrieving one bit from x. With this formalization, many results have been produced in recent years. Depending on whether trusted hardware is employed or not, we classify PIR schemes into two categories: *traditional PIR* which does not utilize any trusted hardware and *hardware-based PIR* which employs trusted hardware in order to reduce communication and computation complexities.

The major body of PIR work focuses on the traditional PIR. Interested readers are referred to a survey [10] for a thorough review. A big challenge in PIR design is to minimize the communication complexity, which measures the number of bits transmitted between the user and the server(s) per query. A trivial solution of PIR is for the server to return the entire database. Therefore, the upper bound of communication complexity is $O(n)$ while the lower bound is $O(\log n)$, since by all means the user has to provide an index. For a single server PIR with information theoretic privacy, it is proven in [6] that the communication complexity is at least $O(n)$ and therefore confirming that $O(n)$ is the lower bound. Two approaches are used to reduce the communication cost. One is to duplicate the database in different servers, with the assumption that the servers do not communicate with each other. Without assuming any limit the servers' computation capability, PIR schemes with multiple database copies are able to offer information theoretic security with lower communication cost. The best known result is [3] due to Beimel et. al., with communication complexity $O(n^{\log\log\omega/\omega\log\omega})$, where ω is the number of database copies. The other approach still uses single server model but assumes that the server's computation capability is bounded. Schemes following this approach offer computational security with relatively low communication complexity. The best result to date is due to Lipmaa [15], where the user and the server communication complexity are $O(\kappa\log^2 n)$ and $O(\kappa\log n)$ respectively, with κ being the secure parameter of the underlying computationally hard problem.

Another key performance metric of PIR schemes is their computation complexity. All existing traditional PIR schemes require high computation cost at the server(s) end. Beimel et al [4] proved that the expected computation of the server(s) is $\Omega(n)$[1], which implies that any study on traditional PIR schemes is only able to improve its computation cost by a constant factor.

To the best of our knowledge, two *hardware-based PIR* constructions exist in literature. The earlier scheme [19] due to Smith and Safford is proposed solely to reduce the communication cost. On each query, a trusted hardware reads all the data items from an external database and returns the requested one to the user. The other hardware-based scheme [13,14] is due to Iliev and Smith. The scheme facilitates an efficient online query process by offloading heavy computation load offline. For each query, its online process costs $O(1)$. Nonetheless, depending on the hardware's internal storage size, for every k ($k << n$) queries the external database needs to be reshuffled with a computation cost $O(n\log n)$. For convenience, we refer to the first scheme as SS01 and the latter as IS04. Both schemes have $O(\log n)$ communication complexity.

Our Contributions. The contributions of this paper are three-fold: (1) We present a new PIR scheme using the same trusted hardware model as in [13] and

[1] Ω is the notation for asymptotic lower bound. $f(n) = \Omega(g(n))$ if there exists a positive constant c and a positive integer n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$.

prove that it is secure in the information theoretical sense; (2) Among all existing PIR constructions, our scheme achieves the best performance in all aspects: $O(\log n)$ communication complexity, $O(1)$ online computation cost and $O(n)$ offline computation cost; (3) We prove that our average computation complexity per query, $O(n/k)$, is the lower bound for hardware-based PIR schemes using the same model, where k ($k << n$) is the maximum number of data items stored by the trusted hardware.

2 Models and Definitions

Database Model and Its Permutation. We use π to denote a permutation of n integers: $(1, 2, \cdots, n)$. For $1 \leq i \leq n$, the image of i under π is denoted by $\pi(i)$. A database $\mathcal{D}$ is modelled as an array, represented by $\mathcal{D} = [d_1, d_2, \cdots d_n]$, where d_i is the i-th data item in its original form, for $1 \leq i \leq n$. A permuted $\mathcal{D}$ under π is denoted by $\mathcal{D}_\pi$ and its i-th data record is denoted by $\mathcal{D}_\pi[i]$, for $1 \leq i \leq n$. The database $\mathcal{D}$ is permuted into $\mathcal{D}_\pi$ by using π in such a way that *the i-th element of $\mathcal{D}_\pi$ is the $\pi(i)$-th element in $\mathcal{D}$*, i.e.

$$\mathcal{D}_\pi[i] = d_{\pi(i)} \tag{1}$$

In other words, $\mathcal{D}_\pi = \pi^{-1}(\mathcal{D})$. To protect the secrecy of π, the permutation is always coupled with encryption operations. In the rest of the paper, we use $\mathcal{D}_\pi[i] \simeq d_{\pi(i)}$ to denote that $\mathcal{D}_\pi[i]$ is the ciphertext of $d_{\pi(i)}$. To illustrate the idea, a trivial example is as follows. Let $\pi = (1324)$, which means $\pi(1) = 3, \pi(2) = 4, \pi(3) = 2, \pi(4) = 1$. Then for $\mathcal{D} = [d_1, d_2, d_3, d_4]$, we have $\mathcal{D}_\pi \simeq [d_3, d_4, d_2, d_1]$. To distinguish the entries in the original plaintext database and the permuted and encrypted database, we use the convention throughout the paper that data *items* refer to those in $\mathcal{D}$ and data *records* refer to those in $\mathcal{D}_\pi$.

Architecture. As shown in Figure 1 below, our hardware-based PIR scheme comprises of three types of entities: a group of *users*; a *server* and a *trusted hardware* denoted by TH. The server hosts a permuted and encrypted version of a database $\mathcal{D} = [d_1, \cdots, d_n]$, which consists of n items of equal length[2]. TH is a secure and tamper-resistant device residing on the server. With limited computation power and storage cache, TH shuffles the original database $\mathcal{D}$ into database $\mathcal{D}_\pi$ based on a permutation π; it remembers the permutation π and answers users' queries.

Each user interacts with TH via a secure channel, e.g. a SSL connection. When a user wants to retrieve the i-th data item of $\mathcal{D}$, she sends a query q to TH through the channel. On receiving q, TH accesses the permuted database $\mathcal{D}_\pi$ and retrieves the intended item d_i. Throughout the paper, by using "$q = i$", we mean the query q requests the i-th item of $\mathcal{D}$. By saying "the index of a (data) item", we refer to its index in the original database $\mathcal{D}$, by saying "the index of a (data) record", we refer to its index in the shuffled database in which the record locates.

[2] If necessary, we use padding for those data items with different length. Different to the bit model in [6], we extend it to the block model.

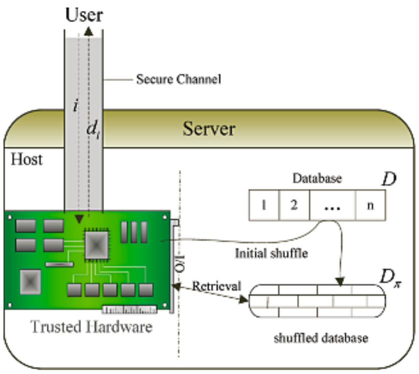


Fig. 1. Hardware-based PIR Model

Trusted Hardware. TH is trusted in the sense that it honestly executes the PIR protocol. Neither outside adversaries nor the server is able to tamper its execution or access its private space. Its limited private cache is able to store up to k ($k << n$) data *items* along with their indices. The indices of those cached items are managed by TH using a list denoted by Γ. In other words, Γ stores the original indices. We assume TH is capable of performing CPA-secure (i.e., secure under Chosen Plaintext Attacks) symmetric key encryption/decryption and to generate random numbers or secret keys.

Access Pattern. As in [11], the access pattern for a time period is defined $\mathcal{A} = (a_1, \cdots, a_N)$, where a_i is the data record read in the i-th database access, for $i \in [1, N]$. We observe that a record in the access pattern is essentially a probabilistic result of both the current query and the query history. In fact, the latter results in the current state of TH and the database.

Adversary. We consider adversaries who attempt to derive non-trivial information from the user's queries. Possible adversaries include both outside attackers and the server. Note that we do not assume any trust on the server. The adversary is able to monitor all the input and output of TH. Moreover, the adversary is allowed to query the database at her will and receives the replies from TH.

Stained Query and Clean Query. A query is stained if its content, e.g. the index of the requested data, is known to the adversary without observing the access pattern. This may occur in several scenarios. For instance, a query is compromised or revealed accidently; or the query could be originated from the adversary herself. On the other hand, a query is clean if the adversary does not know its content before observing the access pattern.

Security Model. Our security model follows the security notion in ORAM [11]. We measure the information leakage from PIR query executions. A secure PIR scheme ensures that the adversary does not gain additional information to determine the distribution of queries. Formally, we define it as follows.

Definition 1. *A hardware-based PIR scheme is secure, if given a* clean *query q and an access pattern $\mathcal{A}$, the conditional probability for the event that the query is on index j (i.e., q=j) is the same as its a-priori probability, i.e.*

$$\Pr(q = j|\mathcal{A}) = \Pr(q = j), \text{ for all } j \in [1, n].$$

The equation implies that the access pattern $\mathcal{A}$ reveals to the adversary no information on the target query q's content.

Table 1 below highlights the notations used throughout this paper.

Table 1. Notations

Notation	Description
k	The maximum number of data items cached in TH.
$\mathcal{D}$	The original database in the form of $(d_1, d_2, \cdots, d_n)$.
$\pi_0, \pi_1, \cdots$	A sequence of secret random permutations of n elements $\{1, 2, \cdots, n\}$.
$\mathcal{D}_{\pi_s}$	A permuted database of $\mathcal{D}$ using permutation π_s such that $\mathcal{D}_{\pi_s}[j] \simeq d_{\pi_s(j)}$, for $1 \le j \le n$, where $\mathcal{D}_{\pi_s}[j]$ denotes the j-th record in $\mathcal{D}_{\pi_s}$.
a_i	The retrieved data record by TH during its i-th access to a shuffle database.
$\mathcal{A}$	The access pattern comprising all the retrieved records $(a_1, \cdots, a_N)$ during a fixed time period.
$\mathcal{A}_s$	The access pattern comprising all the retrieved records during the s-th *session*, as defined in Section 3.
Γ	The list of (original) indices of all data items stored in TH's cache.

3 The PIR Scheme

System Setup

We consider applications where a trusted third party (TTP) is available to initialize the system. This TTP is involved only in the initialization phase and then stays offline afterwards. For other scenarios where the TTP is not available, an alternative solution is provided in Section 6.

TTP secretly selects a random permutation π_0 and a secret key sk_0. It permutes the original database $\mathcal{D}$ into $\mathcal{D}_{\pi_0}$, which is encrypted under sk_0, such that $\mathcal{D}_{\pi_0}[j] \simeq d_{\pi_0(j)}$ for $j \in [1, n]$. $\mathcal{D}_{\pi_0}$ is then delivered to the server. TTP secretly assigns π_0 and sk_0 to TH, which completes the system initialization.

The outline of our PIR scheme is as follows. Every k consecutive query executions are called a *session*. For the s-th session, $s \ge 0$, let $\pi_s, \mathcal{D}_{\pi_s}$ and sk_s denote the permutation, the database, and the encryption key respectively. On receiving a query from the user, TH retrieves a data record from $\mathcal{D}_{\pi_s}$, decrypts it with sk_s to get the data item, and stores the item in its private cache. Then TH replies to the user with the desired data item. The detailed operations on data retrieval are shown in Algorithm 1. After k queries are executed, TH generates a new random permutation π_{s+1} and an encryption key sk_{s+1}. It reshuffles $\mathcal{D}_{\pi_s}$ into $\mathcal{D}_{\pi_{s+1}}$ by employing π_{s+1} and sk_{s+1}. Note that in the newly shuffled database $\mathcal{D}_{\pi_{s+1}}$, all data items are encrypted under the secret key sk_{s+1}. The details on database reshuffle are given in Algorithm 2.

The original database $\mathcal{D}$ is not involved in any database retrieval operations. Since TH always performs a decryption for every read operation and an encryption for every write operation, we omit them in the algorithm description in order to keep the presentation compact and concise.

Retrieval Query Process

The basic idea of our retrieval algorithm is the following. TH always reads a different record on every query and every record is accessed at most once. Thus, if the database is well permutated (in the sense of oblivious permutation), all database accesses within the session appear random to the adversary.

Without loss of generality, suppose that the user intends to retrieve d_i in $\mathcal{D}$ during the s-th session ($s \geq 0$). On receiving the query for index i, TH performs the following: If the requested d_i is not in TH's cache, it locates the corresponding record in the permutated database $\mathcal{D}_{\pi_s}$ by computing the record index as $\pi_s^{-1}(i)$. If the requested item resides in TH, it reads from $\mathcal{D}_{\pi_s}$ a random record which is not accessed before [3]. The algorithm is elaborated in Figure 2 below.

```
Algorithm 1: Retrieving record $d_i$ using $\mathcal{D}_{\pi_s}$
1. TH decrypts the query and gets the requested index $i$.
2. If $i \notin \Gamma$
3.     TH reads $\pi_s^{-1}(i)$-th record of $\mathcal{D}_{\pi_s}$ and stores the item $d_i$ into the cache;
4.     $\Gamma = \Gamma \cup \{i\}$;
5. Else
6.     TH selects a random index $j$, $j \in_R \{1, \cdots, n\} \setminus \Gamma$
7.     TH reads the $\pi_s^{-1}(j)$-th record from $\mathcal{D}_{\pi_s}$ and stores the item $d_j$ into the cache;
8.     $\Gamma = \Gamma \cup \{j\}$;
9. TH returns $d_i$ to the user.
```

Fig. 2. Retrieval Query Processing Algorithm

Access Pattern. The access pattern $\mathcal{A}_s$ produced by Algorithm 1 is a sequence of data records which are retrieved from $\mathcal{D}_{\pi_s}$ during the s-th session. It is clear from Figure 2 that on each query, TH reads exactly one data record from $\mathcal{D}_{\pi_s}$. Therefore, when the s-th session terminates, $\mathcal{A}_s$ has exactly k records.

[3] The operation should be coded so that both "if" and "else" situations take the same time to stand against side-channel attack. This requirement is applied at the similar situation of reshuffle algorithm later.

Reshuffle Process

After k retrievals, TH's private cache reaches its limit, which demands a reshuffle of the database with a new permutation. Note that simply using cache substitution introduces a risk of privacy exposure. The reason is that when a discarded item is requested again, the adversary knows that a data record is retrieved more than once by TH from the same location. Therefore, a reshuffle procedure must be executed at the end of each session.

TH first secretly chooses a new random permutation π_{s+1}. The expected database $\mathcal{D}_{\pi_{s+1}}$ satisfies $\mathcal{D}_{\pi_{s+1}}[j] \simeq d_{\pi_{s+1}(j)}$, $j \in [1, n]$. The correlation between $\mathcal{D}_{\pi_s}$ and $\mathcal{D}_{\pi_{s+1}}$ is

$$\mathcal{D}_{\pi_{s+1}}[j] \simeq \mathcal{D}_{\pi_s}[\pi_s^{-1} \circ \pi_{s+1}(j)], \tag{2}$$

for $1 \leq j \leq n$, where $\pi_s^{-1} \circ \pi_{s+1}(j)$ means $\pi_s^{-1}(\pi_{s+1}(j))$.

The basic idea of our reshuffle algorithm is as follows. We sort the items in TH's cache in ascending order based on their new positions in $\mathcal{D}_{\pi_{s+1}}$. We observe that those un-cached items in $\mathcal{D}_{\pi_s}$ will also be logically sorted in the ascending order based on their new indexes, because the database supports index-based record retrieval. The reshuffle process is similar to a merge-sort of the sorted cached items and un-cached items. TH plays two roles: (1) participating in the merge-sort to initialize $\mathcal{D}_{\pi_{s+1}}$; (2) obfuscating the read/write pattern to protect the secrecy of π_{s+1}.

TH first sorts indices in Γ based on the ascending order of their images under π_{s+1}^{-1}. It assigns database $\mathcal{D}_{\pi_{s+1}}$ sequentially, starting from $\mathcal{D}_{\pi_{s+1}}[1]$. For the first $n - k$ assignments, TH always performs one read operation and one write operation; for the other k assignments, TH always performs one write operation. The initialization of $\mathcal{D}_{\pi_{s+1}}[j]$, $j \in [1, n]$, falls into one of the following two cases, depending on whether its corresponding item is in the cache or not.

Case(i) The corresponding item is not cached (i.e., $\pi_{s+1}(j) \notin \Gamma$): TH reads it (i.e., the record $\mathcal{D}_{\pi_s}[\pi_s^{-1} \circ \pi_{s+1}(j)]$) from $\mathcal{D}_{\pi_s}$ and writes it to $\mathcal{D}_{\pi_{s+1}}$ as $\mathcal{D}_{\pi_{s+1}}[j]$.

Case (ii) The corresponding item is in the cache (i.e., $\pi_{s+1}(j) \in \Gamma$): Before retrieving $\mathcal{D}_{\pi_s}[\pi_s^{-1} \circ \pi_{s+1}(j)]$ from the cache and writing it into $\mathcal{D}_{\pi_{s+1}}$ as $\mathcal{D}_{\pi_{s+1}}[j]$, TH also performs a read operation for two purposes: (a) to demonstrate the same reading pattern as if in Case (i) so that the secrecy of π_{s+1} is protected; (b) to save the cost of future reads. Moreover, instead of randomly reading a record from the $\mathcal{D}_{\pi_s}$, TH looks for the smallest index which has not been initialized and falls in Case (i). It then retrieves the corresponding data record from $\mathcal{D}_{\pi_s}$. Since Γ is sorted, this searching process totally costs k comparisons for the entire reshuffle process. The benefit of this approach coupled with a sorted Γ is that both the costs for testing if $\pi_{s+1}(j) \in \Gamma$ and the item retrieval from the cache are $O(1)$; Otherwise, both cost $O(k)$ per item and $O(nk)$ in total.

The details of the reshuffle algorithm are shown in Figure 3, where *min* denotes the head of sorted Γ. We use `sortedel`(i)/ `sortins`(i) to denote the sorted deletion/insertion of index i from/to Γ and subsequent adjustments.

```
Algorithm 2: Reshuffle $\mathcal{D}_{\pi_s}$ into $\mathcal{D}_{\pi_{s+1}}$, executed by TH
 1. secretly select a new random permutation $\pi_{s+1}$;
 2. sort indices in $\Gamma$ based on the order of their images under $\pi_{s+1}^{-1}$.
    set $j = 1; j' = 1$.
 3. while $1 \leq j \leq n-k$ do
 4.    while $\pi_{s+1}(j') \in \Gamma$ do $j' = j' + 1$ end;
 5.    set $r = \pi_s^{-1} \circ \pi_{s+1}(j')$; read $\mathcal{D}_{\pi_s}[r]$ from $\mathcal{D}_{\pi_s}$;
 6.    if $j = j'$          /* Case (i): $\pi_{s+1}(j) \notin \Gamma$ */
 7.       write $\mathcal{D}_{\pi_{s+1}}$ by setting $\mathcal{D}_{\pi_{s+1}}[j] \simeq \mathcal{D}_{\pi_s}[r]$;
 8.    else                 /* Case (ii): $\pi_{s+1}(j) \in \Gamma$ */
 9.       write $\mathcal{D}_{\pi_{s+1}}$ by setting $\mathcal{D}_{\pi_{s+1}}[j] \simeq d_{min}$; [a]
10.       sortedel($j$); Insert $\mathcal{D}_{\pi_s}[r]$ into cache and sorteins($j'$);
11.    $j = j + 1; j' = j' + 1$;
12. end{while};
13. while $n - k + 1 \leq j \leq n$ do
14.     set $\mathcal{D}_{\pi_{s+1}}[j] = d_{min}$; sortdel($j$);
15.    $j = j + 1$;
16. end
```

[a] d_{min} is exactly $\mathcal{D}_{\pi_s}[\pi_s^{-1} \circ \pi_{s+1}(j)]$ since $\mathcal{D}_{\pi_{s+1}}$ is filled in by an increasing order.

Fig. 3. Database Reshuffle Algorithm

The reshuffle algorithm is secure and efficient. An intuitive explanation of its security is as follows. After a reshuffle, the new database is reset to its initial status. If an item has been accessed in the previous session, it is placed at a random position by the reshuffle. Other items are indeed relatively linkable between sessions. However, the linkage does not provide the adversary any advantage since they have not been accessed at all. Note that the addition in the inner loop is executed at most $n - 1$ times in total, since j' never decreases. Because Γ is a sorted list and the inserted and deleted indices are in an ascending order, the insertion and deletion are of constant cost. Totally at most n comparisons are needed for the whole execution. Therefore, the overall computation complexity of Algorithm 2 is $O(n)$.

Reshuffle Pattern. The access pattern produced by Algorithm 2 is denoted by $\mathcal{R}_s$. Since TH only reads $n - k$ data records, $\mathcal{R}_s$ has exactly $n - k$ elements. Note that the writing pattern is omitted because it is in a fixed order, i.e., sequentially writing from position 1 to position n.

4 Security

We now proceed to analyze the security of our scheme based on the notion defined in Section 2. Our proof essentially goes as follows. Lemma 1 proves that the reshuffle procedure is oblivious in the same notion as in Oblivious RAM [11]. Thus after each reshuffle, the database is reset into the initial state such that the accesses between different sessions are not correlated. Then in Theorem 1 we show that each individual query session does not leak information of the query, which leads to the conclusion on user privacy across all sessions.

Lemma 1. *The reshuffle algorithm in Figure 3 is oblivious. For any non-negative integer s, any integer $j \in [1, n]$,*

$$\Pr(\mathcal{D}_{\pi_s}[j] = d_l \mid \mathcal{A}_0, \mathcal{R}_0, \cdots, \mathcal{A}_{s-1}, \mathcal{R}_{s-1}) = 1/n, \tag{3}$$

for all $l \in [1, n]$, where $\mathcal{A}_i$ and $\mathcal{R}_i$, $i \in [0, s-1]$, are the access pattern and reshuffle pattern for i-th session respectively.

PROOF. We prove Lemma 1 for a fixed j by induction on the session index s. The proof applies to all $j \in [1, n]$.
I. $s = 0$. Since $\mathcal{D}_{\pi_0}$, the initial shuffled database, is constructed in advance under a secret random permutation π_0, the probability $\Pr(\mathcal{D}_{\pi_0}[j] = d_l \mid \emptyset) = 1/n$ holds for all $1 \le l \le n$.

II. Suppose the lemma is true for $s = i$, that is, $\Pr(\mathcal{D}_{\pi_i}[j] = d_l \mid \mathcal{A}_0, \mathcal{R}_0, \cdots, \mathcal{A}_{i-1}, \mathcal{R}_{i-1}) = 1/n$. We proceed to prove that it holds for $s = i + 1$, i.e.

$$\Pr(\mathcal{D}_{\pi_{i+1}}[j] = d_l \mid \mathcal{A}_1, \mathcal{R}_1, \cdots, \mathcal{A}_i, \mathcal{R}_i) = 1/n, \tag{4}$$

for all $l \in [1, n]$.

In order to use the recursive assumption, We link the two databases $\mathcal{D}_{\pi_{i+1}}$ and $\mathcal{D}_{\pi_i}$ by the following conditional probability,

$$\begin{aligned}&\Pr(\mathcal{D}_{\pi_{i+1}}[j] = d_l \mid \mathcal{A}_1, \mathcal{R}_1, \cdots, \mathcal{A}_i, \mathcal{R}_i)\\ =&\sum_{x=1}^{n} \Pr(\mathcal{D}_{\pi_{i+1}}[j] = \mathcal{D}_{\pi_i}[x] \mid \mathcal{A}_1, \mathcal{R}_1, \cdots, \mathcal{A}_i, \mathcal{R}_i) \cdot \Pr(\mathcal{D}_{\pi_i}[x] = d_l \mid \mathcal{A}_1, \mathcal{R}_1, \cdots, \mathcal{A}_i, \mathcal{R}_i).\end{aligned} \tag{5}$$

Then the formula is evaluated depending on cases that whether or not l is stained and whether or not the item corresponding to x is in the cache. The conclusion is obtained by showing that the right hand side of the equation sums to be $1/n$ in any case. Due to page limitation, the detailed proof which can be found in the full version [22] is omitted here. □

Lemma 1 implies that the reshuffle procedure resets the observed distribution of the data items. Therefore, the events occurring during separated sessions are independent of each other. Theorem 1 below addresses the security of the proposed PIR scheme.

Theorem 1. *Given a time period, the observation of the access pattern $\mathcal{A} = (a_1, a_2, \cdots, a_N)$, $N > 0$, provides the adversary no additional knowledge to determine any clean query q, i.e. for all $j \in [1, n]$,*

$$\Pr(q = j \mid \mathcal{A}) = \Pr(q = j) \tag{6}$$

where $\Pr(q = j)$ is an a-priori probability of query q being on index j.

PROOF. For $1 < t \leq N$, let $\Pr(a_t \mid a_1, \cdots, a_{t-1})$ denote the probability of the event that data a_t is accessed immediately after the access of $t-1$ records. Let $\Pr(a_t \mid a_1, \cdots, a_{t-1}; q = j)$ denote the probability of the same event with additional knowledge that the requested index of query q is j. Note that we do not assume any temporal order of the query q and the t-th query. We proceed to show below that

$$\Pr(a_t \mid a_1, \cdots, a_{t-1}) = \Pr(a_t \mid a_1, \cdots, a_{t-1}; q = j) \tag{7}$$

Without loss of generality, suppose a_t is read from $\mathcal{D}_{\pi_s}$ during the s-th session. Consider the following two cases:

1. $a_t \in \mathcal{R}_s$, i.e. a_t is accessed during a reshuffle process: Obviously $\Pr(a_t \mid a_1, \cdots, a_{t-1}) = \Pr(a_t \mid a_1, \cdots, a_{t-1}; q = j)$, due to the fact that the access to a_t is completely determined by permutation π_s and π_{s+1}.
2. $a_t \in \mathcal{A}_s$, i.e. a_t is accessed during a query process: Let this query be the l-th query in this session, $l \in [1, k]$. Therefore, $l-1$ data items are cached by TH before a_t is read. We consider two scenarios based upon Algorithm 1:
   (a) The requested data is cached in TH: a_t is randomly chosen from those data items not cached in TH. Therefore, $\Pr(a_t \mid a_1, \cdots, a_{t-1}) = \frac{1}{n-(l-1)}$.
   (b) The requested data is not cached in TH: a_t is retrieved from D_{π_s} based on the permutation π_s. According to Lemma 1, the probability that a_t is selected is $\frac{1}{n-(l-1)}$.

   Note that the compromise of a query, i.e. knowing $q = j$, possibly helps an adversary to determine whether a_t is in case (2a) or (2b). Nonetheless, this information does not change $\Pr(a_t \mid a_1, \cdots, a_{t-1})$, since their values are $\frac{1}{n-(l-1)}$ in both cases. Thus, $\Pr(a_t \mid a_1, \cdots, a_{t-1}) = \Pr(a_t \mid a_1, \cdots, a_{t-1}, q = j)$ when $a_t \in \mathcal{A}_s$.

In total, we conclude that for any t and $q = j$,

$$\Pr(a_t \mid a_1, \cdots, a_{t-1}) = \Pr(a_t \mid a_1, \cdots, a_{t-1}; q = j).$$

As a result,

$$\begin{aligned} \Pr(\mathcal{A} \mid q = j) &= \Pr(a_1, \cdots, a_N \mid q = j) \\ &= \Pr(a_N \mid a_1, \cdots, a_{N-1}; q = j) \cdot \Pr(a_1, \cdots, a_{t-1} \mid q = j) \\ &= \prod_{t=1}^{N} \Pr(a_t \mid a_1, \cdots, a_{t-1}; q = j) \\ &= \prod_{t=1}^{N} \Pr(a_t \mid a_1, \cdots, a_{t-1}) \\ &= \Pr(\mathcal{A}). \end{aligned} \tag{8}$$

Then,

$$\begin{aligned}\Pr(q = j \mid \mathcal{A}) &= \Pr(q = j, \mathcal{A})/\Pr(\mathcal{A}) \\ &= \frac{\Pr(\mathcal{A} \mid q = j) \cdot \Pr(q = j)}{\Pr(A)} \\ &= \Pr(q = j).\end{aligned}$$

The result shows that, given the access pattern, the a-posteriori probability of a query equals to its a-priori probability, which concludes the security proof for our PIR scheme. □

5 Performance

We proceed to analyze the communication and computation complexity of our PIR scheme. They are evaluated with respect to the database size n. Both complexities of our scheme reach the respective lower bounds for hardware-based PIR schemes.

Communication. We consider the user/system communication cost per query. In our scheme, the users only inputs an index of the desired data item and TH returns exactly one data item. Therefore, its communication complexity per query is $O(\log n)$. Note that $O(\log n)$ is the lower bound of communication cost for all PIR constructions.

Computation. For simplicity purpose, each reading, writing, encryption, and decryption of a data item is treated as one operation. The computation cost is measured by the average number of operations per session and per query. We also measure the online cost which excludes the expense of offline reshuffle operations.

As evident in Figure 2 and 3, it costs the trusted hardware $O(1)$ operations to process a query and $O(n)$ operations to reshuffle the database. Table 2 compares the computation cost of our scheme against [19] and [13,14].

Table 2. Comparison of Computation Cost of Three Hardware-based PIR Schemes

Schemes	**Total cost per session (k queries)**	**Online cost per query**	**Average Cost per query**
Our scheme	$O(n)$	$O(1)$	$O(n/k)$
IS04 [13,14]	$O(n \log n)$	$O(1)$	$O(\frac{n}{k} \log n)$
SS01 [19]	$O(kn)$	$O(n)$	$O(n)$

Our scheme outperforms the other two hardware-based PIR schemes in all three metrics. The advantage originates from our reshuffle algorithm which utilizes the hardware's cache in a more efficient manner. Moreover, we prove that the average cost per retrieval of our scheme reaches the lower bound for all information-theoretic PIR schemes with the same trusted hardware system model. Our result is summarized in the following theorem.

Theorem 2. *For any information-theoretically secure PIR scheme with a trusted hardware storing maximum k data items, the average computation cost per retrieval is $\Omega(n/k)$.*

Proof. Our proof is constructed in a similar manner to the proof in [4] which shows the computational lower bound for traditional PIR schemes.

Fix a PIR scheme, let B_i denote the set of all indices that the hardware reads in order to return d_i. B_i is essentially a random variable probabilistically determined by both the user query on d_i and the items that the hardware has already stored inside. Consequently, $\mathrm{E}(|B_i|)$ denotes the expect number of data items to read by the hardware to process a query on index i. We evaluate $\mathrm{E}(|B_i|)$ as the computation cost for PIR schemes.

For $1 \leq l \leq n$, let $\Pr(l \in B_i)$ be the probability that the hardware reads d_l in order to answer the query on index i. We define $\mathrm{P}(l)$ as the maximum of these probabilities for all $1 \leq i \leq n$, i.e.

$$\mathrm{P}(l) = \max_{1\leq i\leq n} \{\Pr(l \in B_i)\}.$$

Note that for an information-theoretically secure PIR scheme, user privacy implies that $B_1, B_2, \cdots, B_n$ have the identical distribution. Therefore, for convenience purpose, let

$$\mathrm{P}(l) = \Pr(l \in B_1).$$

Due to Lemma 5 of [4][4],

$$\mathrm{E}(|B_i|) = \sum_{l=1}^{n} \mathrm{P}(l). \tag{9}$$

Our target now is to show $\mathrm{E}(|B_i|) = \Omega(n/k)$. We prove it by contradiction.

Suppose that among $\mathrm{P}(1), \cdots, \mathrm{P}(n)$, there at least exist $k+1$ of them whose values are less than $1/(k+1)$. Without loss of generality, let the $k+1$ probabilities be $\mathrm{P}(1), \mathrm{P}(2), \cdots, \mathrm{P}(k+1)$. Now consider the probability $\Pr(1 \notin B_1 \cap 2 \notin B_2 \cdots \cap (k+1) \notin B_{k+1})$. We have,

$$\begin{aligned}
&\Pr(1 \notin B_1 \cap 2 \notin B_2 \cdots \cap (k+1) \notin B_{k+1}) \\
&= 1 - \Pr(1 \in B_1 \cup 2 \in B_2 \cdots \cup (k+1) \in B_{k+1}) \\
&\geq 1 - \sum_{l=1}^{k+1} \Pr(l \in B_l) = 1 - \sum_{l=1}^{k+1} \mathrm{P}(l) \\
&> 1 - (k+1)\frac{1}{k+1} = 0.
\end{aligned}$$

[4] It can be proved by defining the random variables $Y_1, \cdots, Y_n$ where $Y_l = 1$ if $l \in B_i$ and $Y_l = 0$ otherwise.

On the other hand, note that TH only caches k data items in maximum. As a consequence, there always exists one data item which must be read from the database during the $k+1$ queries on $1, 2, \cdots, k+1$. Thus, the event that $1 \notin B_1 \cap 2 \notin B_2 \cdots \cap (k+1) \notin B_{k+1}$ never occurs, i.e. $\Pr(1 \notin B_1 \cap 2 \notin B_2 \cdots \cap (k+1) \notin B_{k+1}) = 0$, which contradicts the probability computation above.

Thus, at most k elements in $\{\mathrm{P}(1), \cdots, \mathrm{P}(n)\}$ whose values are less than $1/(k+1)$. As a result,

$$\sum_{l=1}^{n} \mathrm{P}(l) \geq (n-k) \cdot \frac{1}{k+1}, \tag{10}$$

which shows the lower bound for average computation cost is $\Omega(n/k)$. $\square$

6 Discussion

Database Initialization Using TH

For applications where no trusted third party exists, TH can be used to initialize the database. TH first chooses a random permutation π_0. For $1 \leq i \leq n$, TH tags the i-th item d_i with its new index $\pi_0^{-1}(i)$. Using the merge-sort algorithm [8], $d_1, d_2, \cdots, d_n$ are sorted based on their new indices by TH. With the limited cache size in TH, Batcher's odd-even merges sorter [1] is an appropriate choice which requires $(\log^2 n - \log n + 4)n/4 - 1$ comparisons. One may argue that Beneš network [21] and Goldstein et al's switch networks [12] incur less comparisons. Unfortunately, neither is feasible in our system since the first one requires at least $n \log n$-bit ($>> k$) memory in TH while the latter has a prohibitively high setup cost. Note that encryption is applied during tagging and merging so that the process is oblivious to the server.

A simple example is presented in Fig. 4. The database in the example has 4 items d_1, d_2, d_3, d_4. The permutation is $\pi_0 = (1324)$, i.e. $\pi_0(1) = 3, \pi_0(2) = 4, \pi_0(3) = 2$ and $\pi_0(4) = 1$. The circles denote TH and the squares denote encrypted data items. After initialization, the original four items are permuted as shown on the right end. All the encrypted items are stored on the host. In

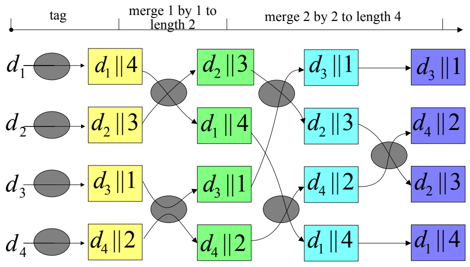


Fig. 4. Initial oblivious shuffle example using odd-even merges

every operation, only two items are read into TH's cache and then written back to the server.

Instantiation of Encryption and Permutation Algorithms

An implicit assumption of our security proof in Section 4 is the semantic security of the encryption of the database. Otherwise, the encryption reveals the data information and consequently exposes user privacy. Our adversary model in Section 2 allows the adversary to submit queries and observe the subsequent access patterns and replies. Thereafter, the adversary is able to obtain k pairs of plaintext and ciphertext in maximum for each encryption key, since different random keys are selected in different sessions. Thus, it is demanded to have an encryption algorithm semantically secure under CPA (Chosen Plaintext Attack) model. In practice, CPA secure symmetric ciphers such as AES, are preferred over public key encryptions, since the latter have more expensive computation cost and higher storage space demand.

For the permutation algorithm, we argue that it is impractical for a hardware-based PIR to employ a *true* random permutation, since it requires $O(n \log n)$ bits of storage, comparable to the size of the whole database. As a result, we opt for a pseudo-random permutation with a light computation load.

Since a cipher secure under CPA is transformed into an invertible pseudo-random permutation, we choose a CPA secure block cipher, e.g. AES, to implement the needed pseudo-permutation. With a block cipher, a message is encrypted by blocks. When $n \neq 2^{\lceil \log n \rceil}$, the ciphertext may be greater than n. In that case, the encryption is repeated until the output is in the appropriate range. Since $2^{\lceil \log n \rceil} \leq 2n$, the expected number of encryptions is less than 2. Black and Rogaway's result in [5] provides more information on ciphers with arbitrary finite domains.

Service Continuity

The database service is disrupted during the reshuffle process. The duration of a reshuffle is non-negligible since it is an $O(n)$ process. A trivial approach to maintaining the continuity of service is to deploy two pieces of trusted hardware. While one is re-permuting the database, the other deals with user queries. In case that installing an extra hardware is infeasible, an alternative is to split the cache of the hardware into two halves with each having the capacity of storing $k/2$ items. Consequently, the average computation cost will be doubled.

Update of Data Items

A byproduct of the reshuffle process is database update operations. To update d_i, the trusted hardware reads d_i obliviously in the same way as handling a read request. Then, d_i is updated inside the hardware's cache and written into the new permuted database during the upcoming reshuffle process. Though the new value of d_i is not written immediately into the database, data consistency is ensured since the hardware returns the updated value directly from the cache upon user requests.

7 Conclusion

In summary, we present in this paper a novel PIR scheme with the support of a trusted hardware. The new PIR construction is provably secure. The observation of the access pattern does not offer additional information to adaptive adversaries in determining the data items retrieved by a user.

Similar to other hardware-based PIR schemes, the communication complexity of our scheme reaches its lower bound, $O(\log n)$. In terms of computation complexity, our design is more efficient than all other existing constructions. Its online cost per query is $O(1)$ and the average cost per query is only $O(n/k)$, which outperforms the best known result by a factor $O(\log n)$ (though using a big-O notation, the hidden constant factor is around 1 here). Furthermore, we prove that $O(n/k)$ is the lower bound of computation cost for PIR schemes with the same trusted hardware based architecture.

The Trusted Computing Group (TCG) [20] defines a set of Trusted Computing Platform (TCP) specifications aiming to provide hardware-based root of trust and a set of primitive functions to propagate trust to application software as well as across platforms. How to extend and improve our proposed PIR scheme based on trusted computing technologies will be one of our future research directions.

Acknowledgement

This research is partly supported by the Office of Research, Singapore Management University.

We would like to thank anonymous reviewers for their valuable suggestions and criticisms on an earlier draft of this paper.

References

1. Kenneth E. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computing Conference*, pages 307–314, 1968.
2. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among Notions of Security for Public-key Encryption Schemes. In *Proceedings of Crypto '98*, LNCS 1462, pages 26–45, Berlin, 1998.
3. Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Jean-François Raymond. Breaking the $o(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In *FOCS*, pages 261–270. IEEE Computer Society, 2002.
4. Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: Pir with preprocessing. In *CRYPTO*, pages 55–73, 2000.

5. John Black and Phillip Rogaway. Ciphers with arbitrary finite domains. In Bart Preneel, editor, *CT-RSA*, volume 2271 of *Lecture Notes in Computer Science*, pages 114–130. Springer, 2002.
6. Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. In *FOCS*, pages 41–50, 1995.
7. Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.
8. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to Algorithms, Second Edition. ISBN 0-262-03293-7.
9. Joan Feigenbaum. Encrypting problem instances: Or ..., can you take advantage of someone without having to trust him? In Hugh C. Williams, editor, *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 477–488. Springer, 1985.
10. William Gasarch. A survey on private information retrieval. *The Bulletin of the European Association for Theoretical Computer Science*, Computational Complexity Column(82), 2004.
11. Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.
12. J. L. Goldstein and S. W. Leibholz. On the synthesis of signal switching networks with transient blocking. IEEE Transactions on Electronic Computers. vol.16, no.5, 637-641, 1967.
13. Alexander Iliev and Sean Smith. Private information storage with logarithm-space secure hardware. In *International Information Security Workshops*, pages 199–214, 2004.
14. Alexander Iliev and Sean W. Smith. Protecting client privacy with trusted computing at the server. *IEEE Security & Privacy*, 3(2):20–28, 2005.
15. Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *ISC*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328. Springer, 2005.
16. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988.
17. Jacques Patarin. Luby-rackoff: 7 rounds are enough for $2^{n(1-\epsilon)}$ security. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 513–529. Springer, 2003.
18. Ronald L. Rivest, Leonard Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms.
19. Sean W. Smith and David Safford. Practical server privacy with secure coprocessors. *IBM Systems Journal*, 40(3):683–695, 2001.
20. TCG Specification Architecture Overview. Available from http://www.trustedcomputinggroup.org.
21. Abraham Waksman. A permutation network. *J. ACM*, 15(1):159–163, 1968.
22. Shuhong Wang, Xuhua Ding, Robert H. Deng and Feng Bao. Private Information Retrieval Using Trusted Hardware Online available at http://eprint.iacr.org/.

Bridging the Gap Between Inter-communication Boundary and Internal Trusted Components

Yuji Watanabe, Sachiko Yoshihama, Takuya Mishina, Michiharu Kudo, and Hiroshi Maruyama

IBM Research, Tokyo Research Laboratory
1623-14 Shimotsuruma, Yamato-shi,
Kanagawa, 242-8502, Japan
{muew, sachikoy, tmishina, kudo, maruyama}@jp.ibm.com

Abstract. Despite increasing needs for the coalition-based resource sharing, establishing trusted coalition of nodes in an untrusted computing environment is a long-standing yet increasingly important issue to be solved. The Trusted virtual domain (TVD) is a new model for establishing trusted coalitions over heterogeneous and highly decentralized computing environment. The key technology to enable TVD is the integrity assurance mechanism, which allows a remote challenger to verify the configuration and state of a node.

A modern computer system consists of a multi-layer stack of software, such as a hypervisor, a virtual machine, an operating system, middleware, etc. The integrity assurance of software components is established by chains of assurance from the trusted computing base (TCB) at the *lowest* layer, while the communication interface provided by nodes should be properly abstracted at a *higher* layer to support interoperable communication and the fine-grained handling of expressive messages.

To fill the gap between "secure communication between nodes" and "secure communication between trusted components", a notion of "*Secure Message Router (SMR)*", domain-independent, easy to verify, multi-functional communication wrapper for secure communication is introduced in this paper. The SMR provides essential features to establish TVDs : end-to-end secure channel establishment, policy-based message translation and routing, and attestability using fixed clean implementation. A virtual machine-based implementation with a Web service interface is also discussed.

Keywords: Trusted Virtual Domain, Distributed Coalition, Trusted Computing, Mandatory Access Control.

1 Introduction

1.1 Background

In computer and information security history, there have been many efforts to achieve controlled sharing of digital resources. Recent advances in hardware platforms and networking technology increase the need for sharing resources securely

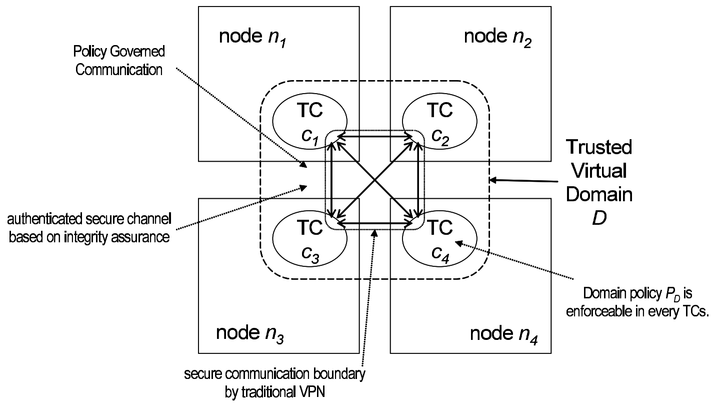

node n_1
node n_2
node n_3
node n_4
TC c_1
TC c_2
TC c_3
TC c_4
Policy Governed Communication
authenticated secure channel based on integrity assurance
Trusted Virtual Domain D
Domain policy P_D is enforceable in every TCs.
secure communication boundary by traditional VPN

Fig. 1. Trusted Virtual Domain Model

across decentralized administrative nodes in an insecure network world. Moreover, there is a growing tendency for nodes to form coalitions in order to collaborate by sharing some of their resources, or by coordinating some of their activities. Such coalitions are quite common in various scenarios, such as grid computing, on-demand working environments, and virtual enterprises.

With the increasing needs for the coalition-based resource sharing, establishing trusted coalitions in untrusted computing environments is becoming increasingly important concern. The client-server-style authenticated key establishment techniques such as SSL/TLS and IPSec provide end-to-end secure communication channels to build virtual private networks in open but untrusted network environments. Traditional authentication has focused on identifying the identity of the subject, i.e., who is at the other end of the communication channel, in order to control what information can be released to each subject, while not dealing with how the information is managed in the container at the end-point after the release of the information. Thus, once information is released through the channel, the program at the container may, through error or malice, propagate the information improperly. This implies identity-based authentication provides the secure communication channel only between nodes (or the interfaces of nodes) participating in the coalition, while it does not provide any assurances on the information handling (or how information flows in the coalition) in an end-to-end manner.

1.2 Trusted Virtual Domain

To tackle this problem in a universal and flexible way, we are developing an integrity-based end-to-end trusted computing infrastructure called *Trusted Virtual Domain* (TVD), which was originally introduced in an ongoing project in our research division (see [1] for details). A number of recent research projects[2,3,4,5] support realization of the TVD concept in various application domains. The basic idea of TVD is to establish a coalition of trusted components running on the nodes participating in the coalition in a decentralized, heterogeneous environment in order to allow them to simplify the management and to provide explicit infrastructure-level containment and trust guarantees. Our own view of TVD is formalized by the following definition (illustrated in Figure 1).

Definition 1 (Trusted Virtual Domain). *A coalition C which consists of a set $\{c_1, \ldots, c_l\}$ of (trusted) software components running on the nodes $\{n_1, \ldots, n_l\}$ respectively is a trusted virtual domain D as long as the following properties are satisfied:*

– *(Enforceability) Each component c_i works as a reference monitor for a set R_i of resources in the node n_i.*
– *(Authenticity of Enforceability) Any component c_i in C can provide strong evidence that domain policy P_D is properly enforced for any access to the resources R_i.*
– *(Authenticated secure channel) Any component in C can establish an end-to-end secure communication channel with the other components in C by checking the evidence given above.*
– *(Policy-Governed Communication) Communications between software components through the secure channels established above conforms to the domain policy P_D.*

We refer to a software component which satisfies the above properties as a "trusted component" ("TC" for short).

The major difference between a TVD-based coalition and a VPN-based one is that the software components in TVD are authorized to participate in the domain not only by authenticating their identities but also by attesting to the integrity of the other components (see remarks below). A security assurance of the coalition based on the VPN only covers the channel between nodes, while it does not cover the internal behavior of the nodes because there is no assurance after passing through the channel interfaces to the internal of the node. Meanwhile, the TVD expands its coverage of assurance in such a way that the trusted boundary is not limited to the interfaces of the node, but also includes the trusted software components in the nodes.

Remarks: In general, assurance of the integrity of components does not directly imply any assurance for the security properties of the components. The integrity assurance of components is considered as evidence that assures no unintended behavior will occur, but the intended behavior should be mapped to certain security properties with assurance based on a mathematical or analytical evaluation mechanism. For instance, a rigorous language-based information flow analysis shows the software component as a whole enforces the policies with respect to certain security properties such as confidentiality, integrity, or non-interference [6]. For more complex computer systems, an evaluation under the Common Criteria, which is an internationally recognized ISO standard used by governments and other organizations to assess the security and assurance of components, provides weaker,but in practice useful, evidence of the assurance of security properties □ .

A modern computer system consists of multi-layer stacks of software, such as a hypervisor, virtual machine, an operating system, and application sandboxes including middleware stack. Without assuming assurance based on tamper-resistant software, no integrity assurance can be provided for any software component which is running on an untrusted software runtime environment, because the behavior of the software running in such an environment can easily be compromised. This characteristic implies that the integrity assurance of software components is established by the chain of assurance from the trusted computing base (TCB) at the *lowest* layer, which could be a hardware layer such as a Trusted Platform Module (TPM) [7][8]. Therefore, the higher the layer the software component is placed in, the harder it is to establish integrity assurance for it.

In contrast, the communication interface provided by nodes should be properly abstracted at the *highest possible* layer to support interoperable communications and fine-grained handling of expressive messages between TCs. For example, highly interoperable Web-service interfaces are implemented on Web application servers, which could be huge, multi-layered, highly complex, and thus often difficult to manage securely. To assure the integrity of the software behind the interface, the TC must include the trusted implementation of its interfaces. However, this could raise serious concerns:

– *Differences of abstraction layers between communication interfaces and TCs*: It is difficult to build an assured implementation which handles both enforceability and communication control, because of the gap between the layer of abstraction for interfaces and that for the trusted components which can be build from the bottom. The communication stack is usually provided by different mechanism from the TC.
– *Multiple TCs in a node*: It is quite common that a communication stack implementing an interface is directly controlled by a node, not by any TCs. In such cases, the interface implementation is never included in a single TC.

Due to these observations, there exists a non-trivial gap between "secure communications between nodes" and "secure communications between trusted components". This gap brings the needs for an independent implementation of the interface to be placed at the boundary and for a secure intra-node communication mechanism to be placed between that implementation and the trusted components.

1.3 Our Contribution

To fill the gap between the TC and the interfaces, this paper introduces the notion of a *Secure Message Router (SMR)*, a domain-independent, easy to verify, multi-functional communication wrapper which mediates the communications between trusted components.

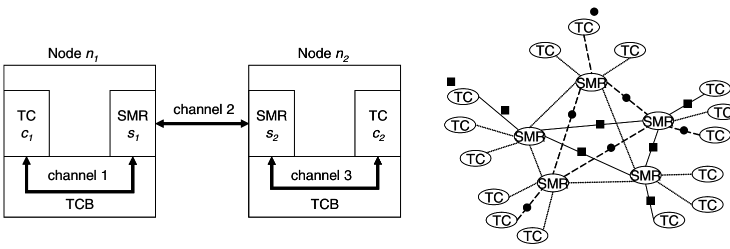


Fig. 2. Secure Messaging Router model

The model of SMR is described in Figure 2. The SMR is placed in each node in a coalition, and mediates the communications between TCs. The SMR works as a component which is governed by the TCB in the same node. The SMR employs the TCB's functions in order to control communication done by the TCs and to detect the state change of the TCs. The TCB controls inbound-outbound communication with TCs so that the communication cannot be done without SMR mediating it. This property provided by the TCB allows the SMR to perform complete mediation of communication done by TCs in the same node. Also, the SMR can manage the list of TCs and their states, and detect change of their states which are continuously observed by the TCB.

We say a communication channel from c_1 to c_2 is *end-to-end secure* if the rule derived from the policy for a domain which c_1 belongs to is enforced all of the messages which are transferred from c_1 to c_2 over the channel. The end-to-end secure communication channel between TCs is mediated by the SMRs s_1 and s_2. The secure channel between c_1 and c_2 consists of the three independent communication channels between c_1 and s_1 (channel 1), betweens_1 and s_2 (channel 2), and between s_2 and c_2 (channel 3). The SMR s_1 works as the end-point of channel 2 between nodes n_1 and n_2, while different secure channels such as channel 1 inside the node n_1 or channel 3 inside the node n_2 between SMR are established. We call channel 2 an *inter-communication channel*, and channel 1 or 3 an *intra-communication channel*. Thus, the inter-communication channel and the two intra-communication channels are collaboratively integrated with the mediation of the SMRs in order to establish an end-to-end secure channel between TCs. The correctness of the behavior of s_1 and s_2 is crucial to connect three secure channels in an end-to-end secure manner. This correctness is attested by integrity assurance which is checked by the TCs c_1 or c_2. This check is domain-independent, in the sense that the correct behavior of the SMR is defined regardless of the design or policy of any specific TVD. On the other hand, the attestation between TCs is domain-specific, which allows the designer of a TVD to support application-specific, finer-grained requirements for TVD.

Besides these features, the SMR provides the following capabilities which are required to establish a TVD.

– *Policy-based message routing*: In principle, only a single SMR is running on each node, while multiple TCs could be contained in the node. The end-

point of an inter-communication channel is an SMR, and thus the single SMR needs to be shared by multiple TCs. A capability for message routing allows nodes to participate in multiple coalitions simultaneously, just by each node maintaining multiple TCs and by the SMRs controlling the destinations of the message transfers according to the domain policy.
- *Secure message translation*: Some messages transmitted over the inter-communication channel could be domain-specific, in other words, dependent on which domain the message is transferred to. For example, to enable mandatory access control (MAC) in the TVD, a security label should be attached to the data which is conveyed in the messages transmitted. If heterogeneous implementations of TCs in coalitions are considered, the security label should be translated to appropriate security labels that can be handled by each TC. The SMR properly translates the messages from and to each TC according to the domain policy.
- *State Management*: Some of the domain policy is stateful in the sense that the destination of a message transfer could be dependent on the state of the coalition, (considering the history of message transfers, status of the TC, etc.) In addition, application-specific policies could be transactional, in the sense that the series of messages needs to be handled sequentially in order to form a unit of execution. The SMR provides stateful and transactional operations for routing and translating messages. The state of the domain is maintained by the SMR or possibly by the TCs in a decentralized manner. The SMR provides a function to integrate the domain state among the SMRs and TCs, which could be used through a unified interface when the SMRs or TCs in the domain evaluating the domain policy.
- *Thin clean implementation*: The SMR provides only a small set of functions for mediation: routing, translation, and state handling. This limited functionality allows us to implement SMR as a domain-independent and reusable trusted software component. Moreover, the feather-weight implementation makes it easy to obtain security assurance in the integrity of the SMR.

This paper is structured as follows: In Section 2, we describe the framework for realizing trusted virtual domain and discuss issues affecting the end-to-end secure communication addressed in this paper. We present our new approach, the Secure Messaging Router (SMR) and describe several protocols based on the SMR in Section 3. Section 4 discusses possible implementations of the SMR. Related works is discussed in Section 5. Finally, we conclude in Section 6 with a summary of our results and consider issues that still remain to be addressed.

2 Flexible Framework for Trusted Virtual Domain

The notion of a TVD can be implemented independently and concurrently using a variety of underlying technologies and mechanisms. One of the promising implementations of TVD is an approach using hypervisor-based isolation coupled with Trusted Computing Group (TCG)-based verification.

TCG technology provides hardware-based assurance of software integrity. This is based on a security module called the Trusted Platform Module (TPM) which is usually implemented as a tamper-resistant hardware module. The TPM measures and reports platform integrity correctly in a manner that cannot be compromised even by the platform owners or the software running on it. The platform measurement mechanism provides strong evidence for enforceability in Definition 1. Also, a challenge-response type confirmation protocol called TCG-attestation allows a remote challenger to verify the precise configuration and state of a computing platform in a reliable way, by using a tamper-resistant secure subsystem. This feature supports the realization of authenticity in Definition 1.

Another important requirement for enforceability and its authenticity is to guarantee the security property that any access to resources must be governed by the domain policy. This can be viewed as stating that all resources controlled by the TC are virtually isolated within exclusively controlled compartments. This fundamental feature is called "containment" in the context of TVD [1]. Hypervisor technologies such as the Xen Hypervisor provides very strong isolation among TCs on the same node, while secure OSes such as SELinux or sandbox mechanisms such as a JVM can also provide isolation among applications running in the same execution environment.

How to realize the authenticated secure channel and policy governed communication in Definition 1 depends on who mediates the communications between TCs and what kinds of information the messages contain. For example, sHype[2] presents how to establish secure communication channels between virtual machines running on Xen hypervisor, where the communications are governed by the MAC policy and the messages transmitted over the channel are associated with MAC labels. In this case as illustrated in Figure 3 (a), a secure communication channel is directly established between VMs only with the mediation of the hypervisor which can be verifiably assumed to be a TCB. This type of communication channel provides a very strong security assurance since it requires no mediation except for a TCB. However, some situations require finer-grained control of the channel and the messages on them. Typical examples include a policy for information flow control or a usage control policy. For example, a member in a coalition may be authorized to send information categorized as an "announcement" at most once a day, as long as the approvals from at least k of the members for that day are attached to the message, but the recipients will be limited to the members who gave approvals. This kind of fine-grained, stateful, and application-specific control is not suitable for the TCB for the following reasons.

– Configuration of the TCB, in general, is controlled only by strictly authorized subjects (such as node administrator).
– Flexible configuration of the TCB degrades its assurance, especially if the assurance of the TCB is attested by using the integrity measurement. (For example, under the Common Criteria evaluation, certification should be issued only for a rigorously fixed configuration of the system.)
– The TCB is placed at the lowest layer, and therefore the granularity of the control can be coarse compared with the application-specific requirements.

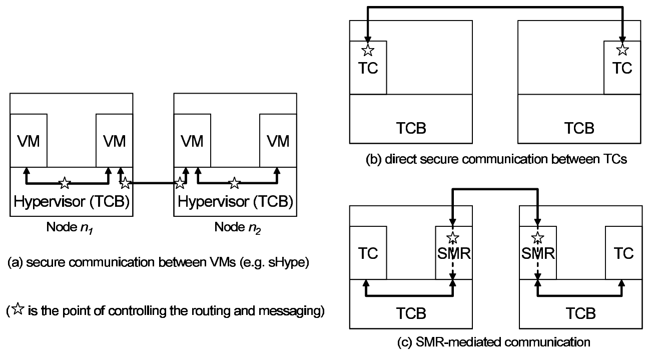

VM
VM
VM
VM
Hypervisor (TCB)
Hypervisor (TCB)
Node n_1
Node n_2
(a) secure communication between VMs (e.g. sHype)
(☆ is the point of controlling the routing and messaging)
TC
TC
TCB
TCB
(b) direct secure communication between TCs
TC
SMR
SMR
TC
TCB
TCB
(c) SMR-mediated communication

Fig. 3. Models of inter-communication channel

> For example, some domain policy might require not only the security label associated with the data, but also more detailed information on who, when, and how the data has been created or manipulated in order to determine if the data transfer is allowed. In general, such state information is finer-grained compared with the information which is observable from the TCB. Furthermore, expressive representations of the data need to be handled in order to support heterogeneous implementations of TCs.

Therefore, an independent mechanism which handles the communications between TCs needs to be provided outside the TCB in order to protect the security of the TCB. There are two types of approach as illustrated in Figure 3 ((b) and (c)). In the first approach, the TC contains the communication stack to establish an end-to-end secure communication channel directly between the TCs. It is easy to establish a secure channel over a public channel by using existing techniques such as SSL/TLS or Web Service (WS) security. Also, arbitrary granularity of the messages can be handled if every TC in a coalition is aware of how to communicate with the others. This is reasonable in a single domain setting, but not in a collaborative multi-domain setting. Specifically, if a node manages multiple TCs in order to participate in multiple independent coalitions and wants to enjoy collaborative activities (such as data transfers, sharing resources, or event notifications) across the multiple coalitions, every TC need to prepare communication stacks on a domain-by-domain basis. This limitation is because the control point of routing and messaging provided by communication stack is tightly bound to a core part of a TC. To overcome the limitations of the first approach and to deal with interoperable communications for multi-domain settings, our proposed approach using the Secure Message Router (SMR) can be used. The SMR can be seen as a divided portion of the TC which only deals with the mediation of secure communications (which is why we call it a "router"). Figure 3 (c) illustrates our approach using SMR, where an end-to-end communications link between TCs is established with the mediation of the SMRs, and this is easily extensible to support multiple TCs simply by routing the messages at the SMR.

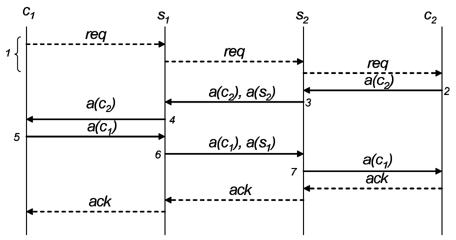


Fig. 4. Establishing end-to-end secure channel between TCs

3 Secure Messaging Router

This section gives an informal description of our proposed approach, *Secure Message Router (SMR)*, which provides complete mediation for the secure communication between TCs to establish a coalition based on the Trusted Virtual Domain (TVD) model.

3.1 End-to-End Secure Channel

Figure 4 depicts the protocol to establish a secure communication channel by using the remote attestation sub-protocol of the integrity measurement, which allows a remote challenger to verify the precise configuration and state of a computing platform. TCG-attestation is one of the promising implementations of this sub-protocol based on a tamper resistant secure subsystem such as TPM. Here, we will discuss how to use the sub-protocol to achieve our goal rather than how the sub-protocol works in detail. Without loss of generality, we focus on a protocol between two nodes hereafter though we believe there is no significant difficulty to extend the mechanism to multi-party scenarios. Suppose the two nodes are n_1 and n_2 and each node n_i, for $i = 1, 2$, manages a TC c_i and a SMR s_i, where c_i has an internal communication channel only with s_i, while s_i also has an external communication channel. We refer to the remote attestation sub-protocol which allows a challenger x to verify y as $RA(x, y)$.

This protocol allows c_1 to establish an end-to-end secure communication channel with the trusted mediation of s_1 and s_2. The basic idea of this protocol is to establish mutual trust based on the integrity assurance of the end points as well as the assurance of the SMRs. Since the message transfers are mediated, c_1 needs to invoke the remote attestation sub-protocol $RA(s_1, s_2)$ and $RA(c_1, c_2)$ simultaneously in order to check the capability of mediation. To establish mutual trust, the counterpart c_2 also needs to confirm the capabilities of c_1 and s_1 by using $RA(c_2, c_1)$ and $RA(s_2, s_1)$. Note that the TC does not need to confirm the integrity assurance of the SMR within the same node since the capability of the SMR for complete mediation is guaranteed by the TCB in the same node. Therefore, in this protocol, s_i and s_2 work as trusted communication wrappers for c_1 and c_2, respectively. On the other hand, the SMR acts as the liaison for the

TC in the same node from the viewpoint of the remote nodes. Thus, the SMR might need to monitor the integrity of the TC in the same node, since the SMR is responsible for the uninterrupted assurance as of the correct configuration and state of the TC in the same node within a session of the secure channel.

3.2 Multi-domain Handling

The basic functions provided by an SMR are the support of *translation* and *routing*, which allows the node to join with multiple coalitions based on the TVD model. Figure 5 illustrates the intuitive scenario for the two nodes n_1 and n_2. Suppose n_1 manages the SMR s_1 and two TCs c_1^A and c_1^B where c_i^x indicates a trusted component which is managed within c_i as a member of a coalition x. For this example, n_1 is participating in two coalitions A and B, and n_2 is participating in A and E.

The specifications of the TC could be node-specific in a heterogeneous computing environment. For example, to enforce a MAC policy such as the Bell-LaPadula policy in a coalition, a security label associated with the data needs to be transferred from node to node. If more fine-grained control is considered, messages which contain more information, such as contexts, states, or obligations, need to be exchanged between nodes in an expressive but interoperable way. The SMR translates the node-specific messages to interoperable messages and vice versa according to translation rules compliant with the domain policies. In the case of Figure 5, when n_1 joins those coalitions, n_1 sets up a translation function t_1 and its inverse t_1^{-1} in the SMR s_1. The function t_1 translates an outbound message into an interoperable message, while t_1^{-1} translates an in-bound message into a node-specific (application-specific) representation. These translations support not only simple replacement of the term but also finer-grained, constraint-based control complying with the domain policy, e.g., certain constraints prohibit translation, or a part of the message is encrypted under certain conditions. The translation functions t_1 and t_1^{-1} could be changed over time if the node joins or leaves the domain. When those functions are changed, all of the secure channels which has been already established and mediated by the SMR must be re-established because the security of the channel is dependent on the integrity assurance of the SMR.

Routing is another important function of an SMR. For example, in Figure 5, if c_1^A wants to send a message to all of the members in a coalition A, c_1 can do so simply by designating the destination in the message and by sending it to s_1, c_1^A never manage any list of members in the coalition A. Similarly, c_1^A can send the message to c_2^A with the mediation of s_2 simply by designating the destination c_2^A. Note that the correctness of the behavior of the SMR ensures that the message is never transmitted to any other TC such as c_2^E. Intuitively, as illustrated in Figure 5, SMRs in a TVD collaboratively establish a virtual boundary which protects the TCs in the TVD in order for the coalition of the TCs to meet the TVD requirements described in Definition 1. The interactions among multiple domains are fully controlled over their boundaries, and the SMRs work as a gateway across the boundaries. The SMR's abstraction of the communication

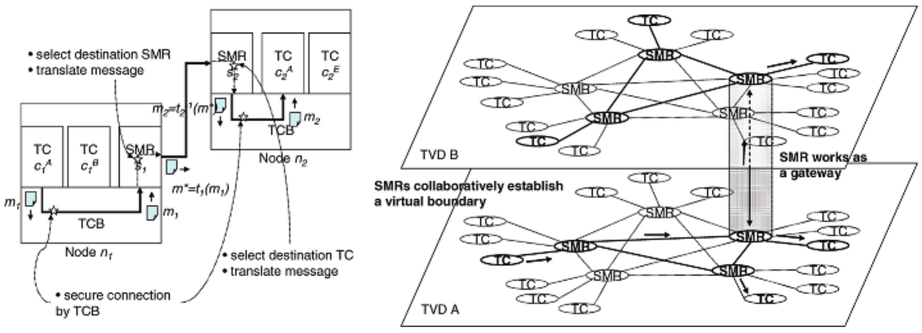


Fig. 5. Multi-domain handling

layer from the domain-specific layer allows the nodes in a coalition to simplify the management of the TVD and the specification of the domain policy.

3.3 TVD Management

When a node wants to join an existing coalition, a node prepares a TC and sets up rules for routing and translation in the SMR, and establishes end-to-end secure communication channels with all of the members in the coalition. This should be done at the time of joining, but some delegation of authority reduces the overhead. For example, all members in a coalition can give some special member (an administrative node) the right to check the membership qualifications of the new member.

The functionality of SMR is optimized for the objective of establishing TVD, in the sense that the SMR provides only the minimal but commonly required functionality for TVD management. In order to facilitate the management of the coalition, the SMR provides a reporting interface which can be accessed from any other node or administration service. This can be used in order to observe the configuration and status inside the node and to offer state management. This interface is also used for the remote attestation sub-protocol as mentioned in Section 3.1. The reporting interface does not allow any input such that it causes a state transition of the SMR, and therefore it does not introduce any side effects affecting the integrity assurance of the SMR.

The SMR is responsible for continuously observing the members in a coalition in order to confirm if the assurance is still valid. To prevent unnoticed alteration of the configuration or the state of the TC, the SMR provides the current status of the TC through the reporting interface. This can be implemented by periodically using the internal attestation protocol (with support by the TCB [7][8]).

4 Implementation of the SMR

Our model of the SMR does not limit the applicability to any specific implementation, though the feature of domain-independent attestability of the SMR requires a formally-verifiable implementation with minimal but sufficient functionality. A possible implementation could be a thin clean implementation of a Web service handler running over a micro-kernel OS with the smallest set of functionality. In this case, the SMR is introduced within a virtual machine fully controlled by the hypervisor, which is considered to be a TCB. The attestability of the implementation running over the micro-kernel OS has been thoroughly discussed in the literature (e.g. [2]). The reason to exploit the Web service is its interoperability and security[9]. The Web service standard provides interoperability to realize easy connectivity over heterogeneous computing environments, and in addition, the Web service security mechanism allows us to establish secure communications even in the presence of an intermediary mediator, which could be regarded as a significant advantage compared with the other secure channel technologies such as SSL/TLS. Another extreme approach could be a hardware implementation such as an appliance, secure coprocessor, or hardware extension to the node, because it is widely accepted that hardware is more difficult to crack than software. In that case, the assurance of the capability of the SMR must be provided by a different mechanism, such as the Common Criteria evaluation.

A TCB needs to provide two functions; 1) controlling inbound-outbound communication with TCs so that the communication cannot be done without SMR mediating it, 2) monitoring the states of the TCs so that the SMR can detect the change of their states. Furthermore, according to Definition 1, the TC needs to work as a reference monitor for the resources under the control of the TC. A virtual-machine-based implementation meets these requirements. The Virtual Machine Monitor (VMM) works as the TCB, and the Virtual Machines (VMs) running on the VMM are considered as the TCs. In this case, the VM isolation and communication control mechanism provided by the VMM allow the SMR to fully mediate the communication done by the TCs. The VMM can measure the state of the VM by the integrity measurement mechanism such as the TCG technology. The resources in the VM are governed by the policy enforced by the OS which is introduced in the VM.

A VMM supports hardware-level isolation between VMs. This allows the explicit control of communication, while the granularity of the control is bound to the VM-level, because the VMM is not aware of any internal detail of the VM. In the meanwhile, the TCs could be placed at the higher layer. For example, the Integrity Measurement Architecture (IMA) [8] enhances Linux by a TPM-based Linux Security Module in order to generate verifiable representative information about the software stack running on the system. In this case, a process can work as a TC if the integrity of the process is verified by the IMA, which works as a TCB in the sense that the TCB guarantees the TC's correct behavior by allowing the process to access only privileged resources and to communicate with outside only with the mediation of the SMR.

A distributed information flow control is one of the most attractive application of the TVD. Though a TC controls the information flow inside it, the SMR controls the information flow across the TCs. Furthermore, even in the internal information flow, some style of the control requires a domain-wide context which is never in the TC. The SMR integrates such domain-wide context by accessing the TC internally or by communicating with the other SMRs in the domain. The TC provides the SMR with an implementation of a callback interface which is predefined by the SMR. The TC can also access the SMR through an interface for accessing the domain-wide information. This allows the TC to delegate the domain-wide task such as the membership service or state management.

The SMRs collaboratively manage and share the list of the TCs and their state in the domain. When the node joins or leaves a domain, the state of the SMR in the node is modified because the SMR needs to maintain the state of the domain and the domain policy. Since the end-to-end secure channel between TCs requires that the correctness of the SMR's behavior is properly verified by an integrity assurance mechanism, the state change of the SMR triggers the channel re-establishment. This might be concern when the size of the domain is huge or the network structure is complicated. Another issue is the policy representation for multiple domains. Currently, we are assuming the existence of the mutual agreement on how to represent and evaluate the policy, while this might be too strong when we support highly heterogeneous environment. We need a common understanding on the representation of the policy, as well as a negotiation mechanism to establish mutual agreement on it.

5 Related Work

The Trusted Computing Group [7] has been gaining more attention than before, and various use-cases leveraging TCG technologies have been identified. Sailer et al. [10] utilizes TCG integrity measurements and attestation to protect remote access points, in order to enforce corporate security policies on remote clients in a seamless and scalable manner. One of the most active areas in the TCG is the Trusted Network Connect (TNC), with the main idea of using platform integrity information for network authorization control. One binding of TNC to an existing protocol is to use the TLS extension header [7] to extend EAP-TLS to support TCG attestation within that protocol.

NetTop [11] uses VMWare [12] to isolate execution environments, and allows connecting isolated environments to each other to establish a network of secure environments, and leverages secure OS such as SELinux to enhance the security on the host and the guest OS. Terra [13] realizes isolated trusted platforms on top of a virtual machine monitor, and allows attestation by using a binary image of each virtual machine, such as virtual disks, virtual BIOS, PROM, and VM descriptions. Terra exploits non-TCG based attestation to check the software stacks running in the guest OS, to build trusted relationship between multiple VMs.

Recent efforts on mitigating the drawbacks of TCG attestation include the Semantic Remote Attestation [14], which leverages language-based security and trusted virtual machines to verify the characteristics of a VM in a more semantic manner including attestation of dynamic, arbitrary, and system properties as well as the behavior of the portable code. Property-based Attestation [5,15] proposes an attestation model with a trusted third party that translates low-level integrity information into a set of properties. WS-Attestation [9] proposes to exchange attestation in the form of a credential which asserts properties and binds those properties to hash-value-based attestation generated by a TPM chip, while protecting the configuration details from potentially malicious challengers.

The notion of an SMR and its goals have a lot in common with the Law-Governed Interaction (LGI), a notion that originally introduced by Minsky [16] with its prototype developed in subsequent work [17,18,19,20]. LGI is a decentralized coordination and control mechanism for distributed systems. The policy management mechanism in the LGI allows a distributed group of heterogeneous nodes to engage in a mode of interaction governed by an explicitly specified policy, called a *law*, which is enforced on every node in order to create a coalition in which members can rely on each other to comply with the given law. Their Moses middleware implements the concept of LGI. Even though our trusted virtual domain based on the SMR provides not only the policy-governed communications among nodes, but also infrastructure-level support for coalition management fully integrated with integrity-based assurance of the configuration and status of the node, nevertheless, the notion of LGI could significantly reduce the complexity of inter-node communication management. In fact, the statements of domain policy in the TVD model are initially specified at an abstract level, and decomposed in order to be enforced on the communications and behaviors of the TCs in each coalition.

Yin and Wang proposed an application-aware IPsec policy system as middleware to provide Internet applications with network-layer security protection [21]. In order to introduce application context into the IPsec policy model, they use a socket monitor which detects the socket activities of applications and reports them to the application policy engine. They also propose an application specification language to configure and distribute application-specific policies. Our approach has some similarity with their approach in terms of bringing the higher-layer's context and requirements such as interoperability to the lower-layer's base security model (the TCB in our case). Their approach employs the network layer's security mechanism as the basic security infrastructure, while our approach employs the isolation mechanism such as hypervisor technology and the integrity-based assurance mechanism such as the TCG technology. Furthermore, the SMR supports a coalition of TCs to establish a trusted virtual domain.

6 Conclusions

Despite the increasing needs for coalition-based resource sharing, establishing trusted coalitions of nodes in untrusted computing environments is a long-standing yet increasingly important problem. A Trusted Virtual Domain (TVD) could be a solution to this problem by establishing trusted coalitions based on integrity assurances in heterogeneous and highly decentralized computing environments. However, the integrity assurance mechanism which is the basis of the TVD needs to be designed carefully with consideration of modern computer architectures that consist of multi-layer stacks of software, such as hypervisors, virtual machines, operating systems, JVMs, middleware, etc. The gap between ”secure communication between nodes” and ”secure communication between trusted components” requires a new notion of an SMR, which provides a number of functions such as end-to-end secure channel establishment, policy-based message translation and routing, and attestability through configuration-fixed, clean implementations.

In this paper, we considered virtual machine-based implementations with Web service interfaces as a possible implementation, but the issues related to formal verification of such implementations still remains to be solved. The approach using the language-based information-flow analysis[6] could be the next direction of this research.

The authors wish to thank our colleagues at IBM who helped us with their insights in earlier discussions of our collaborative work on TVD. In particular, we thank Matthias Schunter, Reiner Sailer, Ronald Perez, John L. Griffin, Yasuharu Katsuno, Megumi Nakamura, and Seiji Munetoh. We also benefited from insightful comments by Christian Stueble and Chris I. Dalton. Part of this work has been sponsored by the Ministry of Economy, Trade and Industry, Japan (METI) under contract, New-generation Information Security R&D Program.

References

1. Anthony Bussani, John Linwood Griffin, Bernhard Jansen, Klaus Julisch, Guenter Karjoth, Hiroshi Maruyama, Megumi Nakamura, Ronald Perez, Matthias Schunter, Axel Tanner, Leendert Van Doorn, Els A. Van Herreweghen an Michael Waidner, and Sachiko Yoshihama. Trusted Virtual Domains: Secure Foundations For Business and IT Services. In *IBM Research Report RC23792, IBM Corporation, November 2004 (available from `http://www.research.ibm.com/ssd_tvd`).*
2. Reiner Sailer, Trent Jaeger, Enriquillo Valdez, Ramón Cáceres, Ronald Perez, Stefan Berger, John Linwood Griffin, and Leendert van Doorn. Building a MAC-based security architecture for the Xen open-source hypervisor. In *Proc. of 21st Annual Computer Security Applications Conference (ACSAC 2005), Tucson, AZ, USA, pages 276–285, December 2005.*
3. John Linwood Griffin, Trent Jaeger, Ronald Perez, Reiner Sailer, Leendert van Doorn, and Ramon Caceres. Trusted virtual domains: Toward secure distributed services. In *IEEE First Workshop on Hot Topics in System Dependability (HotDep2005), Yokohama, Japan, June 2005.*
4. Hiroshi Maruyama, Frank Seliger, Nataraj Nagaratnam, Tim Ebringer, Seiji Munetoh, Sachiko Yoshihama, and Taiga Nakamura. Trusted platform on demand. In *IBM Research Report RT0564, IBM Corporation, February 2004.*
5. Jonathan Poritz, Matthias Schunter, Els Van Herreweghen, and Michael Waidner. Property attestation - scalable and privacy-friendly security assessment of peer computers. In *IBM Research Report RZ3548, IBM Corporation, May 2004.*
6. Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. In *IEEE Journal on Selected Areas in Communications, Vol 21, No. 1, January 2003.*

7. Trusted Computing Group, `http://www.trustedcomputinggroup.org/`.
8. Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proc. of the 11th USENIX Security Symposium. USENIX, San Diego, California, Augest 2004.*
9. Sachiko Yoshihama, Tim Ebringer, Megumi Nakamura, Seiji Munetoh, and Hiroshi Maruyama. WS-attestation: Efficient and fine-grained remote attestation on web services. In *Proc. of International Conference on Web Services (ICWS 2005), Orlando, Florida, USA, July 2005.*
10. Reiner Sailer, Trent Jaeger, Xiaolan Zhang, and Leendert van Doorn. Attestation-based policy enforcement for remote access. In *Proc. of the 11th ACM Conference on Computer and Communications Security (CCS2004), Washington, October, 2004.*
11. HP NetTop: A Technical Overview, `http://h71028.www7.hp.com/enterprise/downloads/hp_nettop_whitepaper2.pdf`.
12. VMWare, `http://www.vmware.com/`.
13. Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proc. of the 19th Symposium on Operating System Principles(SOSP 2003), October, 2003.*
14. Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation - virtual machine directed approach to trusted computing. In *Proc. of the 3rd Virtual Machine Research and Technology Symposium, San Jose, CA, USA, May 2004.*
15. Ahmad-Reza Sadeghi and Christian Stuble. Property-based attestation for computing platforms: Caring about properties, not mechanisms. In *Proc. of the 20th Annual Computer Security Applications Conference (ACSAC2004) December, 2004.*
16. Naftaly H. Minsky. The imposition of protocols over open distributed systems. *IEEE Trans. Softw. Eng.*, 17(2):183–195, 1991.
17. Naftaly H. Minsky and Victoria Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering and Methodology*, 9(3):273–305, 2000.
18. Xuhui Ao and Naftaly H. Minsky. Flexible regulation of distributed coalitions. In *Proc. of the European Symposium on Research in Computer Security (ESORICS2003), Norway, October 2003.*
19. Moses - LGI, `http://www.moses.rutgers.edu/`.
20. Law Governed Interaction (LGI): A Distributed Coordination and Control Mechanism, `http://www.moses.rutgers.edu/documentation/manual.pdf`.
21. Heng Yin and Haining Wang. Building an application-aware IPsec policy system. In *Proc. of USENIX Security Symposium '05, Baltimore, MD, August 1-5, 2005.*

License Transfer in OMA-DRM

Cheun Ngen Chong[1], Sorin Iacob[2], Paul Koster[1],
Javier Montaner[3], and René van Buuren[2]

[1] Philips Research, Prof. Holstlaan 100, 5656 AA Eindhoven, The Netherlands
{jordan.chong,r.p.koster}@philips.com
[2] Telematica Instituut, P.O. Box 589, 7500 AN Enschede, The Netherlands
{sorin.iacob,rene.van.buuren}@telin.nl
[3] Vodafone Group Research & Development, Gelissendomein 5, 6201 BM Maastricht, The Netherlands
javier.montaner@vodafone.com

Abstract. Many novel usage scenarios for protected digital content are emerging. Some of these are content transfer, by which users give away, borrow, or re-sell their protected content to other users. The transfer must abide by the rules defined by a rights issuer. The concept of authorized domain complicates the transfer operation. The main contribution of this paper is that we present a full-fledged design and implementation of the transfer operation for one of the major DRM standards, namely Open Mobile Alliance (OMA) DRM. The transfer operations include device compliance check, transfer between two devices (from two different domains), and license revocation within a domain. Our approach extends the user experience in using the DRM-protected digital content.

1 Introduction

Many novel digital content usage scenarios of Digital Rights Management (DRM) [1, 17] are emerging. The concept of authorized domain is proposed by DVB Consortium (http://www.dvb.org/) to support high usability in DRM [7, 10]. The user is able to use the digital content on any device, which belongs to her authorized domain, in any manner she likes. For instance, she can move the content freely or make and distribute replicas of the content within her authorized domain.

Nowadays, the users interact with each other more frequently in using the digital content. One of the user-to-user interaction is content *transfer*, as a gift, a loan or a resale. After transfer, the license (and the associated content) is bound to another user. Intuitively, this grants a user more freedom in using her digital content. There are potential benefits for the content providers as well. A content provider could distinguish between the costs of a license that can be transferred or cannot be transferred. Additionally, an increased distribution of the digital content by reselling might attract the users to buy original content from the content providers.

We envisage that the transfer operation in a DRM system especially in an authorized domain will become increasingly important. In this paper, we propose a theoretical approach and provide an implementation based on one of the main DRM standards, namely Open Mobile Alliance (OMA) (`http://www.openmobilealliance.com`). We propose our approach based on OMA-DRM because OMA-DRM provides a full-fledged implementation to develop DRM systems on mobile devices. Thereby, by extending OMA-DRM to facilitate the transfer operation, we are able to develop a complete DRM system that is capable of supporting wide variety of usage scenarios.

Our approach improves user's experience of using the DRM-protected digital content in her authorized domain. As far as we know, this paper is the first proposal of a full-fledged design implementation of transfer operation in a DRM standard, namely OMA-DRM.

The remainder of the paper is organized as follows: Section 2 discusses our approach and its implementation in OMA-DRM. Section 3 analyzes the security of our approach. Section 4 describes our prototype. Section 5 briefly explains some related work. Finally, section 6 concludes the paper.

2 Our Approach

In this section, we elaborate on our approach to implement license and content transfer in OMA-DRM. There are several approaches available supporting transfer of license, as discussed in section 5. We propose an approach that can be seamlessly implemented in OMA-DRM at the same time fulfilling some reasonable requirements as listed in section 2.3.

2.1 Players

We identify four main players corresponding to the OMA-DRM v.2.0 [14], as shown in Figure 1, namely Compliance Server, Rights Issuer, and two users namely Giver and Taker. The Compliance Server acts as a trusted third party that proves the compliance of the devices a user possess in her domain. The Compliance Server has a direct (business) interaction with the Rights Issuer (RI).

The Rights Issuer manages Giver's and Taker's authorized domain. When a user adds a device in her domain, she has to initiate the ROAP Join Domain Protocol [14] from the Device or from a Website of a Rights Issuer that can send a trigger, etc. The management of OMA-DRM authorized domain is centralized, i.e., the rights issuer can keep a record of all the devices in the user's domain.

The Rights Issuer also generates and distributes licenses (and the associated digital content) to a user's device or domain. The Giver initiates the transfer of a license, and Taker receives the transferred license (and the associated digital content). Giver and Taker have their own domain. There are a number of devices whose compliance is checked and verified using proofs obtained from the Compliance Server. Transfer is performed between Taker's Device and Giver's Device, as shown in Figure 1.

As shown in Figure 2, a connected Device can reach the Rights Issuer by establish a network connection with the Rights Issuer directly, e.g. via wired or wireless connection. On the other hand, a unconnected Device can reach the

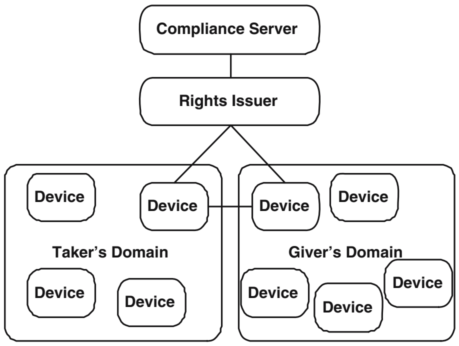


Fig. 1. The overview of the license transfer concept

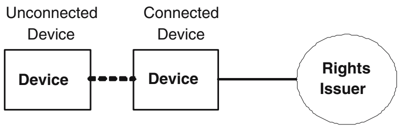


Fig. 2. The reachability of a device to a rights issuer

Rights Issuer via a connected Device, which acts as a proxy having a direct network connection to the Rights Issuer. This is as defined by OMA-DRM v.2.0 [14].

When a device is capable of reaching the rights issuer in real-time to perform transfer operation, we call it an *online transfer*. Otherwise, it is an *offline transfer*. The offline transfer can be performed when two devices can connect each other, e.g. via wired (e.g. USB) or wireless (e.g. Bluetooth) connection without connecting to the rights issuer. However, the rights issuer can still "participate" in offline transfer by using pre-defined usage rights (e.g. by specifying in the license that off-line transfer is not allowed).

2.2 Assumptions

We make the following assumptions under which our solution is valid:

– An OMA authorized domain has been established for the users. All devices belonging to the domain may maintain a replica of each domain-based license and therefore can render the domain-based digital content.
– Device compliance can be verified by the Compliance Server. After compliance verification, the device is trusted, i.e., it enforces the rights stated on the license correctly.
– All devices have OMA-DRM Agents [15].
– We assume that Giver's and Taker's domains are managed by the same Rights Issuer and the content stays under control of the Rights Issuer.
– We do not consider stateful license [14] in this paper.

2.3 Requirements

In order to support the rightful and secure transfer of licenses between users, we define the following requirements that have to be satisfied:

- Rights Issuer Control: The Rights Issuer shall be able to control the transfer operations, either in real-time or after the transfer is complete. The Rights Issuer should be able to keep track of the digital content before and after the transfer.
- Compliance Check of Participants: The Rights Issuer, the devices of Giver and Taker can achieve mutual compliance check to perform the transfer operations. This is only required in some specific cases, i.e., when transferring some secrets such as the content encryption keys.
- High Usability: The users, i.e., both Giver and Taker can perform the transfer operations with high convenience. The users do not need to connect their devices to the Rights Issuer in real-time (i.e., without network connection) to perform the transfer operations.
- Successful Completion: All transfer operations should be completed successfully. If errors occur during the transfer operation, the Giver, Taker and Rights Issuer will not suffer from any severe loss e.g. permanent loss of a license.

2.4 Four Steps

As shown in Figure 3, our approach has four steps. In this section, we elaborate on the protocols to achieve license and content transfer. In OMA-DRM, a license is called Rights Object (RO) [16]. Hereafter in this paper licenses will be referred to as Rights Objects, or RO. There are two types of RO, namely domain-based RO and device-based RO. A user can render the domain-based RO in every device that belongs to her domain, whereas the user can only render the device-based RO on a particular device.

Device Compliance Check. As mentioned in section 2.3, compliance check is only required for some specific cases. Both the Taker's and Giver's device connect to the Compliance Server via a Rights Issuer (RI) to obtain a proof of compliance, i.e., compliance token.

Online Certificate Status Protocol (OCSP) [11] is a method for determining the revocation status of X.509 certificates [9]. The mechanism requires a connection to an OCSP server (also called OCSP responder) in order to check the validity of a given certificate. OMA DRM clients (v.2.0) [15] already use this functionality to determine the revocation status of Rights Issuers.

Certificate Revocation Lists (CRLs) [4] are another mechanism that can be used to control the status of certificates. The advantage of OCSP over CRL is that it does not require the maintenance of long revocation lists in every device. An OCSP Responder (OR) returns a signed response indicating that the status of a certificate supplied in the request is 'good', 'revoked' or 'unknown'.

The compliance token is retrieved with the help of a RI as shown in Figure 3. It is assumed that the device has already registered with the RI with ROAP 4-Pass

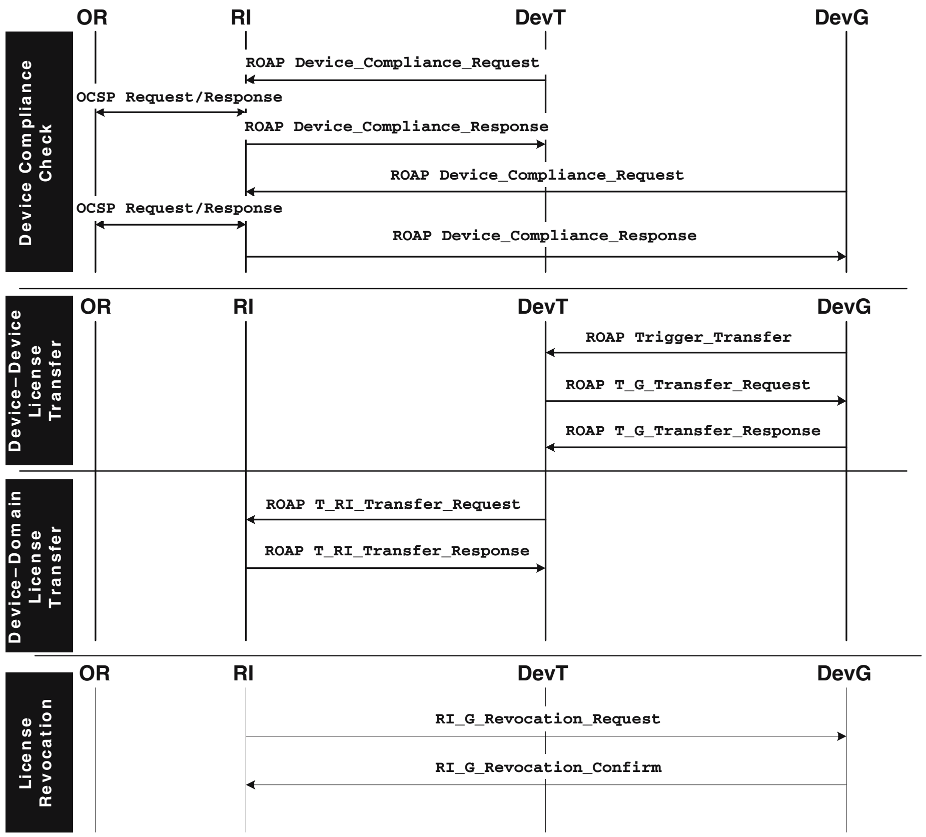

Device Compliance Check
OR
RI
DevT
DevG
ROAP Device_Compliance_Request
OCSP Request/Response
ROAP Device_Compliance_Response
ROAP Device_Compliance_Request
OCSP Request/Response
ROAP Device_Compliance_Response
Device–Device License Transfer
ROAP Trigger_Transfer
ROAP T_G_Transfer_Request
ROAP T_G_Transfer_Response
Device–Domain License Transfer
ROAP T_RI_Transfer_Request
ROAP T_RI_Transfer_Response
License Revocation
RI_G_Revocation_Request
RI_G_Revocation_Confirm

Fig. 3. OMA implementation of transfer

Registration Protocol [14]. The messages are similar to the ROAP Registration Request and Registration Response messages.

The processing at the RI will be exactly the same as for the ROAP registration protocol. The only difference is the certificate (chain) that is being validated. The ROAP registration protocol validates the RI certificate while the new ROAP device compliance protocol validates the device certificate. We elaborate on the messages of the protocol listed in Table 1:

M#1. The message is based on the ROAP Registration Request message. Since device and RI have already exchanged and validated their certificates at the ROAP Registration Protocol, it is not required to exchange this data again. The `trustedAuthorities` element is kept since the `DeviceCertificateChain` might differ from the RI certificate chain received at the ROAP 4-Pass Registration Protocol. Upon receipt of this message the RI will contact an OCSP responder as specified in the ROAP Registration protocol.

Table 1. ROAP protocol for Device Compliance Request, where the element DevX designates either Giver's or Taker's device

M#	Parties	ROAP Message
1	DevX → RI	`ROAP Device_Compliance_Request { trustedAuthorities }`
2	RI → DevX	`ROAP Device_Compliance_Response { DeviceCertificateChain, ocspResponse_DevX }`

M#2. The message is based on the ROAP Registration Response message. It includes a complete certificate chain for the device and the corresponding OCSP response, i.e., `ocspResponse_DevX`. The compliance token that is used in the Device–Device License Transfer is composed of the certificate chain and the OCSP response received in this message.

Device–Device License Transfer. To perform transfer, both the Taker and Giver's devices can be connected to each other (e.g. via wired connection with USB cable or wireless with Bluetooth). The Taker's device and Giver's device exchange messages, which include a valid compliance token. At the same time, the Taker's device checks its current status of reachability to the Rights Issuer. If the Taker's device is currently offline, the Giver's device transfers a device-based RO, i.e., one that the Taker can only access on her receiving device (for a period of time).

The permission for the Giver's device to issue a device-based RO for the Taker's device can be exclusively granted by the RI (e.g. through an additional usage rights specified in the original RO). The RO on the Giver's device is revoked temporarily through a process elaborated later. The RO replicas in the Giver's domain are then revoked permanently after the Taker requests for a domain-based RO from the RI.

Table 2 depicts the message exchange between the Taker's device (DevT) and the Giver's device (DevG) in transferring the RO.

M#1. The device DevG initiates the transfer operation by sending the ROAP Trigger Transfer message to DevT. The message includes the identity of the RO `RO_ID`, which indicates the RO that Giver wants to transfer to Taker.

M#2. After receiving the ROAP Trigger Transfer message, the device DevT sends a `ROAP T_G_Transfer_Request` message to the device DevG. The message must include the item `RO_ID` received from the device DevG earlier.

The message includes an indication of whether it requires a device-based RO it can use without contacting RI (typically DevT only sets this flag if it cannot reach RI in real-time – the decision is left to the device or the user if the device can reach RI and if it needs an RO for offline usage, respectively), i.e., `requestOfflineDeviceRO_DevT`. Compliance token must only be present if the item `requestOfflineDeviceRO_DevT` is set (i.e., `ocspResponse_DevT` and `certificateChain_DevT`).

Table 2. The contents of each ROAP message exchange in the protocol of Device–Device License Transfer

M#	Parties	ROAP Message
1	DevG → DevT	`ROAP Trigger_Transfer { RO_ID }`
2	DevT → DevG	`ROAP T_G_Transfer_Request { RO_ID, certificateChain_DevT, requestOfflineDeviceRO_DevT, ocspResponse_DevT }`
3	DevG → DevT	`ROAP T_G_Transfer_Response { original_RO_G, transferContext, device_RO_T, certificateChain_DevG, ocspResponse_DevG, certificateChain_RI, ocspResponse_RI }`

M#3. The device DevG constructs the message `ROAP T_G_Transfer_Response`. The message contains the original RO that is given away, i.e., `original_RO_G`. The message must contain the transfer context, which asserts that G transfers this RO to Taker and has or will revoke the RO replicas in G's domain (`transferContext`). The `transferContext` has the following form:

```
transferContext =
     { RO_ID, DevG_ID, DevT_ID, transferTime,
       requestOfflineDeviceRO_DevT
     }sig-DevG_PriKey
```

The item `transferTime` indicates the moment Giver creates the `transferContext` and is used as part of the revocation process. By making `transferTime` mandatory the protocol is limited to the Giver's devices that support OMA-DRM v.2.0 secure time.

The device DevG includes the items `device_RO_T` and `certificateChain_DevG`, `ocspResponse_DevG` if the message M#2 includes the `requestOfflineDeviceRO_DevT` flag. The device DevG verifies that the original RO allows the creation of `device_RO_T` (which is decided by the RI when Giver acquires the RO from RI). The device DevG takes its own capabilities into account such as its capability to create `device_RO_T`. The device DevG also verifies in this case that DevT's request is authentic, and verifies the compliance of DevT, both using `certificateChain_DevT` and `ocspResponse_DevT`.

If DevT requires a `device_RO_T` for offline use, but it cannot be honoured due to lack of capabilities then `errorNoOfflineROCapabilities` is returned. If the reason is based on policy then `errorPolicyDeniesOfflineRO` is returned. If DevT prefers a `device_RO_T` for offline use but it cannot be honoured due to any reason then the protocol continues with the next message, but without `device_RO_T` and any other parameter that is only required for offline use.

In case the item `device_RO_T` is present DevT verifies that also the other optional fields, namely `certificateChain_DevG`, `ocspResponse_DevG`, `certificateChain_RI`, and `ocspResponse_RI` are present.

In case `device_RO_T` must be generated, the device DevG processes the original RO, i.e., `original_RO_G` to make it a RO to Taker, which Taker can only use on

DevT. DevG adds the limitations in the rights expression of `device_RO_T` if the transfer policy prescribes this. This policy can be hard-coded and prescribed by for example CMLA [15], or it is present as part of `original_RO_G` (stated in the REL [16], which is defined by RI).

Upon receipt of the M#3 DevT verifies that the DevT receives `original_RO_G` and `transferContext` that correspond with the requested `RO_ID`. If DevT requires `device_RO_T` but does not receive it within a specified timeout then DevT aborts the protocol. This leaves both the Giver's device and Taker's device in a state that equals to the state before initiation of the transfer protocol.

Upon receipt of the M#3 that includes the device-based RO `device_RO_T`, the device DevT attempts to access the content according to the following rules. The device DevT verifies the authenticity and integrity of the item `original_RO_G` using the items `certificateChain_RI` and `ocspResponse_RI`. The device DevT verifies the authenticity and integrity of the message `transferContext` using `certificateChain_DevG` and `ocspResponse_DevG`. The device DevT verifies the authenticity and integrity of `device_RO_T` using `certificateChain_DevG` and `ocspResponse_DevG`.

The device DevT verifies that `RO_ID` of `original_RO_G` and `device_RO_T` are identical. The device DevT evaluates the rights expression of `device_RO_T` as defined by OMA-REL [16] and must in addition check that this rights expression is not broader than the rights expression contained in `original_RO_G`. In case of successful evaluation DevT uses the content encryption key (CEK) of `device_RO_T` to decrypt the content.

Device–Domain License Transfer. Device–Domain License Transfer is performed for the Taker to obtain a domain-based RO from the Rights Issuer. The Taker must present the transfer context to the RI. The RI can use the transfer context to further process the revocation on the Giver's domain.

Table 3 depicts the message exchange between DevT and RI to perform Device–Domain License Transfer.

M#1. The device DevT sends the `ROAP T_RI_Transfer_Request` message to RI, which includes some message items received from the device DevG at the Device–Device License Transfer operation (Table 2). The `ROAP T_RI_Transfer_Request` message is very similar to an OMA DRM v.2.0 ROAP acquisition request message. The message must include the message `transferContext`, and the Giver's original RO, i.e., `original_RO_G`.

The message also includes the Taker's domain information, i.e., `domainInfo_T`. The device DevT should have user consent which `domainInfo_T` to use if multiple options are available. As stated in section 2.2, we assume that the domains of G and T are managed by the same RI. Thus, the device DevT can initiate M#1 (in Table 3). Note that if T has no prior relation with RI or if it has no domain context for this RI, the device DevT must initiate the required ROAP Registration and/or Join Domain protocol(s) as specified by OMA-DRM v.2.0 before sending M#1.

Table 3. The content of each ROAP message exchange in the protocol of Device–Domain License Transfer

M#	Parties	ROAP Message
1	DevT → RI	`ROAP T_RI_Transfer_Request{ original_RO_G, transferContext, certificateChain_DevG, domainInfo_T}`
2	RI → DevT	`ROAP T_RI_Transfer_Response{ domain_RO_T, certificateChain_RI, ocspResponse_RI}`

M#2. The Rights Issuer RI processes `original_RO_G` and produces `domain_RO_T`, i.e., the domain-based RO dedicated to the Taker's domain. Before generating M#2, RI should verify the authenticity and integrity of `ROAP T_RI_Transfer_Request`, `original_RO_G`, `transferContext` including verification of the revocation status of the signers.

RI should also verify that `transferContext` indicates the device DevT as the recipient of the transfer and signee of the request. RI should also verify that DevT is a member of the indicated domain in `domainInfo_T`. RI uses `transferContext` to determine if the RI wants to deliver the domain-based RO.

If RI decides not to honour the request it must return an error indicating the type, e.g. `errorWaitForSuccessfullRevocation` to indicate that revocation must be finished first, `errorTransferFailed` to indicate that transferring failed persistently. Otherwise, RI must create a RO as it would do for the OMA DRM v.2.0 ROAP Acquisition Protocol and send this in `ROAP T_RI_Transfer_Response` message to the device DevT. The device DevT must process this message as if it concerns a ROAP Acquisition Protocol, including verification of authenticity, integrity and compliance of `domain_RO_T`, RI, etc.

License Revocation. In this section, we describe a best-effort license revocation mechanism. The purpose of license revocation is to revoke the license replicas in a user's domain before or after the transfer of the license and the associated digital content. License revocation makes the license (and replicas thereof) unavailable to the current owner's domain. This ensures that the transferred license becomes unavailable to each device in the Giver's domain. Since the completion of the revocation process cannot be instantaneous due to the offline devices in the Giver's domain, it is required to ensure the completion of revocation within a time interval that would be set by the RI.

To support a correct license revocation, all devices must maintain a copy of the license revocation list, that each device uses to check the validity of stored licenses. The license revocation list indicates the domain licenses that cannot be further used within that domain. This may raise the storage problem of huge license revocation list. However, with the rapidly increasing capacity of current storage space with the relatively reasonable price, this can be solved to a certain extent. We can also apply some optimization techniques, such as a compression method to efficiently store the revocation list on small devices. This, however, we leave outside the scope of this paper.

The license revocation process involves updating the revocation list stored on the device. The update is triggered by all domain devices when they connect to the RI, through a revocation list refresh request message. The actual update can be done either by allowing the Rights Issuer to push the new revocation list, or to piggyback the new revocation list to the devices when they request for a fresh compliance token.

During the Device–Device License Transfer operation discussed earlier, the revocation is initiated by the Giver's device (DevG). The item `transferContext` is generated at DevG, asserting that DevG has transferred the RO and has/will revoke the RO replicas in the Giver's domain. Both the DevG and Taker's device (DevT) possess the transfer context.

When either device connects to the RI, the transfer context is sent to RI as a notification that the RO has been transferred from Giver to Taker. Thereby, RI can initiate the license revocation – the RI sends a request for refreshing their RO revocation list (RORL) to all the Giver's devices in her domain (that happen to be online), i.e., `RI_G_Revocation_Request`.

All devices receiving the revocation request must replace their RORLs with the new one and where possible must delete the newly revoked RO. Other devices that are offline will receive the revocation request from the RI whenever they are online. Until then, the RO remains usable on those devices. However, the RI notifies Giver which of her device has not updated the RORL, as the RI keeps a record of devices that belong to her domain.

Until the new RORL is updated on all the devices in the domain (for some specific time) the RI maintains the following state information for the Giver's domain, as shown in Table 4. The timestamps (`Time_n`) is used to indicate the last time the corresponding device updates the RORL.

Each device receiving the request from the RI for revocation must update its RORL and must respond with the message `G_RI_Revocation_Confirm`, so that the RI can update the state information (as shown in Table 4). The message structure of the request and confirm messages are listed in Table 5. Note that the label DevG here indicates each of the devices belong to the Giver's domain.

M#1. The `RI_G_Revocation_Request` message is initiated by RI to DevG, i.e., one of the devices in Giver's domain, which connects to the RI to request fresh compliance token. When the user requests for a fresh compliance token, the user's device initiates the `RORLRefreshTrigger`. The `RI_G_Revocation_Request` message includes a new RORL, i.e., `RORL_new`, signed by the RI for integrity and authenticity.

M#2. Upon receipt the message, DevG updates the stored RORL, by replacing the old one with the newly received RORL from the `RI_G_Revocation_Request` message. To confirm the revocation, the Giver's device DevG responds to the RI with the `G_RI_Revocation_Confirm` message. The message includes the version, identifier, and update time of the RORL, i.e., `RORL_version`, `RORL_identifier`, and `RORL_time`, respectively. RI verifies if the `RORL_version`, `RORL_identifier`, and `RORL_time` received are correct to ensure the revocation has been done properly.

Table 4. State information of the Giver's domain maintained by the RI

Domain Composition	Active RORL
Current RI RORL	`RORL(Time_n)`
Device 1	`RORL(Time_n)`
...	...
Device n	`RORL(Time_some-time-ago)`

Table 5. The message structure of revocation request and confirm

M#	Parties	ROAP Message
1	RI $\rightarrow$ DevG	`RI_G_Revocation_Request { { RORL_new }sig-RI_PriKey }`
2	DevG $\rightarrow$ RI	`G_RI_Revocation_Confirm { { RORL_version, RORL_identifier, RORL_time }sig-DevG_PriKey }`

RI checks if all the recorded devices (see Table 4) have already updated the RORL.

We can also implement revocation locally prior to the transfer. In other words, the user *must* revoke all the license replicas within her domain before transferring. This can be expanded easily with our approach – the user must connect all devices in her domain to the Rights Issuer for revocation prior to the transfer. However, this is more inconvenient than our best-effort approach from the user's perspective. The user normally cannot anticipate which content she wants to transfer especially when the user cannot connect the Rights Issuer.

3 Threat Analysis

In this section, we analyze several threats to our proposed solution. Section 3.1 and section 3.2 are considered as a form of replay attacks.

3.1 Store on Non-compliant Storage

The user can move a license (or its replica) to a (non-compliant) storage medium. After the transfer and revocation, the user moves the license back to the device in her domain.

We propose using a license revocation list (or rights object revocation list, i.e., RORL) to keep a record of *all* the revoked license (see section 2.4). However, this solution seems to be confined by some limitations, which we believe can be overcome:

- The RORL may grow to a size that is unmanageable by devices with limited resources. Therefore, we need an optimization mechanisms to manage the RORL.

– The device must be *obliged* to re-connect the rights issuer for an up-to-date RORL. We must make sure that without an up-to-date RORL, the device (with the DRM agent built-in) cannot further distribute and/or render any content.

3.2 Illegitimate Transfer Proliferation

The user can perform *multiple* offline transfers from every device in her domain to all the devices of another user's domain. For instance, the Giver can use Device$_1$ to transfer a license to Taker's Device offline. After that, without connecting to the the Rights Issuer, the Giver could use Device$_2$ to transfer the replica of the transferred license to Taker's *another* Device. Instead of the same Taker (say Taker$_1$), the user can offline transfer to another Taker (Taker$_2$) device. These threats are caused by the license revocation approach discussed in section 2.4.

As discussed in section 2.4, to perform the transfer, the device must request a fresh compliance token from the Rights Issuer. Additionally, the Giver and Taker must connect to the Rights Issuer for the new digital content, etc. on a regular basis. Thus, the Rights Issuer can detect this form of replay attack if there are more than one transfer context for the same license has been generated (Note: only one transfer context is generated to transfer the license). Furthermore, there is a limited number of devices allowed in a user's domain (as defined by OMA-DRM [15]), which further alleviates this threat.

To tackle the threat of offline transfer to Taker$_1$ and Taker$_2$, if either Taker contacts the Rights Issuer requesting for the domain-based license, the Rights Issuer can notice there are two *same* transfer context. Thereby, the Rights Issuer can then detect the Giver's unauthorized multiple offline transfer. Moreover, the device-based license can only be rendered in within a short period.

Additionally, the Rights Issuer in our approach (based on OMA-DRM architecture [15]) keeps a record of all the available devices in the user's domain. Therefore, it is easy for the Rights Issuer to check if a particular device has been offline for a long period of time.

4 Prototype

To serve the purpose of proof-of-concept, a prototype is built implementing the proposed solution for content transfer. In the prototype, we have defined a use case as follows: Aaron and Jasmine love collecting music tracks from the 1950s. Each has a mobile terminal that contains the some music tracks and the associated rights. They want to trade some tracks using their mobile. They connect the mobiles and swap the content. After trading, both Aaron and Jasmine are not able to access their original content and licenses anymore.

Figure 4 illustrates part of the prototype that facilitates transfer operation between two mobile phones. We use the state-of-the-art broadband network protocols, Bluetooth protocols, Near-Field Communication (NFC) and General Packet Radio Service (GPRS) to facilitate the communications between these components. There are three main components in the prototype, namely:

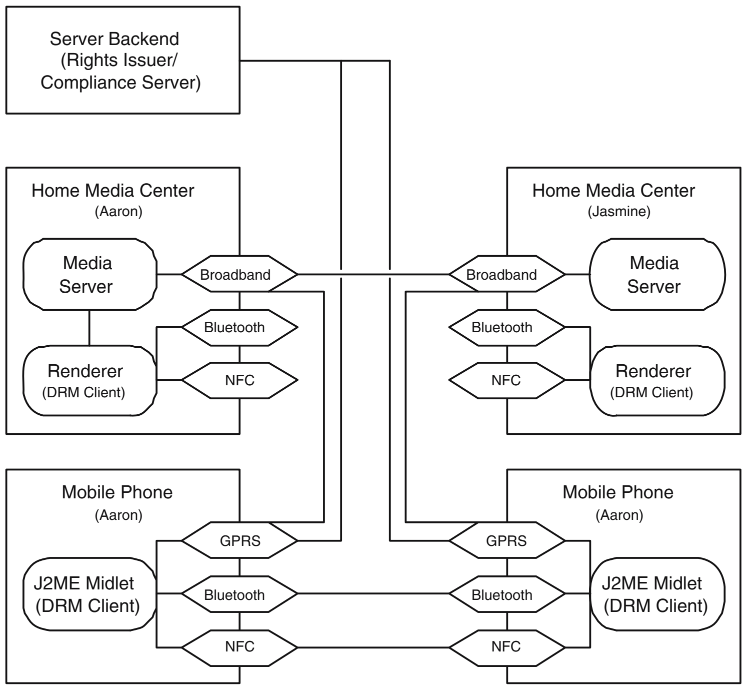


Fig. 4. The architecture of the prototype that implements transfer

– Backend Server: It implements the Rights Issuer and Compliance Server.
– Home Media Center (HMC): It implements the Media Server, which manages the user's content; and Renderer, which implements the OMA DRM Client.
– Mobile Phone: It implements J2ME Midlet, which is an OMA DRM Client.

Aaron and Jasmine each has his/her own Home Media Center and Mobile Phone. When Aaron initiates transfer operation to Jasmine on their mobile phones, the Device–Device License Transfer operation is initiated. After the Device–Device License Transfer operation, Jasmine can only render the transferred content on her Mobile Phone. She can connect her mobile phone to the Backend requesting for a domain-based RO so that she can render the content on her HMC. If the request is granted, the Backend contacts Aaron's HMC (assuming now that Aaron's HMC is connected) to initiate the revocation process. As the Backend Server keeps a record of all Aaron's devices in his domain, thus, the Backend Server can contact Aaron (e.g. via emails) for updating the revocation list on all his devices.

5 Related Work

The transfer rights can already be expressed in the available rights expression languages (REL) [6], such as XrML [5], ODRL [8], OMA-REL [16], and LicenseScript [2]. However, the underlying DRM architecture enforcing the aforementioned RELs does not support explicitly transfer.

Nair et al. [12] have proposed a set of protocols, which can achieve redistribution of the content from one user to another. However, they merely focus on the device-to-device distribution, i.e., they do not consider the concept of authorized domain. They also propose several payment methods for the redistribution, which considers the benefits of the rights issuer. However, the rights issuer does not participate in the process of redistribution in real-time.

Conrado et al. [3] have proposed an approach that supports anonymous transfer of the digital content (and the associated license) to another user. To perform transfer operation, the user must initiate a connection with the rights issuer requesting an anonymous license, which is a license that is not associated to any user. The user can thus transfer the anonymous license to another user. Thereby, the privacy of the user who receives the license is protected from the rights issuer. Prior to transfer, the user must first revoke all the license replicas in her authorized domain, i.e., pre-revocation.

Our approach does not consider privacy protection for user due to the requirement of Rights Issuer Control (as described in section 2.3). The Rights Issuer would like to keep track of the digital content before and after the transfer. However, our approach can be easily customized to support pre-revocation.

6 Conclusions

Digital Rights Management (DRM) attempts to control the user's access over the digital content, protecting the benefits of the rights issuers. The drawback is that the users are constrained in the usage of digital content. Thus, the concept of authorized domain is proposed to grant the users more freedom - the user is allowed to make replicas of the license (and the associated digital content) and use the content on all devices that belong to her domain.

The transfer operation is useful from the user's and also the rights issuer's perspective. The user can transfer a license and the associated content to another user as a gift, loan or resale. However, license transfer between two users becomes complicated due to the user's freedom granted by the concept of authorized domain. There are a number of license replicas in a user's domain, which have to be taken care of for the transfer operations.

We propose an approach for the transfer of license and content. The approach facilitates the compliance check of devices performing transfer, the secure license transfer between users' devices, and the efficient license revocation in the user's domain. We design a set of protocols, which extend the OMA-DRM interacting the Giver, Taker and Rights Issuer. The proposed approach is able to fulfill the following requirements:

– Rights Issuer Control: The Rights Issuer can decide if the Giver can issue a device-based license by stating the transfer rights on the license for offline transfer. When the Taker requests for a domain-based license for the received content, the Rights Issuer can determine to grant the Taker's request.

- Compliance Check of Participants: The Giver's and Taker's devices must possess an up-to-date compliance token to perform offline transfer. Thus, the devices must connect to the Rights Issuer to request for an up-to-date compliance tokens on a regular basis for offline transfer.
- High Usability: The Giver and Taker can perform offline transfer when there is no network connection, and the Taker can use the transferred license on the receiving device (within a limited time period).
- Successful Completion: We make the protocols for the transfer operations context-free. If errors happen, the user can resume the protocols by resending the messages (within a timeout).

As discussed in section 2, we assume that the Giver and Taker's domain are maintained by the same Rights Issuer. We envision the possibility to separate the domain management from the Rights Issuer, which may provide more business opportunities and high scalability. We consider this as our future work.

Acknowledgement

We would like to thank James Irwin, Tim Kerins, Milan Petkovic, Andrew Tokmakoff, Koen Vrielink and Tim Wright for their valuable comments.

References

[1] Chong, C.N., van Buuren, R., Hartel, P.H., Kleinhuis, G.: Security attribute based digital rights management (SABDRM). In Boavida, F., Monteiro, E., Orvalho, J., eds.: Joint Int. Workshop on Interactive Distributed Multimedia Systems/Protocols for Multimedia Systems (IDMS/PROMS). Volume 2515 of LNCS., Springer-Verlag (2002) pp. 339–352

[2] Chong, C.N., Corin, R., Etalle, S., Hartel, P.H., Jonker, W., Law, Y.W.: LicenseScript: A novel digital rights language and its semantics. In Ng, K., Busch, C., Nesi, P., eds.: 3rd International Conference on Web Delivering of Music (WEDELMUSIC), Los Alamitos, California, United States, IEEE Computer Society Press (2003) pp. 122–129

[3] Conrado, C., Petkovic, M., Jonker, W.: Privacy-preserving digital rights management. In: Secure Data Management 2004. Volume 3178., Springer-Verlag (2004) pp. 83–99

[4] Feghhi, J., Williams, P.: Digital Certificates: Applied Internet Security. Addison-Wesley (1998)

[5] Guo, H.: Digital rights management (DRM) using XrML. In: T-110.501 Seminar on Network Security 2001. (2001) Poster paper 4

[6] Guth, S.: Rights expression languages. In Becker, E., Buhse, W., Günnewig, D., Rump, N., eds.: Digital Rights Management: Technological, Economic, Legal and Political Aspects. Volume 2770 of LNCS., Springer-Verlag (2003) pp. 101–112

[7] Heuvel, S. van den, Jonker, W., Kamperman, F., Lenoir, P.: Secure content management in authorised domains. In: Int. Broadcasting Convention (IBC), Amsterdam, The Netherlands, Broadcastpapers Pty Ltd, PO Box 259, Darlinghurst, NSW, 1300, AUSTRALIA (2002) pp. 467–474

[8] Iannella, R.: Open digital rights management. In: World Wide Web Consortium (W3C) DRM Workshop. (2001) Position paper 23

[9] ITU: ITU-T Recommendation X.509, Information Technology, Open Systems Interconnection, The Directory: Authentication Framework. International Telecommunication Union, Place des Nations 1211 Geneva 20 Switzerland. (1997)

[10] Koster, R. P., Kamperman, F., Lenoir, P., Vrielink, K.: Identity based drm: Personal entertainment domain. In: Communications and Multimedia Security (CMS), 9th IFIP TC-6 TC-11 International Conference. Volume 3677 of LNCS., Springer-Verlag (2005) pp. 42–54

[11] Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, C.: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol (OCSP). Technical Report RFC 2560, VeriSign, CertCo, ValiCert, My CFO, and Entrust Technologies (1999)

[12] Nair, S., Popescu, B., Gamage, C., Crispo, B., Tanenbaum, A.: Enabling drm-preserving digital content redistribution. In: CEC 2005. Seventh IEEE International Conference on E-Commerce Technology (CEC 2005), IEEE (2005) pp. 151–158

[13] Niehüser, L.: The right to sell. INDICARE Monitor **1**(3) (2004)

[14] OMA: DRM specification: Candidate version 2.0. Technical Report OMA-TS-DRM-DRM-V2_0-20060303-A, Open Mobile Alliance (2004)

[15] OMA: DRM architecture: version 2.0. Technical Report OMA-DRM-ARCH-V2_0-20060303-A, Open Mobile Alliance (2004)

[16] OMA: DRM rights expression language v.2.0: version 2.0. Technical Report OMA-TS-DRM-REL-V2_0-20060303-A, Open Mobile Alliance (2005)

[17] Rosenblatt, B., Trippe, B., Mooney, S.: 4. In: Digital Rights Management: Business and Technology. M&T Books, New York, United States (2002)

Enhanced Security Architecture for Music Distribution on Mobile

Abdellatif Benjelloun Touimi[1], Jean-Bernard Fischer[2,⋆], Caroline Fontaine[3,⋆⋆], Christophe Giraud[4], and Michel Milhau[1]

[1] France Telecom R&D Div.,
2, avenue Pierre Marzin, 22 307 Lannion, France
{abdellatif.benjellountouimi, michel.milhau}@francetelecom.com
[2] Nagra France,
28, rue du Colonel Pierre Avia, 75 015, Paris, France
jeanbernard.fischer@nagra.fr
[3] CNRS/IRISA,
Campus de Beaulieu, 35 042 Rennes cedex, France
caroline.fontaine@irisa.fr
[4] Oberthur Card Systems,
4, allée du doyen Georges Brus, 33 600, Pessac, France
c.giraud@oberthurcs.com

Abstract. This paper presents the results of the French collaborative SDMO (Secured Diffusion of Music on mObiles) project, which aims at combining traditional security tools such as cryptography and smartcard with digital rights management and watermarking to reinforce the overall security of an audio content distribution service for mobile phones. The paper gives an overview of the system and of the security architecture and describes summarily the watermarking techniques employed.

Keywords: Content distribution, audio, mobile, digital rights management (DRM), smartcard, cryptography, watermarking, OMA DRM.

1 Introduction

The new generation of mobile networks and devices (2.5G, 3G and 4G) has opened up a promising market for multimedia content distribution services on mobile phones. However, before such services can be launched, the issues related to digital rights management (DRM) and the protection of copyrighted work have to be sorted out. Indeed, the existence of efficient compression formats for digital content, broadband capacity networks, and Peer-To-Peer file exchange have encouraged the piracy phenomena in the Internet world. The fear of duplicating this model in the mobile world is a major obstacle for the growth of a

⋆ Work done while at Oberthur Card Systems.

⋆⋆ This work was done when she was with CNRS/LIFL, Cité Scientifique, 59 655 Villeneuve d'Ascq cedex, France.

market for digital content distribution : if the different value chain actors (content providers, content retailers, network operators) cannot expect a fair income from their efforts, they will not enter this new market and the ultimate looser is the consumer.

To fight piracy, different approaches have been used to provide security for the distribution of audio content. Watermarking is one line of research that has been seen as very promising by the content industry to solve all their piracy ails. A lot of research has been done on watermarking during these past years [3]. However, if the technology, which consists of dissimulating information in a host signal, is mature, its applications in the real word are far from effective, especially for the protection against illegal copies. The failed experiment of the SDMI Forum in defining a system to protect copies based essentially on the watermarking technology has shown the limits of this approach [4].

The approach taken in the SDMO (Secured Diffusion of Music on mObile) project[1] aims at combining tightly the traditional security tools of cryptography and smartcard together with watermarking to strengthen the overall security of the audio content distribution service (streaming and download) for mobile phones. The main idea is to use watermarking as an active security tool to reinforce and broaden the scope of classical DRM. The project also takes advantage of the closed nature of the mobile environment, both at the network and the terminal sides, to build secure end-to-end architecture to protect audio content.

The paper first gives an overview of the protection system devised by the SDMO project for music distribution on mobiles. Section 2 describes the specifications of the system and the main usage scenarios. Section 3 describes the security architecture and analyzes how the astute interworking of cryptography, smartcards and watermarking produces a secure environment for content distribution.

2 SDMO System Description

2.1 SDMO Service and Requirements

Unlike the open Internet network, mobile networks are easier to control; indeed, the centralized and authoritarian control the operator has over his network allows for more possibilities for monitoring the traffic and possibly blocking data. Since some nodes handle all the data traffic, the usage of efficient tools to track illegal contents is really effective there. Moreover, the client in this context is well identified as a subscriber to the mobile service. This gives more means to prevent the consumption of illegal content at the end-user mobile phone.

The main objective of the SDMO project has been to define a secure audio content diffusion service in the controlled mobile world that reaches beyond to the open world of Internet. The problem was approached by first defining some ideal yet reasonable security requirements. In this context, the SDMO project defined two requirements (Figure 1):

[1] Funded by “Réseau National de Recherche en Télécommunication”, the French National Research Network on Telecommunication.

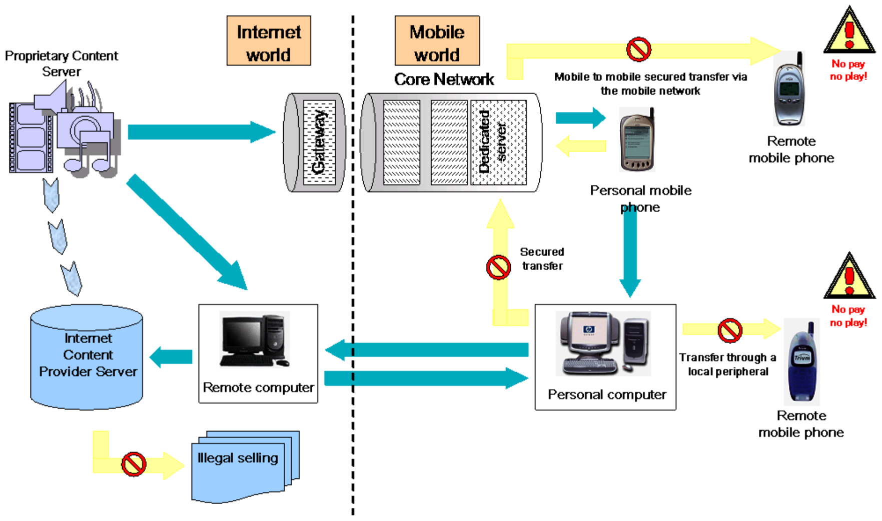


Fig. 1. Operational architecture

– *Maximizing security in the mobile domain.* This is done by prohibiting the consumption of illegal audio content on mobile devices, and the circulation thereof on the mobile operator's network. The aim here is to provide an end-to-end protection in this domain including the network and the terminal device.
– *Restricting the spread of illegal audio content in the Internet domain.* This objective is fulfilled by a constant background checking and tracing of the usage of illegal content on commercial web sites.

For a mobile operator aiming at offering and developing new services around a music catalogue, the assurance that no pirated content will circulate on his network will encourage the development of new business with the majors producing content. It is well known that the entertainment industry is very sensitive to such arguments. Checking the legality of the content during the transport through the network and in the terminal to prevent piracy is of prime importance.

To reach this purpose, SDMO intends to be a content security service provider offering the necessary tools to the actors of the content distribution value chain. In Figure 2, illustrating such a value chain, SDMO is positioned as a service for a mobile operator in agreement with an audio content provider, offering them:

– the *SDMO content preparation server*, ensuring the watermarking, compression, encryption and formatting of a music content file;
– the *SDMO Client* implemented on the mobile device allowing the playback of SDMO content and blocking illegal content;
– a *SDMO Web scrutator*;
– *SDMO Mobile network control* to stop illegal content from circulating on the mobile network.

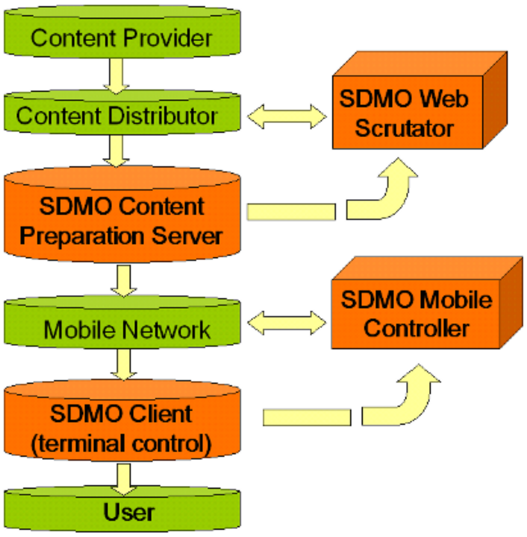


Fig. 2. The SDMO service elements and their position in the music content distribution value chain

2.2 Use Case Scenario

The project considers a classical content delivery scenario in which a mobile user buys a piece of music and acquires it from the content server. A pre-listening phase of limited duration of the audio content is possible at the same quality as the original one. Two content delivery modes are provided: download of the music file or streaming it by buying the listening rights. The usage rights are expressed and delivered in a separate licence file. The usages of the content by the user can be distinguished as follow:

– *Locally at the terminal device*: in that case, any use (play, copy, ...) is constrained by the usage rights granted in the licence;
– *Redistribution*:
  – *through the mobile network*: the user can transfer the audio content to another mobile phone. The content should not be played by the new user device before paying the corresponding rights. In the case of pirated content, it should not be played on mobile devices neither be circulating in the mobile network. A rigorous control of the content should be placed in some core network nodes.
  – *through the device peripherals (Infrared, Bluetooth, memory stick, ...)*: the mobile user can also transfer this content to and from his own personal computer via a peripheral. From there on, it becomes easy to transfer the content to other remote computers. It is also possible for this content to be placed in some web sites on Internet; for this reason, the content provider

servers should be continually examined in order to verify that they are not retailing illegal content. Another possible strategy consists in prohibiting such type of transactions by locking the device peripherals.

In the case of content exchange between two mobiles terminals, the *Mobile Network Controller* has the task of filtering illegal content.

2.3 SDMO Content Format

The audio content is compressed in MPEG-4 AAC-LC (Low Complexity profile) format at 24kbps bit rate. This audio codec is recommended by 3GPP as an audio codec for Packet Switch Services in the mobile network [1]. The resulting compressed bitstreams are encrypted using the AES-128 CBC algorithm, then put into 3GP file format for streaming or downloading. This file is associated with a licence in OMA DRM REL v2 format (based on ODRL language) indicating the usage rights and restrictions as well as the necessary decryption keys and the content identifier [8]. Such format, including the content and the licence, is called SDMO DRM format. To allow the detection of illegal content, two types of audio content are distinguished in the framework of SDMO system:

– *Controlled content*: necessarily in SDMO DRM format (*i.e.* encrypted and wrapped with SDMO DRM header + a separated licence) or another format provided by a legal distribution service.
– *Uncontrolled content*: in any format not associated with a DRM service.

With this distinction, a content can be (i) encrypted and in SDMO DRM format and hence it is controlled, (ii) or in clear and hence either free or pirated. In the latter case, the content was encrypted and the cryptographic protection has been circumvented; this is the situation where the watermark comes into play to distinguish between legal and pirated content.

3 Security

3.1 Separating Security Contexts

Analysis. The security of a mobile phone is relatively low, especially since the trend is towards more open operating systems. Mobile phones are more and more like PCs and are prone to the same ills: viruses, Trojan horses, etc. Moreover, the user might be inclined to deliberately load hacking programs in order to access protected content without payment of the licences. That is why the keys and the cryptographic algorithms are usually stored in the USIM (Universal Subscriber Identity Module), the smartcard which is in the mobile phone.

For a system relying on the super-distribution model, where the same content is always encrypted with the same key, the biggest threat is the compromise of the content key. If that occurs, it is easy to give everyone access to the content key, while still using the original ciphered content.

In order to prevent this threat, one has to provide a means to protect the content until the moment of the audio rendering. However, since the content is in compressed format, the decompression operation requires the deciphering of the content. Ideally, these two operations, deciphering and decompression, should be performed in a tamper-proof environment, the smartcard in our case. However, due to the limitation of the smartcard in terms of processing power, the file has to be (for the time being) decompressed in the mobile phone.

It is a classical paradigm, often seen in PayTV applications, to have the smartcard deliver the content keys to the decoder in order for it to decrypt the video stream. There, the keys are changed very frequently, so that intercepting the content keys by hacking a decoder and distributing them to would-be pirates is not a good solution, as by the time the key arrives at the destination, a new key is already in use. However, this model does not stand for a content that has a long lifetime, like in the distribution of musical content destined to be stored and played at will. If the key is compromised, not only is the content available in clear, but the pirates can use the legitimate content (*i.e.* encrypted content) for free and even sell the keys.

Solution. In the SDMO project, the smartcard is used to separate the security contexts of the SDMO distribution architecture and the SDMO player system. Indeed, the smartcard is the secure holder of the licence, and thus of the coveted file encryption key `ContentKey`. If the key were to be sent to the mobile phone to decrypt the file, it would be easy to hack one phone to recover the content key and then distribute it. So the simple solution is to have the smartcard decrypt the file internally.

In order to avoid the clear content to appear on the smartcard-to-mobile interface, the smartcard and the mobile phone set up a local secure channel, *i.e.* they set up a local session key which is updated from time to time, e.g. for each new musical content or at each start-up. Thus, after decrypting the content with the key `ContentKey` stored in the licence, the smartcard re-encrypts it with the local session key in order to send it to the player for further decryption, decompression and audio rendering.

Now, if a mobile phone is hacked and its keys are compromised, it does not affect the whole system and may not lead to massive piracy where content keys are distributed to be used to decrypt original content files.

A simple key establishment protocol. To allow session key establishment, every USIM card contains a common SIM-Player Authentication key `SimPlayerAuth-Key` which is stored in the smartcard in a secure environment (as for `UserKey`). On the other hand, each SDMO player is personalized with an individual key `PlayerKey` and its certificate, consisting simply in the `PlayerKey` encrypted with AES using the key `SimPlayerAuthKey`.

During the first step of the session key establishment, the Player generates a 16-byte random challenge and encrypts it with its key `PlayerKey`. This value concatenated with the Player certificate is sent to the USIM. The latter is able

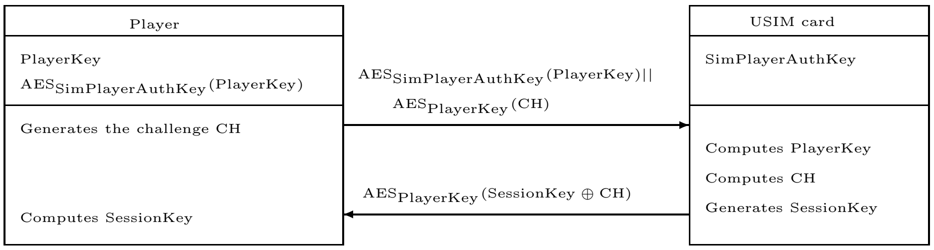

Player
PlayerKey
$AES_{SimPlayerAuthKey}(PlayerKey)$
Generates the challenge CH
Computes SessionKey
USIM card
SimPlayerAuthKey
Computes PlayerKey
Computes CH
Generates SessionKey
$AES_{SimPlayerAuthKey}(PlayerKey)||$
$AES_{PlayerKey}(CH)$
$AES_{PlayerKey}(SessionKey \oplus CH)$

Fig. 3. The session key establishment protocol

to decrypt the Player certificate in order to recover `PlayerKey` which is used to decrypt the challenge.

Secondly the USIM generates a 16-byte session key `SessionKey` which is XORed with the challenge; the result is encrypted by using `PlayerKey` and is sent to the Player.

Finally the Player recovers the session key by decrypting with his key the value sent by the card and by XORing the resulting value with the challenge.

In order to avoid a zero session key attack which can be mounted by a hacking card by sending in the last step the encrypted challenge received at the first step, we prohibit session keys equal to zero. Figure 3 summarizes the session key establishment.

As players are not tamper-proof devices, hackers may succeed in recovering a couple (Player key, Player certificate) and propose programs that authenticate themselves to the USIM and thus gain access to the content in clear. To counteract this kind of piracy, a revocation list is maintained in the data base of the SDMO server in order to blacklist the keys from hacked Players and is send to the USIM with the licences.

The question now is: is this enough to ensure reliable mutual authentication and good session keys? In our setting, where the basic premise is that the smartcard is secure, the session key is unilaterally chosen by the smartcard. A more complex authentication, such as SSL, would be time consuming and would not take advantage of the dissymmetry in the tamper-proof character of the smartcard versus the mobile. Clearly, a pirate could set up a connection with a card by sending a random message and would thus establish a secure channel for which he does not know the key, which is useless in our case as the only thing that will happen is that he will have the content encrypted with an unknown key, meaning he gained nothing.

3.2 Adapting OMA ROAP Security Context for Smartcards

In order to protect the musical content, the first level of protection in SDMO involves a classical DRM architecture, more precisely the OMA (Open Mobile Alliance) DRM v2 architecture [6].

The goal for OMA is to provide an open system, where any device of any kind can connect to any server and negotiate a licence. In the SDMO architecture, the only device that is supposed to enter in communication with the server is the user's smartcard. This smartcard has been personalized in order to fulfil this mission, with applications and data, especially keys, that permit access to the server.

Our goal has been to make the most out of the tamper resistance of the smartcard in order to simplify the protocols and the computation power needed, yet still remaining compatible with OMA DRM v2 protocols [7] and licence formats [8].

The ROAP protocol defined by OMA DRM v2 uses a 4-pass registration protocol in order to authenticate a device and set up a security context, *i.e.* a shared session key that will be used to secure the subsequent communications. This protocol makes heavy use of public key infrastructure in order to certify the device and thus is very demanding in computation for both server and device. This is an issue when the user's device is a mobile phone, since public key cryptography demands intensive computations and is accordingly power hungry. Moreover, it is difficult to store and manipulate secret keys inside a mobile, because it is not a secure computing environment, opening the door to key compromising and account hacking. That is why the keys and cryptographic algorithms are stored in the USIM.

Looking at the ROAP protocol, it appears that once the device is registered to the server, there is no further need to run through the registration again as long as the security context is not compromised. So that if the keys are well protected, as in a smartcard, there is no need to ever register again once the first security context is established. Furthermore, there is only one security context per couple server-device, so that, in a system like SDMO where the server is unique, each smartcard has only one security context to manage. Therefore, this once-in-a-lifetime registration protocol can occur before the smartcard is issued to the user in the secure environment of the personalization process of the smartcard. The protocol can then be simplified to the extreme: the shared key is simply generated by the SDMO server and stored both in the smartcard and in the users account database on the server. From that moment on, no one will have access to the shared key outside of the SDMO licence server and the SDMO application in the smartcard.

With regards to security, one gets thus the full security of the OMA DRM v2 ROAP protocol without the hassle of public key cryptography and with the added security that no operation involving the keys is performed outside a secure environment. OMA specifies the AES as the algorithm to use in case of symmetrical encryption cryptography, so we choose the 128-bit AES in Cipher Block Chaining mode with an initial value set to zero.

Our scheme works as follows. In preparation of the system, the SDMO licence server manages a secure database of users accounts, whereas each user is issued an ID and a key. For each of the users, a smartcard is personalized with his specific data: `UserID` and `UserKey`, according to the ROAP (Right Acquisition

Object Protocol) registration protocol [7]. From there on, the user's smartcard and the SDMO licence server can communicate securely by using the shared `UserKey` with the ROAP protocol. This key cannot be compromised as it is stored in the server and in the smartcard only, both secure environments, and the key never leaves these.

Each content file is encrypted with its particular key `ContentKey` according to OMA DRM specifications [5]. Once the content is encrypted, it has no intrinsic value any more without the corresponding key; so the encrypted file can be freely distributed by any means, and in particular using super-distribution. The file is encapsulated in an OMA DRM envelope, so that the content identifier `ContentID` can be easily recovered. On the system side, the final step is to store both `ContentID` and `ContentKey` in a secure database managed by the SDMO licence server.

When a legitimate user wants to play a file, he purchases the right to do so from the SDMO licence server which downloads the corresponding licence containing the key `ContentKey` into the user's smartcard, using the ROAP protocol with the security session (based on `UserKey`) established at the smartcard personalization.

3.3 Improving Security on the Local Link with Watermarking Techniques

Let us first remark that an audio file can be transferred through any interface, like the radio (operator's network), Bluetooth or Infra-Red. This does not have any impact on the lawfulness of the download. For example, a legally purchased audio file can be temporally stored on a memory card or a computer in order to free space on the mobile phone. The crucial information at the terminal level is to know if the hardware should render the file or not. So, the right question is: "Has the use of this file *originally* been protected by SDMO?"

- If no, we assume that it is a content which can be played freely.
- If yes, we have to check that a licence has been purchased before playing.

It is difficult to find a technical solution based only on cryptographic tools to such a question, especially because of the "originally" term. Due to the fact that the decompression process has to be performed on a content in clear, cryptographic protection is of no use after this step. So that there are several means for a pirate to either recover the content in clear or to play pirated content:

- hack the keys of one player and replace the player's software by its own
- monitor the buffers of the player and read or write data on the fly
- insert a pirate software on the interface of the software modules of the player or of the audio rendering hardware

Hence, we decided to supplement the encryption based security with the embedding of a watermark signal, which remains imperceptible but present in the whole audio signal, even when it is converted to an analogical format and played.

The captured signal still contains a watermark, and the use of such a pirated content can be detected. On the content server side, the watermark is embedded before the encryption and DRM encapsulation.

In order to overcome the security hole at the decompression step, we use a watermarking technique whereby the content is identified by a specific identification number, `WmID` (*SDMO Watermark Identifier*). This number is recovered in the rendering module, at the very last moment before actual rendition of the signal.

The system works as follows. When the content file is prepared for distribution, the `WmID` is computed and inserted as a watermark in the signal, then the file is ciphered with the `ContentKey`. The `WmID` is stored in the content database along with the `ContentKey`. When a user asks for a licence, the `WmID` is added in an specific ciphered field of the licence. When the mobile starts playing a content file, the `WmID` is transmitted to the rendering module via a secure link (that has been set up in the same manner as on the smartcard-to-mobile interface). Thus, the rendering module simply extracts the watermark of the content currently playing and compares it to the content it is supposed to be playing. As long as they coincide, rendition goes on. As soon as there is a discrepancy, rendition stops independently of the player software. In order to be able to play free non-SDMO content, as long as no watermark is detected, rendition is performed.

So a normal execution works as follows. The encrypted file is delivered to the player and its licence is delivered to the USIM (file and licence can be sent separately). When the user select the content to be played, the player sends the file to the USIM. Upon receiving the header of the file, the smartcard checks the licence (key and usage rights), decrypts the content and sends a new version of it to the player, encrypted with the session key. At the same time, it sends an encrypted command containing `WmID` to the rendering module. The player decrypts the content, decompresses it and sends it to the rendering module. The hardware starts rendering the content and checks the watermark on the fly.

When a user wants to play an audio file illegally, several cases can occur.

- The USIM does not contain the licence corresponding to the content. The file will not be decrypted by the smartcard.
- The player is not a SDMO player. It will not get a session key from the USIM card and will not be able to decrypt the content.
- The file is pirated (it was originally protected by the DRM system, but the DRM protection has been stripped off and the file decrypted); however, it still contains a SDMO watermark. The player will decompress the content, but the hardware module will stop render it as soon as the watermark is detected.

If the file is not encrypted and does not contain a SDMO watermark, it is considered as a legal clear content. Thus, the player will decompress it and the hardware module will render it since no watermark is detected.

3.4 Improving Security in the Network with Watermarking Techniques

Given the insertion of the SDMO Watermark Stamp, bound to the content in a discrete way (without any additional signaling bit), an appropriate probe can check the content in the mobile network and will be able to discriminate between SDMO pirated content and non-pirated content and act in an appropriate way.

As the SDMO Watermark Stamp has to be checked at the network level, it has to be just one bit indicating that the signal is legal or not. This test must be simple, fast enough and robust.

Several cases can occur:

– If a content moving through the mobile network is encrypted (in SDMO DRM format), then no watermark check has to be performed as the content is in a protected format.
– If a content is not encrypted, a watermark check has to be performed. If the audio signal carries the SDMO Watermark Stamp, it is a SDMO pirated content that has been decrypted or "captured" somehow (e.g. in the vicinity of the digital to analogue conversion). The network should take action, for example block the pirated content, reject it or report some information to the provider for further use.
– If the content is not encrypted and does not carry the SDMO Watermark Stamp, then the content is assumed not to be pirated and can transit through the mobile network.

3.5 Specifications of the Watermarking Techniques

We considered two kinds of watermarks, both hidden in the audio files in an imperceptible and robust way: a *SDMO Watermark Stamp* and a *SDMO Watermark Identifier*. The first one can be extracted to prove that the file has been pirated and is used for network control; the second one gives an identifier which is compared during audio rendering with the one extracted from the licence in the terminal.

The main specifications of these watermarks, resulting from the network and the terminal requirements, are given in Table 1.

Table 1. SDMO watermarks properties

Specifications	SDMO Stamp	SDMO Identifier
Capacity (bit)	1	32
Duration (seconds)	15	15
Desired bit error rate	10^{-5}	10^{-5}
False alarm rate	10^{-7}	10^{-7}

Table 2. BER performances of the SDMO watermarking algorithms

	Spread Spectrum Scheme		Scalar Costa Scheme	
Attack/Compression	Condition	BER	Condition	BER
Without attacks		0		0
White Gaussian Noise	SNR = 40 dB	$4 \cdot 10^{-5}$	WNR = 39 dB	$3 \cdot 10^{-6}$
	SNR = 25 dB	$5.1 \cdot 10^{-3}$		
Level adjustment	Gain = +6 dB	$1.2 \cdot 10^{-6}$	Gain = +10 dB	$1.2 \cdot 10^{-5}$
	Gain = −6 dB	$2.5 \cdot 10^{-6}$	Gain = −10 dB	$2.3 \cdot 10^{-3}$
Cropping	Rate = 10^{-5}	$3.7 \cdot 10^{-6}$		
	Rate = 10^{-4}	$7.4 \cdot 10^{-4}$		
De-synchronization			1 to 10 sample/s	$3.17 \cdot 10^{-6}$
Compression	AAC@24 kbit/s	10^{-3}	AAC@24 kbit/s	10^{-5}

The watermark payload was set to 32 bits every 15 seconds, to prevent listening to a significant part of the audio content in case of unauthorized listening.

The watermarking detection key stored in the terminal could be compromised. The use of an asymmetric watermarking algorithm is hence recommended at the level of the terminal. However, the actual state of the art of asymmetric watermarking shows that no algorithm is robust enough. As the watermark robustness is the most important criterion, we decided to use a symmetric algorithm and maximize key's security in the terminal. Two different blind symmetric watermarking technologies have been explored and developed: algorithm based on *time domain Spread Spectrum* [2] and on *Scalar Costa Scheme* [10, 11]. As it is not the purpose of this article to detail these techniques, we will only expose their behavior. We considered robustness according to ordinary or malicious transformations (noise addition, filtering, cropping, de-synchronization, ...), and of course took into account the AAC compression step.

The performances of the two algorithms have been also evaluated in terms of BER against different attacks (see Table 2).

4 Conclusion

This paper presents the SDMO system and its specifications aiming to maximize the audio content security in the mobile domain. This objective is reached by defining an end-to-end security architecture involving both the terminal and the network in a complementary way. Such architecture uses watermarking in close relationship with cryptography, rights management and smartcard in the terminal.

The implementation of the proposed architecture has shown how flexible the SDMO specifications are when combined with an existing DRM standard like

OMA DRM v2. According to this security domain, the SDMO Content protection solution must be seen as an added value security tools.

Acknowledgements

This work was funded by the French ministry of industry through the RNRT project SDMO. The authors would like to thanks all the project participants from Laboratoire Signaux et Systèmes (LSS), Laboratoire d'Informatique Fondamentale de Lille (LIFL), France Telecom Research and Development Division, Orange, Faith Technology and Oberthur Card Systems.

References

1. 3GPP TS 26.234. "Transparent end-to-end Packet-switched Streaming Services (PSS); Protocols and codec". Release 6, 2005.
2. C. Baras, N. Moreau. "An audio Spread Spectrum data hiding system with an informed embedding strategy adapted to a Wiener filtering based structure". In Proceedings of IEEE International Conference on Multimedia and Expo, Amsterdam, July, 2005.
3. M. Barni, F. Bartolini, "Watermarking Systems Engineering: Enabling Digital Assets Security and Other Applications", Marcel Dekker Ed., February 1, 2004.
4. S. Craver, J. Stern, "Lessons learned from SDMI", Workshop on Multimedia Signal Processing, October 3-5, 2001, Cannes, France.
5. Open Mobile Alliance, Content Format, Candidate Version 2.0, OMA-TS-DRM-DCF-V2_0_20050901-C.
6. Open Mobile Alliance, DRM Architecture, Candidate Version 2.0, OMA-TS-DRM-AD-V2_0_20050908-C.
7. Open Mobile Alliance, DRM Specifications, Candidate Version 2.0, OMA-TS-DRM-DRM-V2_0_20050915-C.
8. Open Mobile Alliance, Rights Expression Language, Candidate Version 2.0, OMA-TS-DRM-DRM-V2_0_20050825-C.
9. RNRT SDMO Project, `http://www.telecom.gouv.fr/rnrt/rnrt/projets/res_02_74.htm`
10. A. Zaidi, R. Boyer, and P. Duhamel, "Audio Watermarking under Desynchronization and Additive Noise Attacks". IEEE Transactions on Signal Processing, vol. 54, no 2, February 2006, pp. 570–584.
11. A. Zaidi, J. Pablo Piantanida, and P. Duhamel, "Scalar Scheme for Multiple User Information Embedding". Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2005, Philadelphia, USA, March 17-23.

A Formal Model of Access Control for Mobile Interactive Devices

Frédéric Besson, Guillaume Dufay, and Thomas Jensen*

IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France

Abstract. This paper presents an access control model for programming applications in which the access control to resources can employ user interaction to obtain the necessary permissions. This model is inspired by and improves on the Java security architecture used in Java-enabled mobile telephones. We consider access control permissions with multiplicities in order to allow to use a permission a certain number of times. An operational semantics of the model and a formal definition of what it means for an application to respect the security model is given. A static analysis which enforces the security model is defined and proved correct. A constraint solving algorithm implementing the analysis is presented.

1 Introduction

Access control to resources is classically described by a model in which an access control matrix specifies the actions that a subject (program, user, applet, ...) is allowed to perform on a particular object. Recent access control mechanisms have added a dynamic aspect to this model: applets can be granted permissions temporarily and the outcome of an access control depends on both the set of currently held permissions and the state of the machine. The most studied example of this phenomenon is the stack inspection of Java (and the stack walks of C♯) together with the *privileged method calls* by which an applet grants all its permissions to its callers for the duration of the execution of a particular method call, see *e.g.* [1, 3, 5, 8]. Another example is the security architecture for embedded Java on mobile telephones, defined in the Mobile Information Device Profile (MIDP) [13] for Java, which uses interactive querying of the user to grant permissions on-the-fly to the applet executing on a mobile phone so that it can make internet connections, access files, send SMSs *etc.* An important feature of the MIDP model are the "one-shot" permissions that can be used once for accessing a resource. This quantitative aspect of permissions raises several questions of how such permissions should be modeled (*e.g.*, "do they accumulate?" or "which one to choose if several permissions apply?") and how to program with such permissions in a way that respects both usability and security principles

* This work was partly funded by the IST-FET programme of the European Commission, under the IST-2005-015905 MOBIUS project.

such as Least Privilege [12] and the security property stated below. We review the MIDP model in Section 2.

In this paper, we present a formal model for studying such programming mechanisms with the purpose of developing a semantically well-founded and more general replacement for the Java MIDP model. We propose a semantics of the model's programming constructs and a logic for reasoning about the flow of permissions in programs using these constructs. This logic will notably allow to prove the basic security property that *a program will never attempt to access a resource for which it does not have permission.* Notice that this is stronger than just ensuring that the program will never actually access the resource. Indeed, the latter property can be trivially achieved in systems with run-time checks—at the expense of accepting a security exception when an illegal access is detected. The basic security property is pertinent to systems with or without such dynamic controls. For systems without any run-time checks, it guarantees the absence of illegal accesses. For dynamically monitored systems, it guarantees that access control exceptions will never be raised.

The notion of permission is central to our model. Permissions have an internal structure (formalised in Section 3) that describes the actions that it enables and the set of objects to which it applies. The "one-shot" permissions alluded to above have motivated a generalisation in which permissions now have *multiplicities*, stating how many times the given permission can be used. Multiplicities are important for controlling resource access that has a cost, such as sending of SMSs and establishing network connections on mobile telephones. For example, in our model it is possible for a user to grant an applet the permission to send 3 SMSs during a transaction. Furthermore, with the accompanying analyses we propose, it is possible to verify formally that such a number of permissions are sufficient for completing the transaction.

The security model we propose has two basic constructs for manipulating permissions:

– `grant` models the interactive querying of the user, asking whether he grants a particular permission with a certain multiplicity to the applet, and
– `consume` models the access to a method which requires (and hence consumes) permissions.

In this model, we choose *not* to let permissions accumulate *i.e.*, the number of permissions available of a given type of permissions are those granted by the most recently executed `grant`. To avoid the potential confusion that may arise when several permissions can be used by a `consume` we introduce a typing of permissions that renders this situation impossible.

An important feature of this model is that an application can request one or more permissions in advance instead of having to ask permission just before consuming it, as with the "one-shot" permissions. The choice of where to insert requests for user-granted permissions now becomes important for the usability of an applet and has a clear impact on its security. We provide a static analysis that will verify automatically that a given choice of placement will ensure that an applet always has the permissions necessary for its further execution.

The analysis is developed by integrating the `grant` and `consume` constructs into a program model based on control-flow graphs. The model and its operational semantics is presented in Section 4. In this section, we also formally define what it means for an execution trace (and hence for a program) to respect the basic security property. Section 5 defines a constraint-based static analysis for safely approximating the flow of permissions in a program with the aim of computing what permissions are available at each program point. Section 7 describes how to solve the constraints produced by the analysis. Section 8 describes related formal models and verification techniques for language-based access control and Section 9 concludes.

2 The Java MIDP Security Model

The Java MIDP programming model for mobile telephones [13] proposes a thoroughly developed security architecture which is the starting point of our work. In the MIDP security model, applications (called *midlets* in the MIDP jargon) are downloaded and executed by a Java virtual machine. Midlets are made of a single archive (a jar file) containing complete programs. At load time, the midlet is assigned a protection domain which determines how the midlet can access resources. It can be seen as a labelling function which classifies a resource access as either `allowed` or `user`.

- `allowed` means that the midlet is granted unrestricted access to the resource;
- `user` means that, prior to an access, an interaction with the user is initiated in order to ask for permission to perform the access and to determine how often this permission can be exercised. Within this protection domain, the MIDP model operates with three possibilities:
  - `blanket`: the permission is granted for as long as the midlet remains installed;
  - `session`: the permission is granted for as long as the midlet is running;
  - `oneshot`: the permission is granted for a single use.

The `oneshot` permissions correspond to dynamic security checks in which each access is protected by a user interaction. This clearly provides a secure access to resources but the potentially numerous user interactions are at the detriment of the usability and makes social engineering attacks easier. At the other end of the spectrum, the `allowed` mode which gets granted through signing provides a maximum of usability but leaves the user with absolutely no assurance on how resources are used, as a signature is only a certificate of integrity and origin.

In the following we will propose a security model which extends the MIDP model by introducing permissions with multiplicities and by adding flexibility to the way in which permissions are granted by the user and used by applications. In this model, we can express:

- the `allowed` mode and `blanket` permissions as initial permissions with multiplicity ∞;

– the `session` permissions by prompting the user at application start-up whether he grants the permission for the session and by assigning an infinite number of the given permission;
– the `oneshot` permissions by prompting the user for a permission with a `grant` just before consuming it with a `consume`.

The added flexibility is obtained by allowing the programmer to insert user interactions for obtaining permissions at any point in the program (rather than only at the beginning and just before an access) and to ask for a batch of permissions in one interaction. The added flexibility can be used to improve the usability of access control in a midlet but will require formal methods to ensure that the midlet will not abuse permissions (security concern) and will be granted by the programmer sufficient permissions for a correct execution (usability concern). The analysis presented in section 5 is addressing these two concerns.

3 The Structure of Permissions

In classical access control models, permissions held by a subject (user, program, ...) authorise certain *actions* to be performed on certain *resources*. Such permissions can be represented as a relation between actions and resources. To obtain a better fit with access control architectures such as that of Java MIDP we enrich this permission model with multiplicities and resource types. Concrete MIDP permissions are strings whose prefixes encode package names and whose suffixes encode a specific permission. For instance, one finds permissions `javax.microedition.io.Connector.http` and `javax.microedition.io.Connector.sms.send`which enable applets to make connections using the http protocol or to send a SMS, respectively. Thus, permissions are structured entities that for a given resource type define which actions can be applied to which resources of that type and how many times.

To model this formally, we assume given a set $\mathit{ResType}$ of resource types. For each resource type rt there is a set of resources Res_{rt} of that type and a set of actions Act_{rt} applicable to resources of that type. We incorporate the notion of multiplicities by attaching to a set of actions a and a set of resources r a multiplicity m indicating how many times actions a can be performed on resources from r. Multiplicities are taken from the ordered set:

$$\mathit{Mul} \triangleq (\mathbb{N} \cup \{\bot_{\mathit{Mul}}, \infty\}, \leq).$$

The 0 multiplicity represents absence of a given permission and the ∞ multiplicity means that the permission is permanently granted. The $\bot_{\mathit{Mul}}$ multiplicity represents an error arising from trying to decrement the 0 multiplicity. We define the operation of decrementing a multiplicity as follows:

$$m - 1 = \begin{cases} \infty & \text{if } m = \infty \\ m - 1 & \text{if } m \in \mathbb{N}, m \neq 0 \\ \bot_{\mathit{Mul}} & \text{if } m = 0 \text{ or } m = \bot_{\mathit{Mul}} \end{cases}$$

Several implementations of permissions include an *implication ordering* on permissions. One permission implies another if the former allows to apply a particular action to more resources than the latter. However, the underlying object-oriented nature of permissions imposes that only permissions of the same resource type can be compared. We capture this in our model by organising permissions as a dependent product of permission sets for a given resource type.

Definition 1 (Permissions). *Given a set ResType of resource types and Res Type-indexed families of resources Res_{rt} and actions Act_{rt}, the set of atomic permissions $Perm_{rt}$ is defined as:*

$$Perm_{rt} \stackrel{\triangle}{=} (\mathcal{P}(Res_{rt}) \times \mathcal{P}(Act_{rt})) \cup \{\bot\}$$

relating a type of resources with the actions that can be performed on it. The element $\bot$ represents an invalid permission. By extension, we define the set of permissions Perm as the dependent product:

$$Perm \stackrel{\triangle}{=} \prod_{rt \in ResType} Perm_{rt} \times Mul$$

relating for all resource types an atomic permission and a multiplicity stating how many times it can be used.

For $\rho \in Perm$ and $rt \in ResType$, we use the notations $\rho(rt)$ to denote the pair of atomic permissions and multiplicities associated with rt in ρ. Similarly, $\mapsto$ is used to update the permission associated to a ressource type, i.e., $(\rho[rt \mapsto (p, m)])(rt) = (p, m)$.

Example 1. *Given a ressource type $SMS \in ResType$, the permission $\rho \in Perm$ satisfying $\rho(SMS) = ((+1800*, \{send\}), 2)$ grants two accesses to a send action of the resource +1800∗ (phone number starting with +1800) with the type SMS.*

Definition 2. *The ordering $\sqsubseteq_p \subseteq Perm \times Perm$ on permissions is given by*

$$\rho_1 \sqsubseteq_p \rho_2 \stackrel{\triangle}{=} \forall rt \in ResType \quad \rho_1(rt) \sqsubseteq \rho_2(rt)$$

where $\sqsubseteq$ is the product of the subset ordering $\sqsubseteq_{rt}$ on $Perm_{rt}$ and the $\leq$ ordering on multiplicities.

Intuitively, being higher up in the ordering means having more permissions to access a larger set of resources. The ordering induces a greatest lower bound operator $\sqcap : Perm \times Perm \to Perm$ on permissions. For example, for $\rho \in Perm$

$$\rho[File \mapsto ((/tmp/*, \{read, write\}), 1)] \sqcap \rho[File \mapsto ((*/dupont/*, \{read\}), \infty)] = \rho[File \mapsto ((/tmp/*/dupont/*, \{read\}), 1)]$$

Operations on Permissions

There are two operations on permissions that will be of essential use:

- consumption (removal) of a specific permission from a collection of permissions;
- update of a collection of permissions with a newly granted permission.

Definition 3. *Let $\rho \in Perm$, $rt \in ResType$, $p, p' \in Perm_{rt}$, $m \in Mul$ and assume that $\rho(rt) = (p, m)$. The operation consume* : $Perm_{rt} \to Perm \to Perm$ *is defined by*

$$consume(p')(\rho) = \begin{cases} \rho[rt \mapsto (p, m-1)] & \text{if } p' \sqsubseteq_{rt} p \\ \rho[rt \mapsto (\bot, m-1)] & \text{otherwise} \end{cases}$$

There are two possible error situations when trying to consume a permission. Attempting to consume a resource for which there is no permission ($p' \not\sqsubseteq_{rt} p$) is an error. Similarly, consuming a resource for which the multiplicity is zero will result in setting the multiplicity to $\bot_{Mul}$.

Definition 4. *A permission $\rho \in Perm$ is an error, written $Error(\rho)$, if:*

$$\exists rt \in ResType, \exists (p, m) \in Perm_{rt} \times Mul, \rho(rt) = (p, m) \wedge (p = \bot \vee m = \bot_{Mul}).$$

Granting a number of accesses to a resource of a particular resource type is modeled by updating the component corresponding to that resource type.

Definition 5. *Let $\rho \in Perm$, $rt \in ResType$, the operation grant* : $Perm_{rt} \times Mul \to Perm \to Perm$ *for granting a number of permissions to access a resource of a given type is defined by*

$$grant(p, m)(\rho) = \rho[rt \mapsto (p, m)]$$

Notice that granting such a permission erases all previously held permissions for that resource type, *i.e.*, permissions do not accumulate. This is a design choice: the model forbids that permissions be granted for performing one task and then used later on to accomplish another. The *grant* operation could also add the granted permission to the existing ones rather than replace the corresponding one. Besides cumulating the number of permissions for permissions sharing the same type and resource, this would allow different resources for the same resource type. However, the `consume` operation becomes much more complex, as a choice between the overlapping permissions may occur. Analysis would require handling multisets of permissions or backtracking.

Another consequence of the fact that permissions do not accumulate is that our model can impose scopes to permissions. This common programming pattern is naturally captured by inserting a `grant` instruction with null multiplicity at the end of the permission scope.

4 Program Model

We model a program by a control-flow graph (CFG) that captures the manipulations of permissions (grant and consume), the handling of method calls and returns, as well as exceptions. These operations are respectively represented by the instructions $\mathtt{grant}(p, m)$, $\mathtt{consume}(p)$, $\mathtt{call}$, $\mathtt{return}$, $\mathtt{throw}(ex)$, with $ex \in EX, rt \in ResType, p \in Perm_{rt}$ and $m \in Mul$. Exceptions aside, this model has been used in previous work on modelling access control for Java—see [1, 3, 8].

Definition 6. *A control-flow graph is a 7-tuple*

$$G = (NO, EX, KD, TG, CG, EG, n_0)$$

where:

- *NO is the set of nodes of the graph;*
- *EX is the set of exceptions;*
- *$KD : NO \to \{\mathtt{grant}(p, m), \mathtt{consume}(p), \mathtt{call}, \mathtt{return}, \mathtt{throw}(ex)\}$, associates a kind to each node, indicating which instruction the node represents;*
- *$TG \subseteq NO \times NO$ is the set of intra-procedural edges;*
- *$CG \subseteq NO \times NO$ is the set of inter-procedural edges, which can capture dynamic method calls;*
- *$EG \subseteq EX \times NO \times NO$ is the set of intra-procedural exception edges that will be followed if an exception is raised at that node;*
- *n_0 is the entry point of the graph.*

In the following, given $n, n' \in NO$ and $ex \in EX$, we will use the notations $n \overset{TG}{\to} n'$ for $(n, n') \in TG$, $n \overset{CG}{\to} n'$ for $(n, n') \in CG$ and $n \overset{ex}{\to} n'$ for $(ex, n, n') \in EG$.

Example 2. *Figure 1 contains the control-flow graph of* `grant` *and* `consume` *operations during a flight-booking transaction (for simplicity, actions related to permissions, such as $\{connect\}$ or $\{read\}$, are omitted). In this transaction, the user first transmits his request to a travel agency, site. He can then modify his request or get additional information. Finally he can either book or pay the desired flight. Corresponding permissions are summarised in the initial permission p_{init}, but they could also be granted using the* `grant` *operation. In the example, the developer has chosen to delay asking for the permission of accessing credit card information until it is certain that this permission is indeed needed. Another design choice would be to grant this permission from the outset. This would minimise user interaction because it allows to remove the querying* `grant` *operation. However, the initial permission p_{init} would then contain $file \mapsto (/wallet/*, 2)$ instead of $file \mapsto (/wallet/id, 1)$ which violates the Principle of Least Privilege.*

Operational Semantics

We define the small-step operational semantics of CFGs in Figure 2. The semantics is stack-based and follows the behaviour of a standard programming

$p_{init}[http \mapsto (*, \infty); https \mapsto (site, 1); file \mapsto (/wallet/id, 1)]$

$$\begin{tikzcd}
& \texttt{consume}(http(site)) \arrow[loop left, "1."] \arrow[dr, "2."] \arrow[dd, "3."'] & \\
& & \texttt{consume}(http(*)) \arrow[ul, bend right, "1."'] \arrow[dl, "3."] \\
& \texttt{consume}(file(/wallet/id)) \arrow[dl, "4."'] \arrow[dr, "5."] & \\
\texttt{consume}(http(site)) & & \texttt{grant}(file(/wallet/visa\#), 1) \arrow[d] \\
& & \texttt{consume}(file(/wallet/visa\#)) \arrow[d] \\
& & \texttt{consume}(https(site))
\end{tikzcd}$$

1. Modify request
2. Get info
3. Choose flight
4. Book only
5. Book and pay

Fig. 1. Example of grant/consume permissions patterns

language with exceptions, e.g., as Java or C♯. Instantiating this model to such languages consists of identifying in the code the desired `grant` and `consume` operations, building the control-flow graph and describing the action of the other instructions on the stack.

The operational semantics operates on a state consisting of a standard control-flow stack of nodes, enriched with the permissions held at that point in the execution. Thus, the small-step semantics is given by a relation $\twoheadrightarrow$ between elements of $(NO^* \times (EX \cup \{\epsilon\}) \times Perm)$, where NO^* is a sequence of nodes. For example, for the instruction `call` of Figure 2, if the current node n leads through an inter-procedural step to a node m, then the node m is added to the top of the stack $n{:}s$, with $s \in NO^*$.

Instructions may change the value of the permission along with the current state. *E.g.*, for the instruction `grant` of Figure 2, the current permission ρ of the state will be updated with the new granted permissions. The current node of the stack n will also be updated, at least to change the program counter, depending on the desired implementation of `grant`. Note that the instrumentation is *non-intrusive*, *i.e.* a transition will not be blocked due to the absence of a permission. Thus, for s in NO^*, e in $(EX \cup \{\epsilon\})$, ρ' in $Perm$, if there exists s' in NO^*, e' in $(EX \cup \{\epsilon\})$, ρ' in $Perm$ such that $s, e, \rho \twoheadrightarrow s', e', \rho'$, then for all ρ and ρ', the same transition holds.

This operational semantics will be the basis for the notion of program execution traces, on which global results on the execution of a program will be expressed.

Definition 7 (Trace of a CFG). *A partial trace* $tr \in (NO, (EX \cup \{\epsilon\}))^*$ *of a CFG is a sequence of nodes* $(n_0, \epsilon) :: (n_1, e_1) :: \ldots :: (n_k, e_k)$ *such that for*

$$\frac{KD(n) = \texttt{grant}(p,m) \quad n \overset{TG}{\rightarrow} n'}{n{:}s, \epsilon, \rho \twoheadrightarrow n'{:}s, \epsilon, grant(p,m)(\rho)} \quad \frac{KD(n) = \texttt{consume}(p) \quad n \overset{TG}{\rightarrow} n'}{n{:}s, \epsilon, \rho \twoheadrightarrow n'{:}s, \epsilon, consume(p)(\rho)}$$

$$\frac{KD(n) = \texttt{call} \quad n \overset{CG}{\rightarrow} m}{n{:}s, \epsilon, \rho \twoheadrightarrow m{:}n{:}s, \epsilon, \rho} \qquad \frac{KD(r) = \texttt{return} \quad n \overset{TG}{\rightarrow} n'}{r{:}n{:}s, \epsilon, \rho \twoheadrightarrow n'{:}s, \epsilon, \rho}$$

$$\frac{KD(n) = \texttt{throw}(ex) \quad n \overset{ex}{\rightarrow} h}{n{:}s, \epsilon, \rho \twoheadrightarrow h{:}s, \epsilon, \rho} \quad \frac{KD(n) = \texttt{throw}(ex) \quad \forall h, n \overset{ex}{\not\rightarrow} h}{n{:}s, \epsilon, \rho \twoheadrightarrow n{:}s, ex, \rho}$$

$$\frac{\forall h, n \overset{ex}{\not\rightarrow} h}{t{:}n{:}s, ex, \rho \twoheadrightarrow n{:}s, ex, \rho} \qquad \frac{n \overset{ex}{\rightarrow} h}{t{:}n{:}s, ex, \rho \twoheadrightarrow h{:}s, \epsilon, \rho}$$

Fig. 2. Small-step operational semantics

all $0 \leq i < k$ *there exists* $\rho, \rho' \in Perm$, $s, s' \in NO^*$ *such that* $n_i{:}s, e_i, \rho \twoheadrightarrow n_{i+1}{:}s', e_{i+1}, \rho'$.

For a program P represented by its control-flow graph G, we will denote by $[\![P]\!]$ *the set of all partial traces of G.*

To state and verify the safety of a program that acquires and consumes permissions, we first define what it means for an execution trace to be safe. We define the permission set available at the end of a trace by induction over its length.

$$\begin{array}{ll}
PermsOf(nil) & \triangleq p_{init} \\
PermsOf(tr :: (\texttt{consume}(p), e)) & \triangleq consume(p, PermsOf(tr)) \\
PermsOf(tr :: (\texttt{grant}(p,m), e)) & \triangleq grant((p,m), PermsOf(tr)) \\
PermsOf(tr :: (n, e)) & \triangleq PermsOf(tr) \qquad \text{otherwise}
\end{array}$$

p_{init} is the initial permission of the program, for the state n_0. By default, if no permission is granted at the beginning of the execution, it will contain $((\emptyset, \emptyset), 0)$ for each resource type. The `allowed` mode and `blanket` permissions for a resource r of a given resource type can be modeled by associating the permission $((\{r\}, Act), \infty)$ with that resource type.

A trace is *safe* if none of its prefixes end in an error situation due to the access of resources for which the necessary permissions have not been obtained.

Definition 8 (Safe trace). *A partial trace* $tr \in (NO, (EX \cup \{\epsilon\}))^*$ *is* safe*, written* $Safe(tr)$*, if for all prefixes* $tr' \in prefix(tr)$*,* $\neg Error(PermsOf(tr'))$*.*

5 Static Analysis of Permission Usage

We now define a constraint-based static flow analysis for computing a safe approximation, denoted P_n, of the permissions that are guaranteed to be available at each program point n in a CFG when execution reaches that point. Thus, safe

$$\frac{}{P_{n_0} \sqsubseteq_p p_{init}} \qquad \frac{KD(n) = \mathtt{grant}(p,m) \quad n \overset{TG}{\to} n'}{P_{n'} \sqsubseteq_p grant(p,m)(P_n)}$$

$$\frac{KD(n) = \mathtt{consume}(p) \quad n \overset{TG}{\to} n'}{P_{n'} \sqsubseteq_p consume(p)(P_n)} \qquad \frac{KD(n) = \mathtt{call} \quad n \overset{CG}{\to} m \quad n \overset{TG}{\to} n'}{P_{n'} \sqsubseteq_p R_m(P_n)}$$

$$\frac{KD(n) = \mathtt{call} \quad n \overset{CG}{\to} m}{P_m \sqsubseteq_p P_n} \qquad \frac{KD(n) = \mathtt{call} \quad n \overset{CG}{\to} m \quad n \overset{ex}{\to} h}{P_h \sqsubseteq_p R_m^{ex}(P_n)}$$

$$\frac{KD(n) = \mathtt{call} \quad n \overset{CG}{\to} m \quad \forall h, n \overset{ex}{\not\to} h}{P_n \sqsubseteq_p R_m^{ex}(P_n)} \qquad \frac{KD(n) = \mathtt{throw}(ex) \quad n \overset{ex}{\to} m}{P_m \sqsubseteq_p P_n}$$

Fig. 3. Constraints on minimal permissions

means that P_n underestimates the set of permissions that will be held at n during the execution. The approximation will be defined as a solution to a system of constraints over P_n, derived from the CFG following the rules in Figure 3. The rules for P_n are straightforward data flow rules: *e.g.*, for `grant` and `consume` we use the corresponding semantic operations *grant* and *consume* applied to the start state P_n to get an upper bound on the permissions that can be held at end state $P_{n'}$. Notice that the set $P_{n'}$ can be further constrained if there is another flow into n'. The effect of a method call on the set of permissions will be modeled by a transfer function R that will be defined below. Finally, throwing an exception at node n that will be caught at node m means that the set of permissions at n will be transferred to m and hence form an upper bound on the set of available permissions at this point.

Our CFG program model includes procedure calls which means that the analysis must be inter-procedural. We deal with procedures by computing *summary functions* for each procedure. These functions summarise how a given procedure consumes resources from the entry of the procedure to the exit, which can happen either normally by reaching a `return` node, or by raising an exception which is not handled in the procedure. More precisely, for a given CFG we compute the quantity $R : (EX \cup \{\epsilon\}) \to NO \to (Perm \to Perm)$ with the following meaning:

- the partial application of R to ϵ is the effect on a given initial permission of the execution from a node until return;
- the partial application of R to $ex \in EX$ is the effect on a given initial permission of the execution from a node until reaching a node which throws an exception ex that is not caught in the same method.

Given nodes $n, n' \in NO$, we will use the notation R_n and R_n^{ex} for the partial applications of $R\ \epsilon\ n$ and $R\ ex\ n$. The rules are written using diagrammatic function composition ; such that $F; F'(\rho) = F'(F(\rho))$. We define an order $\sqsubseteq$ on functions $F, F' : Perm \to Perm$ by extensionality such that $F \sqsubseteq F'$ if $\forall \rho \in Perm, F(\rho) \sqsubseteq_p F'(\rho)$.

$$\frac{KD(n) = \mathtt{grant}(p,m) \quad n \overset{TG}{\rightarrow} n'}{R^e_n \sqsubseteq \mathit{grant}(p,m); R^e_{n'}} \qquad \frac{KD(n) = \mathtt{consume}(p) \quad n \overset{TG}{\rightarrow} n'}{R^e_n \sqsubseteq \mathit{consume}(p); R^e_{n'}}$$

$$\frac{KD(n) = \mathtt{return}}{R_n \sqsubseteq \lambda\rho.\rho} \qquad \frac{KD(n) = \mathtt{call} \quad n \overset{CG}{\rightarrow} m \quad n \overset{TG}{\rightarrow} n'}{R^e_n \sqsubseteq R_m; R^e_{n'}}$$

$$\frac{KD(n) = \mathtt{call} \quad n \overset{CG}{\rightarrow} m \quad \forall n', n \overset{ex}{\not\rightarrow} n'}{R^{ex}_n \sqsubseteq R^{ex}_m} \quad \frac{KD(n) = \mathtt{call} \quad n \overset{CG}{\rightarrow} m \quad n \overset{ex}{\rightarrow} h}{R_n \sqsubseteq R^{ex}_m; R_h}$$

$$\frac{KD(n) = \mathtt{throw}(ex) \quad n \overset{ex}{\rightarrow} h}{R^e_n \sqsubseteq R^e_h} \qquad \frac{KD(n) = \mathtt{throw}(ex) \quad \forall n', n \overset{ex}{\not\rightarrow} n'}{R^{ex}_n \sqsubseteq \lambda\rho.\rho}$$

Fig. 4. Summary functions of the effect of the execution on initial permission

As for the entities P_n, the function R is defined as solutions to a system of constraints. The rules for generating these constraints are given in Figure 4 (with $e \in EX \cup \{\epsilon\}$). The rules all have the same structure: compose the effect of the current node n on the permission set with the function describing the effect of the computation starting at n's successors in the control flow. This provides an upper bound on the effect on permissions when starting from n. As with the constraints for P, we use the functions *grant* and *consume* to model the effect of `grant` and `consume` nodes, respectively. A method call at node n is modeled by the R function itself applied to the start node of the called method m. The combined effect is the composition $R_m; R^e_{n'}$ of the effect of the method call followed by the effect of the computation starting at the successor node n' of call node n.

6 Correctness

The correctness of our analysis is stated on execution traces. For a given program, if a solution of the constraints computed during the analysis does not contain errors in permissions, then the program will behave safely. Formally,

Theorem 1 (Basic Security Property). *Given a program P:*

$$(\forall n \in NO, \neg Error(P_n)) \Rightarrow \forall tr \in [\![P]\!], Safe(tr)$$

The proof of this theorem relies on a big-step operational semantics which is shown equivalent to the small-step semantics of Figure 2. This big-step semantics is easier to reason with (in particular for method invocation) and yields an accessibility relation *Acc* that also captures non-terminating methods. The first part of the proof of Theorem 1 amounts to showing that if the analysis declares that if no abstract state indicates an access without the proper permission then this is indeed the case for all the accessible states in program.

Lemma 1.

$$(\forall n \in NO, \neg Error(P_n)) \Rightarrow \forall (n, \rho) \in Acc, \neg Error(\rho)$$

To do this, we first show (by induction over the definition of the big-step semantics) that summary functions R correctly model the effect of method calls on permissions. Then, we show a similar result for the permissions computed for each program point by the analysis:

Lemma 2.

$$\forall n \in NO, \forall \rho \in Perm,\ (n, \rho) \in Acc \Rightarrow P_n \sqsubseteq_p \rho$$

Lemma 1 is a direct consequence of Lemma 2. Using proof by contradiction, we suppose $(n, \rho) \in Acc$ with $Error(\rho)$, then we get $P_n \sqsubseteq_p \rho$, which contradicts $\neg Error(P_n)$ given $Error(\rho)$.

The second part links the trace semantics with the big-step instrumented semantics by proving that if no accessible state in the instrumented semantics has a tag indicating an access control error then the program is safe with respect to the definition of safety of execution traces. This part amounts to showing that the instrumented semantics is a monitor for the $Safe$ predicate.

A more detailed proof is given in the online version of this paper at `http://hal.inria.fr/inria-00083453`.

7 Constraint Solving

Computing a solution to the constraints generated by the analysis in Section 5 is complicated by the fact that solutions to the R-constraints (see Figure 4) are functions from *Perm* to *Perm* that have infinite domains and hence cannot be represented by a naive tabulation [14]. To solve this problem, we identify a class of functions that are sufficient to encode solutions to the constraints while restricted enough to allow effective computations. Given a solution to the R-constraints, the P-constraints (see Figure 3) are solved by standard fixpoint iteration.

The rest of this section is devoted to the resolution of the R-constraints. The resolution technique consists in applying solution-preserving transformations to the constraints until they can be solved either symbolically or iteratively.

7.1 On Simplifying R-Constraints

In our model, resources are partitioned depending on their resource type. At the semantic level, *grant* and *consume* operations ensure that permissions of different types do not interfere *i.e.*, that it is impossible to use a resource of a given type with a permission of a different type. We exploit this property to derive from the original system of constraints a family of independent *ResType*-indexed constraint systems. A system modelling a given resource type, say *rt*, is a copy of the original system except that *grant* and *consume* are indexed by *rt* and are specialized accordingly:

$$grant_{rt}(p'_{rt'}, m') = \begin{cases} \lambda(p,m).(p',m') & \text{si } rt = rt' \\ \lambda(p,m).(p,m) & \text{sinon} \end{cases}$$

$$consume_{rt}(p'_{rt'}) = \begin{cases} \lambda(p,m).(\text{if } p' \sqsubseteq_{rt'} p \text{ then } p \text{ else } \bot, m-1) & \text{si } rt = rt' \\ \lambda(p,m).(p,m) & \text{sinon} \end{cases}$$

Further inspection of these operators shows that multiplicities and atomic permissions also behave in an independent manner. As a result, each *ResType* indexed system can be split into a pair of systems: one modelling the evolution of atomic permissions; the other modelling the evolution of multiplicities. Hence, solving the R-constraints amounts to computing for each exception e, node n and resource type rt a pair of mappings:

– an atomic permission transformer ($Perm_{rt} \to Perm_{rt}$) and
– a multiplicity transformer ($Mul \to Mul$).

In the next sections, we define syntactic representations of these multiplicity transformers that are amenable to symbolic computations.

7.2 Constraints on Multiplicity Transformers

Before presenting our encoding of multiplicity transformers, we identify the structure of the constraints we have to solve. Multiplicity constraints are terms of the form $x \dot{\leq} e$ where $x : Mul \to Mul$ is a variable over multiplicity transformers, $\dot{\leq}$ is the point-wise ordering of multiplicity transformers induced by $\leq$ and e is an expression built over the terms

$$e ::= v | grant_{Mul}(m) | consume_{Mul}(m) | id | e; e$$

where

– v is a variable;
– $grant_{Mul}(m)$ is the constant function $\lambda x.m$;
– $consume_{Mul}(m)$ is the decrementing function $\lambda x.x - m$;
– id is the identity function $\lambda x.x$;
– and $f; g$ is function composition ($f; g = g \circ f$).

We define $MulF = \{\lambda x.min(c, x-d) | (c, d) \in Mul \times Mul\}$ as a restricted class of multiplicity transformers that is sufficiently expressive to represent the solution to the constraints. Elements of *MulF* encode constant functions, decrementing functions and are closed under function composition as shown by the following equalities:

$$grant_{Mul}(m) = \lambda x.min(m, x - \bot_{Mul})$$
$$consume_{Mul}(m) = \lambda x.min(\infty, x - m)$$
$$\lambda x.min(c, x - d'); \lambda x.min(c', x - d') = \lambda x.min(min(c - d', c'), x - (d' + d))$$

We represent a function $\lambda x.min(c, x - d) \in MulF$ by the pair (c, d) of multiplicities. Constraint solving over *MulF* can therefore be recast into constraint solving over the domain $MulF^\sharp = Mul \times Mul$ equipped with the interpretation $[\![(c, d)]\!] \triangleq \lambda x.min(c, x-d)$ and the ordering $\sqsubseteq^\sharp$ defined as $(c, d) \sqsubseteq^\sharp (c', d') \triangleq c \leq c' \wedge d' \leq d$.

7.3 Solving Multiplicity Constraints

The domain $MulF^\sharp$ does not satisfy the *descending chain condition.* This means that iterative solving of the constraints might not terminate. Instead, we use an elimination-based algorithm. First, we split our constraint system over $MulF^\sharp = Mul \times Mul$ into two constraint systems over *Mul*. Example 3 shows this transformation for a representative set of constraints.

Example 3. $C = \{Y \sqsubseteq^\sharp (c, d), Y' \sqsubseteq^\sharp X, X \sqsubseteq^\sharp Y ;^\sharp Y'\}$ *is transformed into* $C' = C_1 \cup C_2$ *with* $C_1 = \{Y_1 \leq c, Y'_1 \leq X_1, X_1 \leq min(Y_1 - Y'_2, Y'_1)\}$ *and* $C_2 = \{Y_2 \geq d, Y'_2 \geq X_2, X_2 \geq Y'_2 + Y_2\}$.

Notice that C_1 depends on C_2 but C_2 is independent from C_1. This result holds generally and, as a consequence, these sets of constraints can be solved in sequence: C_2 first, then C_1.

To be solved, C_2 is converted into an equivalent system of fixpoint equations defined over the complete lattice $(Mul, \leq, max, \perp_{Mul})$. The equations have the general form $x = e$ where $e ::= var \mid max(e, e) \mid e + e$. The elimination-based algorithm unfolds equations until a direct recursion is found. After a normalisation step, recursions are eliminated using a generalisation of Proposition 1 for an arbitrary number of occurences of the x variable.

Proposition 1. $x = max(x + e_1, e_2)$ *is equivalent to* $x = max(e_2 + \infty \times e_1, e_2)$.

Given a solution for C_2, the solution of C_1 can be computed by standard fixpoint iteration as the domain $(Mul, \leq, min, \infty)$ does not have infinite descending chains. This provides multiplicity transformer solutions of the R-constraints.

8 Related Work

To the best of our knowledge, there is no formal model of the Java MIDP access control mechanism. A number of articles deal with access control in Java and C♯ but they have focused on the stack inspection mechanism and the notion of granting permissions to code through privileged method calls. Earlier work by some of the present authors [3, 8] proposed a semantic model for stack inspection but was otherwise mostly concerned with proving behavioural properties of programs using these mechanisms. Closer in aim with the present work is that of Pottier *et al.* [11] on verifying that stack inspecting programs do not raise security exceptions because of missing permissions. Bartoletti *et al.* [1] also aim at proving that stack inspecting applets will not cause security exceptions and propose the first proper modelling of exception handling. Both these works prove properties that allow to execute the program without dynamic permission checks. In this respect, they establish the same kind of property as we do in this paper. However, the works cited above do not deal with multiplicities of permissions and do not deal with the aspect of permissions granted on the fly through user interaction. The analysis of multiplicities leads to systems of numerical constraints which do not appear in the stack inspecting analyses.

Language-based access control has been studied for various idealised program models. Igarashi and Kobayashi [7] propose a static analysis for verifying that resources are accessed according to access control policies specified *e.g.* by finite-state automata, but do not study specific language primitives for implementing such an access control. Closer to the work presented in this article is that of Bartoletti *et al.* [2] who propose with $\lambda^{[]}$ a less general resource access control framework than Igarashi and Kobayashi, and without explicit notions of resources, but are able to ensure through a static analysis that no security violations will occur at run-time. They rely for that purpose on a type and effect system on $\lambda^{[]}$ from which they extract history expressions further model-checked. In the context of mobile agent, Hennessy and Riely [6] have developed a type system for the π-calculus with the aim of ensuring that a resource is accessed only if the program has been granted the appropriate permission (capability) previously. In this model, resources are represented by locations in a π-calculus term and are accessed via channels. Permissions are now capabilities of executing operations (*e.g.* read, transmit) on a channel. Types are used to restrict the access of a term to a resource and there is a notion of sub-typing akin to our order relation on permissions. The notion of multiplicities is not dealt with but could probably be accommodated by switching to types that are multi-sets of capabilities.

Our permission model adds a quantitative aspect to permissions which means that our analysis is closely related to the work by Chander *et al.* [4] on dynamic checks for verifying resource consumption. Their safety property is similar to ours and ensure that a program always acquires resources before consuming them. However, their model of resources is simpler as resources are just identified by name. Because their approach requires user-provided invariants, their analysis of numeric quantities (multiplicities) is very precise. In contrast to this, our analysis is fully automatic.

9 Conclusions

We have proposed an access control model for programs which dynamically acquire permissions to access resources. The model extends the current access control model of the Java MIDP profile for mobile telephones by introducing multiplicities of permissions together with explicit instructions for granting and consuming permissions. These instructions allow to improve the usability of an application by fine-tuning the number and placement of user interactions that ask for permissions. In addition, programs written in our access control model can be formally and statically verified to satisfy the fundamental property that a program does not attempt to access a resource for which it does not have the appropriate permission. The formalisation is based on a model of permissions which extends the standard object $\times$ action model with multiplicities. We have given a formal semantics for the access control model, defined a constraint-based analysis for computing the permissions available at each point of a program, and shown how the resulting constraint systems can be solved. To the best of our knowledge, it is the first time that a formal treatment of the Java MIDP model has been proposed.

The present model and analysis has been developed in terms of control-flow graphs and has ignored the treatment of data such as integers *etc.* By combining our analysis with standard data flow analysis we can obtain a better approximation of integer variables and hence, e.g., the number of times a permission-consuming loop is executed. In the present model, we either have to require that there is a `grant` executed for each `consume` inside the loop or that the relevant permission has been granted with multiplicity ∞ before entering the loop. Allowing a `grant` to take a variable as multiplicity parameter combined with a relational analysis (the *octagon* analysis by Miné [9]) is a straightforward extension that would allow to program and verify a larger class of programs.

This work is intended for serving as the basis for a Proof Carrying Code (PCC) [10] architecture aiming at ensuring that a program will not use more resources than what have been declared. In the context of mobile devices where such resources could have an economic (via premium-rated SMS for instance) or privacy (via address-book access) impact, this would provide improved confidence in programs without resorting to third-party signature. The PCC certificate would consist of the precomputed P_n and R_n^e. The host device would then check that the transmitted certificate is indeed a solution. Note that no information is needed for intra-procedural instructions other than `grant` and `consume`—this drastically reduces the size of the certificate.

References

[1] Massimo Bartoletti, Pierpaolo Degano, and Gian Luigi Ferrari. Static analysis for stack inspection. *Electronic Notes in Computer Science*, 54, 2001.

[2] Massimo Bartoletti, Pierpaolo Degano, and Gian Luigi Ferrari. History-based access control with local policies. In *Proceedings of FOSSACS 2005*, volume 3441 of *Lecture Notes in Computer Science*, pages 316–332. Springer-Verlag, 2005.

[3] Frédéric Besson, Thomas Jensen, Daniel Le Métayer, and Tommy Thorn. Model ckecking security properties of control flow graphs. *Journal of Computer Security*, 9:217–250, 2001.

[4] Ajay Chander, David Espinosa, Nayeem Islam, Peter Lee, and George C. Necula. Enforcing resource bounds via static verification of dynamic checks. In *Proceedings of the 14th European Symposium on Programming, ESOP 2005*, volume 3444 of *Lecture Notes in Computer Science*, pages 311–325. Springer-Verlag, 2005.

[5] Cedric Fournet and Andy Gordon. Stack inspection: theory and variants. In *Proceedings of the 29th ACM Symp. on Principles of Programming Languages (POPL'02).* ACM Press, 2002.

[6] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173(1):82–120, 2002.

[7] Atsushi Igarashi and Naoki Kobayashi. Resource usage analysis. In *Proceedings of the 29th ACM Symp. on Principles of Programming Languages (POPL'02)*, pages 331–342, 2002.

[8] Thomas Jensen, Daniel Le Métayer, and Tommy Thorn. Verification of control flow based security properties. In *Proceedings of the 20th IEEE Symp. on Security and Privacy*, pages 89–103. New York: IEEE Computer Society, 1999.

[9] Antoine Miné. The octogon abstract domain. In *Proceedings of the 8th Working Conference On Reverse Engineering (WCRE 01)*, pages 310–320. IEEE, 2001.

[10] George C. Necula. Proof-carrying code. In Neil D. Jones, editor, *Proceedings of the 24th ACM Symp. on Principles of Programming Languages (POPL'97)*, pages 106–119, Paris, France, January 1997. ACM Press.

[11] François Pottier, Christian Skalka, and Scott F. Smith. A systematic approach to static access control. In *Proceedings of the 10th European Symposium on Programming, ESOP 2001*, volume 2028 of *Lecture Notes in Computer Science*, pages 30–45. Springer-Verlag, 2001.

[12] Jerry H. Saltzer and Mike D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63:1278–1308, 1975.

[13] Sun Microsystems, Inc., Palo Alto/CA, USA. *Mobile Information Device Profile (MIDP) Specification for Java 2 Micro Edition, Version 2.0*, 2002.

[14] Hisao Tamaki and Taisuke Sato. OLD resolution with tabulation. In *Proceedings on Third International Conference on Logic Programming*, volume 225 of *Lecture Notes in Computer Science*, pages 84–98. Springer-Verlag, 1986.

Discretionary Capability Confinement

Philip W.L. Fong

Department of Computer Science, University of Regina, Regina, SK, Canada
pwlfong@cs.uregina.ca

Abstract. Motivated by the need of application-level access control in dynamically extensible systems, this work proposes a static annotation system for modeling capabilities in a Java-like programming language. Unlike previous language-based capability systems, the proposed annotation system can provably enforce capability confinement. This confinement guarantee is leveraged to model a strong form of separation of duty known as hereditary mutual suspicion. The annotation system has been fully implemented in a standard Java Virtual Machine.

1 Introduction

Dynamic extensibility is a popular architectural feature of networked or distributed software systems [1]. In such systems, code units originating from potentially untrusted origins can be linked dynamically into the core system in order to augment its feature set. The protection infrastructure of a dynamically extensible system is often language based [2]. Previous work on language-based access control largely focuses on infrastructure protection via various forms of history-based access control [3,4,5,6,7,8,9]. The security posture of infrastructure protection tends to divide run-time principals into a trusted "kernel" vs untrusted "extensions", and focuses on controlling the access of kernel resources by extension code. This security posture does not adequately address the need of ***application-level security***, that is, the imposition of collaboration protocols among peer code units, and the enforcement of access control over resources that are defined and shared by these code units. This paper reports an effort to address this limitation through a language-based capability system.

The notion of ***capabilities*** [10,11] is a classical access control mechanism for supporting secure cooperation of mutually suspicious code units [12]. A capability is an unforgeable pair comprised of an object reference plus a set of access rights that can be exercised through the reference. In a capability system, possession of a capability is the necessary and sufficient condition for exercising the specified rights on the named object. This inherent symmetry makes capability systems a natural protection mechanism for enforcing application-level security.

Previous approaches to implement language-based capability systems involve the employment of either the proxy design pattern [13] or load-time binary rewriting [14] to achieve the effect of interposition. Although these "dynamic" approaches are versatile enough to support ***capability revocation***, they are not without blemish. Leaving performance issues aside, a common critique [13,15] is that an unmodified capability model fails to address the need of ***capability confinement***: once a capability is granted to a receiver, there is no way to prevent further propagation.

An alternative approach is to embed the notion of capabilities into a static type system [16]. In a ***capability type system*** [17,18], every object reference is statically assigned a capability type, which imposes on the object reference a set of operational restrictions that constrains the way the underlying object may be accessed. When a code unit delegates a resource to an untrusted peer, it may do so by passing to the peer a resource reference that has been statically typed by a capability type, thereby exposing to the peer only a limited view of the resource.

The class hierarchy of a Java-like programming language [19,20] provides non-intrusive building blocks for capability types. Specifically, one may exploit ***abstract types*** (i.e., abstract classes or interfaces in Java) as capability types. An abstract type exposes only a limited subset of the functionalities provided by the underlying object, and thus an object reference with an abstract type can be considered a capability of the underlying object. A code unit wishing to share an object with its peer may grant the latter a reference properly typed with an abstract type. The receiver of the reference may then access the underlying object through the constrained interface. This scheme, however, suffers from the same lack of capability confinement. The problem manifests itself in two ways.

1. ***Capability Theft.*** A code unit may "steal" a capability from code units belonging to a foreign protection domain, thereby amplifying its own access rights. Worst still, capabilities can be easily forged by unconstrained object instantiation and dynamic downcasting.
2. ***Capability Leakage.*** A code unit in possession of a capability may intentionally or accidentally "push" the capability to code units residing in a less privileged protection domain.

This paper proposes a lightweight, static annotation system called ***Discretionary Capability Confinement (DCC)***, which fully supports the adoption of abstract types as capability types and provably prevents capability theft and leakage. Targeting Java-like programming languages, the annotation system offers the following features:

– While the binding of a code unit to its protection domain is performed statically, the granting of permissions to a protection domain occurs dynamically through the propagation of ***capabilities***.
– Inspired by Vitek *et al* [21,22,23,24], a protection domain is identified with a ***confinement domain***. Once a capability is acquired, it roams freely within the receiving confinement domain. A capability may only escape from a confinement domain via explicit capability granting.
– Following the design of Gong [25], although a method may freely exercise the capabilities it possesses, its ability to grant capabilities is subject to discretionary control by a ***capability granting policy***.

- Under mild conditions, capability confinement guarantees such as ***no theft*** and ***no leakage*** can be proven. Programmers can achieve these guarantees by adhering to simple annotation practices.
- An application-level collaboration protocol called ***hereditary mutual suspicion*** is enforced. This protocol entails a strong form of ***separation of duty*** [26,27]: not only is the notion of mutually-exclusive roles supported, collusion between them is severely restricted because of the confinement guarantees above.

The contributions of this paper are the following:

- A widely held belief among security researchers is that language-based capability systems adopting the reference-as-capability metaphor cannot address the need of capability confinement [13,15]. Employing type-based confinement, this work has successfully demonstrated that such a capability system is in fact feasible.
- The traditional approach to support separation of duty is through the imposition of mutually exclusive roles [28,27]. This work proposes a novel mechanism, hereditary mutual suspicion, to support separation of duty in an object-oriented setting. When combined with confinement guarantees, this mechanism not only implements mutually exclusive roles, but also provably eliminate certain forms of collusion.

Organization. Sect. 2 motivates DCC by an example. Sect. 3 outlines the main type constraints. Sect. 4 states the confinement guarantees. Sect. 5 discusses extensions and variations. The paper concludes with related work and future work. Appendix A reports implementation experiences.

2 Motivation

The Hero-Sidekick Game. Suppose we are developing a role-playing game. Over time, a playable character, called a *hero* (e.g., Bat Man), may acquire an arbitrary number of *sidekicks* (e.g., Robin). A sidekick is a non-playable character whose behavior is a function of the state of the hero to which it is associated. The intention is that a sidekick augments the power of its hero. The number of sidekicks that may be attached to a hero is a function of the hero's experience. A hero may adopt or orphan a sidekick at will. New hero and sidekick types may be introduced in future releases.

A possible design is to employ the Observer pattern [29] to capture the dynamic dependencies between heros and sidekicks, as is shown in Fig. 1, where sidekicks are observers of heros. The `GameEngine` class is responsible for creating instances of `Hero` and `Sidekick`, and managing the attachment and detachment of `Sidekicks`. This set up would have worked had it not been the following requirement: *users may download new hero or sidekick types from the internet during a game play.* Because of dynamic extensibility, we must actively ensure fair game play by eliminating the possibility of cheating through the downloading of malicious characters. Two prototypical cheats are the following.

```
public interface Character {  /* Common character behavior ... */  }
public interface Observable {
   State getState();
}
public abstract class Hero implements Character, Observable {
   protected Sidekick observers[];
   public final void attach(Sidekick sidekick) { /* Attach sidekick */ }
   public final void detach(Sidekick sidekick) { /* Detach sidekick */ }
   public final void broadcast() {
      for (Sidekick observer : observers)
         observer.update(this);
   }
}
public interface Sidekick extends Character {
   void update(Observable hero);
}
public class GameEngine {  /* Manage life cycle of characters ... */  }
```

Fig. 1. A set up of the hero-sidekick game

Cheat I: Capability Theft. A `Sidekick` reference can be seen as a capability, the possession of which makes a `Hero` instance more potent. A malicious `Hero` can augment its own power by creating new instances of concrete `Sidekick`s, or stealing existing instances from unprotected sources, and then attaching these instances to itself.

Cheat II: Capability Theft and Leakage. A `Hero` exposes two type interfaces: (i) a sidekick management interface (i.e., `Hero`), and (ii) a state query interface (i.e., `Observable`). While the former is intended to be used exclusively by the `GameEngine`, the latter is a restrictive interface through which `Hero`s may be accessed securely by `Sidekick`s. This means that a `Hero` reference is also a capability from the perspective of `Sidekick`. Upon receiving a `Hero` object through the `Observable` argument of the `update` method, a malicious `Sidekick` may downcast the `Observable` reference to a `Hero` reference, and thus exposes the sidekick management interface of the `Hero` object (i.e., capability theft). This in turn allows the malicious `Sidekick` to attach sidekicks to the `Hero` object (i.e., capability leakage).

Solution Approach. To control capability propagation, DCC assigns the `Hero` and `Sidekick` interfaces to two distinct ***confinement domains*** [21], and restricts the exchange of capability references between the two domains. Specifically, capability references may only cross confinement boundaries via explicit argument passing. Capability granting is thus possible only under conscious ***discretion***. Notice that the above restrictions shall not apply to `GameEngine`, because it is by design responsible for managing the life cycle of `Hero`s and `Sidekick`s, and as such it requires the rights to acquire instances of `Hero`s and `Sidekick`s. This motivates the need to have a notion of ***trust*** to discriminate the two cases.

To further control the granting of capabilities, a ***capability granting policy*** [25] can be imposed on a method. For example, a capability granting policy can be imposed on the `broadcast` method so that the latter passes only `Observable` references to `update`, but never `Hero` references.

Our goal is not only to prevent capability theft and leakage between `Hero` and `Sidekick`, but also between the subtypes of `Hero` and those of `Sidekick`. In other words, we want to treat `Hero` and `Sidekick` as ***roles***, prescribe capability confinement constraints between them, and then require that their subtypes also conform to the constraints. DCC achieves this via a mechanism known as ***hereditary mutual suspicion***.

3 Discretionary Capability Confinement

This section presents the DCC annotation system for the JVM bytecode language. The threat model is reviewed in Sect. 3.1, the main type constraints are specified in Sect. 3.2, and the utility of DCC in addressing the security challenges of the running example is discussed in Sect.3.3.

3.1 Threat Model

As the present goal is to restrict the forging and propagation of abstractly typed references, we begin the discussion with an exhaustive analysis of all means by which a reference type A may ***acquire*** a reference of type C. We use metavariables A, B and C to denote *raw* JVM reference types (i.e., after erasure). We consider class and interface types here, and defer the treatment of array types and genericity till Sect. 5.1.

1. B ***grants*** a reference of type C to A when B invokes a method[1] declared in A, passing an argument via a formal parameter of type C.
2. B ***shares*** a reference of type C with A when one of the following occurs: **(a)** A invokes a method declared in B with return type C; **(b)** A reads a field declared in B with field type C; **(c)** B writes a reference into a field declared in A with field type C.
3. A ***generates*** a reference of type C when one of the following occurs: **(a)** A creates an instance of C; **(b)** A dynamically casts a reference to type C; **(c)** an exception handler in A with catch type C catches an exception.

3.2 Type Constraints

We postulate that the space of reference types is partitioned by the programmer into a finite number of ***confinement domains***, so that every reference type C is assigned to exactly one confinement domain via a domain label $l(C)$. We use metavariables $\mathcal{D}$ and $\mathcal{E}$ to denote confinement domains. The confinement domains are further organized into a ***dominance hierarchy*** by a programmer-defined partial order $\blacktriangleright$. We say that $\mathcal{D}$ ***dominates*** $\mathcal{E}$ whenever $\mathcal{E} \blacktriangleright \mathcal{D}$. The

[1] By a ***method*** we mean either an instance or static method, or an instance or class initializer. By a ***field*** we mean either an instance or static field.

dominance hierarchy induces a pre-ordering of reference types. Specifically, if $l(B) = \mathcal{E}$, $l(A) = \mathcal{D}$, and $\mathcal{E} \blacktriangleright \mathcal{D}$ then we write $B \rhd A$, and say that B ***trusts*** A. We write $A \bowtie B$ iff both $A \rhd B$ and $B \rhd A$. The binary relation $\bowtie$ is an equivalence relation, the equivalence classes of which are simply the confinement domains. If $C \rhd A$ does not hold, then a reference of type C is said to be a ***capability*** for A. Intuitively, capabilities should provide the sole means for untrusted types to access methods declared in capability types. The following constraint is imposed to ensure that an untrusted access is always mediated by a capability:

($\mathcal{DCC}1$) Unless $B \rhd A$, A shall not invoke a static method declared in B.

Acquiring non-capability references is harmless. Capability acquisition, however, is restricted by a number of constraints, the first of which is the following:

($\mathcal{DCC}2$) The following must hold:
1. A can generate a reference of type C only if $C \rhd A$. [No capability generation is permitted.]
2. B may share a reference of type C with A only if $C \rhd A \vee A \bowtie B$. [Capability sharing is not permitted across domain boundaries.]

In other words, capability acquisition only occurs as a result of explicit capability granting. Once a capability is acquired, it roams freely within the receiving confinement domain. Escape from a confinement domain is only possible when the escaping reference does not escape as a capability, or when it escapes as a capability via argument passing.

We also postulate that there is a ***root domain*** $\top$ so that $\top \blacktriangleright \mathcal{D}$ for all $\mathcal{D}$. All Java platform classes are members of the root domain $\top$. This means they can be freely acquired by any reference type[2].

Capability granting is regulated by discretionary control. We postulate that every declared method has a unique designator, which is denoted by metavariables m and n. We occasionally write $A.m$ to stress the fact that m is declared in A. Associated with every method m is a programmer-supplied label $l(m)$, called the ***capability granting policy*** of m. The label $l(m)$ is a confinement domain. (If $l(n) = \mathcal{E}$, $l(m) = \mathcal{D}$, and $\mathcal{E} \blacktriangleright \mathcal{D}$, then we write $n \rhd m$. Similarly, we write $m \rhd A$ and $A \rhd m$ for the obvious meaning.) Intuitively, the capability granting policy $l(m)$ dictates what capabilities may be granted by m, and to whom m may grant a capability.

($\mathcal{DCC}3$) If $A.m$ invokes[3] $B.n$, and C is the type of a formal parameter of n, then $C \rhd B \vee A \bowtie B \vee (B \rhd m \wedge C \rhd m)$.

[2] Notice that the focus of this paper is not to protect Java platform resources. Instead, our goal is to enforce application-level security policies that prescribe interaction protocols among dynamically loaded software extensions. The organization of the domain hierarchy therefore reflects this concern: platform classes and application core classes belong respectively to the least and the most dominating domain.

[3] In the case of instance methods, if $A.m$ invokes $B.n$, the actual method that gets dispatched may be a method $B'.n'$ declared in a proper subtype B' of B. Constraints ($\mathcal{DCC}3$) and ($\mathcal{DCC}4$) only regulate method invocation. Dynamic method dispatching is regulated by controlling method overriding through ($\mathcal{DCC}6$).

That is, capability granting ($\neg\, C \rhd B$) across domain boundaries ($\neg\, A \bowtie B$) must adhere to the capability granting policy of the caller ($B \rhd m \wedge C \rhd m$). Specifically, a capability granting policy $l(m)$ ensures that m only grants capabilities to those reference types B satisfying $B \rhd m$, and that m only grants capabilities of type C for which $C \rhd m$.

A method may be tricked into invoking another method that does not honor the caller's capability granting policy. This classical anomaly is known as the Confused Deputy [30]. The following constraint ensures that capability granting policies are always preserved along a call chain.

($\mathcal{DCC}4$) A method m may invoke another method n only if $n \rhd m$.

We now turn to the interaction between capability confinement and subtyping. We write $A <: B$ whenever A is either B itself or one of B's subtypes. A subtype exposes the interface of its supertypes. Specifically, if a reference type A has acquired a reference of type B, then A has effectively acquired a reference of every type B' that is a supertype of B. This is because implicit widening conversion is not considered a reference acquisition event in our threat model. The following constraint is imposed to ensure that widening does not turn a non-capability into a capability.

($\mathcal{DCC}5$) If $A <: B$ then $B \rhd A$.

When an instance method $B.n$ is invoked, the method that gets dispatched may be a method $B'.n'$ declared in a subtype B' of B. This allows $B'.n'$ to "*impersonate*" $B.n$, potentially allowing $B'.n'$ to **(i)** grant capabilities in a way that violates the capability granting policy of $B.n$, **(ii)** return a capability to a caller with whom B' is not supposed to share capabilities, or **(iii)** accept a capability argument that is intended for B rather than B'. The following constraint prevents impersonation.

($\mathcal{DCC}6$) Suppose $B.n$ is overridden by $B'.n'$. The following must hold:

1. $n' \rhd n$. [Overriding never relaxes capability granting rights.]
2. If the method return type is C, then $C \rhd B \vee B \bowtie B'$. [A method that returns a capability may not be overridden by a method declared in a different domain.]
3. If C is the type of a formal parameter, then $C \rhd B' \vee B \bowtie B'$. [A method may be granted a capability only if it does not override a method declared in a different domain.]

If reference types A and B do not trust each other (i.e., neither $A \rhd B$ nor $B \rhd A$ hold), they are said to be ***mutually suspicious***. The following constraint requires that mutual suspicion is preserved by subtyping.

($\mathcal{DCC}7$) **Hereditary mutual suspicion.** Suppose A and B are mutually suspicious. If $A' <: A$ and $B' <: B$, then A' and B' are also mutually suspicious.

($\mathcal{DCC}7$) results in a strong form of static separation of duty [27]. Firstly, as $\rhd$ is reflexive, no reference type can be a subtype of both A and B. This renders A and B mutually exclusive roles. Secondly, Sect. 4.2 shows that a class of collusion between A and B can be provably eliminated.

3.3 Addressing the Security Challenges

The challenge of capability theft and leakage described in our running example (Sect. 2) can be fully addressed by DCC. A dominance hierarchy for the hero-sidekick game application is given in Fig. 2. Because `HeroDomain` and `SidekickDomain` are incomparable in the dominance hierarchy, `Hero` and `Sidekick` are capabilities for each other. Consequently, not only are `Sidekick`s not allowed to downcast an `Observable` reference to a `Hero` capability (i.e., Cheat II), `Hero`s are also forbidden to create new `Sidekick` capabilities or to steal such capabilities through aliasing (Cheat I). Furthermore, the dominance hierarchy also renders `GameEngineDomain` the most dominating confinement domain, thereby allowing `GameEngine` to have full access to the reference types declared in the rest of the confinement domains. We also annotate every method $A.m$ displayed in Fig. 1 with a capability granting policy of $l(m) = l(A)$: e.g., $l(\texttt{update}) = \texttt{SidekickDomain}$. Consequently, even if a `Sidekick` obtains a `Hero` reference, it is still not allowed to attach any sidekick to the `Hero` instance (Cheat II). Lastly, hereditary mutual suspicion allows us to turn `Hero` and `Sidekick` into mutually suspicious roles, so that their subtypes cannot conspire to communicate capabilities.

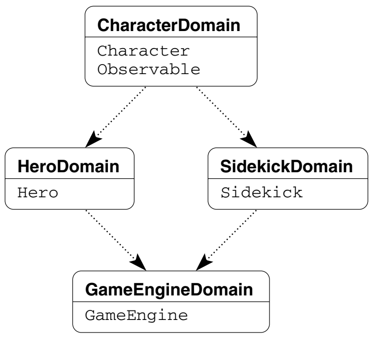


Fig. 2. Dominance hierarchy for the hero-sidekick application. Arrows represent "dominated-by" relationships (▶).

4 Confinement Properties

Given a discretionary access control mechanism such as DCC, safety analysis [31,32,33] must be conducted to characterize the conditions under which access rights are not granted to unintended parties. This section reports the confinement properties that have been established for DCC [34].

4.1 Featherweight JVM

Our confinement results are formalized in a lightweight model of the JVM called Featherweight JVM (FJVM) [35]. FJVM is a nondeterministic production system that describes how the JVM state evolves in reaction to access events. Nondeterminism is employed because we are not modeling the execution of a specific bytecode sequence, but rather all possible access events that may be generated by the JVM when well-typed bytecode sequences are executed. FJVM manipulates *object references*. Every object reference is an instance of exactly one class. An object has an arbitrary number of fields, each of which is declared either

$A, B, C \in \mathcal{C}$	raw reference types
$m, n \in \mathcal{M}$	method designators
$p, q, r \in \mathcal{O}$	object references
$S, T ::= \langle \Pi, \Gamma; \Phi, A.m, \sigma \rangle$	VM states
$\Pi ::= \emptyset \mid \Pi \cup \{r : C\}$	object pools
$\Gamma ::= \emptyset \mid \Gamma \cup \{p : B \rightsquigarrow q : C\}$	link graphs
$\Phi ::= \emptyset \mid \Phi \cup \{r : C\}$	stack frames
$\sigma ::= \diamond \mid push(\Phi, A.m, C, \sigma)$	proper stacks

Fig. 3. FJVM states

in the class of the object or one of the supertypes. Each field in turn stores an object reference. A field may only be initialized once but never updated.

Fig. 3 summarizes the structure of a VM state $\langle \Pi, \Gamma; \Phi, A.m, \sigma \rangle$. The *object pool* Π is a finite set of *allocations* $r : C$, recording the objects allocated by the VM, together with their class membership. The *link graph* Γ is a finite set of *links*. A link $p : B \rightsquigarrow q : C$ asserts that p has a field declared in B, with field type C, storing the object reference q. The *stack frame* Φ is a finite set of *labeled references* $r : C$. The set Φ models the references accessible in a JVM stack frame, and tracks the type interfaces that are visible to the execution context. The *execution context* $A.m$ is the currently executing method. The *proper stack* σ models the call chain that leads to the current VM state. Specifically, σ is either an *empty stack*, $\diamond$, or a *non-empty stack*, $push(\Phi, A.m, C, \sigma)$, where Φ is the caller stack frame, $A.m$ is the caller execution context, C is the callee return type, and σ is another proper stack.

In the following, we write $\overline{x}$ for a list $x_1, \ldots, x_k$. We also write $X \vdash x$ if $x \in X$. Obvious variations shall be clear from the context.

Fig. 4 defines the state transition relation $\rightarrow_\Sigma$, which is parameterized by a *safety policy* Σ. Intuitively, Σ specifies for each execution context $A.m$ the set $\Sigma[A.m]$ of permitted events. The transition rules ensure that $\rightarrow_\Sigma$ observes Σ. We model the type rules of DCC by the policy in Fig. 5. ($\mathcal{DCC}$5) and ($\mathcal{DCC}$7) are not modeled: ($\mathcal{DCC}$5) is implicitly assumed in the proofs [34], and ($\mathcal{DCC}$7) is orthogonal to the confinement results.

4.2 Confinement Theorem

To help articulate confinement guarantees, a family of *Accessible* judgments are defined in Fig. 6 to assert that a labeled reference ($r : C$ or $q : C$) is accessible from a domain ($\mathcal{D}$) in a given VM state. The main confinement theorem is stated below (consult [34] for a detailed proof).

Theorem 1 (Discretionary Capability Confinement). *Suppose* $\langle \Pi, \Gamma; \Phi, A.m, \diamond \rangle \xrightarrow{*}_\Sigma \langle \Pi', \Gamma'; \Phi', A'.m', \sigma' \rangle$. *Let* $\mathcal{D}$ *be an arbitrary domain. If* $Accessible_{[\mathcal{D}]}(r : C \mid \langle \Pi', \Gamma'; \Phi', A'.m', \sigma' \rangle)$, *then at least one of the following conditions holds:*

$$\frac{\Phi \vdash r : C \qquad C <: B}{\langle \Pi, \Gamma; \Phi, A.m, \sigma\rangle \rightarrow_{\Sigma} \langle \Pi, \Gamma; \Phi \cup \{r : B\}, A.m, \sigma\rangle} \quad \text{(T-WIDEN)}$$

$$\frac{\begin{array}{c} r \text{ is a fresh object reference from } \mathcal{O} \\ \mathbf{new}\langle B\rangle \in \Sigma[A.m] \end{array}}{\langle \Pi, \Gamma; \Phi, A.m, \sigma\rangle \rightarrow_{\Sigma} \langle \Pi \cup \{r : B\}, \Gamma; \Phi \cup \{r : B\}, A.m, \sigma\rangle} \quad \text{(T-NEW)}$$

$$\frac{\begin{array}{c} \Phi \vdash r : C \qquad \Pi \vdash r : C' \qquad C' <: B \\ \mathbf{checkcast}\langle B\rangle \in \Sigma[A.m] \end{array}}{\langle \Pi, \Gamma; \Phi, A.m, \sigma\rangle \rightarrow_{\Sigma} \langle \Pi, \Gamma; \Phi \cup \{r : B\}, A.m, \sigma\rangle} \quad \text{(T-CHECKCAST)}$$

$$\frac{\begin{array}{c} \Phi \vdash p : B_0 \qquad B_0 <: B \qquad \Gamma \vdash p : B \rightsquigarrow q : C \\ \mathbf{getfield}\langle B : C\rangle \in \Sigma[A.m] \end{array}}{\langle \Pi, \Gamma; \Phi, A.m, \sigma\rangle \rightarrow_{\Sigma} \langle \Pi, \Gamma; \Phi \cup \{q : C\}, A.m, \sigma\rangle} \quad \text{(T-GETFIELD)}$$

$$\frac{\begin{array}{c} \Phi \vdash p : B_0 \qquad B_0 <: B \qquad \Phi \vdash q : C \\ \mathbf{putfield}\langle B : C\rangle \in \Sigma[A.m] \end{array}}{\langle \Pi, \Gamma; \Phi, A.m, \sigma\rangle \rightarrow_{\Sigma} \langle \Pi, \Gamma \cup \{p : B \rightsquigarrow q : C\}; \Phi, A.m, \sigma\rangle} \quad \text{(T-PUTFIELD)}$$

$$\frac{\begin{array}{c} \Phi \vdash \overline{r} : \overline{C} \\ \mathbf{invokestatic}\langle B.n : \overline{C} \rightarrow C\rangle \in \Sigma[A.m] \end{array}}{\begin{array}{c}\langle \Pi, \Gamma; \Phi, A.m, \sigma\rangle \rightarrow_{\Sigma} \langle \Pi, \Gamma; \Phi', B.n, \sigma'\rangle \\ \text{where } \Phi' = \{\overline{r} : \overline{C}\} \text{ and } \sigma' = \mathit{push}(\Phi, A.m, C, \sigma)\end{array}} \quad \text{(T-INVOKESTATIC)}$$

$$\frac{\begin{array}{c} \Phi \vdash r_0 : C_0 \qquad C_0 <: B \qquad \Phi \vdash \overline{r} : \overline{C} \\ \Pi \vdash r_0 : B'' \qquad B'' <: B' \qquad B' <: B \\ \mathbf{invokemethod}\langle B.n : \overline{C} \rightarrow C\rangle[B'.n'] \in \Sigma[A.m] \end{array}}{\begin{array}{c}\langle \Pi, \Gamma; \Phi, A.m, \sigma\rangle \rightarrow_{\Sigma} \langle \Pi, \Gamma; \Phi', B'.n', \sigma'\rangle \\ \text{where } \Phi' = \{r_0 : B', \overline{r} : \overline{C}\} \text{ and } \sigma' = \mathit{push}(\Phi, A.m, C, \sigma)\end{array}} \quad \text{(T-INVOKEMETHOD)}$$

$$\frac{\Phi' \vdash r : C}{\langle \Pi, \Gamma; \Phi', B.n, \mathit{push}(\Phi, A.m, C, \sigma)\rangle \rightarrow_{\Sigma} \langle \Pi, \Gamma; \Phi \cup \{r : C\}, A.m, \sigma\rangle} \quad \text{(T-RETURN)}$$

Fig. 4. FJVM transitions

1. $\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \langle \Pi, \Gamma; \Phi, A.m, \diamond\rangle)$ *(previously accessible)*
2. $l(C) \blacktriangleright \mathcal{D}$ *(not a capability)*
3. $C \triangleright m \wedge \mathcal{D} \blacktriangleright l(m)$ *(controlled capability propagation)*

The theorem above ensures that capability propagation honors the capability granting policy of a method. In the following, we describe how one may annotate methods with capability granting policies to preserve useful confinement properties. Specifically, a method $A.m$ is said to be ***safe*** iff $m \triangleright A$. *Executing a safe method $A.m$ will only cause those domains dominated by $l(A)$ to acquire capabilities that A can generate.* Programmers concerned with capability confinement may then arrange their code to invoke untrusted software extensions only via safe method interfaces.

Theft. Capability theft occurs when executing code in a domain causes the domain to acquire capabilities it does not already possess. The absence of theft

$$\frac{B \rhd A}{\mathbf{new}\langle B\rangle \in \mathit{\Sigma}[A.m]} \quad \text{(P-New)}$$

$$\frac{B \rhd A}{\mathbf{checkcast}\langle B\rangle \in \mathit{\Sigma}[A.m]} \quad \text{(P-CheckCast)}$$

$$\frac{C \rhd A \vee A \bowtie B}{\mathbf{getfield}\langle B : C\rangle \in \mathit{\Sigma}[A.m]} \quad \text{(P-GetField)}$$

$$\frac{C \rhd B \vee A \bowtie B}{\mathbf{putfield}\langle B : C\rangle \in \mathit{\Sigma}[A.m]} \quad \text{(P-PutField)}$$

$$\frac{\begin{array}{c} n \rhd m \qquad B \rhd A \qquad C \rhd A \vee A \bowtie B \\ (\forall i \,.\, C_i \rhd B) \vee A \bowtie B \vee (B \rhd m \;\wedge\; \forall i \,.\, C_i \rhd m) \end{array}}{\mathbf{invokestatic}\langle B.n : \overline{C} \to C\rangle \in \mathit{\Sigma}[A.m]} \quad \text{(P-InvokeStatic)}$$

$$\frac{\begin{array}{c} n \rhd m \qquad n' \rhd n \qquad C \rhd A \vee A \bowtie B \\ C \rhd B \vee B \bowtie B' \qquad (\forall i \,.\, C_i \rhd B') \vee B \bowtie B' \\ (\forall i \,.\, C_i \rhd B) \vee A \bowtie B \vee (B \rhd m \;\wedge\; \forall i \,.\, C_i \rhd m) \end{array}}{\mathbf{invokemethod}\langle B.n : \overline{C} \to C\rangle[B'.n'] \in \mathit{\Sigma}[A.m]} \quad \text{(P-InvokeMethod)}$$

Fig. 5. A safety policy for DCC

$$\frac{l(B) = \mathcal{D} \qquad \mathit{\Gamma} \vdash p : B \rightsquigarrow q : C}{\mathit{Accessible}_{[\mathcal{D}]}(q : C \mid \mathit{\Gamma})} \qquad \frac{\mathit{\Phi} \vdash r : C' \qquad C' <: C}{\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \mathit{\Phi})}$$

$$\frac{\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \mathit{\Phi})}{\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \mathit{push}(\mathit{\Phi}, A.m, C', \sigma))} \qquad \frac{\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \sigma)}{\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \mathit{push}(\mathit{\Phi}, A.m, C', \sigma))}$$

$$\frac{\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \mathit{\Gamma})}{\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \langle \mathit{\Pi}, \mathit{\Gamma}; \mathit{\Phi}, A.m, \sigma\rangle)}$$

$$\frac{\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \mathit{\Phi})}{\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \langle \mathit{\Pi}, \mathit{\Gamma}; \mathit{\Phi}, A.m, \sigma\rangle)} \qquad \frac{\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \sigma)}{\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \langle \mathit{\Pi}, \mathit{\Gamma}; \mathit{\Phi}, A.m, \sigma\rangle)}$$

Fig. 6. Accessibility judgments

makes capabilities unforgeable. Theorem 1 entails that executing safe methods always guarantees the absence of capability theft.

Corollary 2 (No Theft). *Suppose* $m \rhd A$ *and* $\langle \mathit{\Pi}, \mathit{\Gamma}; \mathit{\Phi}, A.m, \diamond\rangle \xrightarrow{*}_{\mathit{\Sigma}} \langle \mathit{\Pi}', \mathit{\Gamma}'; \mathit{\Phi}', A'.m', \sigma'\rangle$*. Let* $\mathcal{D}$ *be* $l(A)$*. If* $\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \langle \mathit{\Pi}', \mathit{\Gamma}'; \mathit{\Phi}', A'.m', \sigma'\rangle)$*, then at least one of the following conditions holds:*

1. $\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \langle \mathit{\Pi}, \mathit{\Gamma}; \mathit{\Phi}, A.m, \diamond\rangle)$ *(previously accessible)*
2. $l(C) \blacktriangleright \mathcal{D}$ *(not a capability)*

Leakage. Capability leakage occurs when executing code in a domain causes a foreign domain to acquire a capability that the foreign domain does not already possess. When a capability is granted to a domain, it is in the interest of the granter that the grantee will not leak the granted capability. Theorem 1 entails that a safe method never leaks capabilities to a domain that is not dominated by the home domain of the method.

Corollary 3 (No Leakage). *Suppose* $m \rhd A$ *and* $\langle \Pi, \Gamma; \Phi, A.m, \diamond \rangle \xrightarrow{*}_{\Sigma} \langle \Pi', \Gamma'; \Phi', A'.m', \sigma' \rangle$. *Let* $\mathcal{D}$ *be a domain such that* $\mathcal{D} \blacktriangleright l(A)$ *is not true. If* $\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \langle \Pi', \Gamma'; \Phi', A'.m', \sigma' \rangle)$*, then at least one of the following conditions holds:*

1. $\mathit{Accessible}_{[\mathcal{D}]}(r : C \mid \langle \Pi, \Gamma; \Phi, A.m, \diamond \rangle)$ *(previously accessible)*
2. $l(C) \blacktriangleright \mathcal{D}$ *(not a capability)*

Mutual Suspicion. Suppose A and B are mutually suspicious, and a safe method $A.m$ is invoked. By Corollary 2, no reference of type B will be acquired by A as a result of the invocation. Similarly, by Corollary 3, no reference of type A will be acquired by B. Consequently, mutually suspicious types never exchange capabilities as a result of invoking safe methods. If the two types have never been explicitly granted capabilities of one another, then they cannot invoke methods declared in each others type interface. Collusion of this kind is therefore completely eliminated.

5 Extensions and Variations

5.1 Accommodating Other Language Constructs

Arrays. The array types $C[\,]$, $C[\,][\,]$, ... are said to be ***carrier types*** for declared type C. An object reference with a carrier type is a ***carrier***. If $\mathcal{D}$ acquires a carrier (e.g., of type $C[\,]$) for a capability type C, while $\mathcal{E}$ obtains a carrier-type reference (e.g., of type `Object`$[\,]$) to the same object, then $\mathcal{E}$ can store references into the carrier, while $\mathcal{D}$ can retrieve the said references as type-C capabilities. Special type constraints must be introduced into DCC to avoid the misuse of carriers as covert channels for capability communication. A solution is to allow the aliasing of carriers across domain boundaries so long as the acquisition of capability carriers is categorically denied. This can be enforced easily by minor revisions to the type constraints in Sect. 3.2: (a) assume $C \bowtie C[\,]$; (b) adapt ($\mathcal{DCC}3$) to forbid the granting of capability carriers across domain boundaries. Further details concerning the treatment of arrays can be found in [34].

Genericity. Genericity does not present any security challenge to the present design of DCC. Genericity is a purely source-level construct that is translated into bytecode via type erasure. The source-level generic type `Set<`C`>` is translated into the raw reference type `Set`. Set members are retrieved as `Object` references. The compiler introduces a dynamic cast to convert the retrieved `Object` reference into a type-C reference. There are two implications to this set up: (1) since generic containers such as `Set` belong to the root domain, DCC permits the acquisition and transmission of capability containers; (2) if C is a capability type, then P-CheckCast will effectively forbid the retrieval of any type-C

capabilities from generic containers. This is consistent with the overall design philosophy of DCC: capability acquisition must only occur as a result of explicit granting (i.e., argument passing). In summary, there is no security motivation for any additional type constraint to account for genericity.

5.2 Modular Enforcement of Hereditary Mutual Suspicion

Hereditary mutual suspicion ($\mathcal{DCC}7$) interacts with dynamic linking in a non-trivial manner. Specifically, ($\mathcal{DCC}7$) is universally quantified over all subtypes of two mutually exclusive roles. The enforcement of ($\mathcal{DCC}7$) thus involves a time complexity quadratic to the number of subtypes of the mutually exclusive roles, making it very inefficient. Worst still, because of the dynamic linking semantics of the JVM, some of these subtypes may not have been completely loaded, making it impossible to enforce ($\mathcal{DCC}7$) at link time. This section addresses these two issues by examining a reformulation of ($\mathcal{DCC}7$) that facilitates ***modular enforcement***. A conservative solution is adopted. Specifically, we want to check reference type A only once at link time, and then conclude that it will not participate in the violation of ($\mathcal{DCC}7$) in the future. To this end, we (1) lift the reasoning of mutual suspicion from the level of reference types to the level of confinement domains, and (2) capture in a binary relation the sufficient condition by which mutual suspicion is preserved in subtyping. A programmer-supplied partial order $:\blacktriangleright$ is postulated, so that:

$$\begin{array}{llll} (\mathcal{HMS}1) & \top :\blacktriangleright \mathcal{D} & (\mathcal{HMS}2) & \mathcal{D} :\blacktriangleright \mathcal{E} \Rightarrow \mathcal{D} \blacktriangleright \mathcal{E} \\ (\mathcal{HMS}3) & \multicolumn{3}{l}{(\mathcal{D} :\blacktriangleright \mathcal{E} \wedge \mathcal{D}' \blacktriangleright \mathcal{E}) \Rightarrow (\mathcal{D} \blacktriangleright \mathcal{D}' \vee \mathcal{D}' \blacktriangleright \mathcal{D})} \end{array}$$

We say that $\mathcal{D}$ ***strongly dominates*** $\mathcal{E}$ whenever $\mathcal{E} :\blacktriangleright \mathcal{D}$. The $:\blacktriangleright$ relation induces a pre-ordering of Java reference types: we write $B :\triangleright A$ iff $l(B) = \mathcal{E}$, $l(A) = \mathcal{D}$ and $\mathcal{E} :\blacktriangleright \mathcal{D}$. It follows readily from definition that $:\triangleright$ is reflexive and transitive, and $B :\triangleright A \Rightarrow B \triangleright A$. We restate ($\mathcal{DCC}7$) in a form that facilitates modular enforcement:

($\mathcal{DCC}7''$) If $A <: B$, then $B :\triangleright A$.

The companion technical report [34] shows that ($\mathcal{DCC}7$) follows from ($\mathcal{DCC}7''$). That is, ($\mathcal{DCC}7''$) is sound but incomplete: programs satisfying ($\mathcal{DCC}7''$) are guaranteed to satisfy ($\mathcal{DCC}7$), but some programs satisfying ($\mathcal{DCC}7$) may not satisfy ($\mathcal{DCC}7''$). We trade completeness for tractability.

6 Concluding Remarks

Related Work. Previous language-based capability systems [16,13,14] lack confinement guarantees. This work combines the idea of confinement domains [21] with capability granting policies [25] to achieve confinement.

The design of DCC has been influenced by confined types [21,22,23,24]. While the confinement boundaries of confined types are uniform, those in DCC are discriminatory, allowing reference acquisition through dominance and capability granting through discretion. This difference is due to the fact that confined types is designed to uniformly confine all instances of a given *concrete class*, but DCC is designed to selectively confine those references that would otherwise escape with a privileged *static type*.

The static type system pop [36] supports the reference-as-capability metaphor in an inheritance-less object calculus. Contrary to "communication-based" schemes of object confinement (e.g., confined types), an "used-based" approach has been adopted by pop to impose a custom "user interface" over an object. The user interface specifies how individual protection domains may access the object. DCC can be seen as a hybrid of communication-based and use-based approaches to capabilities: use-based views are modeled as static types imposed on references, and references may only escape from a confinement domain so long as they do not escape with a view that grants privileged accesses to the receiving domain.

Stack inspection [5] is an access control model for program execution that involves code units belonging to distinct protection domains. A common assumption behind most existing models of stack inspection [5,6,7,9] is that the binding of permissions to code units is performed statically. DCC identifies protection domains with confinement domains. While the binding of code units to their protection domains is performed statically, the granting of permissions to protection domains occurs dynamically through capability acquisition. Notice, however, *the right to grant capabilities* is still modeled statically in DCC in the form of a stack invariant.

Although this work is primarily concerned with access control, and thus orthogonal to language-based information flow control [37], one may see the **No Theft** and **No Leakage** properties as playing the analogous roles of **Simple Security** and ***-Property** in information flow control.

Separation of duty is foundational in ensuring system integrity [26]. Establishing mutually-exclusive roles is a popular means [28] for implementing separation of duty. The underlying assumption is that collusion between multiple agents is *unlikely*. In DCC, hereditary mutual suspicion not only establishes mutually exclusive roles, but *provably* prevents a class of collusion. To the best of the author's knowledge, this is the first work to enforce such a strong form of separation of duty in a language-based environment.

Future Work. To ease exposition, a simple representation of capability granting policy has been adopted. A future direction is to explore finer-grained representations of capability granting policies, and study the collaboration idioms thus enabled. First ideas are reported in [34].

The right to grant capability is always diminishing along a call chain. This restricts the reusability of methods, and causes methods deep in a call chain incapable of granting capabilities. Can we allow the amplification of capability granting right while preserving confinement? A helpful observation is that the reasoning of capability granting right is akin to stack inspection. Exploring this connection belongs to future work.

A limitation of DCC is the lack of support for capability revocation. It is obviously impossible to "revoke" a reference that has already been acquired by a confinement domain. However, the lack of revocation can be alleviated by carefully regulating authority delegation. Constrained delegation is a well-studied topic in role-based access control and trust management (see, particularly, [38,39,40]). Controlling delegation in DCC belongs to future work.

Acknowledgments. This work is supported by an NSERC Discovery Grant.

References

1. Carzaniga, A., Picco, G.P., Vigna, G.: Designing distributed applications with mobile code paradigms. In: Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA (1997) 22–32
2. Schneider, F.B., Morrisett, G., Harper, R.: A language-based approach to security. In: Informatics: 10 Years Back, 10 Years Ahead. Volume 2000 of LNCS. Springer (2000) 86–101
3. Edjlali, G., Acharya, A., Chaudhary, V.: History-based access control for mobile code. In: Proceedings of the 5th ACM Conference on Computer and Communications Security, San Francisco, California, USA (1998) 38–48
4. Gong, L., Schemers, R.: Implementing protection domains in the Java development kit 1.2. In: Proceedings of the Internet Society Symposium on Network and Distributed System Security, San Diego, California, USA (1998) 125–134
5. Wallach, D.S., Appel, A.W., Felten, E.W.: SAFKASI: A security mechanism for language-based systems. ACM Transactions on Software Engineering and Methodology **9** (2000) 341–378
6. Úlfar Erlingsson, Schneider, F.B.: IRM enforcement of Java stack inspection. In: Proceedings of the 2000 IEEE Symposium on Security and Privacy, Berkeley, California (2000) 246–255
7. Fournet, C., Gordon, A.D.: Stack inspection: Theory and variants. ACM Transactions on Programming Languages and Systems **25** (2003) 360–399
8. Abadi, M., Fournet, C.: Access control based on execution history. In: Proceedings of the 10th Annual Network and Distributed System Security Symposium, San Diego, California, USA (2003)
9. Pottier, F., Skalka, C., Smith, S.: A systematic approach to static access control. ACM Transactions on Programming Languages and Systems **27** (2005) 344–382
10. Dennis, J.B., van Horn, E.C.: Programming semantics for multiprogrammed computations. Communications of the ACM **9** (1966) 143–155
11. Miller, M.S., Yee, K.P., Shapiro, J.: Capability myths demolished. Technical Report SRL2003-02, System Research Lab, Department of Computer Science, The John Hopkins University (2003)
12. Rees, J.A.: A security kernel based on the lambda-calculus. A. I. Memo 1564, MIT (1996)
13. Wallach, D.S., Balfanz, D., Dean, D., Felten, E.W.: Extensible security architectures for Java. In: Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP'97), Saint Malo, France (1997) 116–128
14. Hawblitzel, C., Chang, C.C., Czajkowski, G., Hu, D., von Eicken, T.: Implementing multiple protection domains in Java. In: Proceedings of the USENIX Annual Technical Conference, New Orleans, Louisiana, USA (1998)

15. Chander, A., Dean, D., Mitchell, J.C.: A state-transition model of trust management and access control. In: Proceedings of the 14th IEEE Computer Security Foundations Workshop, Cape Breton, Nova Scotia, Canada (2001) 27–43
16. Jones, A.K., Liskov, B.H.: A language extension for expressing constraints on data access. Communications of the ACM **21** (1978) 358–367
17. Boyland, J., Noble, J., Retert, W.: Capabilities for sharing: A generalization of uniqueness and read-only. In: Proceedings of the 2001 European Conference on Object-Oriented Programming, Budapest, Hungary (2001) 2–27
18. Crary, K., Walker, D., Morrisett, G.: Typed memory management in a calculus of capabilities. In: Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, Texas, USA (1999) 262–275
19. Arnold, K., Gosling, J., Holmes, D.: The Java Programming Language. 3rd edn. Addison Wesley (2000)
20. ECMA: Standard ECMA-335: Common Language Infrastructure (CLI). 2nd edn. (2002)
21. Vitek, J., Bokowski, B.: Confined types in Java. Software - Practice & Experience **31** (2001) 507–532
22. Grothoff, C., Palsberg, J., Vitek, J.: Encapsulating objects with confined types. In: Proceedings of the 16th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, Tampa Bay, FL, USA (2001) 241–253
23. Zhao, T., Palsberg, J., Vitek, J.: Lightweight confinement for Featherweight Java. In: Proceedings of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, Anaheim, California, USA (2003) 135–148
24. Zhao, T., Palsberg, J., Vitek, J.: Type-based confinement. Journal of Functional Programming **16** (2006) 83–128
25. Gong, L.: A secure identity-based capability system. In: Proceedings of the 1989 IEEE Symposium on Security and Privacy, Oakland, California, USA (1989) 56–63
26. Clark, D.D., Wilson, D.R.: A comparison of commercial and military computer security policies. In: Proceedings of the 1987 IEEE Symposium on Security and Privacy. (1987) 184–194
27. Li, N., Bizri, Z., Tripunitara, M.V.: On mutually-exclusive roles and separation of duty. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, Washington DC, USA (2004) 42–51
28. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Transactions on Information and System Security **4** (2001) 224–274
29. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley (1994)
30. Hardy, N.: The confused deputy: or why capabilities might have been invented. Operating Systems Review **22** (1988) 36–38
31. Lipton, R.J., Snyder, L.: A linear time algorithm for deciding subject security. Journal of the ACM **24** (1977) 455–464
32. Sandhu, R.S.: The schematic protection model: Its definition and analysis for acyclic attenuating schemes. Journal of the ACM **35** (1988) 404–432
33. Sandhu, R.S.: The typed access matrix model. In: Proceedings of the 1992 IEEE Symposium on Security and Privacy. (1992) 122–136
34. Fong, P.W.L.: Discretionary capability confinement. Technical Report CS-2006-03, Department of Computer Science, University of Regina, Regina, Saskatchewan, Canada (2006)

35. Fong, P.W.L.: Reasoning about safety properties in a JVM-like environment. Technical Report CS-2006-02, Department of Computer Science, University of Regina, Regina, Saskatchewan, Canada (2006)
36. Skalka, C., Smith, S.: Static use-based object confinement. International Journal of Information Security **4** (2005) 87–104
37. Sabelfeld, A., Meyers, A.C.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications **21** (2003) 5–19
38. Bandmann, O., Dam, M., Firozabadi, B.S.: Constrained delegation. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy, Berkeley, California, USA (2002) 131–140
39. Li, N., Grosof, B.N., Feigenbaum, J.: Delegation logic: A logic-based approach to distributed authorization. ACM Transactions on Information and System Security **6** (2003) 128–171
40. Wainer, J., Kumar, A.: A fine-grained, controllable, user-to-user delegation method in RBAC. In: Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, Stockholm, Sweden (2005) 59–66

A Implementation Experience

A.1 Source-Level Annotations

Although DCC is formulated and enforced at the bytecode level, a specification mechanism has been devised to facilitate the annotation of Java source files with such DCC typing information as domain membership ($l(C)$), capability granting policy ($l(m)$), dominance relationship ($\blacktriangleright$), and strong dominance relationship ($:\blacktriangleright$). These source-level annotations are encoded using the JDK 5.0 metadata facility. For example, Fig. 7 illustrates how the domain hierarchy in Fig. 2 is encoded at the source level as an interface hierarchy. Specifically, a confinement domain is represented as an empty public interface with a `@Domain` annotation. The dominance relation is represented by interface extension: if a domain interface $\mathcal{E}$ extends another domain interface $\mathcal{D}$, then $\mathcal{D} \blacktriangleright \mathcal{E}$. The root domain $\top$ is represented by the predefined domain interface `Root`, which must be a superinterface of every user-defined domain interface. Strong dominance is specified via the `allowSubtyping` element of a `@Domain` annotation. Specifically, the value of an `allowSubtyping` element is a list of domain interfaces. If domain interface $\mathcal{D}$ appears in the `allowSubtyping` list of domain interface $\mathcal{E}$, then we intend it to mean $\mathcal{D} :\blacktriangleright \mathcal{E}$. If no `allowSubtyping` element is supplied, then, by default, the domain interface is strongly dominated only by `Root`. Lastly, domain membership and capability granting policies are indicated by the `@Confined` and `@Grants` annotations respectively. For example, the following declaration confines the `Robin` class to `SidekickDomain` and sets the capability granting policy of the `update` method to `SidekickDomain`:

```
@Confined( SidekickDomain.class )
public class Robin implements Sidekick {
   @Grants( SidekickDomain.class )
   public void update(Observable hero);
}
```

```
@Domain
public interface CharacterDomain extends Root { }
@Domain( allowSubtyping = { CharacterDomain.class } )
public interface HeroDomain extends CharacterDomain { }
@Domain( allowSubtyping = { CharacterDomain.class } )
public interface SidekickDomain extends CharacterDomain { }
@Domain
public interface GameEngineDomain extends HeroDomain, SidekickDomain { }
```

Fig. 7. An interface hierarchy representing the dominance hierarchy of the hero-sidekick game application

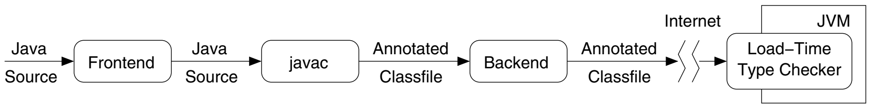


Fig. 8. The DCC software development environment

A.2 Type Checkers

We envision a programming environment (Fig. 8) in which Java source files embedded with DCC annotations are partially validated by a compiler frontend, and subsequently translated into annotated classfiles by the JDK 5.0 compiler. The annotated classfiles are then type-checked at the bytecode level by a compiler backend prior to shipping. To guard against malicious code generators, type checking is also conducted by the JVM at load time, against classfiles, at the bytecode level. All the three DCC type checkers depicted in Fig. 8 have been implemented. The ***frontend*** component is a source-level type checker based on the JDK 5.0 annotation processing tool (`apt`). It ensures that the type interface of Java classes and interfaces conform to type constraints ($\mathcal{DCC}5$), ($\mathcal{DCC}6$) ($\mathcal{DCC}7''$), as well as the $\mathcal{HMS}$ rules. The ***backend*** component is an offline, bytecode-level type checker based on the Apache ByteCode Engineering Library (BCEL). It ensures that classfiles or JAR files conform to *all* the DCC type constraints. Lastly, the ***load-time type checker*** is obtained by embedding the backend type checking engine into a Java class loader, which type-checks classfiles as they are loaded into the JVM.

The present design of DCC is optimized for enforcement efficiency, and as such it requires no iterative analysis of method bodies. All type constraints are enforced by a linear-time scan of classfiles.

Minimal Threshold Closure

Xi-Bin Zhao[1], Kwok-Yan Lam[1], Guimin Luo[1], and Siu-Leung Chung[2], and Ming Gu[1]

[1] School of Software, Tsinghua University, Beijing, PR China
{zxb, lamky, gluo, guming}@tsinghua.edu.cn
[2] School of Business Administration, The Open University of Hong Kong
slchung@ouhk.edu.hk

Abstract. Access structure is a flexible mechanism for representing complex access control and authorization policies [1]. Numerous efforts have been devoted to the research of efficient schemes for implementing access structures in a scalable manner. Threshold closure was invented as an efficient way to implement access structures that represent complex authorization policies [4]. In essence, threshold closure is an efficient and scalable implementation of access structure using a reduced collection of threshold schemes [5]. A practical application of threshold closure was presented in [6] where the use of threshold closure for addressing the complex security needs of Grid Computing Systems was explained. One major deficiency of threshold closure is that a threshold closure generated from the corresponding access structure is not minimal in size, thus the collection of threshold schemes is not optimized for efficiency. In this connection, an operation called minimal covering was proposed to minimize the size of a threshold closure once it is formed from its corresponding access structure [4]. Unfortunately, the minimal covering of a threshold closure is no longer a threshold closure, thus is not scalable in terms of addition/deletion of access control rules. This paper presents a way for constructing minimal threshold closure. It defines a new structure called enhanced threshold closure. The paper proves that the enhanced threshold closure of an access structure is a threshold closure and is minimal, hence it is also called a minimal threshold closure. The paper also presents a mechanism for constructing minimal threshold closure from a basis access structure.

Keywords: Access Control, Authorization, Threshold Schemes, Threshold Closure.

1 Introduction

Threshold scheme was proposed for implementing access control [5]. With a (t, l) threshold scheme, a secret key needed for accessing the resource is split into l pieces such that any t (or more) of the l pieces are needed to reconstruct the secret, hence enabling access to the resource. The beauty of the threshold scheme is that it is a simple mechanism for describing the "t out of l" authorization rules. More importantly, such rules can be directly implemented very efficiently using threshold cryptography. Hence, a collection of threshold schemes may be used to efficiently implement complex policies of access structures. The threshold scheme is attractive because it is computationally efficient and only involves the computation of a simple Lagrange interpolation. However, its expressive power is still very limited as it was proven that threshold schemes cannot express authorization policies in many cases [7]. For example, it cannot specify exactly which subset of participants is allowed to determine the secret and which is not. Therefore, the concept of access structure for representing complex secret sharing schemes was proposed by [1, 2, 3].

Access structure is a flexible mechanism for representing complex access control and authorization policies [1]. An access structure is composed of a collection of authorized set which can be implemented as a collection of threshold schemes [5]. Unfortunately, access structures are difficult to implement. Furthermore, the use of threshold schemes will be tedious if the policies they represent are dynamic. For example, when more "t out of l" rules are added to the system, it is highly likely that the overall collection of threshold schemes are redundant, thus leading to serious security management problems [4].

Numerous efforts have been devoted to the research of efficient schemes for implementing access structures in a scalable manner. The strong expressive power of access structure makes it an attractive approach for representing access control policies. However, the collection of threshold schemes of an access structure is hard to maintain especially when the authorized set needs to be updated regularly. In practical situations, dynamic updating of the authorized set is more of the norm than exception, unfortunately. In order to realize the potential benefits of access structure, some efficient schemes for implementing the dynamic authorized set in a scalable manner is highly desirable.

To allow efficient implementation of access structures and at the same time address the security management issues of threshold schemes, threshold closure was proposed by [4] as an efficient and flexible approach to secret sharing. A threshold closure is an efficient approach for representing an access structure by specifying a collection of threshold schemes. Complex authorization policies may be represented using an access structure (with each authorized set in the access structure represented by a threshold scheme) which is then translated to a threshold closure which in turn can be implemented efficiently using a minimal collection of threshold schemes. In essence, threshold closure is an efficient and scalable implementation of access structure by representing an access structure with a reduced collection of threshold schemes. [4] proved that a threshold closure generated from a basis access structure is equivalent to that basis access structure. Besides, operators are provided to maintain this one-to-one correspondence when access control rules and added/deleted from the authorized set. A practical application of threshold closure for addressing the complex security needs of Grid Computing Systems was presented in [6].

One major deficiency of threshold closure is that a threshold closure generated from the corresponding access structure is not minimal in size, thus the collection of threshold schemes is not optimized for efficiency. In this connection, an operation called minimal covering was proposed by [4] to minimize the size of a threshold closure once it is formed from its corresponding access structure. Unfortunately, the minimal covering of a threshold closure is no longer a threshold closure. Thus the threshold closure operators will not be applicable to the minimal covering, hence it is not scalable in terms of addition/deletion of access control rules and fails to maintain the scalability of threshold closure.

This paper addresses the deficiency of threshold closure by presenting a way for constructing minimal threshold closure. In this paper, we define a new structure called enhanced threshold closure, and prove that the enhanced threshold closure of an access structure is a threshold closure and is minimal. As a result of this prove, an enhanced threshold closure is also called a minimal threshold closure. The paper also presents a mechanism for constructing minimal threshold closure from a basis access structure.

This paper is organized as follows: An overview of threshold closure and the minimization issues of threshold closures are explained in Section 2. In Section 3, a new structure for implementing access structure, namely the enhanced threshold closure, will be introduced. This section also proves that an enhanced threshold closure is indeed a threshold closure. Section 4 proves that the enhanced threshold closure is minimal and unique. Section 5 describes the mechanism for constructing minimal threshold closure from access structure. The discussion of this paper is concluded in Section 6.

2 Threshold Closure

A threshold closure, denoted as ε, is a collection of (t, S)-threshold schemes (S is a set of l users such that $0 < t \leq |S|, S \subseteq P$ where P is the set of all potential participants/users), and satisfies the three conditions:

1. Redundant-free i.e. there do not exist two distinct $(t_1, S_1), (t_2, S_2) \in \varepsilon$ such that
$$S_1 \subseteq S_2 \ \texttt{or} \ |S_1 \cap S_2| \geq \min\{t_1, t_2\}, t_1 \neq t_2.$$
2. Reduced i.e. there do not exist $(t, S_1), (t, S_2), \ldots, (t, S_m) \in \varepsilon$ such that
$$\bigcup_{i=1}^{m} [S_i]_t = [\bigcup_{i=1}^{m} S_i]_t.$$
where
$$[S]_t = \{S' : |S'| = t, S' \subseteq S\}.$$
3. Closed i.e. $\forall (t, S_1), (t, S_2), \ldots, (t, S_m) \in \varepsilon$ and $S_1' \subseteq S_1, S_2' \subseteq S_2, \ldots, S_m' \subseteq S_m$ ("=" cannot be held by all) if
$$\bigcup_{i=1}^{m} [S_i']_t = [\bigcup_{i=1}^{m} S_i']_t$$

then

$$(t, \bigcup_{i=1}^{m} S_i') \in \varepsilon, \texttt{ or}$$

$$(t, \bigcup_{i=1}^{m} S_i') \notin \varepsilon \quad \texttt{and} \quad (\exists (t, S) \in \varepsilon) \bigcup_{i=1}^{m} S_i' \subset S.$$

It was proven in [4] that there exists a one-to-one correspondence between access structure Γ_0 and threshold closure ε. After implementing the algorithm of converting Γ_0 to ε, the number of the thresholds in the destination threshold closure is very much smaller than the number of the authorized sets in the access structure.

Besides, [4] also introduced four kinds of operation on Γ_0 and ε to allow authorization policies to be dynamically changed efficiently. The four operations are:

1. Add (t, S) into ε.
2. Add S into Γ_0.
3. Delete (t, S) from ε.
4. Delete S from Γ_0.

The consistency between Γ_0 and ε can be maintained using these four operations. By exploring these convenient operations, the threshold closure not only can expand and contract freely but also preserve its permanent consistency with the dynamic access structure. Therefore we can see that threshold closure has better efficiency and scalability while it keep the high express power of other general access structure schemes.

In addition, the $\min(\varepsilon)$ which is the minimal covering of threshold closure ε can be obtained. As pointed out by [4], the threshold closure is not minimal in size. In order to reduce the size of the threshold closure, a minimization mechanism was introduced and the result, namely minimal covering of the threshold closure, is a minimal collection of threshold schemes that represent the same access control policies as the original threshold closure. However, a minimal covering is not a threshold closure by itself. To illustrate this, we look at the following counter-example: For instance,

$$\begin{aligned}\varepsilon = \{&(2, P_1P_2P_5), (2, P_1P_3P_5), (2, P_1P_4P_6),\\ &(2, P_2P_3P_6), (2, P_2P_4P_7), (2, P_3P_4P_7),\\ &(2, P_1P_2P_3P_4)\}\end{aligned}$$

is a threshold closure [4]. According to the definition of the sufficient covering of threshold closure, it is easy to see that a collection

$$\begin{aligned}\varepsilon_1 = \{&(2, P_1P_2P_5), (2, P_1P_3P_5), (2, P_1P_4P_6),\\ &(2, P_2P_3P_6), (2, P_2P_4P_7), (2, P_3P_4P_7)\}\end{aligned}$$

is a sufficient covering of the threshold closure ε. It is the minimal covering of threshold closure ε, too. However, it is not a threshold closure.

Because of the contradiction between condition (3) of Definition 4.1 and condition (2) of sufficient covering (defined in Definition 4.5 of [4]), in general, a sufficient covering of threshold closure is not a threshold closure.

As a consequence, the minimal covering reduces the size of the threshold closure at the cost of scalability. Note that a threshold closure is scalable because it is consistent with its associated access structure, and such consistency can be maintained with the help of four conventional operations. Now that the minimal covering is not a threshold closure, the consistency cannot be maintained efficiently and hence the scheme is not scalable.

In the next section we define the enhanced threshold closure as a tool for representing a complex access structure. We then prove that an enhanced threshold closure is indeed a threshold closure with minimal size.

3 Enhanced Threshold Closure

Let $\mathcal{P} = \{P_1, P_2, \cdots, P_w\}$ be a finite of w participants. $\mathbf{T} \subseteq 2^{\mathcal{P}}$ be a collection of subsets of the set $\mathcal{P}$. Relation $\preceq$ be defined as the set inclusion $\subseteq$. Then the set $\mathbf{T}$ is a finite subset and $\{\mathbf{T}, \preceq\}$ is a poset. It implies that there are some maximal elements on the set $\mathbf{T}$. From the property of the poset we have that if $\overline{a}$ is a maximal element of the poset $\{\mathbf{T}, \preceq\}$, then $\overline{a}$ is unique and there exists no element in $\mathbf{T}$ to contain $\overline{a}$. Denote that:

$$\overline{\mathbf{T}} = \{\overline{a} | \overline{a} \text{ is a maximal element of } \mathbf{T}\}$$

and

$$[S]_t = \{X | |X| = t, X \subseteq S\}.$$

Definition 1. Let $\overline{\mathbf{T}}$ be defined on the set $\mathbf{T} \subseteq 2^{\mathcal{P}}$. A collection of (t,S)-threshold schemes $\varepsilon = \{(t, S) | 0 < t \leq |S|, S \in \overline{\mathbf{T}}\}$ is called an enhanced threshold closure of the set $\mathbf{T}$, if it satisfies the following conditions:

1. For any two distinct threshold schemes $(t_1, S_1), (t_2, S_2) \in \varepsilon$
$$|S_1 \cap S_2| < \min\{t_1, t_2\}.$$
2. If a set $S_0 \subseteq \mathcal{P}$ and $[S_0]_k \subseteq \cup_{(t,S)\in\varepsilon}[S]_t$, then there exists a threshold scheme $(k, S) \in \varepsilon$ such that $S_0 \subseteq S$. □

Now we show that an enhanced threshold closure is indeed a threshold closure. To prove this, we need the following lemmas.

Lemma 1. *If ε is an enhanced threshold closure of the set $\mathbf{T}$, then there do not exist (t, S_1), $(t, S_2), \cdots, (t, S_m) \in \varepsilon$ such that*

$$\cup_{k=1}^{m}[S_k]_t = [\cup_{k=1}^{m} S_k]_t.$$

Proof : Otherwise, if the above relation is held, then

$$[\cup_{k=1}^{m} S_k]_t \subseteq \cup_{(k,S)\in\varepsilon}[S]_k.$$

By condition (2) of Definition 1 there exists a threshold scheme $(t, S) \in \varepsilon$ that satisfies

$$\cup_{k=1}^{m} S_k \subseteq S.$$

Therefore, there exists at least one set $S_k \in \{S_1, S_2, \cdots, S_m\}$ such that $S_k \subset S$. It is contradictory to fact that S_k is a maximal element in the set $\mathbf{T}$. □

Lemma 2. *Suppose that ε is an enhanced threshold closure of the set $\mathbf{T}$. For any $(t, S_1), (t, S_2), \cdots, (t, S_m) \in \varepsilon$ and $S'_1 \subseteq S_1, S'_2 \subseteq S_2, \cdots, S'_m \subseteq S_m$ ("=" cannot be held by all), if*

$$\cup_{k=1}^{m}[S'_k]_t = [\cup_{k=1}^{m} S'_k]_t$$

then

$$(t, \cup_{k=1}^{m} S'_k) \in \varepsilon$$

or there exists a threshold scheme $(t, S) \in \varepsilon$ such that

$$\cup_{k=1}^{m} S'_k \subset S.$$

Proof : If $(t, S_1), (t, S_2), \cdots, (t, S_m) \in \varepsilon$ and $S'_1 \subset S_1, S'_2 \subset S_2, \cdots, S'_m \subset S_m$,

$$\cup_{k=1}^{m}[S'_k]_t = [\cup_{k=1}^{m} S'_k]_t$$

and

$$(t, \cup_{k=1}^{m} S'_k) \notin \varepsilon.$$

Hence

$$[\cup_{k=1}^{m} S'_k]_t = \cup_{k=1}^{m}[S'_k]_t \subseteq \cup_{k=1}^{m}[S_k]_t \subseteq \cup_{(t,S)\in\varepsilon}[S]_t$$

By condition (2) of Definition 1, there exists a threshold scheme $(t, S) \in \varepsilon$ such that

$$\cup_{k=1}^{m} S'_k \subseteq S$$

Since

$$(t, \cup_{k=1}^{m} S'_k) \notin \varepsilon$$

that is

$$\cup_{k=1}^{m} S'_k \neq S,$$

so

$$\cup_{k=1}^{m} S'_k \subset S.$$

□

From Lemma 1 and Lemma 2 we can obtain the following result.

Theorem 1. *If ε is an enhanced threshold closure, then it is also a threshold closure.*

Proof : Lemma 1 shows that an enhanced threshold closure satisfies Condition 2 of threshold closure. Lemma 2 shows that an enhanced threshold closure satisfies condition 3 of threshold closure. Hence, Lemma 1 and Lemma 2 proves that an enhanced threshold closure is a threshold closure. □

4 Minimal Threshold Closure

For a given access structure (i.e. a set of threshold schemes), there may exist more than threshold closure. For instance, the universal set should satisfy the condition of threshold closure [4]. We are interested in the performance of the enhanced threshold closure which could be determined by the number of elements it contains. This section we will answer this problem and illustrate that the enhanced threshold closure of the set $\mathbf{T}$ is minimal and unique if it exists.

Definition 2. Let $\mathcal{A} \subseteq 2^{\mathcal{P}}$ be a finite set. $\xi(\mathcal{A}) \subseteq 2^{\mathcal{P}}$ is a closure of $\mathcal{A}$ if for $\forall A \in \mathcal{A}, \exists C \in \xi(\mathcal{A})$, such that $A \subseteq C \subseteq \mathcal{P}$. $\square$

Definition 3 A closure $\mathcal{B}$ of $\mathcal{A}$ is called a minimal closure of $\mathcal{A}$, if $\xi(\mathcal{A})$ is another closure of $\mathcal{A}$, then $\xi(\mathcal{A})$ must be a closure of $\mathcal{B}$. A minimal closure of $\mathcal{A}$ is called the least closure of $\mathcal{A}$ if it has the least elements among the collection of minimal closure of $\mathcal{A}$, that is, if $\mathcal{B}$ is the least closure of $\mathcal{A}$ and $\mathcal{C}$ is any minimal closure of $\mathcal{A}$, then $|\mathcal{B}| \leq |\mathcal{C}|$. $\square$

Lemma 3. *The least closure of $\mathcal{A}$ is unique.*

Proof : Suppose that $\mathcal{B}$ and $\mathcal{C}$ are two distinct least closures of $\mathcal{A}$. By the definition we have $|\mathcal{B}| = |\mathcal{C}|$. It follows that there exits an element $B \in \mathcal{B}$ and $B \notin \mathcal{C}$. Because $\mathcal{B}$ is a minimal closure and $\mathcal{C}$ is a closure of $\mathcal{A}, \mathcal{C}$ should be a closure of $\mathcal{B}$. There exists an element $C \in \mathcal{C}$, such that $B \subset C$. Because $\mathcal{C}$ is also a minimal closure of $\mathcal{A}$, the same reason as before, there exist an element $B_1 \in \mathcal{B}$, such that $C \subseteq B_1$. It implies that $B, B_1 \in \mathcal{B}$, and $B \subset B_1$. We construct a collection $\mathcal{D}$ as the following:

$$\mathcal{D} = \mathcal{B} - \{B\}$$

It is obvious that $\mathcal{D}$ is also a minimal closure of $\mathcal{A}$ since B_1 is still an element of $\mathcal{D}$. But $|\mathcal{D}| = |\mathcal{B}| - \infty$. It contradicts that $\mathcal{B}$ is the least closure of $\mathcal{A}$. Therefore the least closure of $\mathcal{A}$ is unique. $\square$

According to the previous definition, the enhanced threshold closure is based on a maximal element set $\overline{\mathbf{T}}$. Thus the number of element in the enhanced threshold closure depends on the set $\overline{\mathbf{T}}$. To determine the size of the enhanced threshold closure, we need the following theoretical results.

Lemma 4. *$\overline{\mathbf{T}}$ is the least closure of $\mathbf{T}$.*

Proof : From the definitions of the closure and $\overline{\mathbf{T}}$, it is easy to see that $\overline{\mathbf{T}}$ is a closure of $\mathbf{T}$. If $\overline{\mathbf{T_1}}$ is another closure of $\mathbf{T}$, then for any element $a \in \mathbf{T}$, there exists an element in $\overline{\mathbf{T_1}}$ contained the element a. According to the definition of $\overline{\mathbf{T}}$, it implies $\overline{\mathbf{T}} \subseteq \mathbf{T}$. Therefore, for every $\overline{a} \in \overline{\mathbf{T}}$, there exists an element in $\overline{\mathbf{T_1}}$ contained the element $\overline{a}$. It follows that $\overline{\mathbf{T}}$ is a minimal closure of $\mathbf{T}$.

If there exists a set $\overline{\mathbf{T_2}}$ such that $\overline{\mathbf{T_2}}$ is a minimal closure of $\mathbf{T}$ and $|\overline{\mathbf{T_2}}| < |\overline{\mathbf{T}}|$, by the above argument, $\overline{\mathbf{T_2}}$ is also a closure of $\overline{\mathbf{T}}$. Then for every $\overline{a} \in \overline{\mathbf{T}}$, there exists an element $\overline{b} \in \overline{\mathbf{T_2}}$, such that $\overline{a} \subseteq \overline{b}$.

Because $\overline{\mathbf{T}}$ is the collection of the maximal elements of $\mathbf{T}$ and $|\overline{\mathbf{T_2}}| < |\overline{\mathbf{T}}|$, there exist at least two elements $\overline{a_1}, \overline{a_2} \in \overline{\mathbf{T}}$ and an element $\overline{b_1} \in \overline{\mathbf{T_2}}$, such that $\overline{a_1} \subseteq \overline{b_1}$ and $\overline{a_2} \subseteq \overline{b_1}$.

Since $\overline{\mathbf{T}}$ is a minimal closure of $\mathbf{T}$, it is a closure of $\overline{\mathbf{T_2}}$ too. It follows that for $\overline{b_1} \in \overline{\mathbf{T_2}}$, there exists an element $\overline{a_3} \in \overline{\mathbf{T}}$, such that $\overline{b_1} \subseteq \overline{a_3}$. It implies that $\overline{a_1} \subseteq \overline{a_3}$ and $\overline{a_2} \subseteq \overline{a_3}$. It is contradictory to the fact that $\overline{a_2}$ is a maximal element in $\mathbf{T}$. Therefore, $\overline{\mathbf{T}}$ is the least closure of $\mathbf{T}$. □

From Lemma 3, Lemma 4, and Definition 1 it concludes that:

Theorem 2. *If ε is an enhanced threshold closure of the set* $\mathbf{T}$*, then ε is unique and minimal.*

Proof : According to Definition 1, an enhanced threshold closure is a maximal element collection. The maximal element collection is the least closure (Lemma 3) and the least closure is unique (Lemma 4). □

Because of Theorem 2, the enhanced threshold closure of a collection of threshold schemes is the unique minimal threshold closure of the collection. Hence, from now on, the term "enhanced threshold closure" and "minimal threshold closure" will be used interchangeably.

5 Constructing Minimal Threshold Closure

The collection of authorized sets Γ and unauthorized sets $\widetilde{\Gamma}$, assumed to be disjoint, are called the access structure $(\Gamma, \widetilde{\Gamma})$ of the secret sharing scheme. Further, if every subset of participants belongs to either Γ or $\widetilde{\Gamma}$ then $(\Gamma, \widetilde{\Gamma})$ is called complete. Through this discussion we assume that every access structure is complete.

If Γ is an access structure on $\mathcal{P}$, then $X \in \Gamma$ is a minimal qualified set if $Y \notin \Gamma$ whenever $Y \subset X$. The family of minimal qualified set of Γ is denoted as Γ_0 and is called the basis of Γ. We refer to a minimal qualified set as a basis set.

Definition 4. An access structure Γ_0 and a threshold closure ε are said to be consistent with each other iff

$$\cup_{(t,S)\in\varepsilon}[S]_t = \Gamma_0.$$

If Γ_0 and ε are consistency, we denoted as $\Gamma_0 \sim \varepsilon$ or $\varepsilon \sim \Gamma_0$. □

Now we want to find a way to construct the enhanced threshold closure. First of all, the relationship between the enhanced threshold closure and basis access structure is analyzed. Next theorems are contributed for this purpose.

Theorem 3. *If two enhanced threshold closures ε_1 and ε_2 are distinct, then*

$$\cup_{(t,S)\in\varepsilon_1}[S]_t \neq \cup_{(t,S)\in\varepsilon_2}[S]_t.$$

Proof : Otherwise, if $\varepsilon_1 \neq \varepsilon_2$, and

$$\cup_{(t,S)\in\varepsilon_1}[S]_t = \cup_{(t,S)\in\varepsilon_2}[S]_t \tag{1}$$

We can suppose that there exists a threshold $(t_1, S_1) \in \varepsilon_1 - \varepsilon_2$. By Equation 1,

$$[S_1]_{t_1} \subseteq \cup_{(t,S)\in\varepsilon_2}[S]_t.$$

From condition (2) of Definition 1 it follows that there exists a threshold $(t, S_2) \in \varepsilon_2$ such that $S_1 \subset S_2$. Same reason we get that there exists a threshold $(t, S_3) \in \varepsilon_1$ such that $S_2 \subset S_3$. It implies that $S_1, S_3 \in \varepsilon_1$ and $S_3 \subset S_1$. It is in contradiction with that $S_1 \in \mathbf{T}$ is a maximal element. □

Theorem 4. *Given a basis access structure Γ_0, there is one and only one enhanced threshold closure ε consistent with it.*

Proof : Given a basis access structure Γ_0. Define

$$\Xi = \{(t,S) | [S]_t \subseteq \Gamma_0\}, \quad \Theta = \{(t,S) | (t,S) \in \Xi, S \text{ is a maximal element}\} \tag{3}$$

$$\Pi = \{S | (t,S) \in \Xi\}, \quad \overline{\Pi} = \{S | (t,S) \in \Theta\} \tag{4}$$

Now we will prove that Θ is an enhanced threshold closure consistent with Γ_0. First, we prove that Θ is an enhanced threshold closure. If there exist two thresholds $(t_1, S_1), (t_2, S_2)$:

$$|S_1 \cap S_2| \geq t_1, \quad t_1 < t_2.$$

Then there exist two sets $S_{t_1}, S_{t_2} \in \Gamma_0$, such that

$$|S_{t_1}| = t_1, |S_{t_2}| = t_2, S_{t_1} \subseteq S_1, S_{t_2} \subseteq S_2, S_{t_1} \subset S_{t_2}$$

It is in contradiction with the definition of Γ_0. Therefore, for any two distinct access structures $(t_1, S_1), (t_2, S_2) \in \Theta$ condition (1) of Definition 1 hold. It also follows that the set $\overline{\Pi}$ is the maximal elements on the set Π.

Suppose there exists a set S_0 to satisfy:

$$[S_0]_k \subseteq \cup_{(t,S)\in\Theta}[S]_t$$

Since $(t, S) \in \Theta \subseteq \Xi$. Then $[S]_t \subseteq \Gamma_0$. It implies that $[S_0]_k \subseteq \Gamma_0$. It follows $(k, S_0) \in \Xi$. Thus, there exists a threshold $(k, S) \in \Theta$ and $S_0 \subseteq S$. Therefore, the set Θ is an enhanced threshold closure of the set Π.

Next we prove that $\Gamma_0 \sim \Theta$. By relation (3) $(t, S) \in \Theta \subseteq \Xi$, that is, $[S]_t \subseteq \Gamma_0$. Thus, $\cup_{(t,S)\in\Theta}[S] \subseteq \Gamma_0$.

For any $S_1 \in \Gamma_0$, it is obvious that $[S_1]_{|S_1|} \in \Xi$. By the definition of Θ, there exists a threshold $(|S_1|, S_2) \in \Theta$, such that $S_1 \in [S_2]_{|S_1|}$. Therefore, $\Gamma_0 \sim \Theta$.

Lastly, from Theorem 3 we can conclude that there is only one enhanced threshold closure consistent with the basis access structure Γ_0. □

Theorem 5. *Given an enhanced threshold closure ε, there is one and only one basis access structure Γ_0 consistent with it.*

Proof : Construct

$$\cup_{(t,S)\in\varepsilon}[S]_t = \Gamma.$$

If there exist two sets

$$S_{10} \in [S_1]_{|S_{10}|}, S_{20} \in [S_2]_{|S_{20}|}, S_{10} \subset S_{20}$$

where $(|S_{10}|, S_1), (|S_{20}|, S_2) \in \varepsilon$. It is a contradiction to the condition (1) of Definition 1. Therefore, Γ is a basis access structure. From the construction it is directly to see that Γ is consistent with the enhanced threshold closure ε. The uniqueness of the basis access structure is also followed from the definition and the construction. $\square$

By combining Theorem 3, 4 and 5, it concludes that the enhanced threshold closure and basis access structure are one-to-one correspondence.

6 Conclusion

This paper addresses the deficiency of threshold closure by presenting a way for constructing minimal threshold closure. In this paper, we defined a new structure called enhanced threshold closure, and proved that the enhanced threshold closure of an access structure is a threshold closure and is minimal. As a result of this proof, an enhanced threshold closure is also called a minimal threshold closure. A mechanism for constructing minimal threshold closure from a basis access structure was also presented in this paper.

Acknowledgement. This research was partly funded by the Chinese National Science Foundation (Project Number: 90412007), P. R. China and the Grid Security project of PrivyLink International Limited (Singapore).

References

1. M. Ito, A. Saito, T. Nishizeki. "Secret sharing scheme realizing general access structure". in Globecom'87, Tokyo, Japan, 1987, pp.99-102.
2. K. Tochikubo. "Efficient Secret Sharing Schemes Realizing General Access Structures", IEICE Trans. on Fundamentals, vol.E87-A, no.7, pp.1788-1797.
3. M. Iwamoto, H. Yamamoto, and H. Ogawa. "Optimal Multiple Assignments Based on Integer Programming in Secret Sharing Schemes", IEEE-ISIT2004, p.16, Chicago, June-July, 2004.
4. C.R. Zhang, K.Y. Lam, S. Jajodia. "Scalable threshold closure". *Theoretical Computer Science*, 226(1999) 185-206.
5. A. Shamir. "How to share a secret", *Communications of the ACM*, Vol 22, No 11, 1979, pp. 612-613.

6. X.B. Zhao, K.Y. Lam, S.L. Chung, M. Gu and J.G. Sun. "Authorization Mechanisms for Virtual Organizations in Distributed Computing Systems", 9th Australasian Conference On Information Security and Privacy (ACISP'04), Sydney, Australia, July 13-15, 2004, Springer-Verlag LNCS 3108, pp 414-426.
7. J.C. Benaloh, J. Leichter. "Generalized secret sharing and monotone functions". *Advances in Cryptology-CRYPTO'88, Lecture Notes in Computer Science*, vol.403, Springer, Berlin, 1989, pp27-35.

Reducing the Dependence of SPKI/SDSI on PKI

Hao Wang[1,⋆], Somesh Jha[1,⋆], Thomas Reps[1,⋆], and Stefan Schwoon[2], and Stuart Stubblebine[3]

[1] University of Wisconsin, Madison, U.S.A.
{hbwang, jha, reps}@cs.wisc.edu
[2] Universität Stuttgart, Germany
schwoosn@fmi.uni-stuttgart.de
[3] Stubblebine Research Labs
stuart@stubblebine.com

Abstract. Trust-management systems address the authorization problem in distributed systems. They offer several advantages over other approaches, such as support for delegation and making authorization decisions in a decentralized manner. Nonetheless, trust-management systems such as KeyNote and SPKI/SDSI have seen limited deployment in the real world. One reason for this is that both systems require a public-key infrastructure (PKI) for authentication, and PKI has proven difficult to deploy, because each user is required to manage his/her own private/public key pair. The key insight of our work is that issuance of certificates in trust-management systems, a task that usually requires public-key cryptography, can be achieved using secret-key cryptography as well. We demonstrate this concept by showing how SPKI/SDSI can be modified to use Kerberos, a secret-key based authentication system, to issue SPKI/SDSI certificates. The resulting trust-management system retains all the capabilities of SPKI/SDSI, but is much easier to use because a public key is only required for each SPKI/SDSI server, but no longer for every user. Moreover, because Kerberos is already well established, our approach makes SPKI/SDSI-based trust management systems easier to deploy in the real world.

1 Introduction

Authorization is a central problem in distributed environments where resources are shared among many users across different administrative domains. Trust-management systems [3] are designed to address the authorization problem in distributed environments; they answer the question "Is principal A allowed to perform operation O on a shared resource R?". Existing trust-management systems, such as KeyNote [2] and SPKI/SDSI[1] [9], rely heavily on public-key in-

⋆ Supported by NSF under grants CCF-0524051 and CCR-9986308, and by ONR under grants N00014-01-1-{0796,0708}.

[1] Strictly speaking, SPKI/SDSI would not be considered to be a trust-management system according to the definition given by Blaze *et al.* [3]—if the processing of the certificates is not standardized (i.e., is application specific). In the context of this paper, we assume that certificate processing in SPKI/SDSI is standardized, and hence consider SPKI/SDSI to be a trust-management system.

frastructure (PKI). They use PKI to produce digitally-signed *certificates*, which authorize a principal to perform an operation on a shared resource.

However, PKI-based systems have proved difficult to deploy in practice because of several reasons [17]. Some issues (e.g., naming) have been addressed by trust-management systems, such as KeyNote and SPKI/SDSI. However, each user is still required to possess a public-private key pair, and it is cumbersome to securely transport and retrieve private keys. Complexity of PKI is another issue that makes PKI-based systems difficult to deploy. Implementing PKI-based solutions requires in-depth knowledge of PKI and much modification to existing systems.

Despite the issues mentioned above, trust-management systems are still desirable for authorization in distributed environments because they offer several advantages over traditional centralized authorization systems [2]. For example, because the trust-management system SPKI/SDSI has no conceptual requirement for a central authority and provides the ability to make authorization decisions in a truly distributed fashion [14], it is very scalable—an important requirement in distributed systems. SPKI/SDSI is also simple to use as it supports delegation, which simplifies access control, and provides locally defined name spaces, which allows each user to define his/her own security policies.

We introduce a technique to reduce the dependence of trust-management systems on PKI so that they become easier to deploy in the real-world. We observe that the main use of PKI in trust-management systems is to digitally sign each certificate with the private key of the principal who issues the certificate. The key behind our work is that *the signing process can be achieved using secret-key-based systems as well.* Although the notion of using secret-key cryptography in place of public-key cryptography as the building block of security operations has been studied previously [16,8] and has been used in distributed military and banking systems, to the best of our knowledge, our work is the first to apply this technique in the context of trust-management systems, specifically SPKI/SDSI.

By utilizing existing secret-key-based systems, which are already widely deployed, we can reduce the dependence of trust-management systems on PKI because end users no longer need to have public-private key pairs. In our approach, each site in a distributed environment has a dedicated trust-management server, whose sole purpose is to issue digitally-signed certificates, and this server possesses a public-private key pair. Users at a site authenticate themselves to this server using a secret key, and the server issues digitally-signed certificates on their behalves. Thus, in our solution *just one server per site needs to have a public-private key pair*, as opposed to traditional trust-management systems, where *each principal must possess a public-private key pair.*

In this paper, we focus on the trust-management system SPKI/SDSI and show how to reduce its dependence on PKI by using Kerberos [19], a widely-deployed secret-key-based authentication system. In our approach, we allow authenticated Kerberos users to issue SPKI/SDSI certificates. The Kerberized SPKI/SDSI server[2] (K-SPKI/SDSI) accepts certificate requests from authenticated Kerberos

[2] Here, *Kerberize* means that we modify the SPKI/SDSI server to use Kerberos library.

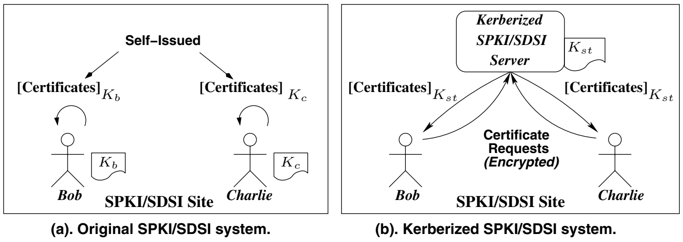

Self–Issued
[Certificates]$_{K_b}$
[Certificates] K_c
K_b
K_c
Bob
SPKI/SDSI Site
Charlie
(a). Original SPKI/SDSI system.
Kerberized SPKI/SDSI Server
K_{st}
[Certificates]$_{K_{st}}$
[Certificates]$_{K_{st}}$
Certificate Requests (Encrypted)
Bob
SPKI/SDSI Site
Charlie
(b). Kerberized SPKI/SDSI system.

Fig. 1. Reducing SPKI/SDSI's dependence on PKI using Kerberos

users, and generates corresponding SPKI/SDSI certificates on their behalves. In the original SPKI/SDSI system, shown in Figure 1(a), each principal has the ability to issue *name certificates* and *auth certificates*, signed using his/her public-private key. In contrast, with our solution, shown in Figure 1(b), each user no longer needs to have a public-private key pair. Instead, a site has a dedicated Kerberized SPKI/SDSI server—with its own public-private key—that is responsible for signing certificates. To issue a SPKI/SDSI certificate, a user first authenticates with the local Kerberos server and obtains a secure communication channel with the K-SPKI/SDSI server. The user can then issue the same certificates, in the form of *certificate requests*, but without the public-private key pair. The certificate requests are sent by the user, through the secure channel, to the K-SPKI/SDSI server, which creates and signs the certificates. The signed certificates can be either stored at the K-SPKI/SDSI server or sent back to the user, depending on the configuration of the system. In the case where the newly issued certificates are sent back to the user, the new system operates identically to the original SPKI/SDSI system because the certificates are stored locally, and authorization decisions can still be made locally. If the certificates are kept on the server, the server would act as a repository for the certificates issued by users in its domain. Such repositories could be used to organize certificate-chain discovery either centrally or in a distributed manner [14], leaving the burden of certificate management completely to the server.

Our technique offers several tangible benefits. Because end-users of the system authenticate themselves with a dedicated trust-management server using secret keys, it rids trust-management systems of the requirement that each principal must possess a public-private key pair. Furthermore, because we use secret keys to authenticate users to the dedicated server, and secret-key cryptography is widely deployed, we believe that the solution we present will make it easier to deploy PKI-based trust-management systems. In addition, only a small change is required at the end-user level to deploy our solution: each Kerberos application can now pass an optional parameter to the Kerberos library function `kuserok` to indicate that it wants to use our K-SPKI/SDSI server to perform an authorization check. Finally, because the dedicated trust-management server still uses a public-private key pair, our solution retains all the advantages of trust-management systems, such as delegation and distributed authorization.

The contributions of this paper are as follows:

- We show how to make SPKI/SDSI easier to deploy in the real world by reducing its dependence on PKI through leveraging Kerberos, a secret-key-based system that is already widely deployed.
- Our approach synthesizes the benefits of both secret-key-based authentication systems, such as Kerberos, and PKI-based trust-management systems, such as SPKI/SDSI. We utilize Kerberos' proven authentication framework while retaining SPKI/SDSI's elegant distributed authorization features, such as *delegation*, *authorization proofs*, *local name spaces*, and *distributed certificate-chain discovery.*
- We have created a prototype that implements the technique; the paper provides a preliminary report about our implementation and its performance.

Background on SPKI/SDSI is given in Section 2; readers with a knowledge of SPKI/SDSI may choose to skip this section. The method for combining SPKI/SDSI and Kerberos is described in Section 3. Section 4 discusses deployment and performance issues of our prototype. Section 5 discusses related work.

2 Background on SPKI/SDSI

SPKI/SDSI [9] is a novel public-key infrastructure designed to address the authorization problem in distributed systems. In SPKI/SDSI, a *principal* can be an individual, process, host, or any other entity. All *principals* are represented by their public keys, i.e., a principal *is* its public key. Let $\mathcal{K}$ denote the set of public keys; specific keys are denoted by K, K_A, K_B, K', etc. An *identifier* is a word over some alphabet Σ. The set of identifiers is denoted by $\mathcal{A}$. Identifiers will be written in typewriter font, e.g., `A` and `Bob`. A *term* is a key followed by zero or more identifiers. Terms are either keys, local names, or extended names. A *local name* is of the form K `A`, where $K \in \mathcal{K}$ and `A` $\in \mathcal{A}$. For example, K `Bob` is a local name. Local names are important in SPKI/SDSI because they create a decentralized name space. The local name space of K is the set of local names of the form K `A`. An *extended name* is of the form $K\ \sigma$, where $K \in \mathcal{K}$ and σ is a sequence of identifiers of length greater than one. For example, K `UW CS faculty` is an extended name.

2.1 Certificates

SPKI/SDSI has two types of certificates, or "certs":

Name Certificates (or *name certs*): A name cert provides a definition of a local name in the issuer's local name space. Only key K may issue or sign a cert that defines a name in its local name space. A name cert is a signed four-tuple $(K, \mathtt{A}, S, V)$. The issuer K is a public key and the certificate is signed by K. `A` is an identifier. The subject S is a term. Intuitively, S gives additional meaning for the local name K `A`. V is the *validity specification* of the certificate. Usually, V takes the form of an interval $[t_1, t_2]$, i.e., the cert is valid from time t_1 to t_2 inclusive.

Authorization Certificates (or *auth certs*): An auth cert grants (with or without delegation privileges) a specific authorization from an issuer to a subject. Specifically, an auth cert is a five-tuple (K, S, D, T, V). The *issuer* K is a public key, which is also used to sign the cert. The *subject* S is a term. If the *delegation bit* D is turned on, then a subject receiving this authorization can delegate this authorization to other keys. The *authorization specification* T specifies the permission being granted; for example, it may specify a permission to read a specific file, or a permission to login to a particular host. The *validity specification* V for an auth cert is the same as in the case of a name cert.

2.2 Certificates as Rewrite Rules

A *labeled rewrite rule* is a triple $L \xrightarrow{T} R$, where L and R are terms and T is an authorization specification. $\hat{T}$ is the authorization specification such that for all other authorization specifications t, $\hat{T} \cap t = t$, and $\hat{T} \cup t = \hat{T}$.[3] Sometimes we will write $\xrightarrow{\hat{T}}$ simply as $\longrightarrow$, i.e., a rewrite rule of the form $L \longrightarrow R$ has an implicit label of $\hat{T}$. We will treat certs as labeled rewrite rules:

- A name cert $(K, \mathtt{A}, S, V)$ will be written as a labeled rewrite rule $K\ \mathtt{A} \longrightarrow S$.
- An auth cert (K, S, D, T, V) will be written as $K\ \square \xrightarrow{T} S\ \square$ if the delegation bit D is turned on; otherwise, it will be written as $K\ \square \xrightarrow{T} S\ \blacksquare$.

Note that in authorization problems, we only consider valid certificates, so, as a pre-processing step, we first check the validity specification V for each certificate in use. For the rest of the paper, we assume that only valid certificates are considered for authorization proofs.

Because we only use labeled rewrite rules in this paper, we refer to them as rewrite rules or simply rules. A term S appearing in a rule can be viewed as a string over the alphabet $\mathcal{K} \cup \mathcal{A}$, in which elements of $\mathcal{K}$ appear only at the beginning. For uniformity, we also refer to strings of the form $S\ \square$ and $S\ \blacksquare$ as terms. Assume that we are given a labeled rewrite rule $L \xrightarrow{T} R$ that corresponds to an auth cert. Consider a term $S = LX$. In this case, the labeled rewrite rule $L \xrightarrow{T} R$ applied to the term S (denoted by $(L \xrightarrow{T} R)(S)$) yields the term RX. Therefore, a rule can be viewed as a function from terms to terms that rewrites the left prefix of its argument, for example,

$$(K_A\ \mathtt{Bob} \longrightarrow K_B)(K_A\ \mathtt{Bob}\ \mathtt{myFriends}) = K_B\ \mathtt{myFriends}.$$

Consider two rules $c_1 = (L_1 \xrightarrow{T} R_1)$ and $c_2 = (L_2 \xrightarrow{T'} R_2)$, and, in addition, assume that L_2 is a prefix of R_1, i.e., there exists an X such that $R_1 = L_2X$. Then the *composition* $c_2 \circ c_1$ is the rule $L_1 \xrightarrow{T \cap T'} R_2X$. For example, consider the two rules:

$$\begin{aligned} c_1 &: \ K_A\ \mathtt{friends} \xrightarrow{T} K_A\ \mathtt{Bob}\ \mathtt{myFriends} \\ c_2 &: \ K_A\ \mathtt{Bob} \xrightarrow{T'} K_B \end{aligned}$$

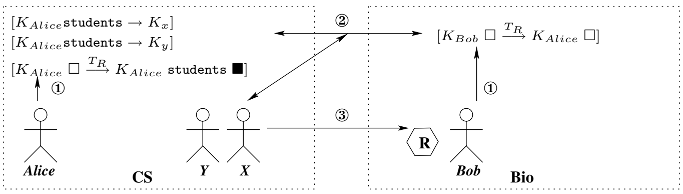
[3] The issue of intersection and union of authorization specifications is discussed in [9,11].

The composition $c_2 \circ c_1$ is K_A `friends` $\xrightarrow{T \cap T'}$ K_B `myFriends`. Two rules c_1 and c_2 are called *compatible* if their composition $c_2 \circ c_1$ is well defined.[4]

A *certificate chain ch* is a sequence of certificates $[c_1, c_2, \cdots, c_k]$. The label of a certificate chain $ch = [c_1, \cdots, c_k]$, denoted by $L(ch)$, is the label obtained from $c_k \circ c_{k-1} \cdots \circ c_1$.

3 Kerberizing SPKI/SDSI

In this section, we explain how we can reduce the dependence of SPKI/SDSI on PKI by utilizing a secret-key-based authentication system, namely *Kerberos*. We first introduce an example that will be used throughout this section. Next, we use this example to illustrate how the original SPKI/SDSI system works. Finally, in Section 3.2, we describe how the reliance of SPKI/SDSI on PKI can be reduced by using Kerberos. We assume that the reader is familiar with Kerberos (for a detailed description of Kerberos see [19]).



$[K_{Alice}$ students $\rightarrow K_x]$
$[K_{Alice}$ students $\rightarrow K_y]$
$[K_{Alice}\ \square \xrightarrow{T_R} K_{Alice}$ students $\blacksquare]$
①
②
$[K_{Bob}\ \square \xrightarrow{T_R} K_{Alice}\ \square]$
①
③
R
Alice
CS
Y
X
Bob
Bio

Fig. 2. Distributed authorization using SPKI/SDSI

Example. Suppose that there are two sites, `Bio` and `CS`, which correspond to the biology and the computer science departments, respectively. Two professors, *Alice* from `CS` and *Bob* from `Bio`, are collaborating on a project. *Bob* wants to delegate to *Alice* full access rights to a shared resource R. In addition, *Alice* plans to delegate access rights to resource R to her students, who are also involved in the project, without allowing them to delegate these rights further.

3.1 Authorization in SPKI/SDSI

In this section, we describe how SPKI/SDSI authorization works in a distributed environment, using the example given above. There are three components to a SPKI/SDSI authorization scenario, denoted by the circled numbers in Figure 2.

Certificate issuance (Figure 2 $\xrightarrow{①}$). First, each user issues auth and name certs. In our example, *Bob* delegates access rights T_R to resource R to *Alice* by issuing the following auth cert, signed with his private key:

[4] In general, the composition operator $\circ$ is not associative. However, when $(c_3 \circ c_2) \circ c_1$ exists, so does $c_3 \circ (c_2 \circ c_1)$; moreover, the expressions are equal when both are defined. Thus, we allow ourselves to omit parentheses and assume that $\circ$ is right associative.

$$\boxed{K_{Bob}\,\Box \xrightarrow{T_R} K_{Alice}\,\Box}$$

At CS, *Alice* grants two of her students, X and Y, access to R by issuing the following two name certs and one auth cert, all signed with *Alice*'s private key:

$$\boxed{\begin{array}{l} K_{Alice}\ \texttt{students} \longrightarrow K_X \\ K_{Alice}\ \texttt{students} \longrightarrow K_Y \\ K_{Alice}\,\Box \xrightarrow{T_R} K_{Alice}\ \texttt{students}\ \blacksquare \end{array}}$$

The two name certs state that X and Y are students of *Alice*; the auth cert states that all of her students (i.e., X and Y) can access resource R with authorization specification T_R, but they cannot delegate the access right.

Now assume that student X wants to access R at site *Bio* according to authorization specification T_R. He needs to perform the following two steps:

Certificate-chain discovery (Figure 2 $\overset{②}{\rightarrow}$). To request access to resource R, a user U first performs certificate-chain discovery to obtain a proof that he can access resource R. This can be achieved by executing a distributed certificate-chain-discovery algorithm [14], and, if the algorithm finds that U is authorized, it returns a proof in the form of a finite set of certificate chains $\{ch_1, \cdots, ch_m\}$. In our example, student X initiates the distributed certificate-chain discovery, which will involve both *Alice* and *Bob*. The distributed certificate-chain discovery returns the singleton set of chains $\{ch_1\}$, where $ch_1 = [c_1, c_2, c_3]$ and c_i are the following certificates:

$$\begin{aligned} c_3 &= K_{Bob}\,\Box \xrightarrow{T_R} K_{Alice}\,\Box \\ c_2 &= K_{Alice}\,\Box \xrightarrow{T_R} K_{Alice}\ \texttt{students}\ \blacksquare \\ c_1 &= K_{Alice}\ \texttt{students} \longrightarrow K_X \end{aligned}$$

Requesting a resource (Figure 2 $\overset{③}{\rightarrow}$). After user U obtains a set of certificate chains $SCH = \{ch_1, \cdots, ch_m\}$ from the previous step, he presents SCH to the owner of the resource R to which T_R refers. The owner authorizes K_U iff $T_R \subseteq \bigcup_{i=1}^{m} L(ch_i)$ (this step is usually called *compliance checking*).

In our example, after making the certificate-chain-discovery request, "Does $K_{Bob}\,\Box$ resolve to $K_X\Box$ or $K_X\blacksquare$ with authorization specification T_R?", student X presents $\{ch_1\}$ to K_{Bob}. K_{Bob} checks that $T_R \subseteq L(ch_1)$, which is true, and hence grants X access to resource R.

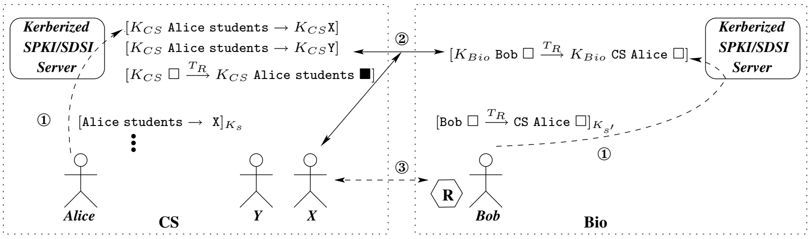
3.2 Authorization in Kerberized SPKI/SDSI

Notice that, to use SPKI/SDSI, *every user* needs to have a public-private key pair. In this section, we describe how to reduce SPKI/SDSI's dependence on PKI by using the distributed authentication system *Kerberos* in a SPKI/SDSI implementation. The key insight behind our work is that *the certificate issuance process in SPKI/SDSI can also be achieved using secret-key-based systems, such as Kerberos.* In SPKI/SDSI, each certificate is signed by its issuer using the private key, and the signature serves as the proof for the authenticity of the certificate. In a secret-key-based system such as Kerberos, the authentication process also produces the evidence for who the user is, and this evidence can be employed by the user to issue certificates. In Kerberos, an authenticated user obtains a token called *Ticket Granting Ticket (TGT)*, which contains digital evidence about the user. This token can be used to obtain a secure communication channel with various Kerberos services. Therefore, in our approach, we use a *Kerberized* SPKI/SDSI server for each site so that an authenticated Kerberos user can securely issue certificate requests through the Kerberized SPKI/SDSI server. In essence, our approach relaxes SPKI/SDSI's binding requirement where each user is identified by its public key. Instead, each principal will be a Kerberos principal in a Kerberos realm. Consequently, with our approach, a SPKI/SDSI user no longer needs to have a public-private key pair, and we only require one public-private key pair *per site*—namely, for the SPKI/SDSI server. Our new system is called *K-SPKI/SDSI*, short for *Kerberized SPKI/SDSI.*

In our system, each SPKI/SDSI site runs a Kerberized SPKI/SDSI server (K-SPKI/SDSI), which shares a public-private key pair with the Kerberos *Key Distribution Center (KDC)* at its site. The public-private key of site *st* is denoted by K_{st}. We now describe the three components of our authorization scenario in this new setting. Figure 3 illustrates the high-level idea behind our approach.

Certificate issuance (Figure 3 $\overset{①}{\rightarrow}$). To issue K-SPKI/SDSI certificates, a Kerberos user first authenticates with the local KDC using the standard Kerberos authentication protocol and receives a *Ticket Granting Ticket (TGT)* from the KDC. Using the TGT, the user requests a Service Granting Ticket (SGT) for accessing the Kerberized SPKI/SDSI (K-SPKI/SDSI) server. Throughout the rest of the section, we assume that the user has obtained an SGT for the K-SPKI/SDSI server at its site. Using the SGT, the user issues requests for generating SPKI/SDSI name certs or auth certs. The session key K_s provided in the SGT is used to protect both the integrity and confidentiality of the requests sent over the communication channel.

To issue an auth cert, a user at site *st* sends a cert request $E_{K_s}[U, S, D, T, V]$, encrypted with the session key K_s from the SGT, to the K-SPKI/SDSI server. Here U is the name of the user, S is the subject, D is the delegation bit, T is the authorization specification, and V is validity information. Upon receiving this encrypted auth-cert request, the K-SPKI/SDSI server ascertains its validity, and if the auth cert is valid, it creates a new K-SPKI/SDSI auth cert of the form $[K_{st}\ U, K_{st}\ S, D, T, V]$, signs it with its private key. The newly issued certificates can be stored in the K-SPKI/SDSI server so that authorization can be done more efficiently. However, to fully emulate SPKI/SDSI, the certs could also be sent back to the users who have requested them. In this scenario, authorization would be carried out exactly as in the original SPKI/SDSI. Notice that in the new auth cert the public key K_{st} of site *st* is added before both U and S. In our example, *Bob* sends the following auth-cert request, encrypted with the appropriate session key (obtained from his SGT), to the K-SPKI/SDSI server S_{Bio}:



Kerberized SPKI/SDSI Server
$[K_{CS}$ Alice students $\rightarrow K_{CS}$X]
$[K_{CS}$ Alice students $\rightarrow K_{CS}$Y]
$[K_{CS}\,\square \xrightarrow{T_R} K_{CS}$ Alice students $\blacksquare]$
②
$[K_{Bio}$ Bob $\square \xrightarrow{T_R} K_{Bio}$ CS Alice $\square]$
Kerberized SPKI/SDSI Server
①
[Alice students $\rightarrow$ X$]_{K_s}$
[Bob $\square \xrightarrow{T_R}$ CS Alice $\square]_{K_{s'}}$
①
③
Alice
CS
Y
X
R
Bob
Bio

Fig. 3. Reducing SPKI/SDSI's dependence on PKI using Kerberos. Dashed lines represent secure Kerberos communication channels.

$$\boxed{\texttt{Bob}\ \square \xrightarrow{T_R} \texttt{CS Alice}\ \square}$$

The auth cert states that *Bob* delegates full access rights to resource R to *Alice* from CS. The K-SPKI/SDSI server S_{Bio} verifies the encrypted auth certs shown above, then creates and signs the following K-SPKI/SDSI auth cert:

$$\boxed{K_{\texttt{Bio}}\ \texttt{Bob}\ \square \xrightarrow{T_R} K_{\texttt{Bio}}\ \texttt{CS Alice}\ \square}$$

To issue a name cert, a user at site *st* sends an encrypted name-cert request $E_{K_s}[U, \texttt{A}, S, V]$ to the K-SPKI/SDSI server. Here U, S, and V are the same as in an auth cert, while A is an identifier. The validation step is exactly the same as that for issuing auth certs. After a request is validated, the K-SPKI/SDSI server creates a new name cert of the form $[K_{st}\ U, \texttt{A}, K_{st}\ S, V]$, signs it with its private key. Similar to auth certs, the name certs can be either stored in the K-SPKI/SDSI server or sent back to the users who have requested them. As before, we will write the name cert as $U\ \texttt{A} \longrightarrow S$. In our example, *Alice* sends two name certs and one auth cert (Figure 3.2 (a)), encrypted with the session key K_s, to the K-SPKI/SDSI server at her site. The K-SPKI/SDSI server verifies the encrypted name certs, creates the corresponding K-SPKI/SDSI name certs, and signs them (Figure 3.2 (b)). Notice that the left-hand sides of K-SPKI/SDSI certificates have three symbols: the left-hand side of an extended auth cert is of the form $K_\alpha\ \texttt{U}\ \square$ or $K_\alpha\ \texttt{U}\ \blacksquare$, where K_α is the public key of site α and U is a user; the left-hand side of an extended name cert is of the form $K_\alpha\ \texttt{U}\ \texttt{A}$, where both U and A are identifiers. In SPKI/SDSI the left-hand sides of auth and name certs have just two symbols. However, the translation from user certificate requests to the actual certificate can be done automatically because this is just a special case of left-prefix rewriting, and the primitives generalize to arbitrary left-prefix rewriting systems [5], which covers the case of K-SPKI/SDSI certs with three left-hand-side symbols.

Besides user-issued certificates, each SPKI/SDSI site also needs to exchange its public key with other SPKI/SDSI sites, represented as name certificates. For example, the site Bio would issue the following name certificate:

$$\boxed{K_{\texttt{Bio}}\ \texttt{CS} \longrightarrow K_{\texttt{CS}}}$$

$$\boxed{\begin{array}{l}\texttt{Alice students} \longrightarrow \texttt{X}\\ \texttt{Alice students} \longrightarrow \texttt{Y}\\ \texttt{Alice}\ \square \xrightarrow{T_R} \texttt{CS Alice students}\ \blacksquare\end{array}} \longrightarrow \boxed{\begin{array}{l}K_{\texttt{CS}}\ \texttt{Alice students} \longrightarrow K_{\texttt{CS}}\ \texttt{X}\\ K_{\texttt{CS}}\ \texttt{Alice students} \longrightarrow K_{\texttt{CS}}\ \texttt{Y}\\ K_{\texttt{CS}}\ \texttt{Alice}\ \square \xrightarrow{T_R} K_{\texttt{CS}}\ \texttt{Alice students}\ \blacksquare\end{array}}$$

(a) Name-cert requests *(b) K-SPKI/SDSI name certs*

Fig. 4. Issuing SPKI/SDSI certificates using K-SPKI/SDSI

Certificate Chain Discovery (Figure 3 $\overset{②}{\rightarrow}$). Suppose that user U at site st_1 wishes to access resource R at site st_2 with access rights given by T. U first initiates a certificate-chain discovery, e.g., using the algorithms from [20] or, in the case where certificates are stored at the K-SPKI/SDSI servers, using the distributed algorithm from [14]. If the search is successful and returns a set of certificate chains SCH, U needs its site st_1 to prepare a proof of authorization that U can present to the owner of R. This is because, while SCH proves that user U at site st_1 has access to R, only st_1 can assure the owner of R (who possibly resides at a different site st_2) that the requesting user is indeed U. Thus, U sends SCH to st_1, and st_1 sends back the following Kerberos tokens:

$$\begin{aligned} Token_U &= E_{K_s}(K_1)\,Ticket_U \\ Ticket_U &= E_{K_{st_2}}(K_2)\ E_{K_2}[st_1, (R, st_2, U, st_1, T, SCH, K_1, TS_1, Lifetime_1)_{K_{st_1}}] \end{aligned}$$

where K_s is the session key for U and st_1, K_1 and K_2 are fresh secret keys generated by st_1, and $(\,\cdot\,)_{K_{st_1}}$ denotes data signed by st_1. Intuitively, $Token_U$ makes the key K_1 known to U, and $Ticket_U$ says that SCH is a proof of authorization for access to R at site st_2 (with access type T) by user U at site st_1. By signing the message, site st_1 confirms that anybody in possession of key K_1 is indeed U. Notice that R, st_2, U, and st_1 are implicitly contained in SCH and can be omitted in practice. In our example, assume that student X receives a token with the set of certificate chains SCH $= \{ch_1\}$, where ch_1 is the certificate chain $[c_1, c_2, c_3, c_4]$: Notice that $(c_4 \circ c_3 \circ c_2 \circ c_1)(K_{Bio}\texttt{Bob}\ \square) \in \{K_{CS}\ \texttt{X}\ \square, K_{CS}\ \texttt{X}\ \blacksquare\}$ and $T_R \subseteq L([c_1, c_2, c_3, c_4])$.

$$\begin{array}{ll} c_1 = K_{Bio}\ \texttt{Bob}\ \square \xrightarrow{T_R} K_{Bio}\ \texttt{CS Alice}\ \square & c_3 = K_{CS}\ \texttt{Alice}\square \longrightarrow K_{CS}\ \texttt{Alice students}\ \blacksquare \\ c_2 = K_{Bio}\ \texttt{CS} \longrightarrow K_{CS} & c_4 = K_{CS}\ \texttt{Alice students} \longrightarrow K_{CS}\ \texttt{X} \end{array}$$

Requesting a resource (Figure 3 $\overset{③}{\rightarrow}$). Upon receiving $Token_U$, user U decrypts $E_{K_s}(K_1)$ and retrieves the session key K_1. He then constructs the following Kerberos authenticator:[5]

$$Authenticator_U = E_{K_1}[\,U, st_1, TS_2, Lifetime_2\,]$$

User U sends the message $[Ticket_U\ Authenticator_U]$ to the owner of resource R (at site st_2), who requests its local K-SPKI/SDSI server st_2 to verify the message. The server performs the following steps:

[5] The actual content of the authenticator is irrelevant in our example.

- Decrypts the message $E_{K_{st_2}}(K_2)$ with its private key and retrieves K_2.
- Decrypts the part of $Ticket_U$ encrypted with K_2.
- Obtains the public key K_{st_1} and verifies the signature of st_1.
- Ascertains freshness and validity of the token using the time-stamp TS_1 and the lifetime $Lifetime_1$; checks that SCH indeed proves the desired access.
- Similarly, the K-SPKI/SDSI server ascertains the validity of the authenticator, thus making sure that the sender is indeed user U at st_1. Notice that the server knows the session key K_1 from $Ticket_U$.

If all the steps given above are successful, then the K-SPKI/SDSI server sends a message to R indicating that U should be granted access, and that communication between R and U should be protected using key K_1.

3.3 Analysis

Correctness of the Protocol. The key idea behind our work is to rely on Kerberos to provide a secure channel for users to submit SPKI/SDSI name certs and auth certs, which are signed and stored at each site. In contrast, in the original SPKI/SDSI system, each user can issue and sign her own certs. We note that there is no conceptual difference between these two approaches; only the underlying security mechanisms used are different (one uses secret-key cryptography and the other uses public-key cryptography). For example, in the original SPKI/SDSI approach, a user U issues a name cert this way: $(K, \mathtt{A}, S, V)_{K'}$. Here the subscript K' denotes that the name cert is signed by the user using the private key K'. In comparison, the corresponding step in our approach is implemented by issuing the certificate request $E_K[U, \mathtt{A}, S, V]$, where E_K denotes that the name-cert request is encrypted by the session key shared between the user and the K-SPKI/SDSI server. The request is first validated, then translated by the K-SPKI/SDSI server into actual certificates, signed with K-SPKI/SDSI server's private key. Thus, in both cases, the possession of a secret (K' in SPKI/SDSI, and K in Kerberos) provides the digital links between the certs issued and the user who has issued them.

Trade Offs. Although previous work has shown that secret keys can be used in place of public keys to implement the same security objectives, such as building a secure broadcast-communication channel [8,16], there are pros and cons with each approach. A secret-key-based system is simpler to set up and use. However, secret-key-based systems often require both communication parties to be online to function properly. For example, sending a message between two Kerberos users usually requires both the sender and receiver to be active at the same time so that they can exchange a secret encryption key.[6] On the other hand, public-key-based systems can operate in offline mode. For example, to send a message using PKI, the sender can simply encrypt the message using the recipient's public key, without contacting the recipient first. However, key management is a major

[6] Unless the two sides have previously agreed upon a shared secret key.

issue that has hindered wider acceptance of PKI-based systems because it is much more cumbersome to maintain public-private key pairs.

Our approach eliminates the public-private key pairs for individual users. Instead, each K-SPKI/SDSI server has a dedicated public-private key pair that is used for signing SPKI/SDSI certificates, whereas users use secret keys in their communication with the server (e.g., for requesting certificates). We chose this hybrid approach for the following reasons. First, the use of PKI in the authorization mechanism of SPKI/SDSI lends itself to offline checking of certificate chains. Indeed, when a user requesting access to a resource presents a set of certificate chains to the owner of that resource, the architecture of SPKI/SDSI ensures that the owner can check the validity of these chains without contacting the owners of the keys involved in the chains (in fact, without even verifying their identities). While it is worth noting that by adopting the ideas of Lampson *et al.* [16] or Davis and Swick [8] it is possible to emulate SPKI/SDSI using secret keys only, such a scheme would not allow offline checking of certificates. Secondly, the communication between users and servers usually happens online, which motivates the use of secret keys in this context. Finally, if the certificates issued by users are stored in the SPKI/SDSI servers, our approach can be very well combined with the distributed certificate-chain algorithm presented in [14].

Threat Analysis. Our message exchange for requesting a resource is very similar to the exchange of messages between the client and KDC in Kerberos. In essence, the ticket $Ticket_U$ states that "anyone who uses K_1 is U". Since in $Token_U$ K_1 is encrypted with K_s, which can only be known by the user U (because K_s is in the SGT issued to U), only U could have known K_1 (assuming that authentication in Kerberos is correct). It is possible for an adversary to replay the message $[Ticket_U\ Authenticator_U]$ to the resource R and masquerade as U. However, this attack fails if R and U communicate using K_1, which is unknown to the attacker.

3.4 Extension to Other PKI-Based Trust-Management Systems

Different trust-management systems have different logics to express security policies. Most of the components (auth and name certs in SPKI/SDSI) of these security policies are signed by principals using their private keys. Recall that our protocol essentially allows a server to sign statements on behalf of an authenticated user. Although we have explained our protocol for SPKI/SDSI, it is clear that it can be used for other trust-management systems. We now demonstrate how our protocol can be extended for the trust-management system KeyNote [2]. Figure 5 shows an example of a KeyNote credential that grants some rights (`RFC822-EMAIL`) to *Alice*, who has the public key `DSA:4401ff92` (Line 2). We can achieve the same goal using our technique, as shown in Figure 6. In our approach, the credential states that if *Alice* is an authenticated Kerberos user with the Kerberos identity `alice@LABS.COM`, then she can have the rights specified in the credential. It must be noted that the two credentials, although they appear similar, have very different operational semantics. In the original KeyNote

```
KeyNote-Version: 2
Local-Constants: Alice="DSA:4401ff92"
Authorizer: "RSA:abc123"
Licensees: Alice
Conditions: (app_domain == "RFC822-EMAIL") && (address ~= ".*@labs\\.com\$");
Signature: "RSA-SHA1:213354f9"
```

Fig. 5. An example of a KeyNote credential. Line 2 represents *Alice*'s public key.

```
KeyNote-Version: 2
Local-Constants: Alice="Kerberos:alice@LABS.COM"
Authorizer: "RSA:abc123"
Licensees: Alice
Conditions: (app_domain == "RFC822-EMAIL") && (address ~= ".*@labs\\.com\$");
Signature: "RSA-SHA1:213354f9"
```

Fig. 6. An example of the same KeyNote credential, but without requiring *Alice* to have a public key

example, *Alice* can further delegate the rights she has received by issuing new credentials directly—without any compliance checking. However, in the Kerberized scenario, because *Alice* no longer has a public-private key pair, she can only delegate her rights by first authenticating herself through Kerberos, and then issue a delegation request through a dedicated Kerberized KeyNote server. In both cases, the *Authorizer* (Line 3) represents the public-private key for the Kerberized KeyNote server.

4 Implementation and Evaluation

We have built a prototype system to evaluate our approach. The implementation uses MIT's *Kerberos* distribution (version 1.3.1 [19]) and the *Distributed SPKI/SDSI* library, which is based on a model checker for weighted pushdown systems [20]. The test environment contains 1500 name certs and 30 auth certs, distributed over different sites. Each site runs on a dedicated machine on a local area network. All test machines have identical configurations: 800 MHz Pentium III with 256 MB RAM, running TAO Linux version 1.0.

We evaluated our approach using two criteria: *ease of deployment* and *performance*. Because our implementation is still a prototype, and we have not deployed the system in a real-world environment, we evaluated the prototype in a simulated environment, using synthetic data. We summarize the results based on these two criteria:

Ease of deployment: Three steps are required to deploy our system, assuming that Kerberos is already installed.

1. *Install a public-private key pair:* In our approach, only one public-private key pair is needed for each Kerberos site. In addition, sites need to exchange their public keys. However, we believe that this is a reasonable requirement

because the exchange is done only once. Alternatively, public keys could be obtained on demand using existing solutions for public-key exchange.

2. *Install the K-SPKI/SDSI server:* Each Kerberos site must have its own K-SPKI/SDSI server. Because each K-SPKI/SDSI server is implemented as a Kerberos service, this does not require any changes to Kerberos besides setting up the secret key between the KDC and the K-SPKI/SDSI server.
3. *Enhance Kerberos clients:* Kerberos clients that want to take advantage of our distributed authorization features can be updated easily by using a new library call to access the K-SPKI/SDSI server.

Performance: We have tested our implementation in a model where all certificates are stored at the K-SPKI/SDSI servers, which then uses the distributed algorithm from [14] for certificate-chain discovery (for results, see Section 4.2). In these experiments, the performance of distributed authorization is highly dependent on how K-SPKI/SDSI certificates are distributed among the sites: the more distributed the certs are, the more sites are needed to resolve authorization queries, and the longer it takes to process an authorization query. In our study, distributed authorization performed well: in a test environment with about 1,500 certificates and eight Kerberos sites, it took about 1 second to process a complex authorization request, and took half as long to process a simple one. Because this is only a prototype implementation, there is still plenty of opportunity for optimizations that would improve the performance. Notice, however, that this issue is slightly orthogonal to the issue of integrating SPKI/SDSI with Kerberos, since certificate-chain discovery could still be done locally.

4.1 Ease of Deployment

The objective of this work is to make SPKI/SDSI, and potentially other PKI-based trust-management systems, less reliant on PKI and easier to deploy in the real world. We achieve this by two means. SPKI/SDSI's reliance on PKI is reduced by using the authentication provided by existing infrastructures, such as Kerberos, that are proven and in use. The approach tries to make SPKI/SDSI fit into existing systems seamlessly instead of introducing substantial changes that would present an impediment to adoption. Deploying our system in environments where Kerberos is installed only requires a few small changes.

Second, in terms of implementation, we tried to minimize the changes to Kerberos, because such changes usually result in additional complications for deployment. We achieved this goal by implementing the K-SPKI/SDSI server as an independent unit, instead of changing the KDC. As a result, our implementation requires no changes to the KDC, and only one minor modification to the Kerberos library.[7]

Our approach also has some drawbacks. First, by using a separate server, clients must be modified to use the provided features—although the change is

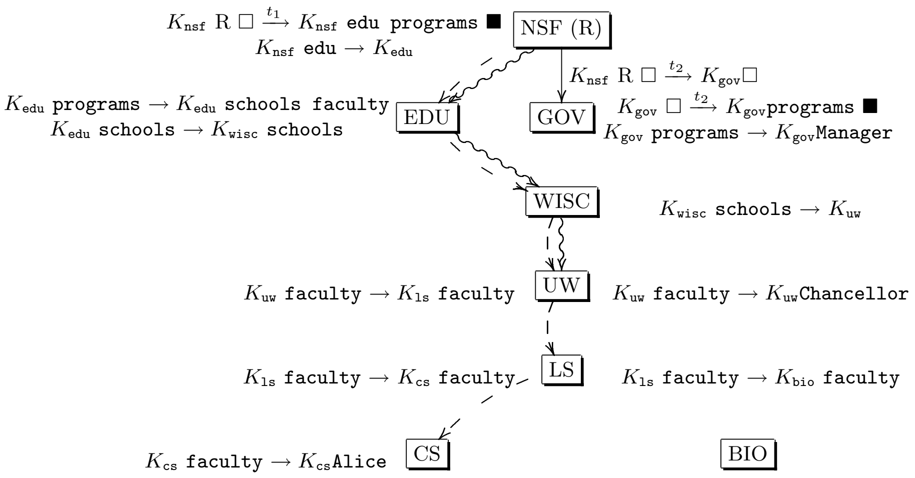
[7] We changed the function `kuserok`, which, when called, evaluates whether a Kerberos principal is allowed to login to a host. Our change provides an option for callers of this function to use the K-SPKI/SDSI server to check for authorization.

very simple. The alternative is to provide these functionalities inside the KDC. When a Kerberos client requests an SGT for a service, the KDC automatically performs the necessary authorization query on behalf of the client and stores the authorization token as part of the SGT. This approach makes the authorization process transparent to the clients, but it does require changes to the KDC. This technique is also used by others for adding authorization support inside Kerberos [4,10,22,15]. We are currently evaluating both approaches.

In addition to the changes above, when deploying our system, each site must install a public-private key pair. Furthermore, each site needs to send its public key to other sites with which it plans to collaborate. However, we believe that this is a reasonable requirement because setting up a collaboration is an administrative task that only needs to be done once for each collaborating site.

4.2 Performance

We also evaluated the performance of our system in a simulated distributed environment using the algorithm from [14]. We only considered the performance for distributed authorization because issuing certificates is an infrequent administrative task. The simulated test environment consisted of eight Kerberos sites, as shown in Figure 7. Each node in the graph represents a Kerberos site; nodes with a symbol `R` represent a service that Kerberos users can access. To illustrate what goes on, some of the certificates used in the experiments are shown next to each site. Because in a distributed environment every Kerberos site stores its own certificates, resolving an authorization request may involve multiple sites, depending on how the K-SPKI/SDSI certificates are distributed. For instance, in Figure 7 when `Manager` from the site `GOV` attempts to access resource `R` from `NSF`,



$K_{\texttt{nsf}}$ R $\square \xrightarrow{t_1}$ $K_{\texttt{nsf}}$ edu programs ■
$K_{\texttt{nsf}}$ edu → $K_{\texttt{edu}}$
NSF (R)
$K_{\texttt{nsf}}$ R $\square \xrightarrow{t_2}$ $K_{\texttt{gov}}\square$
$K_{\texttt{edu}}$ programs → $K_{\texttt{edu}}$ schools faculty
$K_{\texttt{edu}}$ schools → $K_{\texttt{wisc}}$ schools
EDU
GOV
$K_{\texttt{gov}}$ $\square \xrightarrow{t_2}$ $K_{\texttt{gov}}$programs ■
$K_{\texttt{gov}}$ programs → $K_{\texttt{gov}}$Manager
WISC
$K_{\texttt{wisc}}$ schools → $K_{\texttt{uw}}$
$K_{\texttt{uw}}$ faculty → $K_{\texttt{ls}}$ faculty
UW
$K_{\texttt{uw}}$ faculty → $K_{\texttt{uw}}$Chancellor
$K_{\texttt{ls}}$ faculty → $K_{\texttt{cs}}$ faculty
LS
$K_{\texttt{ls}}$ faculty → $K_{\texttt{bio}}$ faculty
$K_{\texttt{cs}}$ faculty → $K_{\texttt{cs}}$Alice
CS
BIO

Fig. 7. Test setup with 1500 name certs and 30 auth certs (only a few are shown due to space constraints)

Table 1. Distributed Authorization Performance Results

Scenario	# of sites	Request	Time (ms)
Manager@GOV ($\rightarrow$)	2	`(fundB apply)`	581
Chancellor@UW ($\rightsquigarrow$)	4	`(fundA apply)`	930
Alice@CS ($\dashrightarrow$)	6	`(fundA apply)`	1128

only these two sites are involved in distributed authorization, as indicated by the solid arrow. In contrast, when `Alice`, from `CS`, wants to access the same resource `R`, multiple sites (along the dashed arrows) must participate in the distributed authorization. Therefore, we expect the number of sites involved in distributed authorization to be an important factor in performance. For this reason, we tested distributed authorization using three different scenarios, indicated by the three types of arrows in Figure 7.

Table 1 shows the results of the experiments. As expected, the number of sites involved in distributed authorization has a direct impact on the performance of the system. In the most complex case (`Alice@CS`), where six Kerberos sites were involved, resolving an authorization request took almost twice as long as the time required in the simplest case (`Manager@GOV`), where only two sites were involved. However, as this is only a prototype, we expect to improve the performance in the future by optimizing the code. Furthermore, our test setup is an extreme case where every Kerberos site has its own physical KDC. In practice, different logical Kerberos sites could share a single physical KDC, which would improve the performance by reducing the communication overhead.

5 Related Work

The key idea behind our approach is that we can use secret-key cryptography to implement the same SPKI/SDSI operations (e.g., issuing certificates) with the same level of security as with public-key cryptography. The notion of using secret keys in place of public-private key pairs as the building block of security operations was first proposed by Lampson *et al.* [16], who showed that, by using a *relay* (an agent that everyone trusts, e.g. the Kerberos KDC), one can build public-key-style secure communication channels. This idea has been extended by Davis and Swick to build other public-key-style security protocols using secret keys [8]. Our work also uses this idea, but applies it in the context of PKI-based trust-management systems, specifically SPKI/SDSI.

Leveraging the advantages of both Kerberos and Public-Key Infrastructure (PKI) has been explored before. *PKINIT* [23], *PKCROSS* [12], and *PKDA* [21] all extend Kerberos by using public-key cryptography for authentication purposes. Our work has a different goal: it is targeted toward authorization rather than authentication; in particular, the goal is *to use Kerberos to reduce the dependence of SPKI/SDSI on PKI.* Furthermore, the approaches of [23,12,21] require modifications to the Kerberos infrastructure itself, while our approach does not.

K-PKI [6,15] addresses the problem of accessing Kerberos services from PKI-based systems, such as web applications. K-PKI provides a special Kerberosserver, KCA, that can generate short-term X.509 certificates for authenticated Kerberos clients. Later on, when a client tries to access Kerberos services through some web applications, (s)he first authenticates with the web services using the generated certificate. The web services, in turn, can obtain necessary Kerberos credentials and access the Kerberos services on behalf of the client. While K-PKI provides a glue between Kerberos and the PKI world, the complexity of the PKI systems is not reduced: all clients are required to manage public-private key pairs. Our work, on the other hand, tries to reduce the reliance of trust-management systems on PKI so that individual users no longer need to have public-private key pairs.

Another aspect of our work is to bring trust management, such as SPKI/SDSI, to Kerberos-based infrastructures. Although there has been previous work on extending Kerberos' authentication framework with authorization services, that work generally assumes a centralized authority and does not address cross-realm authorization. Of these, Neuman's work on *restricted proxy* [18] is the closest to ours. Restricted proxy is a model for building various authorization services such as authorization servers, capabilities, and access control. However, SPKI/SDSI is a superset of restricted proxy, and offers other features, such as distributed trust management. DCE's *Privilege Service (PS)* [22], ECMA's *SESAME* [10], and Microsoft's Kerberos extension [4] provide authorization capability through the use of an optional field (called *authorization data*) provided by Kerberos. For each authenticated Kerberos principal, authorization information (such as group membership, security identifiers) about the principal is stored in the field. This authorization data is used by application servers to check users' access privileges. These systems have the common drawback that, unlike SPKI/SDSI, they rely on a centralized authority for granting access privileges. In contrast, our approach uses SPKI/SDSI, which does not require a central authority, and authorization decisions can be made in a truly decentralized manner [14].

SPKI/SDSI [9], based on public-key infrastructure, was designed to address the *centralized authority* issue of conventional PKI-based systems. SPKI/SDSI provides a novel framework for managing trust (in the form of certificates) using a decentralized approach. In SPKI/SDSI, no central authority is needed because each principal can issue her own certificates. Much of the previous work on SPKI/SDSI focuses on theoretical aspects of SPKI/SDSI [1,7,13,20]. Despite such work, SPKI/SDSI has not been adopted in the real world, primarily due to the difficulty of key-management issues in PKI-based systems. Our work addresses this problem by reducing SPKI/SDSI's reliance on PKI—by making use of Kerberos, essentially unchanged. By relying on Kerberos, a well-accepted and widely used system, our approach should make it possible for SPKI/SDSI to be adopted in the real world more easily.

References

1. L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access-control systems. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 81–95, May 2005.
2. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote trust-management system version 2. RFC 2704, Sept. 1999.

3. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, pages 185–210, 1999.
4. J. Brezak. Utilizing the Windows 2000 authorization data in Kerberos tickets for access control to resources. http://msdn.microsoft.com/library/default.asp?myurl=/library/enus/dnkerb/html/MSDN_PAC.asp.
5. D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, 1992.
6. CITI: Projects. Kerberos leveraged PKI. http://www.citi.umich.edu/projects/kerb_pki/.
7. D. Clarke, J.-E. Elien, C. M. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certficate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(1/2):285–322, 2001.
8. D. Davis and R. Swick. Network security via private-key certificates. In *Proceedings of the 3rd USENIX Security Symposium*, September 1992.
9. C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylönen. *RFC 2693: SPKI Certificate Theory*. The Internet Society, September 1999.
10. European Computer Manufacturers Association (ECMA). Secure European system for applications in a multi-vendor environment (SESAME). https://www.cosic.esat.kuleuven.ac.be/sesame/html/sesame_documents.html.
11. J. Howell and D. Kotz. A formal semantics for SPKI. Technical Report 2000-363, Department of Computer Science, Dartmouth College, Hanover, NH, Mar. 2000.
12. M. Hur, B. Tung, T. Ryutov, C. Neuman, A. Medvinsky, G. Tsudik, and B. Sommerfeld. Public key cryptography for cross-realm authentication in Kerberos, Nov. 2001. Internet-Draft, draft-ieft-cat-kerberos-pk-cross-08.txt.
13. S. Jha and T. Reps. Model checking SPKI/SDSI. *Journal of Computer Security*, 12(3–4):317–353, 2004.
14. S. Jha, S. Schwoon, H. Wang, and T. Reps. Weighted pushdown systems and trust-management systems. In *TACAS*, 2006.
15. O. Kornievskaia, P. Honeyman, B. Doster, and K. Coffman. Kerberized credential translation: A solution to web access control. In *10th USENIX Security Symposium*, pages 235–250, 2001.
16. B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.
17. J. Linn and M. Branchaud. An examination of assorted PKI issues and proposed alternatives. In *Proceedings of the 3rd Annual PKI R&D Workshop*, April 2004.
18. B. C. Neuman. Proxy-based authorization and accounting for distributed systems. In *ICDCS*, pages 283–291, 1993.
19. B. C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, September 1994.
20. S. Schwoon, S. Jha, T. Reps, and S. Stubblebine. On generalized authorization problems. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 202–218. IEEE Computer Society, June 2003.
21. M. Sirbu and J. Chuang. Distributed authentication in Kerberos using public key cryptography, Feb. 1997.
22. The Open Group. DCE 1.1: Authentication and security services. http://www.opengroup.org/onlinepubs/9668899/.
23. B. Tung, C. Neuman, M. Hur, A. Medivinsky, S. Medvinsky, J. Wray, and J. Trostle. Public key cryptography for initial authentication in Kerberos, 2004. Internet-Draft, draft-ieft-cat-kerberos-pk-init-17.txt.

Delegation in Role-Based Access Control

Jason Crampton and Hemanth Khambhammettu

Information Security Group, Royal Holloway, University of London

Abstract. User delegation is a mechanism for assigning access rights available to a user to another user. A delegation operation can either be a *grant* or *transfer* operation. Delegation for role-based access control models have extensively studied grant delegations. However, transfer delegations for role-based access control have largely been ignored. This is largely because enforcing transfer delegation policies is more complex than grant delegation policies. This paper, primarily, studies transfer delegations for role-based access control models. We also include grant delegations in our model for completeness. We present various mechanisms that authorise delegations in our model. In particular, we show that the use of administrative scope for authorising delegations is more efficient than using relations. We also discuss the enforcement and revocation of delegations. Finally, we compare our work with relevant work in the literature.

1 Introduction

Role-based access control (RBAC) is being increasingly recognized as an efficient access control mechanism that facilitates security administration [1]. *Roles* are identified with various job functions in an organization and users are assigned to roles based on their job responsibilities and qualifications. Permissions are associated with roles. Users acquire permissions through the roles allocated for them. This feature of role-based models greatly simplifies the management of permissions.

Delegation is a mechanism of assigning access rights to a user. Delegation may occur in two forms: *administrative* delegation and *user* delegation. An administrative delegation allows an administrative user to assign access rights to a user and does not, necessarily, require that the administrative user possesses the ability to use the access right. A user delegation allows a user to assign a subset of his available rights to another user. However, a user delegation operation requires that the user performing the delegation must posses the ability to use the access right. Furthermore, like Schaad, we believe that an administrative delegation operation is often long-lived and more durable (permanent) than a user delegation operation that is short-lived (temporary) and intended for a specific purpose [2, Chapter 7, Page 117]. This paper studies user delegation. In the rest of the paper, 'delegation' is always understood to be as 'user delegation' unless stated otherwise. The user who performs a delegation is referred to as a 'delegator' and the user who receives a delegation is referred to as a 'delegatee'.

Rights can be delegated in two ways in RBAC: by delegating *roles* or by delegating individual *permissions*. Delegating a permission p gives the delegatee the ability to use p. However, delegating a role r gives the delegatee the ability to act in role r. That is, the delegatee is authorized for role r (and thereby gains the ability to use permissions assigned to role r and roles that are junior to r). In particular, we note that *individually* delegating all permissions explicitly assigned to a role r does not authorize the delegatee to act in role r.

Broadly, delegation of privileges may be classified into (at least) two kinds: *grant* and *transfer* [3]. A *grant delegation* model, following a successful delegation operation, allows a delegated access right to be available to both the delegator and delegatee. As such, this is a *monotonic* model, in which available authorizations are only increased due to successful delegation operations. However, in *transfer delegation* models, following a successful delegation operation, the ability to use a delegated access right is transferred to the delegatee; in particular, the delegated access right is no longer available to the delegator.

Grant delegation models are, primarily, concerned with allowing the delegatee to use the delegated access right. However, in transfer delegation models, besides allowing the delegatee to use the delegated access right, we must prevent the use of the delegated access right by the delegator. It is this feature that makes it more difficult to enforce transfer delegation policies in most access control frameworks [2, 4]. Furthermore, it can be easily seen that, in grant delegation models the availability of access rights increases monotonically with delegations. While some business requirements may support grant delegations, it is often desirable that sensitive access rights may not be available to a large number of users (at any given time). Such requirements are usually expressed as cardinality constraints in an access control policy [5, 6]. Transfer delegation policies prove to be more useful when an access control policy specifies cardinality limits on the availability of access rights between users.

Consider, for example, an access control policy that requires the co-operation of k users to accomplish a given task. The unavailability of (at least) one of the users, which can be for several reasons, makes it impossible to complete the task. A desirable solution would be for users to be able to delegate the access right to another user, who may act on behalf of the former user. In the above example, when assumed that the access control policy specifies that the right r, that is required to complete the task, is available to no more than k users, the reference monitor will always deny an attempt to delegate the right r, using *grant delegation*, to another user to prevent violation of the policy. Such scenarios require that the delegation be made using *transfer delegation.*

Most works that studied delegation in the context of role-based models are grant delegation models [3, 7, 8, 9, 10, 11, 12, 13]. To our knowledge no work has studied temporary transfer delegation for role-based models. This paper, primarily, aims to study transfer delegation for role-based models. Role hierarchies are an important reason for the wide interest in role-based models. Hence, it is natural to consider role hierarchies when studying any aspect of role-based models. The semantics of transfer delegations become more complex when role-hierarchies are considered. We develop a comprehensive delegation model for role-based systems that provides support for both grant and transfer delegation policies. In this paper, we study delegation in the context of both RBAC$_0$ model (flat roles) and RBAC$_1$ model (hierarchical roles) of the RBAC96 family of models [1].

The rest of the paper is organized as follows. In the next section, we develop the background for the rest of the paper and define semantics for grant and transfer delegations. Section 3 describes the mechanism for authorising delegations. Section 4 presents enforcement of delegations. A comparison of our work with related work in the literature is given in Sect. 5. We conclude this work and discuss future work in Sect. 6.

2 Role-Based Delegation

The theoretical development of role-based access control and its standardization has been strongly influenced by the RBAC96 family of models [1]. For this reason, we consider delegation within the context of the RBAC96 family of models. In the next section, we introduce some important prerequisite concepts, including the relevant features of RBAC96 and the concept of administrative scope, which is the building block of the RHA family of administrative models [14].

2.1 Preliminaries

RBAC96. RBAC$_0$ is the simplest model and defines a set of roles R, a set of permissions P, a set of users U, a permission-role assignment relation $PA \subseteq P \times R$, and a user-role assignment relation $UA \subseteq U \times R$. We denote the set of roles explicitly assigned to a user u by R_u; that is, $R_u = \{r \in R : (u, r) \in UA\}$ and the set of roles explicitly assigned to a permission p by R_p; that is, $R_p = \{r \in R : (p, r) \in PA\}$.

RBAC$_1$ introduces the concept of a *role hierarchy* $RH \subseteq R \times R$. The graph of the relation RH is acyclic and the transitive reflexive closure of RH defines a partial order on R.

We write $r \leqslant r'$ in preference to $(r, r') \in RH^*$ (the transitive reflexive closure of RH). We may also write $r' \geqslant r$ whenever $r \leqslant r'$. We write $\downarrow r$ to denote the set $\{r' \in R : r' \leqslant r\}$ and $\downarrow R'$ to denote $\bigcup_{r' \in R'} \downarrow r'$. We write $\uparrow r$ to denote the set $\{r' \in R : r' \geqslant r\}$ and $\uparrow R'$ to denote $\bigcup_{r' \in R'} \uparrow r'$.

Access control decisions are granted within the context of a *user session*, which is determined by the set of roles that a user activates. This set of roles is a subset of the roles for which the user is authorized directly by the UA relation and indirectly by the role hierarchy. We denote a session for user u by $S_u \subseteq \downarrow R_u$. A user u is permitted to invoke a permission p if there exists an activated role $r \in S_u$ and a role $r' \in R$ such that $(p, r') \in PA$ and $r' \leqslant r$.

Administrative scope. While RBAC96 is widely regarded as the *de facto* standard for role-based access control, there is less consensus regarding role-based administration. There are three approaches that are regarded as being relatively mature and sophisticated [15]: the ARBAC97 model [16], which is the administrative counterpart of RBAC96; the RHA family of models [14]; and the role control center [15]. Since delegation can be regarded as "lightweight user-based administration", it is natural that ideas from role-based administration may be useful in models for delegation. Indeed, the `can_delegate` relation in RDM2000 is very similar in structure and meaning to the `can_assign` relation from ARBAC97 [12]. In this section, we introduce the concept of administrative scope from the RHA model and generalize its definition for use in our delegation model.

Definition 1 (Crampton and Loizou [14]). *Let $r \in R$ and define $\sigma(r) = \{s \leqslant r : \uparrow s \subseteq \downarrow r \cup \uparrow r\}$. We say that $\sigma(r)$ is the* administrative scope *of r.*

In Fig. 1(a), for example, $\sigma(b) = \{d\}$; $g \notin \sigma(b)$, because $e > g$ and e is not comparable to b. Administrative scope defines a sub-hierarchy forming a natural unit of administration for the role r.

The success of any administrative operation in the RHA model is determined by the inclusion (or otherwise) of any role parameters in the requesting role's administrative scope. Hence, a user assigned to role b could add a new role i with parent role d, for example, but could not add a new role k with parent role g.

Administrative scope has been shown to be a far more flexible approach to role-based administration than ARBAC97 [14]. As such, it is unsurprising that it turns out to have considerable advantages in role-based delegation. However, it will be necessary to compute the administrative scope of a role in different partially ordered sets, each of which is a sub-hierarchy of the role hierarchy. Hence, we extend our notion of administrative scope to include two parameters: a partially ordered set X and an element $x \in X$. We write $\sigma(x, X)$ to denote the administrative scope of x computed in partially ordered set X. In practice X will be a subset of R. We now discuss the two most important cases.

- If a user u is assigned to a set of roles R_u, this induces a *user view* $\downarrow R_u$ of the role hierarchy containing all the roles to which u is implicitly assigned (via the user-role and role hierarchy relations). This user view is important in defining which roles a user retains following the delegation of a role. In this case, when u delegates r, we compute $\sigma(r, \downarrow R_u)$ (see Proposition 2).
- Similarly, a set of activated roles $S_u \subseteq \downarrow R_u$ defines a *session view* $\downarrow S_u$, which is useful in defining the roles available to a user following certain types of transfer delegation. In this case, when u delegates r, we compute $\sigma(r, \downarrow S_u)$ (see Proposition 3).

2.2 Delegation Operations

We now describe the characteristics of delegation operations in RBAC96. The grant operation is well understood and has been described in several earlier papers [3, 7, 8, 9, 10, 11, 12, 13]. We include it here for completeness. We define three different types of transfer operations in the context of RBAC_1.

RBAC$_0$.

`grantR`$_0(u, v, r)$**:** The delegator u grants the role r to delegatee v. The delegator may continue to use r.

`grantP`$_0(u, v, p)$**:** The delegator u grants the permission p to delegatee v. The delegator may continue to use p.

`xferR`$_0(u, v, r)$**:** The delegator u transfers the role r to delegatee v. The delegator may no longer use r.

`xferP`$_0(u, v, p)$**:** The delegator u transfers the permission p to delegatee v. The delegator may no longer use p.

RBAC$_1$.

`grantR`$_1(u, v, r)$**:** The delegator u grants the role r to delegatee v. The delegator may continue to use r. The delegatee acquires the right to use all roles in $\downarrow r$.

`grantP`$_1(u, v, p)$**:** The delegator u grants the permission p to delegatee v. The delegator may continue to use p.

`xferP`$_1(u, v, p)$**:** The delegator u transfers the permission p to delegatee v. The delegator may not continue to use p.

`xferRstrong`(u, v, r)**:** The delegator u transfers the role r to delegatee v. The delegator may not use any role in $\downarrow r$. The delegatee acquires the right to use all roles in $\downarrow r$. We call this *strong transfer* of a role from the delegator to the delegatee.

`xferRstatic`(u, v, r)**:** The delegator u transfers the role r to delegatee v. The delegator may not use $x \in \downarrow r$ *unless there exists a role* r' *such that* $(u, r') \in UA$ *and* $x \leqslant r'$. (Informally, u retains the use of a role x if there is an alternative path from x to a role to which u is assigned.) We call this *static weak transfer* of a role from the delegator to the delegatee. As before, the delegatee acquires the right to use all roles in $\downarrow r$.

`xferRdynamic`(u, v, r)**:** The delegator u transfers the role r to delegatee v. The roles available to the delegator are determined by the roles he activates in a session. The delegator may not use $x \in \downarrow r$ *unless there exists a role* r' *such that* u *has activated* r' *in the current session and* $x \leqslant r'$. (Informally, u regains the use of a role x if there is an alternative path from x to a role that u has activated in her session.) We call this *dynamic weak transfer* of a role from the delegator to the delegatee. As before, the delegatee acquires the right to use all roles in $\downarrow r$.

Given the above definitions, the following results are used as a basis for deciding whether the delegator is allowed to use a role following a successful transfer delegation operation. We discuss the enforcement of the consequences of transfer delegations in Sect. 4.

Proposition 2. *If* u *has performed a static weak delegation of role* r*, then* u *is denied access to all roles in* $\sigma(r, \downarrow R_u)$*, where* R_u *denotes the set of roles assigned to* u*.*

Proof. Suppose u cannot use role x. Then $x \leqslant r$ and there does not exist a role r' such that $(u, r') \in UA$ and $x \leqslant r'$. Hence, $\uparrow x \subseteq \downarrow r \cup \uparrow r$ in $\downarrow R_u$. The result follows.

Proposition 3. *If u has performed a dynamic weak delegation of role r, then u is denied access to all roles in $\sigma(r, \downarrow S_u)$, where S_u is the set of roles activated by u.*

Proof. The proof is analogous to that of Proposition 2.

Proposition 4. *Let R_{strong}, $R_{dynamic}$ and R_{static} denote the set of roles denied to a user following the operations* $\texttt{xferRstrong}(u, v, r)$, $\texttt{xferRdynamic}(u, v, r)$ *and* $\texttt{xferRstatic}(u, v, r)$, *respectively. Then $R_{strong} \subseteq R_{dynamic} \subseteq R_{static}$.*

Proof. $R_{strong} = \downarrow R_u \setminus \downarrow r$, $R_{dynamic} = \downarrow R_u \setminus \sigma(r, \downarrow S_u)$ and $R_{weak} = \downarrow R_u \setminus \sigma(r, \downarrow R_u)$. Now $\sigma(r) \subseteq \downarrow r$ irrespective of the sub-hierarchy in which σ is computed. Hence $R_{strong} \subseteq R_{dynamic}$. Moreover, $S_u \subseteq R_u$ and it is easy to see that this implies that $\sigma(r, \downarrow R_u) \subseteq \sigma(r, S_u)$ (since if $r' \geqslant x$ for some $x \in \downarrow r$ and some $r' \in S_u$, then $r' \in R_u$).

Consider the role hierarchy depicted in Fig. 1. We assume that some user u is assigned to roles b and f. In the diagram, unfilled nodes represent roles that are available to u (and filled nodes represent nodes that are not available). Each diagram represents different views of the role hierarchy: Fig. 1(a) illustrates the whole hierarchy; Fig. 1(b) illustrates u's view of the hierarchy; and Fig.s 1(d) and 1(e) illustrate two different session views, one in which only role b is activated and one in which only role f is activated.

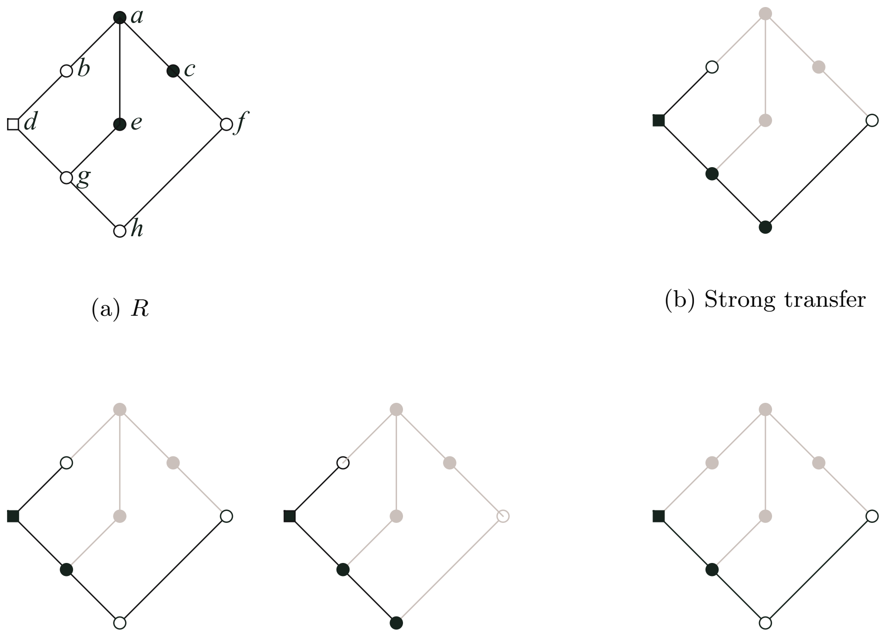
The user delegates role d, represented by the square node, to another user. Figure 1(b) illustrates the roles that are denied to the user in the event that strong transfer is used to delegate the role. If we are using dynamic weak transfer, then the ability to use role h is determined by whether the user activates role f, as shown in Fig. 1(d) and Fig. 1(e). If we are using static weak delegation, the role h is always available, as depicted in Fig. 1(c).

3 Controlling Delegation

We assume that an access control policy specifies whether delegation of a role or permission is permitted. We also assume the presence of a reference monitor that evaluates such access control policies to determine whether a delegation operation is permitted. There are two issues involved in the specification of delegations:

- Is a user (delegator) authorized to delegate a role or permission that is available to him?
- Can a role or permission be delegated to a user (delegatee)?

In this section, we describe two new mechanisms for authorizing delegations, and discuss their advantages over existing approaches.



(a) R

(b) Strong transfer

(c) Weak static transfer

(d) Weak dynamic transfer: b activated

(e) Weak dynamic transfer: f activated

Fig. 1. Transfer delegation patterns for role d

3.1 Delegation Relations

Role delegation relation. We introduce two relations that authorize role delegations in a role-based system: `can-delegate` and `can-receive`. The binary relation `can-delegate` specifies the set of roles that can be delegated by a delegator and the relation `can-receive` authorizes delegation of a role to a delegatee.

The relation `can-delegate` $\subseteq R \times R$ specifies whether a user is authorized to delegate a role. $(r, r') \in$ `can-delegate` specifies that a user u is authorized to delegate role r', if $r \in R_s$, where R_s denotes the set of roles that are active in u's current session s. For example, in Fig. 1(a), if $(b, d) \in$ `can-delegate` then u may delegate role d if $b \in R_s$.

We define a constraint on the tuples in the `can-delegate` relation that guarantees that no user can delegate a role r that the user is not assigned to. Specifically, we require that if $(r, s) \in$ `can-delegate` then $r \geqslant s$. In our example, in Fig. 1(a), an attempt to add the tuple (b, d) to the `can-delegate` relation will succeed since $b \geqslant d$. However, an attempt to add the tuple (d, c) to the `can-delegate` relation will fail since $d \not\geqslant c$.

A delegatee must satisfy a set of conditions, usually role memberships, in order to be authorized to receive a delegation. A *delegation condition* specifies the conditions that must be satisfied by the delegatee to receive a delegation. We adopt the notation of a SARBAC constraint to model a delegation condition [14]. Let $R' = \{r_1, \ldots, r_k\}$ be a subset of R and let $\bigwedge R'$ denote $r_1 \wedge \cdots \wedge r_k$. We model a delegation condition as $\bigwedge C$ for some $C \subseteq R$. A delegation condition $\bigwedge C$ is said to be *satisfied* by a user v if $C \subseteq \downarrow R_v$. In other words, the condition $\bigwedge C$ is satisfied by the delegatee if he is assigned to all the roles in C. The relation `can-receive` $\subseteq R \times C$ specifies whether a user is authorized to receive a role delegation. $(r, C) \in$ `can-receive` means a delegatee v may receive a delegation of role r provided that v satisfies $\bigwedge C$.

It may be more efficient to limit the set of roles that a delegatee may receive due to a successful delegation. For example, in Fig. 1(a), is it reasonable for a delegation request to succeed that delegates a role c to a delegatee v who currently is assigned to role g? We may often require that such delegations are not allowed, since v may lack sufficient expertise to efficiently perform the job requirements of role c. Essentially, we require that a delegation always results in a natural progression for the delegatee with respect to the role hierarchy. We formalize the above requirement by defining the following constraint on the tuples in the `can-receive` relation: $(r, C) \in$ `can-receive` then for all $r' \in C$ we require that $r' \leqslant r$. Note that this constraint does not apply if r has no junior roles (that is, r is a leaf node in the hierarchy). In our example, $(c, \{g\})$ is not a permissible entry in `can-receive`; $(c, \{f\})$, in contrast, is a permissible entry.

In summary, the role-based reference monitor refers to the `can-delegate` and `can-receive` relations to establish whether a delegation operation can succeed. A request by u to delegate r to v will succeed only if there exists $(r', r) \in$ `can-delegate` such that $r' \in R_s$, where s is u's current session and a tuple $(r, C) \in$ `can-receive` such that v satisfies $\bigwedge C$. In our example, let us assume that $(b, d) \in$ `can-delegate`, $(d, \{g\}) \in$ `can-receive`, $(u, b) \in UA$ and $(v, g) \in UA$. Then an attempt by u to delegate role d to v will succeed.

Permission delegation relation. We define a relation `can-delegatep` $\subseteq R \times P$ that specifies the set of permissions that can be delegated by a user u. $(r, p) \in$ `can-delegatep` specifies that a delegator u may delegate a permission p provided that there exists $r \in R_s$. Similar to the `can-delegate` relation, we define a constraint on the `can-delegatep` relation that guarantees that no user u can delegate a permission p that is not available to u. In other words, we require that if $(r, p) \in$ `can-delegatep` then there exists a role r' such that $(p, r') \in PA$ and $r' \leqslant r$.

The relation `can-receivep` $\subseteq P \times C$ specifies whether a user is authorized to receive a permission delegation, where C is a delegation condition as defined above. $(p, C) \in$ `can-receivep` means that a delegatee v is authorized to receive a delegation of a permission p if v satisfies $\bigwedge C$. Similar to the `can-receive` relation, we define a constraint on the `can-receivep` relation that ensures that a delegation always results in a natural progression for the delegatee with respect to the role hierarchy. If $(p, C) \in$ `can-receivep` then there must exist $r' \in C$ and $r \in R$ such that $(p, r) \in PA$ and $r' \leqslant r$.

3.2 Using Administrative Scope

The use of relations for controlling delegations, discussed above, has been used extensively in the literature mainly perhaps because of its simplicity [7, 12, 13]. The use of relations is simple, if we assume that RBAC relations, such as the role hierarchy relation RH, remain static. However, updates to the RBAC relations may lead to inconsistencies in the `can-delegate` and `can-receive` relations. Such inconsistencies are explained in detail in Sect. 3.3. For now, we note that the dynamic nature of various RBAC components increases the complexity of managing the relations that are used for controlling delegations. It is our belief that the mechanism used for controlling delegations must be simple and able to implicitly handle any updates to RBAC relations. In this section, we suggest an alternative method for controlling delegations, which dynamically handles updates to RBAC relations using the concept of administrative scope [14].

We limit the extent to which a user can delegate roles and permissions using the administrative scope of the roles(s) activated by the user. Specifically, we define the administrative scope of a session s to be

$$\sigma(s) = \bigcup_{r \in S} \sigma(r).$$

Then in order for the delegation of role r by user u to succeed we require that $r \in \sigma(s)$, where s is u's current session. In other words, u can only delegate roles that are within his administrative scope. Similarly, in order for the delegation of permission p by user u to succeed we require that there exists $r \in s$ such that $(p, r) \in PA$ and $r \in \sigma(s)$.

We now consider what criteria the delegatee must satisfy to be able to receive a delegation. Informally, for the delegation of r to v to succeed, we require that v is already "sufficiently authorized". Now, the delegation of role r means that the delegatee is authorized for all roles $r' \leqslant r$. These observations lead to the idea that the delegatee should already be assigned to any roles outside the delegator's administrative scope that the delegatee will acquire as a result of the delegation. More formally, for the delegation of role r to user v by user u to succeed we require that for all $r' < r$ such that $r' \notin \sigma(s)$, there exists r'' such that $r' \leqslant r''$ and $(v, r'') \in UA$.

In our example, Fig. 1(a), u (who is assigned to b and f) may delegate role d since $d \in \sigma(b)$. Moreover, the delegation of d to user v will only succeed if v is already assigned to role g. Note, however, that a user w who is assigned (only) to role f will not be able to receive the delegated role d (from u) because $g \notin \sigma(s)$, $g < d$ and w is not assigned to g.

Administrative scope implicitly deals with any updates to various RBAC relations, in particular the role hierarchy relation RH, since administrative scope of a session s is computed, dynamically, with respect to the role hierarchy. Consider, for example, Fig. 2(a) and Fig. 2(b) that depicts our original role hierarchy and a role hierarchy that is obtained after deleting an edge between roles d and b. In the original hierarchy, if a user u has activated b in a session s then u is authorized to delegate role d since $d \in \sigma(s)$. However, following the edge deletion operation, u is no longer able to delegate d since $d \notin \sigma(s)$. In other words, the success of delegation operations adapts in a natural and transparent way to changes in the structure of the role hierarchy. Figure 2(c) depicts the role hierarchy obtained after deleting an edge between roles g and d. In the original hierarchy, the delegatee v, who is assigned to role g, can receive a delegation of role d since $g < d$. Following the edge deletion operation, v will not be able to receive a delegation of d because $g \not< d$. Note that in Fig. 2(c) there are no roles less than d. This observation leads to the result that no user can receive a delegation of a role $\{r \in R : {\downarrow}r \subseteq {\uparrow}r \cap {\downarrow}r\}$.

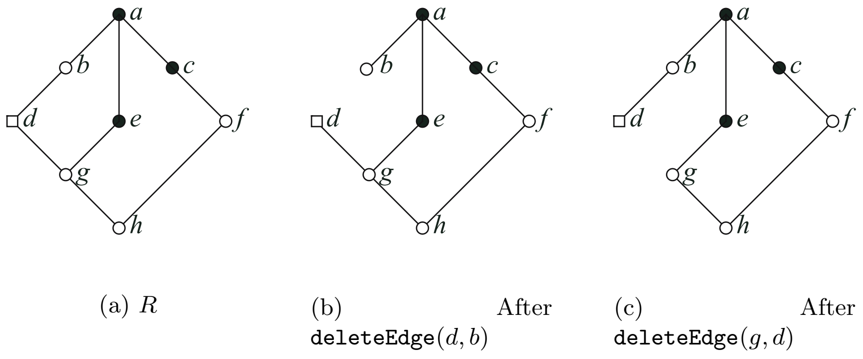


Fig. 2. Role hierarchies *before* and *after* edge deletion

3.3 Discussion

Most work in the literature uses a single relation to control delegation [7, 12, 13]. This relation encodes conditions on both the delegator and delegatee. We have proposed an alternative approach where the above issues are dealt with independently. There are several advantages in dealing with these issues separately. Primarily, it eases the management of delegation specification. Furthermore, such an approach employs separation of tasks, thus, making the model less error prone while updating delegation policies. For example, if we wish to revoke the authority of a role r to perform any delegations, then appropriate tuples are deleted from the `can-delegate` relation. Note that such an operation does not require any updates to the `can-receive` relation that authorizes a delegation to the delegatee. Similarly, if we require that a permission p may no longer be delegated to a delegatee who satisfies a condition $\bigwedge C$, we only delete necessary tuples from the `can-receivep` relation.

The use of relations to control delegation is common to most existing approaches, mainly perhaps because of its simplicity [7, 12, 13]. The use of relations is appropriate, if we assume that RBAC structures such as the role hierarchy remain static. However, updates to the role hierarchy may lead to inconsistencies in the `can-delegate` and `can-receive` relations. Consider, for example, Fig. 2(a) and Fig. 2(b) that depict an original role hierarchy and a role hierarchy obtained after deleting an edge between roles d and b. If $(b, d) \in$ `can-delegate` it will need to be *explicitly* deleted following the deletion of this edge. Similar considerations apply to the operations that involve revocation of a permission p assigned to b. Similarly, if $(b, \{d\}) \in$ `can-receive`, then we must delete this entry following the deletion of the edge between b and d. In summary, the `can-delegate` and `can-receive` relations must be updated after updates to certain RBAC relations to prevent inconsistencies.

Hence the advantages of using relations for controlling delegations are limited if we allow updates to RBAC relations. In contrast, administrative scope is a *dynamic* model and *implicitly* deals with any updates to RBAC relations. It is important to note that, following successful updates to RBAC relations, no explicit updates are required with the use of administrative scope to resolve any inconsistencies involved in controlling delegations. This is because administrative scope (and hence delegation) is determined by the structure of the role hierarchy. In short, the use of administrative scope greatly simplifies delegation in the presence of dynamic RBAC structures.

Furthermore, the administrative scope model for controlling delegations preserves the separation of conditions on delegator and delegatee that we introduced in our relation-based model. The issues that are involved in controlling delegations, such as specifying the roles that a user is authorized to delegate and receive, are still dealt with independently. Note also that the constraint we use to limit the increase of power of the delegatee can be framed very elegantly using administrative scope and existing RBAC relations.

4 Enforcing Delegation Constraints

Enforcing delegation operations that grant a role or permission to a user is quite straightforward as it is a monotonic action. That is, the set of roles or permissions for each user either stays the same or increases. Delegation operations that transfer a role or permission are rather more difficult. In this section, we introduce three relations that are used to guarantee that delegation operations are enforced correctly. The model described in this section can be used for both grant and transfer delegations. To our knowledge, this is the first treatment of the consequences of temporary transfer delegation policies for access control in role-based systems. This is mainly due to the fact that there is much to say on the subject when grant delegations, but not transfer delegations, are allowed in role-based models.

4.1 The Delegation History Relation

The delegation history (DH) relation is used to record all delegations that have been made. The DH relation is used by the delegators and administrative users for administrative purposes. Typically, such activities include auditing and revoking delegations.

A DH tuple has the form (i, u, v, o, C, M) where i is an identifier, u is a delegator, v is the delegatee, o is an object or target of the delegation that can either be a set of roles $R' \subset R$ and/or a set of permissions $P' \subseteq P$, C is the set of conditions, from the `can-receive` relation, that must be satisfied by the delegatee and M is a *delegation mask*. A delegation mask records various internal details of a delegation.

A delegation mask M contains five bits which are used to record certain details of the delegation. More specifically, a delegation mask has the form $b_4b_3b_2b_1b_0$, where $b_i \in \{0, 1\}$, b_4 specifies whether the delegated object can be further delegated[1], b_3 specifies whether the delegated object is a role or permission, b_2 specifies whether the delegation is static or dynamic, b_1 specifies whether the delegation is strong or weak, and b_0 specifies whether the delegation is grant or transfer. The characteristics of a delegation mask are summarized in Table 1, and some examples of masks are shown in Table 2 together with the commands that would give rise to such a mask.

Note that we require the delegator to explicitly set a few values of the bits in the delegation mask while performing a delegation operation. In particular, we require that the bits b_0, b_1 and b_2 are set by the delegator, while the bits b_3 and b_4 are deduced by the access control enforcement system.

Table 1. Delegation mask bit values

	b_4	b_3	b_2	b_1	b_0
0	`undelegatable`	`role`	`static`	`strong`	`grant`
1	`delegatable`	`permission`	`dynamic`	`weak`	`transfer`

4.2 Temporary Delegation Relations

We also introduce two relations *tempUA* and *tempPA*, which record temporary user-role and user-permission assignments that arise from delegation operations. *tempUA* contains tuples of the form (i, u, r, s), where $s \in \{-, +\}$ and i identifies a tuple in the DH relation. The meaning of the tuple $(i, v, r, +)$ is analogous to $(v, r) \in UA$; such a tuple would arise as a result of a grant or transfer of role r to the delegatee v. In contrast, a tuple of the form $(i, u, r, -)$ means that u is prohibited from activating role r; such a tuple only arises when u transfers r to v. The precise set of roles that are unavailable for the delegator u depends on the transfer type used for delegation (strong, weak static or weak dynamic).

tempPA contains tuples of the form (i, u, p, s), where $s \in \{-, +\}$ and i identifies a tuple in the DH relation. The tuple $(i, v, p, +)$ means that v is allowed

[1] An extra boolean-valued parameter can be added to each of the delegation commands; this parameter would be used to set this bit.

Table 2. Examples of valid delegation mask values, their semantics, and associated commands

Mask	**Delegation semantics**	**Command**
`00xx0`	undelegatable, role, grant	`grantR`
`10x01`	delegatable, role, strong, transfer	`xferRstrong`
`00011`	undelegatable, role, static, weak, transfer	`xferRstatic`
`10111`	delegatable, role, dynamic, weak, transfer	`xferRdynamic`
`11xx0`	delegatable, permission, grant	`grantP`
`11x01`	delegatable, permission, strong, transfer	`xferPstrong`
`01111`	undelegatable, permission, dynamic, weak, transfer	`xferPdynamic`
`11011`	delegatable, permission, static, weak, transfer	`xferPstatic`

to use p; such a tuple would arise as a result of a grant or transfer of permission p to the delegatee v. A tuple of the form $(i, u, p, -)$ means that u is prohibited from invoking permission p; such a tuple only arises when u transfers p to v.

For simplicity, we write (i, u, R', s) to denote $\{(i, u, r', s) : r' \in R'\}$. We also write (i, u, P', s) to denote $\{(i, u, p', s) : p' \in P'\}$. We now describe the effects on various relations following the successful execution of a delegation operation. We focus our attention on the more difficult case of RBAC$_1$.

Successful delegation operations are recorded in the delegation history relation DH. Hence, following any delegation of o by u to v, we have $DH \leftarrow DH \cup \{(i, u, v, o, C, M)\}$. In addition we have the following operational semantics:

`grantR`$_1(u, v, r)$**:** u grants role r to v. Such a delegation requires that the delegatee v is allowed to use role r. Hence, $\mathit{tempUA} \leftarrow \mathit{tempUA} \cup \{(i, v, r, +)\}$.

`grantP`$_1(u, v, p)$**:** u grants permission p to the delegatee v. Then, permission p must be available to the delegatee v. Hence, $\mathit{tempPA} \leftarrow \mathit{tempPA} \cup \{(i, v, p, +)\}$.

`xferP`$_1(u, v, p)$**:** delegator u transfers the authority to use permission p to the delegatee v. Hence, we add $\mathit{tempPA} \leftarrow \mathit{tempPA} \cup \{(i, v, p, +), (i, u, p, -)\}$.

`xferRstrong`(u, v, r)**:** u performs a strong transfer of role r to v. The delegator u may not use any role in $\downarrow r$ and the delegatee v acquires the right to use all roles in $\downarrow r$. Hence, $\mathit{tempUA} \leftarrow \mathit{tempUA} \cup \{(i, v, r, +), (i, u, R', -)\}$, where $R' = \downarrow r$.

`xferRstatic`(u, v, r)**:** u performs a static weak transfer of role r to v. The delegator u may not use a role $x \in \downarrow r$ unless there exists a role r' such that $(u, r') \in UA$ and $x \leqslant r'$. Hence, by Proposition 2, $\mathit{tempUA} \leftarrow \mathit{tempUA} \cup \{(i, v, r, +), (i, u, R', -)\}$, where $R' = \sigma(r, \downarrow R_u)$.

`xferRdynamic`(u, v, r)**:** u performs a dynamic weak transfer of role r to v. The set of roles available to the delegator u are computed by the roles that

u activates in a session. Hence, by Proposition 3, $tempUA \leftarrow tempUA \cup \{(i, v, r, +), (i, u, R', -)\}$, where $R' = \sigma(r, \downarrow S_u)$.[2]

4.3 Access Control Decisions

The role-based reference monitor uses the $tempUA$ and $tempPA$ relations, in addition to the UA and PA relations, to make access control decisions. For example, an attempt by a user u to activate a role r is always denied if there exists a tuple $(i, u, R', -) \in tempUA$ such that $r \in R'$; and granted if there exists a tuple $(i, u, r', +) \in tempUA$ or $(u, r') \in UA$ such that $r \leqslant r'$. Similarly, an attempt to invoke a permission p by u is always denied if $(i, u, p, -) \in tempPA$; and granted if $(i, u, p, +) \in tempPA$ or there exist $r, r' \in R$ such that $(p, r) \in PA$, $(u, r') \in UA$ and $r \leqslant r'$.

4.4 Revocation

Successful delegations have a specified lifetime or may be revoked before the delegation ends.[3] In either case, the $tempUA$ and $tempPA$ relations must be updated to prevent any subsequent use of the delegated access right by the delegatee and, if necessary, to allow the delegator to use the delegated access right. Essentially, we require that when a delegation $d \in DH$ ends or is revoked the tuples $(i, u, R', s) \in tempUA$ and $(i, u, P', s) \in tempPA$ are *deleted*, where i identifies the obsolete delegation d. We now describe the effects on various relations when a delegation ends or is revoked.

$\texttt{grantR}_1(u, v, r)$: When such a delegation ends, the delegatee v is no longer allowed to use role r. Hence, $tempUA \leftarrow tempUA \setminus \{(i, v, r, +)\}$.

$\texttt{grantP}_1(u, v, p)$: permission p must no longer be available for the delegatee v. Hence, $tempPA \leftarrow tempPA \setminus \{(i, v, p, +)\}$.

$\texttt{xferP}_1(u, v, p)$: delegatee v must be prevented from using permission p and the delegator u regains the ability to use the delegated permission p. Hence, $tempPA \leftarrow tempPA \setminus \{(i, v, p, +), (i, u, p, -)\}$.

$\texttt{xferRstrong}(u, v, r)$: u performs a strong transfer of role r to v. The delegator u gets back the right to use any role in $\downarrow r$ and the delegatee v can no longer use any role in $\downarrow r$. Hence, $tempUA \leftarrow tempUA \setminus \{(i, v, r, +), (i, u, R', -)\}$ where $R' = \downarrow r$.

[2] It is important to note that, following a dynamic weak transfer, the roles that are to be prevented for the delegator u are determined by the roles that are active in u's current session. Hence, it is necessary to add appropriate tuples to the $tempUA$ relation whenever a session is initiated by a user who has performed a weak dynamic delegation.

[3] In a practical implementation, the DH relation would include some information that would determine the lifetime of the delegation: this information might be a start and an end time, or a start time and a duration, for example.

`xferRstatic`(u, v, r): u performs a static weak transfer of role r to v. Similar to the above delegation, the delegator u gets back the right to use any role in $\downarrow r$ and the delegatee v can no longer use any role in $\downarrow r$. Hence, $\mathit{tempUA} \leftarrow \mathit{tempUA} \setminus \{(i, v, r, +), (i, u, R', -)\}$ where $R' = \sigma(r, \downarrow R_u)$.

`xferRdynamic`(u, v, r): u performs a dynamic weak transfer of role r to v. Again, the delegator u regains the right to use any role in $\downarrow r$ and the delegatee v can no longer use any role in $\downarrow r$. Hence, $\mathit{tempUA} \leftarrow \mathit{tempUA} \setminus \{(i, v, r, +), (i, u, R', -)\}$ where $R' = \sigma(r, \downarrow S_u)$.[4]

5 Related Work

Several delegation models have been proposed for role-based access control [3, 7, 8, 9, 10, 11, 12, 13]. The early work of Barka and Sandhu was instrumental in identifying the important considerations for delegation in RBAC [3]. This included the concepts of *monotonic* and *non-monotonic* delegation, which correspond to our grant and transfer models. To our knowledge, the work that we present in this paper represents the first attempt to deal properly with the consequences of temporary non-monotonic delegation in RBAC.[5]

Of the work in the literature, the RBDM0, RDM2000 and PBDM models are closest to our work [7, 8, 12, 13]. RBDM0 is a model for delegating roles, and is based on the RBAC$_0$ model of the RBAC96 family of models [7]. Another role delegation model for role-based access control is presented in [8]. RDM2000 defines a rule-based framework for delegation and revocation [12]. The model considers role hierarchies and also provides support for multi-step delegations. The PBDM model proposes a delegation model for permissions that supports multi-step delegations [13].

The RDM2000 model uses a relation $\mathit{can\_delegate} \subseteq R \times 2^R \times \mathbb{N}$ to authorize delegations, $\mathbb{N}$ is the set of natural numbers. If $(r, C, n) \in \mathit{can\_delegate}$, a delegator acting in role r may delegate any role $r' \leqslant r$ to a delegatee v provided v satisfies some role assignment conditions specified by C; n is used to define the maximum depth of a delegation. A major limitation of this relation is that no constraints are defined on the tuples in the *can_delegate* relation. That is, r and C are not required to have any relationship with each other. What this means is that the delegatee's power can be arbitrarily increased by a successful delegation. In practice, a delegatee can be assigned to roles for which they lack any relevant expertise or experience.

The PBDM model uses a similarly named relation $\mathit{can\_delegate} \subseteq R \times R \times P \times \mathbb{N}$. If $(r, r', P', n) \in \mathit{can\_delegate}$ then a delegator who is assigned to role r

[4] Recall that, following a dynamic weak transfer of a role r, we determine the roles that are not available for the delegator u, in a session s, by computing $\sigma(r, \downarrow S_u)$ (see Proposition 3). Hence, it may also be necessary to delete appropriate tuples from the tempUA and tempPA relations when the delegator u deactivates a role.

[5] Barka distinguishes between temporary and permanent delegation for role-based delegation models [17]. We believe the latter is more correctly viewed as an administrative activity. Barka does not consider temporary non-monotonic delegation policies (which we call transfer delegation policies).

can delegate the set of permissions P' to a user who is assigned to a role r' with a maximum delegation depth of n. Like the RDM2000 model, the *can_delegate* relation does not require that there is any relationship between P' and r, which means that it is possible for a delegator u to delegate a permission p that is not available to u. As we have already noted, these relation-based approaches to delegation are unlikely to be suitable in environments where the role hierarchy may change.

Unlike the above discussed models, which use a single relation for controlling delegations, we use two different relations: `can-delegate` and `can-receive`. Advantages of using different relations for controlling delegations include flexibility, greater control while specifying delegations, ease of management and is less error prone. Furthermore, our model for controlling delegations defines constraints on the `can-delegate` and `can-receive` relations. Such constraints ensure that the tuples that are added to the `can-delegate` and `can-receive` relations does not give the authority for a delegator u to delegate a right that is not available to u and the rights of a delegatee v can only be incrementally increased and are limited by v's existing rights.

Hence, we believe our delegation specification model is more conservative (and thus safer), more fine-grained and more manageable than these models [7, 8, 12, 13]. However, in Sect. 3.3, we observed that the use of relations for controlling delegations may not be efficient for implicitly handling updates to various RBAC relations. Our model includes an alternative way of controlling delegations using the concept of administrative scope [14]. The administrative scope model is dynamic and implicitly handles any updates to RBAC relations, in particular the role hierarchy relation RH. We have also described the enforcement of both grant and transfer delegations and dealt with revocations.

A delegation model with restricted permission inheritance is proposed in [9]. The model is based on the idea of dividing a role hierarchy into inter-related role hierarchies. A cascaded role delegation model in the context of decentralized trust management systems is presented in [10]. Another model that supports delegation of both roles and permissions is discussed in [11]. We believe that our

Table 3. Delegation characteristics in various delegation models

Characteristic	**RBDM0**	**RDM2000**	**PBDM**	**Our model**
Role delegation	✓	✓	✓	✓
Permission delegation	✗	✗	✓	✓
Grant delegation	✓	✓	✓	✓
Transfer delegation	✗	✗	✗	✓
Controlling delegations	✓	✓	✓	✓
Implicit updates	✗	✗	✗	✓
Delegation history	✗	✓	✗	✓
Temporary delegation assignments	✓	✓	✓	✓
Revocation	✓	✓	✓	✓

model can easily be extended to incorporate these concepts. However, none of the above work considers transfer delegation for role-based systems. Table 3 compares the support provided for key delegation features by well known delegation models.

6 Conclusions and Future Work

We have developed a comprehensive delegation model for role-based access control that provides support for both grant and transfer delegation policies. This is the first attempt in the literature that extensively studies transfer delegations for role-based access control.

We have discussed two mechanisms for controlling delegations: the relations approach and the administrative scope approach. In particular, we have shown that the concept of administrative scope can be used for authorising delegations and is more effective than relations for delegation in dynamic environments.

We have also presented an enforcement mechanism that supports both grant and transfer delegation policies. Our enforcement model uses a delegation history relation DH and temporary delegation relations, $tempUA$ and $tempPA$, to guarantee that delegations are enforced correctly. We also discussed various effects on the above relations following a successful delegation operation and corresponding revocations.

An immediate priority in future work is to enrich the revocation mechanism that has been described in this paper. Most importantly, we wish to provide support for cascaded revocations. That is, if a delegation is revoked and this delegation had been used to generate other delegations, then the latter delegations must also be revoked. We may also consider revocation of a delegation when a user is revoked from a role $r \in C$, where C is a delegation condition. Another future work is to consider supporting revocations such as those discussed in [18]. Furthermore, we wish to define necessary commands that provide an option to set the delegatable flag in delegation mask. A long term goal is to develop abstract Java classes that implement our delegation model.

References

1. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-based access control models. IEEE Computer **29**(2) (1996) 38–47
2. Schaad, A.: A Framework for Organisational Control Principles. PhD thesis, The University of York, York, England (2003)
3. Barka, E., Sandhu, R.: Framework for role-based delegation models. In: Proceedings of Twenty Third National Information Systems Security Conference (NISSC'00). (2000) 101–114
4. Aura, T.: Distributed access-rights management with delegation certificates. In: Secure Internet Programming – Security Issues for Distributed and Mobile Objects. Volume 1603 of LNCS. Springer (1999) 211–235
5. Gligor, V., Gavrila, S., Ferraiolo, D.: On the formal definition of separation-of-duty policies and their composition. In: Proceedings of IEEE Symposium on Research in Security and Privacy. (1998) 172–183

6. Simon, R., Zurko, M.: Separation of duty in role-based environments. In: Proceedings of Tenth IEEE Computer Security Foundations Workshop. (1997) 183–194
7. Barka, E., Sandhu, R.: A role-based delegation model and some extensions. In: Proceedings of Sixteenth Annual Computer Security Applications Conference (ACSAC'00). (2000) 168–177
8. Na, S., Cheon, S.: Role delegation in role-based access control. In: Proceedings of Fifth ACM Workshop on Role-Based Access Control (RBAC'00). (2000) 39–44
9. Park, J., Lee, Y., Lee, H., Noh, B.: A role-based delegation model using role hierarchy supporting restricted permission inheritance. In: Proceedings of the 2003 International Conference on Security and Management (SAM'03). (2003) 294–302
10. Tamassia, R., Yao, D., Winsborough, W.: Role-based casdaded delegation. In: Proceedings of Ninth ACM Symposium on Access Control Models and Technologies (SACMAT'04). (2004) 146–155
11. Wainer, J., Kumar, A.: A fine-grained, controllable, user-to-user delegation method in RBAC. In: Proceedings of Tenth ACM Symposium on Access Control Models and Technologies (SACMAT'05). (2005) 59–66
12. Zhang, L., Ahn, G.J., Chu, B.T.: A rule-based framework for role-based delegation and revocation. ACM Transactions on Information and System Security (TISSEC) **6**(3) (2003) 404–441
13. Zhang, X., Oh, S., Sandhu, R.: PBDM: A flexible delegation model in RBAC. In: Proceedings of Eighth ACM Symposium on Access Control Models and Technologies (SACMAT'03). (2003) 149–157
14. Crampton, J., Loizou, G.: Administrative scope: A foundation for role-based administrative models. ACM Transactions on Information and System Security (TISSEC) **6**(2) (2003) 201–231
15. Ferraiolo, D., Kuhn, D., Chandramouli, S.: Role-Based Access Control. Artech House, Boston, Massachussetts (2003)
16. Sandhu, R., Bhamidipati, V., Munawer, Q.: The ARBAC97 model for role-based administration of roles. ACM Transactions on Information and System Security **1**(2) (1999) 105–135
17. Barka, E.: Framework for Role-Based Delegation Models. PhD thesis, George Mason University, Virginia, USA (2002)
18. Hagström, Å., Jajodia, S., Parisi-Presicce, F.: Revocations– a classification. In: Proceedings of the Fourteenth IEEE Workshop on Computer Security Foundations (CSFW'01). (2001) 44–58

Applying a Security Requirements Engineering Process

Daniel Mellado[1], Eduardo Fernández-Medina[2], and Mario Piattini[2]

[1] Ministry of Labour and Social Affairs; Information Technology Center of the National Social Security Institute; Madrid, Spain
Daniel.Mellado@alu.uclm.es

[2] Alarcos Research Group, Information Systems and Technologies Department, UCLM-Soluziona Research and Development Institute, University of Castilla-La Mancha, Paseo de la Universidad 4, 13071 Ciudad Real, Spain
{Eduardo.FdezMedina, Mario.Piattini}@uclm.es

Abstract. Nowadays, security solutions are mainly focused on providing security defences, instead of solving one of the main reasons for security problems that refers to an appropriate Information Systems (IS) design. In fact, requirements engineering often neglects enough attention to security concerns. In this paper it will be presented a case study of our proposal, called SREP (Security Requirements Engineering Process), which is a standard-centred process and a reuse-based approach which deals with the security requirements at the earlier stages of software development in a systematic and intuitive way by providing a security resources repository and by integrating the Common Criteria into the software development lifecycle. In brief, a case study is shown in this paper demonstrating how the security requirements for a security critical IS can be obtained in a guided and systematic way by applying SREP.

1 Introduction

Present-day information systems are vulnerable to a host of threats. What is more, with increasing complexity of applications and services, there is a correspondingly greater chance of suffering from breaches in security [20]. In our contemporary Information Society, depending as it does on a huge number of software systems which have a critical role, it is absolutely vital that IS are ensured as being safe right from the very beginning [1, 13].

As we know, the principle which establishes that the building of security into the early stages of the development process is cost-effective and also brings about more robust designs is widely-accepted [9]. The biggest problem, however, is that in the majority of software projects security is dealt with when the system has already been designed and put into operation. Added to this, the actual security requirements themselves are often not well understood. This being so, even when there is an attempt to define security requirements, many developers tend to describe design solutions in terms of protection mechanisms, instead of making declarative propositions regarding the level of protection required [4].

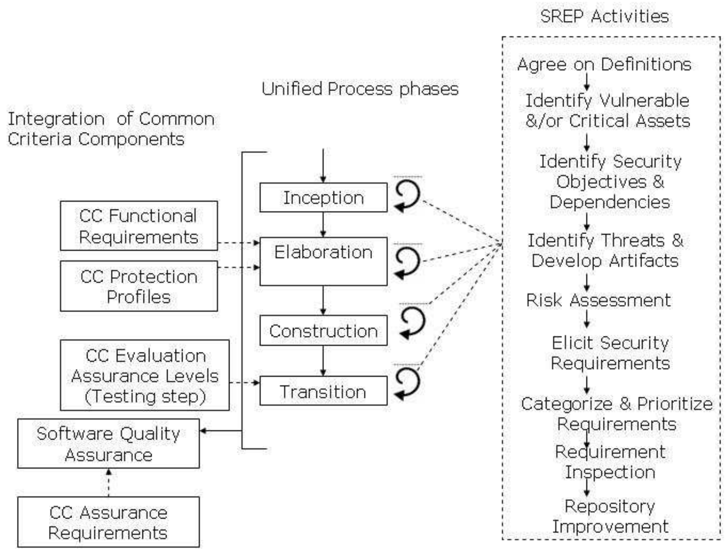
A very important part of the achieving of secure software systems in the software development process is that known as Security Requirements Engineering, which provides techniques, methods and norms for tackling this task in the IS development cycle. It should involve the use of repeatable and systematic procedures in an effort to ensure that the set of requirements obtained is complete, consistent and easy to understand and analyzable by the different actors involved in the development of the system [10]. A good requirement specification document should include both functional requirements and non-functional. As far as security is concerned, it should be a consideration throughout the whole development process, and it ought to be defined in conjunction with the requirements specification [16].

After having performed a comparative analysis of several relevant proposals of IS security requirements, as those of Yu 1997 [21], Toval et al. 2001 [19], Popp et al. 2003 [17], Firesmith 2003 [5], Breu, et al. 2004 [3], etc. in [15], we concluded that those proposals did not reach the desired level of integration into the development of IS, nor are specific enough for a systematic and intuitive treatment of IS security requirements at the early stages of software development. In addition, as yet, only few works (such as the article of Massacci et al. [12]) describes complex case studies which really cope with the complexity required by security standards, such as ISO/IEC 17799 [7] and ISO/IEC 15408 [8], compliance. Therefore, in this paper we briefly present the Security Requirements Engineering Process (SREP) [14] along with a case study of this proposal, which describes how to integrate security requirements into the software engineering process in a systematic and intuitive way. In order to achieve this goal, our approach is based on the integration of the Common Criteria (CC) [8] into the software lifecycle model, because the CC helps us deal with the security requirements along all the IS development lifecycle, together with the reuse of security requirements which are compatible with the CC Framework subset. In addition this proposal integrates other approaches such as UMLSec [17], security use cases [5] or misuse cases [18]. The remainder of this paper is set out as follows: in section 2, we will describe SREP. We will present the case study of SREP in section 3. Next, in section 4 it is presented the lessons learned. Lastly, our conclusions will be set out in section 5.

2 SREP: Security Requirements Engineering Process

To describe our proposal, we will rely on the process description patterns used in the Unified Process (UP) [2], since it is a use-case and risk driven, architecture-centric, iterative and incremental development process frame-work that leverages the Object Management Group's (OMG) UML and that is compliant with the OMG's Software Process Engineering Metamodel (SPEM).

The Security Requirements Engineering Process (SREP) is an asset-based and risk-driven method for the establishment of security requirements in the development of secure Information Systems. Basically, this process describes how to integrate the CC into the software lifecycle model together with the



SREP Activities
Agree on Definitions
Identify Vulnerable &/or Critical Assets
Identify Security Objectives & Dependencies
Identify Threats & Develop Artifacts
Risk Assessment
Elicit Security Requirements
Categorize & Prioritize Requirements
Requirement Inspection
Repository Improvement
Unified Process phases
Inception
Elaboration
Construction
Transition
Integration of Common Criteria Components
CC Functional Requirements
CC Protection Profiles
CC Evaluation Assurance Levels (Testing step)
Software Quality Assurance
CC Assurance Requirements

Fig. 1. SREP Overview

use of a security resources repository to support reuse of security requirements, assets, threats and countermeasures. The focus of this methodology seeks to build security concepts at the early phases of the development lifecycle.

As it is described in Fig. 1, the UP lifecycle is divided into a sequence of phases, and each phase may include many iterations. Each iteration is like a mini-project and it may contain all the core workflows (requirements, analysis, design, implementation, and test), but with different emphasis depending on where the iteration is in the lifecycle. Moreover, the core of SREP is a micro-process, made up of nine activities which are repeatedly performed at each iteration throughout the iterative and incremental development, but also with different emphasis depending on what phase of the lifecycle the iteration is in. Thus, the model chosen for SREP is iterative and incremental, and the security requirements and their associated security elements (threats, security objectives, etc.) evolve along the lifecycle. At the same time, the CC Components are introduced into the software lifecycle, so that SREP uses different CC Components according to the phase of the lifecycle and the activity of SREP, although the Software Quality Assurance (SQA) activities are per-formed along all the phases of the software development lifecycle, and it is in these SQA activities where the most of CC Assurance Requirements might be incorporated into.

2.1 The Security Resources Repository

The purpose of development with requirements reuse is to increase their quality: inconsistency, errors, ambiguity and other problems can be detected and corrected for an improved use in subsequent projects [19]. Thereby, it will guarantee

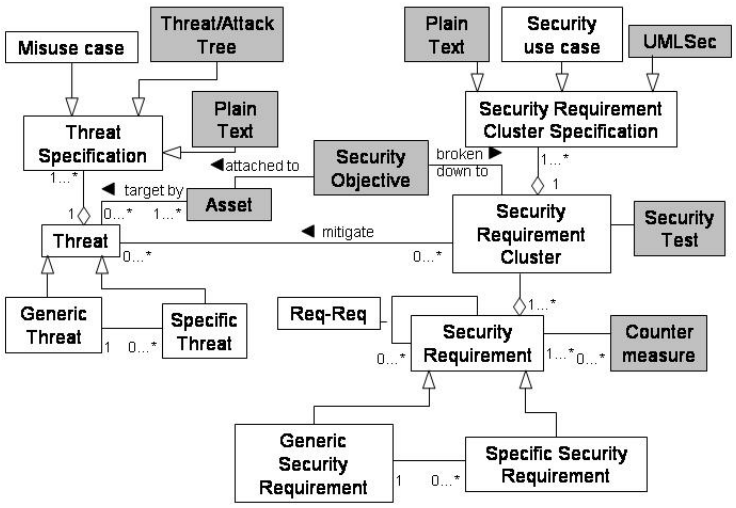


Fig. 2. Meta-model for security resources repository

us the fastest possible development cycles based on proven solutions. Therefore, we propose a Security Resources Repository (SRR), which stores all the security reusable elements. A meta-model, which is an extension of the meta-model for repository proposed by Sindre, G., D.G. Firesmith, and A.L. Opdahl [18], showing the organization of the SRR is exposed in Fig. 2. The dark background in the objects represents our contribution to the meta-model.

As presented in Fig. 2, it is an asset-driven as well as a threat-driven meta-model, because the requirements can be retrieved via assets or threats. Next, we will outline the most important and/or complex aspects of the meta-model:

– 'Generic Threat' and 'Generic Security Requirement' are described independently of particular domains. And they can be represented as different specifications, thanks to the elements 'Threat Specification' and 'Security Requirement Cluster Specification'.
– 'Security Requirement Cluster' is a set of requirements that work together in satisfying the same security objective and mitigating the same threat. We agree with Sindre, G., D.G. Firesmith, and A.L. Opdahl [18] that, in many cases, it is a bigger and more effective unit of reuse.
– The 'Req-Req' relationship allows an inclusive or exclusive trace between requirements. An exclusive trace between requirements means that they are mutually alternative, as for example that they are in conflict or overlapping. Whereas, an inclusive trace between requirements means that to satisfy one, another/other/s is/are needed to be satisfied.

It is known that an important part of the security of an IS can be often achieved through administrative measures, however the CC does not provide us with methodological support, nor contain security evaluation criteria pertaining to administrative security measures not directly related to the IS security measures. Therefore, according to ISO/IEC 17799:2005 [7], we propose to include legal, statutory, regulatory, and contractual requirements that the organization, its trading partners, contractors, and service providers have to satisfy, and their socio-cultural environment. After converting these require-ments into software and system requirements format, these requirements along with the CC security requirements would be the initial subset of security requirements of the SRR.

3 Case Study

The case study we presented here is a representative case of a security critical IS in which security requirements have to be correctly treated in order to achieve a robust IS. It will be analysed the case of an administrative unit of the National Social Security Institute (of Spain), which has the porpoise of providing citizens e-government services. Here it will be studied the case of an e-government service which consists of an application (called Pension-App) that basically allows to provide information about the pension/s of a concrete citizen. Taking into account the constraint of space, this case study is unrealistically simple to enable points of SREP to be easily illustrated in this paper.

PensionApp is an application that allows citizens to obtain an official document which reflects the current amount and the status of their pension/s (whether it is being processed and the stage where it is at the moment of the request, or whether it has been successfully granted or rejected), it also allows citizens to update some personal data, such as their address and bank account number. One of the main design goals was maximum ease of use. Thus, citizens have online access to PensionApp through the Internet or they can go to an office of the National Social Security Institute, where a civil servant will provide them with an official paper document with the information requested about their pension or he/she will update the personal information of the citizens by interacting with other application which has been al-ready developed. Thus, a citizen can only obtain information about his/her own pension and update his/her personal information whereas a civil servant can get pension information and update personal information for a specified social security number of a person by interacting with the IS but through other application different from PensionApp. Thereby, we assume that initial functional requirements have been elicited and that there is only two functional requirements:

– Req 1: On request-1 from an EndUser, the system shall display in-formation about his/her pension. This request shall include the social security number of the EndUser.
– Req 2: On request-2 from an EndUser, the system shall update the personal information of the pensioner. This request shall include the social security number of the EndUser and changed personal data.

In addition we assume that the Organization has already introduced some elements into the Security Resources Repository (SRR), such as legal, statutory, regulatory, and contractual requirements that the organization, its trading partners, contractors, and service providers have to satisfy, and their socio-cultural environment. After converting these requirements into software and system requirements format, these requirements along with the CC security requirements will be the initial subset of security requirements of the SRR, which together with their associated security-elements (security objectives, assets, threats,...) will be the initial subset of security elements of the SRR.

SREP defines nine activities to be carried out as well as several iterations through the software development lifecycle, and each iteration will generate internal or external releases of various artefacts which altogether constitute a baseline, although in the following subsection of this paper we will only describe one iteration at the early stages of the software development lifecycle.

3.1 Activity 1: Agree on Definitions

In this activity we have to agree upon a common set of security definitions, along with the definition of the organizational security policies and the security vision of the IS. The following is a minute subset of the definitions that should be agreed.

– Information security: preservation of confidentiality, integrity and availability of information; in addition, other properties, such as authenticity, accountability, non-repudiation and reliability can be also involved [ISO/IEC 17799:2005].
– Threat: a potential cause of an unwanted incident, which may result in harm to a system or organization [ISO/IEC 13335-1:2004] [6].
– Availability: the property of being accessible and usable upon de-mand by an authorized entity [ISO/IEC 13335-1:2004].
– Confidentiality: the property that information is not made available or disclosed to unauthorized individuals, entities, or processes [ISO/IEC 13335-1:2004].
– Integrity: the property of safeguarding the accuracy and complete-ness of assets [ISO/IEC 13335-1:2004].
– Asset: anything that has value to the organization [ISO/IEC 13335-1:2004].

Then, the *Security Vision Document* will be written, in which it will be outlined the security vision of the IS. In this case, it will state that the most important asset is information, so from the security point of view it is important that confidentiality, availability and integrity of information, as well as authenticity, accountability, non-repudiation of the users and services are ensured.

3.2 Activity 2: Identify Vulnerable and/or Critical Assets

We have to perform an examination of functional requirements (Req1) (because according to CC assurance requirement ADV_FSP.3.1D the developer shall provide a functional specification) and we have realized that there is only one relevant asset type: Information. Other assets would need to be considered in a real case study, including tangible assets such as money or products and intangible assets such as reputation. We can consider different types of Information:

- Personal information about the pensioner: name, social security number, address.
- Personal information about the pension/s: kind of pension (old-age/disability (type of disability)/widow's pension), amount of money, bank account number.

3.3 Activity 3: Identify Security Objectives and Dependencies

In this activity the SRR can be used, so that if the type of assets identified in the previous activity are in the SRR we will be able to retrieve their associated security objectives (SO). Otherwise we will determine the security objectives for each asset and we will take into account the security policy of the Organization as well as legal requirements and constraints in Spain and in the National Social Security Institute. We can identify the following security objectives:

- SO1: Prevent unauthorised disclosure of information. (Confidential-ity). Valuation - High.
- SO2: Prevent unauthorised alteration of information. (Integrity). Valuation - High.
- SO3: Ensure availability of information to the authorised users. Valuation - Medium.
- SO4: Ensure authenticity of users. Valuation - High.
- SO5: Ensure accountability. Valuation - Medium.

This is not a complete list, it should be refined in subsequent iterations (for example by establishing probability and dependencies between the security objectives), but it will be enough for this discussion. These security objectives will be written down in the *Security Objectives* Document with the help of the CC assurance classes (CC class ASE).

3.4 Activity 4: Identify Threats and Develop Artefacts

If the assets identified in the previous activity are in the SRR we will be able to retrieve their associated threats. Otherwise we will find all threats that can prevent the security goal from being achieved by instantiating the business use cases into misuse cases or by instantiating the threat-attack trees associated with the business and application pattern. In addition, we will analyse predefined threat lists for the type of assets selected and following the CC assurance requirement AVA_VAN.5.2E we will search in public domain sources to identify potential vulnerabilities in the IS. In Fig. 3 we present an example of misuse case diagram along with the possible attackers (crackers, thieves, etc.).

Therefore we identify several possible types of threats to Information:

- Generic Threat 1: Unauthorised disclosure of information.
- Generic Threat 2: Unauthorised alteration of information.

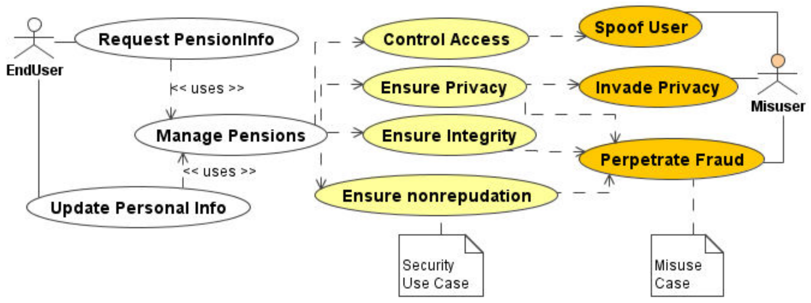


Fig. 3. Security Use Cases and Misuse Cases

– Generic Threat 3: Unauthorised unavailability to information.
– Generic Threat 4: Spoof user.

Then, we will develop the Generic Threats together with the Specific Threats (which are several paths of these Generic Threats) if there are no threats in the SRR that match with the previous identified types of threats. So we will present an example of a specification of a *Generic Threat*(Generic Threat 2) and a *Specific* Threat using misuse cases as a method of specification in Tables 1 and 2.

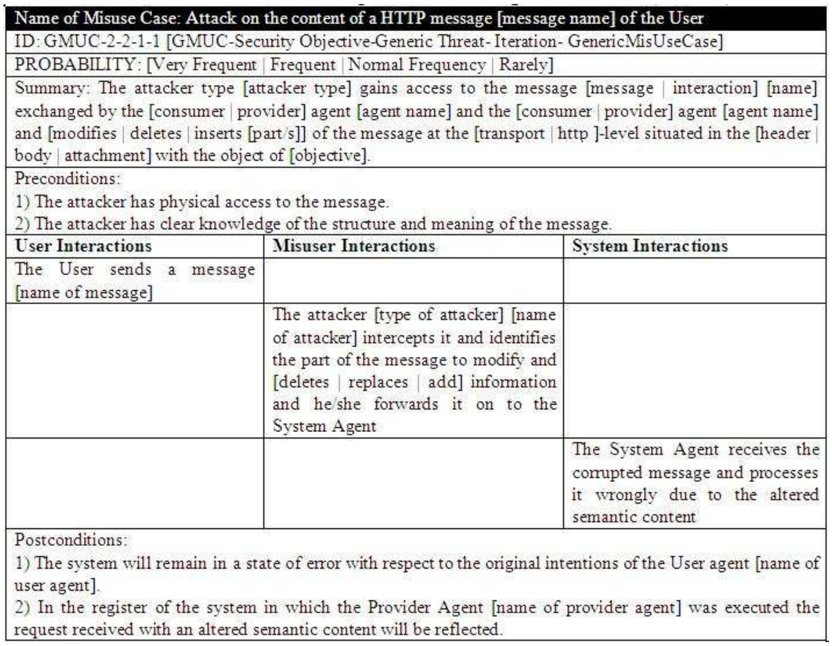
Finally, with the former information we have achieved in this activity, we will constitute the first version of the *Security Problem Definition Document* with the help of the CC assurance classes (CC class ASE). As it is in this document where the assumptions are written down, we would like to reflect the fact that we do not take into consideration, because it is not the main object of this specific work, possible attacks on the provider and consumer organizations, on the network infrastructures or on the infrastructure in use, along with other elements at an organizational level (and not only system-level elements).

3.5 Activity 5: Risk Assessment

Having identified the threats, we shall now go on to determine the probability of each threat and to assess its impact and risk. In order to carry out this task, we will use a technique proposed by the guide of techniques of MAGERIT [11] and which is based on tables to analyse impact and risk of threats.

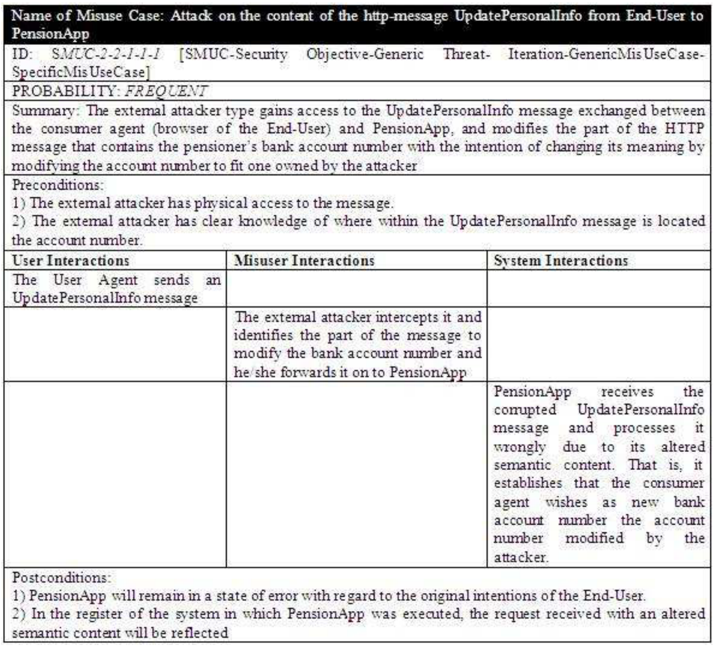
For the time being we are going to evaluate risk and impact with five possible values: Very Low, Low, Medium, High and Very High. The likelihood of a threat could be: Very Frequent (daily event), Frequent (monthly event), Normal Frequency (once a year), Rarely (once in several years). We have therefore to produce a table of threats, attacks (misuse cases: MUC) and risks to register the evaluation of impact and risk regarding the threats we have identified. In Table 3 we will present an example of the analysis of the risk of one threat previously detailed in the former activity. All of this is captured in the *Risk Assessment Document* which will be also refined in subsequence iterations.

Table 1. Generic Threat Specification using misuse cases (GMUC)



Name of Misuse Case: Attack on the content of a HTTP message [message name] of the User		
ID: GMUC-2-2-1-1 [GMUC-Security Objective-Generic Threat- Iteration- GenericMisUseCase]		
PROBABILITY: [Very Frequent | Frequent | Normal Frequency | Rarely]		
Summary: The attacker type [attacker type] gains access to the message [message | interaction] [name] exchanged by the [consumer | provider] agent [agent name] and the [consumer | provider] agent [agent name] and [modifies | deletes | inserts [part/s]] of the message at the [transport | http]-level situated in the [header | body | attachment] with the object of [objective].		
Preconditions: 1) The attacker has physical access to the message. 2) The attacker has clear knowledge of the structure and meaning of the message.		
User Interactions	**Misuser Interactions**	**System Interactions**
The User sends a message [name of message]		
	The attacker [type of attacker] [name of attacker] intercepts it and identifies the part of the message to modify and [deletes | replaces | add] information and he/she forwards it on to the System Agent	
		The System Agent receives the corrupted message and processes it wrongly due to the altered semantic content
Postconditions: 1) The system will remain in a state of error with respect to the original intentions of the User agent [name of user agent]. 2) In the register of the system in which the Provider Agent [name of provider agent] was executed the request received with an altered semantic content will be reflected.		

Table 2. Specific Threat specification



Name of Misuse Case: Attack on the content of the http-message UpdatePersonalInfo from End-User to PensionApp		
ID: *SMUC-2-2-1-1-1* [SMUC-Security Objective-Generic Threat- Iteration-GenericMisUseCase-SpecificMisUseCase]		
PROBABILITY: *FREQUENT*		
Summary: The external attacker type gains access to the UpdatePersonalInfo message exchanged between the consumer agent (browser of the End-User) and PensionApp, and modifies the part of the HTTP message that contains the pensioner's bank account number with the intention of changing its meaning by modifying the account number to fit one owned by the attacker		
Preconditions: 1) The external attacker has physical access to the message. 2) The external attacker has clear knowledge of where within the UpdatePersonalInfo message is located the account number.		
User Interactions	**Misuser Interactions**	**System Interactions**
The User Agent sends an UpdatePersonalInfo message		
	The external attacker intercepts it and identifies the part of the message to modify the bank account number and he/she forwards it on to PensionApp	
		PensionApp receives the corrupted UpdatePersonalInfo message and processes it wrongly due to its altered semantic content. That is, it establishes that the consumer agent wishes as new bank account number the account number modified by the attacker.
Postconditions: 1) PensionApp will remain in a state of error with regard to the original intentions of the End-User. 2) In the register of the system in which PensionApp was executed, the request received with an altered semantic content will be reflected		

Table 3. Table of Threats, Attacks and Risks

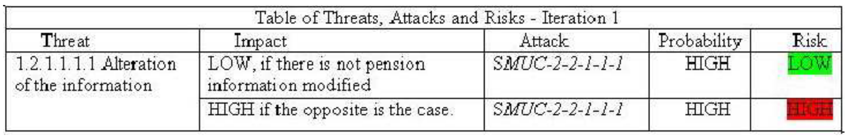


Table of Threats, Attacks and Risks - Iteration 1				
Threat	Impact	Attack	Probability	Risk
1.2.1.1.1.1 Alteration of the information	LOW, if there is not pension information modified	*SMUC-2-2-1-1-1*	HIGH	LOW
	HIGH if the opposite is the case.	*SMUC-2-2-1-1-1*	HIGH	HIGH

3.6 Activity 6: Elicit Security Requirements

In order to derive security requirements, each security objective is analysed for possible relevance together with its threats which imply more risk, so that the suitable security requirements or the suitable cluster of security requirements that mitigate the threats at the necessary levels with regard to the risk assessment are selected. First of all, we will use domain knowledge to transform the entities described in the security objectives into entities in the functional requirement. In this case, it is straightforward, the security objectives refer to information and we know that it is pension information or pensioner information in the context of the functional requirements.

Then, we will transform the security objectives (Confidentiality, Integrity, Availability, Authenticity, Accountability) into constraints on the operations that are used in functional requirements. Additionally, we will search in the CC security functional requirements catalogue (which has been previously introduced together with the CC assurance requirements into the SRR) security requirements which mitigate the threats that can prevent the security objective from being achieved, therefore in this case, we will search for ensuring the integrity, availability, authenticity and accountability of PensionApp. Moreover, we will search in the CC security assurance requirements catalogue to determine the assurance requirements which ensure the secure development of the IS.

The security requirements (SR) that we identify are the following ones:

– SR1: The security functions of PensionApp shall use *cryptography* [assignment: *cryptographic algorithm* and *key sizes*] to protect confidentiality of pension information provided by PensionApp to an EndUser. (CC requirement FCO_CED.1.1)
– SR2: The security functions of PensionApp shall identify and authenticate an EndUser by using *credentials* [assignment: *challenge-response technique based on exchange of encrypted random nonces, public key certificate*] before an EndUser can bind to the shell of PensionApp. (CC requirements FIA_UID.2.1 & FIA_UAU.1.1)
– SR3: When PensionApp transmits pension or pensioner's information to EndUser, the security functions of PensionApp shall provide that user with the *means* [assignment: *digital signature*] to detect [selection: modification, deletion, insertion, replay, other integrity] anomalies. (CC requirement FCO_IED.1.1)
– SR4: The security functions of PensionApp shall ensure the availability of the information provided by PensionApp to an EndUser within [assignment:

a defined availability metric] given the following conditions [assignment: conditions to ensure availability]. (CC requirement FCO_AED.1.1)
– SR5: The security functions of PensionApp shall require evidence that PensionApp has pension information to an EndUser and he/she has received the information. (CC requirement FCO_NRE.1.1)
– SR6: The security functions of PensionApp shall store an audit record of the following events [selection: *the request for pension information, the response of PensionApp*] and each audit record shall record the following information: date and time of the event, [selection: *success, failure*] of the event, and EndUser identity. (CC requirements FAU_GEN)

Due to the former security requirements the first functional requirements (Req1 and Req2) have to be updated, so that they will be as follows:

– Req 1': On request from an EndUser, the system shall display information about his/her pension. This request shall include the social se-curity number of the EndUser and the EndUser's Credentials.
– Req 2': On request-2 from an EndUser, the system shall update the personal information of the pensioner. This request shall include the social security number of the EndUser and the EndUser's Credentials and the changed personal dates.

In Table 4 we will present an example of a *Generic Security Requirement specification* using security use cases as a method of specification. Finally, the *Security Requirements Specification Document* is written in this activity and it will be refined in subsequence iterations because we try to avoid unnecessarily and prematurely architectural/design mechanisms specification.

3.7 Activity 7: Categorize and Prioritize Requirements

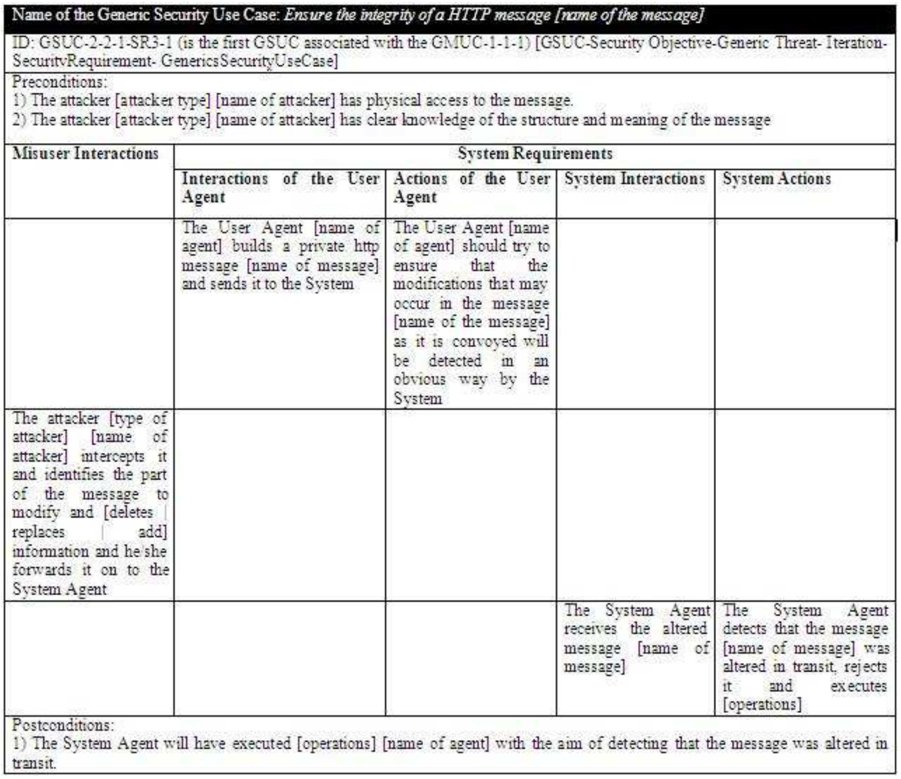
According to the impact and the likelihood of the threats, that is according to the risk, we will rank the security requirements as follows: 1- SR1; 2- SR2; 3- SR3; 4- SR5 and SR6; 5- SR4.

3.8 Activity 8: Requirements Inspection

In this activity, we will generate the *Validation Report*, thereby we will review the quality of the previous work with the help of the CC assurance requirements, these assurance requirements will result from the determined EAL, which was agreed with the stakeholders in the first activity, although it could be modified in subsequent iterations. Supposing we agreed EAL1 (functionally tested) the assurance components that we will use will be presented in the Table 5.

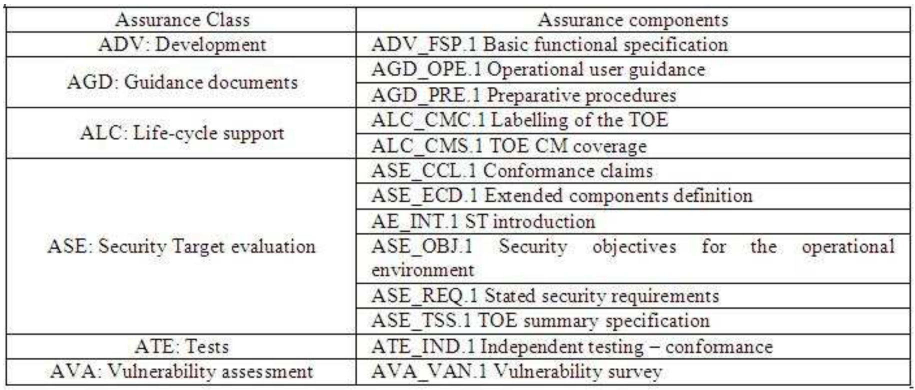
Then we will write the first version of the *Security Requirements Rationale Document* with the help of the CC assurance classes (CC class ASE), showing that if all security requirements are satisfied and all security objectives are achieved, the security problem defined previously is solved: all the threats are countered, the organizational security polices are enforced and all assumptions are upheld.

Table 4. Generic Security Requirement Specification



Name of the Generic Security Use Case: ***Ensure the integrity of a HTTP message [name of the message]***				
ID: GSUC-2-2-1-SR3-1 (is the first GSUC associated with the GMUC-1-1-1) [GSUC-Security Objective-Generic Threat- Iteration-SecurityRequirement- GenericsSecurityUseCase]				
Preconditions: 1) The attacker [attacker type] [name of attacker] has physical access to the message. 2) The attacker [attacker type] [name of attacker] has clear knowledge of the structure and meaning of the message				
Misuser Interactions	**System Requirements**			
	Interactions of the User Agent	**Actions of the User Agent**	**System Interactions**	**System Actions**
	The User Agent [name of agent] builds a private http message [name of message] and sends it to the System	The User Agent [name of agent] should try to ensure that the modifications that may occur in the message [name of the message] as it is convoyed will be detected in an obvious way by the System		
The attacker [type of attacker] [name of attacker] intercepts it and identifies the part of the message to modify and [deletes | replaces | add] information and he/she forwards it on to the System Agent				
			The System Agent receives the altered message [name of message]	The System Agent detects that the message [name of message] was altered in transit, rejects it and executes [operations]
Postconditions: 1) The System Agent will have executed [operations] [name of agent] with the aim of detecting that the message was altered in transit.				

Table 5. EAL 1 Common Criteria Assurance classes and components



Assurance Class	Assurance components
ADV: Development	ADV_FSP.1 Basic functional specification
AGD: Guidance documents	AGD_OPE.1 Operational user guidance
	AGD_PRE.1 Preparative procedures
ALC: Life-cycle support	ALC_CMC.1 Labelling of the TOE
	ALC_CMS.1 TOE CM coverage
ASE: Security Target evaluation	ASE_CCL.1 Conformance claims
	ASE_ECD.1 Extended components definition
	AE_INT.1 ST introduction
	ASE_OBJ.1 Security objectives for the operational environment
	ASE_REQ.1 Stated security requirements
	ASE_TSS.1 TOE summary specification
ATE: Tests	ATE_IND.1 Independent testing – conformance
AVA: Vulnerability assessment	AVA_VAN.1 Vulnerability survey

3.9 Activity 9: Repository Improvement

We will store in the SRR the new elements, in this case in this iteration the Generic and Specific Threats and Requirements which were developed in the activities 4 and 6 will be stored. After all, we will write the *Security Target document* of the CC. This activity will be performed coinciding with the milestone at the end of each phase of the UP.

4 Lessons Learned

Among the most important lessons learned we may stand out from the case study presented above we can highlight the following ones:

– The application of this case study has allowed us to improve and refine the following activities of SREP: identification of security objectives, identification of threats and elicitation of requirements.
– Tool support is critical for the practical application of this process in large-scale software systems due to the number of handled artefacts and the several iterations that have to be carried out.
– As it is an iterative and incremental security requirements engineering process, we have realized that this philosophy lets us take into account changing requirements, facilitates reuse and correct errors over several iterations, risks are discovered and mitigated earlier, and the process itself can be improved and refined along the way.
– Regarding to the experience with the Common Criteria, we have realized that it is sometimes difficult to find the right meaning of the CC requirements, it would be easier if the CC provides examples for each security requirement. However the CC provide us with an important help in order to treat security requirements in a systematic way, in spite of the fact that CC requirements have complex dependencies and the CC does not provide us with any method/guide to include them into the software development process, so that a modification in one document often leads to modify several other documents.

5 Conclusions

In our present so-called Information Society the development of more and more sophisticated approaches to ensuring the security of information is becoming a need. In this paper we demonstrate how the security requirements for a security critical IS can be obtained in a guided and systematic way by applying SREP. Starting from the concept of iterative software construction, we propose a microprocess for the security requirements analysis, made up of nine activities, which are repeatedly performed at each iteration throughout the iterative and incremental development, but with different emphasis depending on where the iteration is in the lifecycle. Therefore the contribution of this work is that of providing a standard-based process that deals with the security requirements at the early stages of software development in a systematic and intuitive way, which is based on the reuse of security requirements, by providing a Security Resources Repository (SRR), together with the integration of the Common Criteria (ISO/IEC 15408) into software development lifecycle. Moreover, it also conforms to ISO/IEC 17799:2005 with regard to security requirements (sections: 0.3, 0.4, 0.6 and 12.1). Hence, it is a very helpful process for security critical Information Systems. Further work is also needed to provide a CARE (Computer-Aided Requirements Engineering) tool which supports the process, as well as a refinement of the theoretical approach by proving it with more real case studies in order to complete and detail more SREP.

Acknowledgements

This paper has been produced in the context of the DIMENSIONS (PBC-05-012-2) Project of the Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha along with the FEDER and the CALIPO (TIC2003-07804-CO5-03) and the RETISTIC (TIC2002-12487-E) projects of the Dirección General de Investigación del Ministerio de Ciencia y Tecnología.

References

1. Baskeville, R. "The development duality of information systems security". *Journal of Management Systems*, 1992, 4(1): p. 1-12.
2. Booch, G., Rumbaugh, J., and Jacobson, I. "The Unified Software Development Process". *ed. Addison-Wesley*, 1999.
3. Breu, R., Burger, K., Hafner, M., and Popp, G. "Towards a Systematic Development of Secure Systems". *Proceedings WOSIS 2004*, 2004: p. 1-12.
4. Firesmith, D.G. "Engineering Security Requirements". *Journal of Object Technology*, 2003. 2(1): p. 53-68.
5. Firesmith, D.G. "Security Use Cases". *Journal of Object Technology*, 2003. p. 53-64.
6. ISO/IEC_JTC1/SC27. "Information technology - Security techniques - Management of information and communications technology security - Part 1: Concepts and models for information and communications technology security management". *ISO/IEC 13335*, 2004.
7. ISO/IEC_JTC1/SC27. "Information technology - Security techniques - Code of practice for information security management". *ISO/IEC 17799*, 2005.
8. ISO/IEC_JTC1/SC27. "Information technology - Security techniques - Evaluation criteria for IT security". *ISO/IEC 15408:2005 (Common Criteria v3.0)*, 2005.
9. Kim., H.-K. "Automatic Translation Form Requirements Model into Use Cases Modeling on UML". *ICCSA 2005, LNCS*, 2005: p. 769-777.
10. Kotonya, G. and Sommerville, I. "Requirements Engineering Process and Techniques". *Hardcover ed*, 1998. 294.
11. MAP. "Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información (MAGERIT - v 2)". *Ministry for Public Administration of Spain*, 2005.
12. Massacci, F., Prest, M., and Zannone, N. "Using a security requirements engineering methodology in practice: The compliance with the Italian data protection legislation". *Computers Standards and Interfaces, 27*, 2005, p.445-455.

13. Dermott, J. and Fox, C."Using Abuse Case Models for Security Requirements Analysis". *Annual Computer Security Applications Conference*, Phoenix (AZ), 1999.
14. Mellado, D., Fernández-Medina, E., and Piattini, M."A Common Criteria Based Security Requirements Engineering Process for the Development of Secure Information Systems". *Computer Standards and Interfaces* , 2006.
15. Mellado, D., Fernández-Medina, E., and Piattini, M."A Comparative Study of Proposals for Establishing Security Requirements for the Development of Secure Information Systems". *The 2006 International Conference on Computational Science and its Applications (ICCSA 2006), Springer LNCS 3982* , 2006. 3: p. 1044-1053.
16. Mouratidis, H., Giorgini, P., Manson, G., and Philp, I."A Natural Extension of Tropos Methodology for Modelling Security". *Workshop on Agent-oriented methodologies, at OOPSLA 2002* ,Seattle (WA), 2003.
17. Popp, G., Jürjens, J., Wimmel, G., and Breu, R."Security-Critical System Development with Extended Use Cases". *10th Asia-Pacific Software Engineering Conference* , 2003, p. 478-487.
18. Sindre, G., Firesmith, D.G., and Opdahl, A.L."A Reuse-Based Approach to Determining Security Requirements". *9th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'03)* , Austria, 2003.
19. Toval, A., Nicolás, J., Moros, B., and García, F."Requirements Reuse for Improving Information Systems Security: A Practitioner's Approach". *Requirements Engineering Journal* , 2001, p. 205-219.
20. Walton, J.P."Developing a Enterprise Information Security Policy". *ACM Press: Proceedings of the 30th annual ACM SIGUCCS conference on User services.* , 2002.
21. Yu, E."Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering". *A3rd IEEE International Symposium on Requirements Engineering (RE'97)* , 1997, p. 226-235.

Modeling and Evaluating the Survivability of an Intrusion Tolerant Database System

Hai Wang and Peng Liu

College of Information Sciences and Technology
Pennsylvania State University
University Park, PA 16802, USA
{haiwang, pliu}@ist.psu.edu

Abstract. The immaturity of current intrusion detection techniques limits the traditional security systems in surviving malicious attacks. Intrusion tolerance approaches have emerged to overcome these limitations. Before intrusion tolerance is accepted as an approach to security, there must be quantitative methods to measure its survivability. However, there are very few attempts to do quantitative, model-based evaluation of the survivability of intrusion tolerant systems, especially in database field. In this paper, we focus on modeling the behaviors of an intrusion tolerant database system in the presence of attacks. Quantitative measures are proposed to characterize the capability of a resilient database system surviving intrusions. An Intrusion Tolerant DataBase system (ITDB) is studied as an example. Our experimental results validate the models we proposed. Survivability evaluation is also conducted to study the impact of attack intensity and various system deficiencies on the survivability.

1 Introduction

Although intrusion tolerance techniques, which gain impressive attention recently, are claimed to be able to enhance the system survivability, survivability evaluation models are largely overlooked in the previous research. Quantifying survivability metrics of computer systems is needed and important to meet the user requirements and compare different intrusion tolerant architectures. Efforts aimed at survivability evaluation have been based on classic reliability or availability models.

The work described in this paper is motivated by the limitations of using the evaluation criteria for availability to evaluate survivability. The evaluation criteria for system availability are quantified by availability modeling, which has a fairly matured literature as summarized in [1]. However, the availability model cannot be used to quantify the survivability of a security system. Besides the differences between security and fault tolerance, a fundamental reason is because the availability model assumes the “fail-stop” semantics, but the “attack-stop” semantics probably can never be assumed in trustworthy data processing systems, not only because of the substantial detection latency, but also because of the needs for degraded services.

The goal of this paper is taking the first step to develop a survivability evaluation model that can systematically address the inherent limitations of classic availability evaluation models in measuring survivability. The approach we proposed is using a state transition graph to model an intrusion tolerant database system. We attempt to model the system in a modular way, so that it can be easily adapted to a wide variety of intrusion tolerant database systems. Quantitative measures are proposed to characterize the capability of a resilient database system surviving intrusions. Furthermore, we are interested in understanding the impact of existing system deficiencies and attack behaviors on the survivability. In this paper, we take the first step to do detailed, quantitative evaluation of the survivability of intrusion tolerant database systems and the impact of system deficiencies and attack behaviors on it.

In particular, the main contributions of this paper are four-fold:

1. We extend the classic availability model to a new survivability (evaluation) model. Comprehensive state transition approaches are applied to study the complex relationships among states and their transition structures encoding sequential response of intrusion tolerant database systems facing attacks.
2. Novel quantitative survivability evaluation metrics are proposed by us. Mean Time to Attack (MTTA), Mean Time to Detection (MTTD), Mean Time to Marking (MTTM), and Mean Time to Repair (MTTR) are proposed as basic measures of survivability. We find that there is a natural mapping between the MTTA-MTTD-MTTM-MTTR model and the steady state probabilities of the system in state transition modeling. This mapping not only provides valuable insights on why the MTTA-MTTD-MTTM-MTTR model can measure survivability, but also provides a convenient way to use mathematical analysis to quantify survivability. Based on the MTTA-MTTD-MTTM-MTTR model, this survivability measuring methodology is no longer ad hoc.
3. To validate the survivability models we proposed, a representative intrusion tolerant database system, ITDB [2], is studied as an empirical example. A real testbed is established to conduct comprehensive validation experiments running TPC-C benchmark transactions. Experimental results show the validity of the survivability models we proposed.
4. To further evaluate the security of ITDB, we have done an empirical survivability evaluation, where maximum-likelihood methods are applied to estimate the values of the parameters used in our state transition models. The impacts of existing system deficiencies and attack behaviors on the survivability are then studied using quantitative measures we defined.

The rest of the paper is organized as follows. In Section 2, we give an overview of the ITDB framework. In Section 3, a series of state transition models are proposed. In Section 4, quantitative measures of database system survivability are proposed. The experiments are conducted in Section 5 to validate the models we established. Survivability evaluation results are reported in Section 6. In Section 7, we discuss the related work. We conclude our paper in Section 8.

2 ITDB: An Motivating Example

ITDB is motivated by the following practical goal: "after the database is damaged, automatically locate the damaged part, contain and repair it as soon as possible, so that the database can continue being useful in the face of attacks or intrusions". The major components of ITDB are shown in Figure 1. Note that in [3], a comprehensive ITDB system has been proposed. In this paper, we only focus on important components of ITDB, namely the damage containment and recovery subsystems. In the rest of this section, we give a brief overview of the functions of major ITDB components.

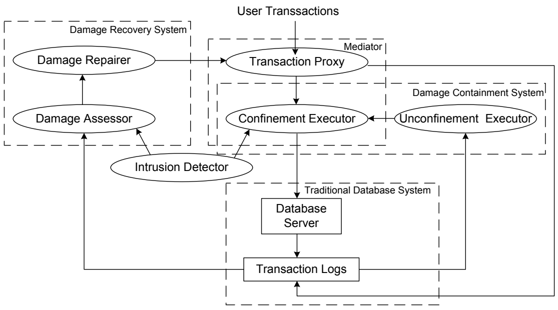


Fig. 1. Basic ITDB System Architecture

The Mediator subsystem functions as a "proxy" for each user transaction and transaction processing call to the database system. Through this proxy, ITDB is able to keep useful information about user transactions, such as information about transactions' read and write operations, which is important to generate the corresponding logs for damage recovery and containment. This part is the foundation of the whole ITDB system. All other subsystems of ITDB rely on this part.

Traditional damage containment approaches are one-phase. An item o will not be contained until it is identified as damaged. However, significant damage assessment latency can cause the damage on o spreading to many other data items before o is contained. To overcome this limitation, ITDB uses a novel technique called multi-phase damage containment as an alternative. This approach has one containing phase, which instantly contains the damage that might have been caused by an intrusion as soon as the intrusion is identified, and one or more later on uncontaining phases, denoted *containment relaxation*, to uncontain the items that are mistakenly contained during the containing phase.

The damage recovery subsystem has the responsibility to perform accurate damage assessment and repair. To do this job, first, the damage recovery subsystem retrieves reported malicious transaction messages from the intrusion detection subsystem. ITDB then traces damage spreading by capturing the dependent-upon relationship among transactions. ITDB repairs the damage caused by T_i using a special *cleaning* transaction which restores each contaminated data item to its latest undamaged version.

The intrusion detection subsystem has the responsibility to detect and report malicious transactions to the damage containment and recovery subsystems. It uses the trails kept in the logs and some relevant rules to identify malicious transactions.

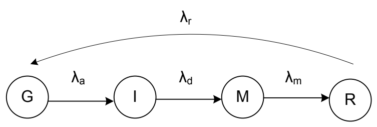
3 Modeling Intrusion Tolerant Database Systems

To analyze and evaluate the survivability of an intrusion tolerant database system, a quantitative evaluation model is required. A variety of modeling techniques can be applied in the research of survivability study. Deterministic models are quite limited in the stochastic behavior. State transition models are much more comprehensive. All possible system states can be captured by state transition models. In this section, we apply state transition models to explore the complex relationships and transition structure of an intrusion tolerant database system.

3.1 Basic State Transition Model

Figure 2 shows the basic state transition model of an intrusion tolerant database system. Traditional computer security leads to the design of systems that rely on prevention to attacks. If the strategies for prevention fail, the system is brought from good state G into the infected state I during the penetration and exploration phases of an attack. If the attack is detected successfully, intrusion tolerance system picks up where attack prevention leaves off. The system enters the containment state M. In this state, all suspicious data items are contained. After marking all the damage made by the attack, undamaged items are released and the system enters to the recovery state R. The repair process will compensate all the damage and the system returns to the good state G. The four phases which are attack penetration, error detection, attack containment, damage assessment and error recovery, describe the basic phenomena that each intrusion tolerant system will encounter. These can and should be the basic requirement for the design and implementation of an intrusion tolerant database system.

Parameters in Figure 2 are: $1/\lambda_a$ is the mean time to attacks (MTTA), the expected time for the system to be corrupted; $1/\lambda_d$ is the mean time to detect (MTTD), the expected time for the intrusion to be detected; $1/\lambda_m$ is the mean time to mark (MTTM), the expect time for the system to mark "dirty" data



$$\begin{tikzcd} G \arrow[r, "\lambda_a"] & I \arrow[r, "\lambda_d"] & M \arrow[r, "\lambda_m"] & R \arrow[lll, bend right, "\lambda_r"'] \end{tikzcd}$$

Fig. 2. Basic State Transition Model

items; $1/\lambda_r$ is the mean time to repair (MTTR), the expect time for the system to repair damaged data items.

3.2 Intrusion Detection System Model

As an important part of an intrusion tolerant system, the Intrusion Detection System (IDS) is largely ignored in intrusion tolerant system modeling. [4] assumes that the IDS can report intrusion without delay or false alarm. Some works only consider part of IDS parameters, like true positive [5]. In this part, we will integrate a comprehensive model of IDS into the whole system.

False alarm rate and detection probability are widely used to evaluate the performance of an IDS in either networking [6] or database field [7]. Detection latency, so called *detection time* in [8], is another metrics to evaluate an IDS. We define detection latency as the duration that elapses from the time when an attack compromises a database system successfully to the time when the IDS identifies the intrusion. All these three metrics are included in our model.

Let T_a and T_{fa}, respectively, denote the times to intrusion and the time to the failure of the IDS. If the IDS fails before the intrusion, then a false alarm is said to have occurred. Let A denote the time to intrusion occurrence. Clearly,

$$A = min\{T_A, T_{fa}\} \tag{1}$$

We assume that T_a and T_{fa} are mutually independent and exponentially distributed with parameter λ_a and α, respectively. Then, clearly, A is exponentially distributed with parameter $\lambda_a + \alpha$.

After the intrusion, it takes a finite time T_d (*detection latency*) to detect the intrusion. We assume that the time to identify one successful intrusion is exponentially distribution with parameter λ_d. For the imperfect detection, we assume that all attacks will be identified by the database administrator eventually. We use state MD and MR to represent the *undetected state* and *manual repair state* respectively. We assume that the detection probability of an IDS to identify a successful intrusion is d. The transition probability that the system transfers from state I to state MD is $(1 - d)$. We assume that the time to manually identify a successful intrusion is exponentially distribution with parameter λ_{md} and the time to manually repair infected data items is exponentially distribution with parameter λ_{mr}. The state transition model considering the deficiencies of the IDS is presented in Figure 3.

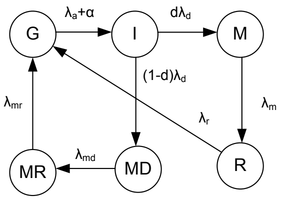


Fig. 3. State Transition Model with IDS

3.3 Damage Propagation and Repair Model

The damage keeps propagating the effect of the intrusion during the detection phase. The purpose of the IDS is to reduce its detection latency. Although damage spreading is a normal phenomenon, little work puts effort on studying the effect of damage propagation on the survivability in their intrusion tolerant system models. In this part, we want to take the first step to study the effect of detection delay on damage propagation, which may affect damage assessment and repair correspondingly.

Let T_{di} denotes the time between the infection of $(i-1)$th and ith data item. Obviously,

$$T_d = \sum_{i=1}^{k} T_{di} \tag{2}$$

where k is the number of infected data items during the detection latency. Let's assume that T_{di} is exponentially distributed with parameter λ_{di} and

$$F_{Di}(t) = 1 - e^{-\lambda_{di} t} \tag{3}$$

As soon as the intrusion is identified, the containing phase instantly contains the damage that might have been caused by an intrusion. At the same time, the damage assessment process begins to scan the contained data items and locate the infected ones. We assume that the time to scan one infected data item is exponentially distributed with parameter λ_m.

After all infected data items are identified via the damage assessment process, the repair system begins to compensate the damage caused by the intrusion. We assume the time to repair one infected data item is exponentially distributed with parameter λ_r

Let $(I : k)$ denote the *infect state* with k infected data items in the database, and $(M : k)$ denote the *mark state* with k infected data need to be located. Figure 4 shows the comprehensive state transition model of ITDB.

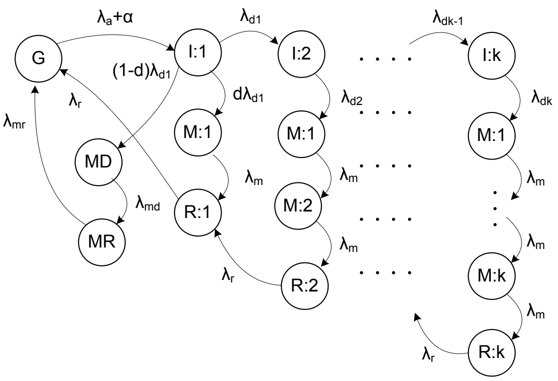


Fig. 4. Comprehensive State Transition Model

4 Survivability Evaluation

Evaluation criteria in trustworthy data processing systems are often referred to as *survivability* or *trustworthiness*. Survivability refers to the capability of a system to complete its mission, in a timely manner, even if significant portions are compromised by attacks or accidents [9]. However, in the context of different types of systems and applications, it can mean many things. This brings a difficulty in the measurement and interpretation of survivability. For a database system, survivability is the quantified ability of a system or subsystem to maintain the integrity and availability of essential data, information, and services. Also a survivable database system should maintain the performance of essential services facing attacks. In this section, based on the models we established in Section 3, quantitative metrics are proposed to facilitate evaluating the survivability of intrusion tolerant database systems from several aspects.

4.1 State Transition Model Analysis

Let $\{X(t), t \geq 0\}$ be a homogeneous finite state Continuous Time Markov Chain (CTMC) with state space S and generator matrix $\mathbf{Q} = [q_{ij}]$. Let $P_i(t) = P\{X(t) = i, i \in S\}$ denote the unconditional probability that the CTMC will be in state i at time t, and the row vector $\mathbf{P}(t) = [P_1, P_2, \cdots, P_n]$ represent the transient state probability vector of the CTMC. The transient behavior of the CTMC can be described by the Kolmogorov differential equation:

$$\frac{d\mathbf{P}(t)}{dt} = \mathbf{P}(t)\mathbf{Q} \tag{4}$$

where $\mathbf{P}(0)$ represents the initial probability vector (at time $t = 0$).

In addition, cumulative probabilities are sometimes of interest. Let $L(t) = \int_0^t P(u)du$; then, $L_i = (t)$ represents the expected total time the CTMC spends in state i during the interval $[0, t)$. $L(t)$ satisfies the differential equation:

$$\frac{d\mathbf{L}(t)}{dt} = \mathbf{L}(t)\mathbf{Q} + \mathbf{P}(0) \tag{5}$$

where $\mathbf{L}(0) = 0$.

The steady-state probability vector $\pi = lim_{t\to\infty}\mathbf{P}(t)$ satisfies:

$$\pi\mathbf{Q} = 0, \sum_{i \in S} \pi_i = 1 \tag{6}$$

By solving the equations 4, 5 and 6, we can get some important survivable metrics of an intrusion tolerant database system.

4.2 Survivability Evaluation Metrics

In our model, survivability is quantified in terms of *integrity* and *availability*. According to survivability, we define integrity in a way different from integrity constraints. In this paper, we define integrity as follow:

Definition 1: Integrity is defined as a fraction of time that all accessible data items in the database are clean.

High integrity means that the intrusion tolerant database system can serve the user with good or clean data at a high probability. Obviously, all data items are clean and accessible in state G. When attacks occur, some data items will be affected. So in state I, part of accessible data items are "dirty". After the intrusion is identified, the ITDB can contain the all damaged data until it finish the repair process. Since the ITDB does selective containment and repair, the database system is still available, and accessible data items are clean during the containment, damage assessment, and repair process.

Consider the model in Figure 2, state space $S = \{G, I, M, R\}$. The generator matrix $\mathbf{Q}$ for the basic state transition model in Section 3.1 is:

$$\mathbf{Q} = \begin{bmatrix} -\lambda_a & \lambda_a & 0 & 0 \\ 0 & -\lambda_d & \lambda_d & 0 \\ 0 & 0 & -\lambda_m & \lambda_m \\ \lambda_r & 0 & 0 & -\lambda_r \end{bmatrix} \tag{7}$$

By solving the equations 5 and 6, we can get:

$$\pi_G = \frac{1/\lambda_a}{1/\lambda_a + 1/\lambda_d + 1/\lambda_m + 1/\lambda_r} = \frac{MTTA}{MTTA + MTTD + MTTM + MTTR}$$
$$\pi_I = \frac{1/\lambda_d}{1/\lambda_a + 1/\lambda_d + 1/\lambda_m + 1/\lambda_r} = \frac{MTTD}{MTTA + MTTD + MTTM + MTTR}$$
$$\pi_M = \frac{1/\lambda_m}{1/\lambda_a + 1/\lambda_d + 1/\lambda_m + 1/\lambda_r} = \frac{MTTM}{MTTA + MTTD + MTTM + MTTR}$$
$$\pi_R = \frac{1/\lambda_r}{1/\lambda_a + 1/\lambda_d + 1/\lambda_m + 1/\lambda_r} = \frac{MTTR}{MTTA + MTTD + MTTM + MTTR}$$

From Definition 1, we can get the integrity for the basic state transition model in Section 3.1:

$$I = \pi_G + \pi_M + \pi_R = \frac{MTTA + MTTM + MTTR}{MTTA + MTTD + MTTM + MTTR} \tag{8}$$

Similarly, we can get the integrity for the comprehensive state transition model we proposed in Section 3.3:

$$I = \pi_G + \sum_{i=1}^{k} \pi_{M_i} + \sum_{i=1}^{k} \pi_{R_i} \tag{9}$$

Availability [1] is defined as a fraction of time that the system is providing service to its users. Since the ITDB does on-the-fly repair and will not stop its service facing attacks, its availability is nearly 100%, which can not show the performance of ITDB clearly. To better evaluate the survivability of ITDB, we define another type of availability, Rewarding-availability:

Definition 2: Rewarding-availability (RA) is defined as a fraction of time that the all clean data items are accessible.

If the clean data can not be accessed, it is a loss of service to users. Rewarding-availability means that the system not only can serve its users, but also do not deny the request for the clean data. ITDB will release the all contained clean data items after damage assessment. For the basic state transition model in Section 3.1, the Rewarding-availability is:

$$RA = \pi_G + \pi_R = \frac{MTTA + MTTR}{MTTA + MTTD + MTTM + MTTR} \tag{10}$$

The Rewarding-availability for the comprehensive state transition model in Section 3.3 is:

$$RA = \pi_G + \sum_{i=1}^{k} \pi_{R_i} \tag{11}$$

5 Empirical Validation

The models we proposed in the above section need to be validated. In this section, we compare the prediction of our model with a set of measured ITDB behaviors facing attacks. For our test bed, we use Oracle $9i$ Server to be the underlying DBMS. The TPC-C benchmark [10] is in general DBMS independent, thus the transaction application can be easily adapted to tolerate the intrusions on a database managed by almost every "off-the-shelf" relational DBMS such as Microsoft SQL Server, Informix, and Sybase.

5.1 Parameters Setting and Estimation

In the models we proposed, some parameters can be controlled by us. In our experiments, the behaviors of attackers, human interaction and the properties of IDS can be controlled by us. So we will set the value of attack hitting rate λ_a, false alarm rate α, detection probability d, detection rate λ_d, manual repair rate λ_{mr} and manual detection rate λ_{md}. We will also vary their value to investigate the impact of them on system survivability.

Assume we generate n attack events and k data items are damaged by the attacks. Let assume the total attack time is A_n, the total detect time is D_k, the total manual detection time is MD_n, and the total manual repair time is MR_n. The transition rates are:

$$\lambda_a = \frac{n}{A_n}, \lambda_d = \frac{k}{D_k}, \lambda_{md} = \frac{(1-d)k}{MD_n}, \lambda_{mr} = \frac{(1-d)k}{MR_n} \tag{12}$$

Some parameters in our model are the characters of ITDB, which are not controlled by us. In this section, we will use the method of maximum-likelihood to produce estimators of these parameters. Assume we observed k scan events

Table 1. Parameter Setting and Estimation

Parameters	Value
Attack Hitting Rate, λ_a	0.5(Low); 1(Moderate); 5(Heavy)
Detect Rate, λ_d	10(Slow); 15(Medium); 20(Fast)
Mark Rate, λ_m	27
Repair Rate, λ_r	22
Manual Detection Rate, λ_{md}	0.02
Manual Repair Rate, λ_{mr}	0.02
False Alarm Rate, α	10%; 20%; 50%
Detection Probability, d	80%; 90%; 99%

and repair events, the total mark time is M_k, and the total repair time is R_k. The maximum-likelihood estimators of λ_m, λ_r are

$$\widetilde{\Lambda}_M = \frac{k}{M_k}, \widetilde{\Lambda}_R = \frac{k}{R_k} \tag{13}$$

Table 1 shows the values of parameter setting and estimation of our experiments.

5.2 Validation

The steady state probability of occupying a particular state computed from the model was compared to the estimated probability from the observed data. The steady state probabilities for the Markov model are computed by using Equation 6. The measured data are estimated as the ratio of the length of time the system was in that state to the total length of the period of observation. The results are shown in Table 2.

Table 2. Comparison of state occupancy probabilities. ($\lambda_a = 0.5$, $\lambda_d = 10$, $\alpha = 10\%$, $d = 90\%$).

State	Observed Value	Value from Model	Difference (%)
G	71.64	72.15	0.7169
I	3.96	3.72	6.4516
M	2.64	2.45	7.7551
R	1.98	1.89	4.7619
U	0.55	0.57	3.5088
M	4.4	4.09	7.5795

It can be seen that the computed values from the model and the actual observed values match quite closely. This validates the model building methodology, and so the Markov model can be taken to model the real system reasonably well.

6 Results

In this section, we use ITDB [2] as an example to study intrusion tolerant database systems' survivability metrics we proposed in section 4. Instead of evaluating the performance of a specified system, we focus on the impact of different system deficiencies on the survivability in the face of attack. Experiments run using different system settings and workloads. The analysis presented here is designed to compare the impact of different parameters of intrusion detection subsystems, such as False Alarm Rate, Detection Latency; and different workload parameters, such as Attack Rates on the relative survivability metrics of ITDB.

6.1 Impact of Attack Intensity

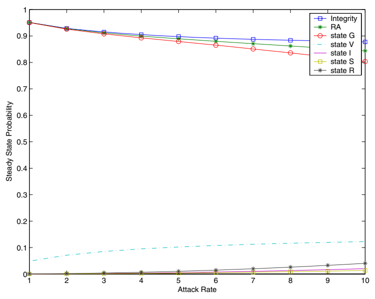
The attack rate can challenge the survivability of an intrusion tolerant system. As an intrusion tolerant system, a key problem is whether ITDB can handle different attack intensity. To answer this question, in this part, we will study the impact of attack rate on survivability of ITDB.

We compare the steady state probabilities of different system configuration of ITDB under different attack rates. In Figure 5(a), an example of a good system, which has a good IDS and fast damage assessment and repair system, is shown. As can be seen, the heavy attacks have little impact on the survivability of ITDB. The damage assessment and repair subsystems can locate and mask the intrusion quickly. As a result, the steady state probabilities of state I, R, and M are very slow. The integrity and rewarding-availability remain at a high level (> 0.8). The only impact of high attack rate is that the probability of ITDB staying at state I is increased. This does not hurt the survivability of ITDB.

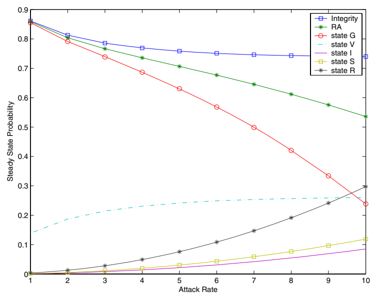
An example of a bad system is shown in Figure 5(b). The high attack rate increases the work load for damage marking and repairing subsystems. As a result, steady state probabilities of state R ($\pi_R > 0.3$) and state M go up quickly. This keeps ITDB busy on analyzing and masking the heavy attacks. However, the system integrity is not impacted by the attacks significantly. The reason is that the ITDB applies the damage containment strategy. This enables the ITDB having the capability to provide clean information to users even facing heavy attacks.

6.2 Impact of False Alarms

False alarm is a key factor to evaluate the performance of an IDS. ITDB adopts the behavior-based intrusion detection techniques. The high false alarm rate is often cited as the main drawback of behavior-based detection techniques since the entire scope of the behavior of an information system may not be covered during the learning phase. High false alarm rate may bring extra workload to the recovery subsystem and waste some system resources. Will ITDB tolerant the relatively high false alarm rates? To answer this question, we will evaluate the impact of false alarms on the steady state of ITDB in this part.

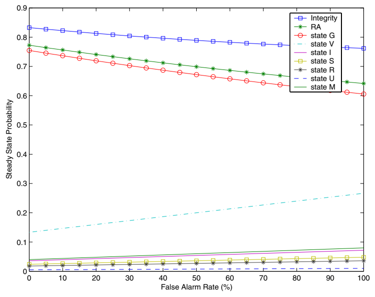


(a) good system ($d = 99\%$, $\alpha = 0.1$, $\lambda_d = 20$)

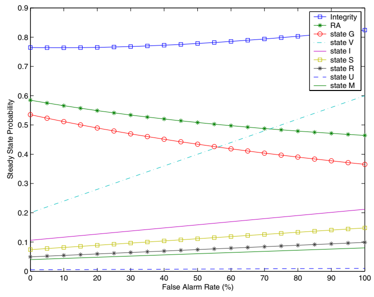


(b) poor system ($d = 80\%$, $\alpha = 0.5$, $\lambda_d = 10$)

Fig. 5. Impact of Attack Intensity



(a) Light Attack ($\lambda_a = 1$, $\lambda_d = 15$, $d = 90\%$)



(b) Heavy Attack ($\lambda_a = 5$, $\lambda_d = 15$, $d = 90\%$)

Fig. 6. Impact of False Alarm Rate

Figure 6(a) shows the variation of steady state probabilities when ITDB is under light attacks ($\lambda_a = 1$). ITDB maintains the integrity (> 0.85) and rewarding-availability (> 0.6) at a high level, even though facing a nearly 100% false alarm rate. This indicates that the system can tolerate a high false alarm rate under light attacks. Also the steady state probabilities of state I, M, R are at a very low level (< 0.1). This indicates that the system can contain, locate, and repair the attacks efficiently and quickly. Another case that ITDB is under heavy attacks ($\lambda_a = 5$) is shown in Figure 6(b). As can be seen, high false alarm brings pressure on ITDB. The steady state probability of state I, π_D, is higher than the probability state G, π_G, when false alarm rate is higher than 60%. The heavy attacks and extra load brought by false alarms increase the steady state probabilities of state I, M, and R. These mean that ITDB spends much more time on state I and keeps busy on analyzing and repairing the damage. The rewarding-availability decreases as the damage containment and assessment process becomes longer.

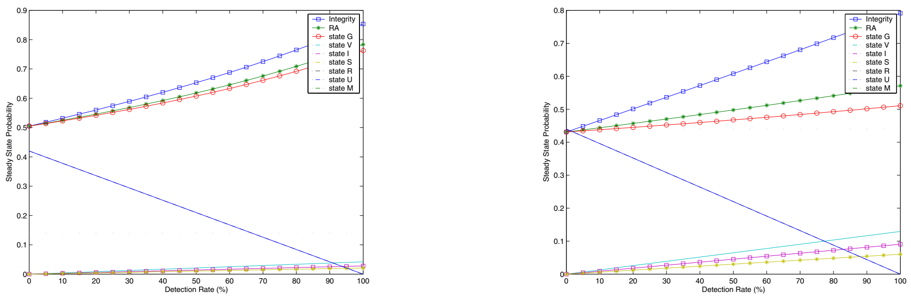
At the same time, the system still can maintain the integrity (> 0.85) at a high level. This means that the probability that the system can provide clean data to some users is high.

6.3 Impact of Detection Probability

Detection probability is another important feature to measure the performance of an intrusion detector. In this section, we will study the impact of detection probabilities on the survivability under different attack intensity.

Figure 7(a) shows that ITDB is under light attack ($\lambda_a = 1$). When detection rate is 0%, the system totally depends on manual detection. Since the manual detection requires human intervention, it takes a relatively long time to detect the intrusion manually. As a result, ITDB has a high probability (> 0.4) staying at state MD and a low probability staying at state G when $d = 0$. The integrity and rewarding-availability are also at a low level (≈ 0.5). The steady state probability of state MD goes back to 0 when the detection probability is 100%. The steady state probabilities of state M and R go up while the detection probability is increasing. This indicates that, with more attacks are identified by the IDS, the system will spend more time on damage assessment and recovery. Since the manual repair is much slower than the repair subsystem of ITDB, the rewarding-availability and integrity go up while the detection probability is increasing. When ITDB faces a heavy attack as shown in Figure 7(b), low detection rate hurts the performance of ITDB. The steady state probability of state G, π_G is lower than 0.5.

Compared with the false alarms, the impact of detection probability on the survivability of an intrusion tolerant database system is severer. The variance of integrity and rewarding-availability is less than 0.2 when the detection probability changes from 0% to 100%, while the variance is nearly 0.4 when changing false alarm rate from 0% to 100%. One reason is that the high false alarms will bring extra load to the security system to contain and repair unaffected data items,



(a) Light Attack ($\lambda_a = 1$, $\alpha = 0.2$, $\lambda_d = 15$)

(b) Heavy Attack ($\lambda_a = 5$, $\alpha = 0.2$, $\lambda_d = 15$)

Fig. 7. Impact of Detection Rate

while low detection probability will bring more work for the administrator to mark and repair the damage manually. If the system can identify and recover the damage faster than manual operation, the impact of low detection probability is severer and more dangerous to the survivability. This result encourages us to consider more on improving the detection probability for the future intrusion tolerant system development.

6.4 Transient Behaviors

Much of the theory developed for solving Markov chain models is devoted to obtaining steady state measures, that is, measures for which the observation interval is "sufficiently large" ($t \to \infty$). These measures are indeed approximations of the behavior of the system for a infinite, but long, time interval, where long means with respect to the interval of time between occurrences of events in the system. However, in some cases the steady state measures are not good approximations for the measures during a relatively "short" period of time.

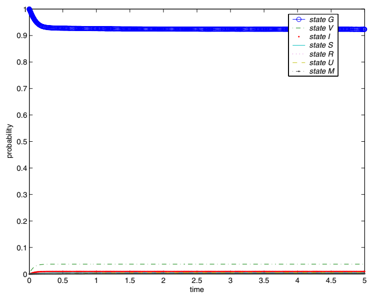
Before reaching the steady state, the system will go through a transient period. If the damage containment and recovery systems are not efficient enough, the system may never reach steady states, or take a very long time. The cumulative time distribution of contain and repair states will be dominant. Even through the steady state probability of good state is high, obviously we can not satisfy the system's performance. The limitation of steady state measures motivates us to observe the transient behaviors of different intrusion tolerant systems in this part. Figure 8 and 9 show the comparison results. We start the system from state G, which means $P_G(0) = 1$.

A better system's behaviors are shown in Figure 8. We assume that a better intrusion tolerant system has a good IDS, which can detect intrusion quickly and have a high detection rate and a low false alarm rate. Damage assessment and repair systems can locate and mask the intrusion quickly. As can be seen in Figure 8(a), a better system reaches steady state quickly. The probability of staying at state G is high, while the probabilities of staying at another states, like state I, R, and M, are very low. From Figure 8(b), we can also find that the cumulative time distribution of staying at state G is dominant, which means the system will spend most of time at good state. Since the damage assessment and repair system can accomplish their tasks quickly, the cumulative time distribution of state I, R, and M are low.

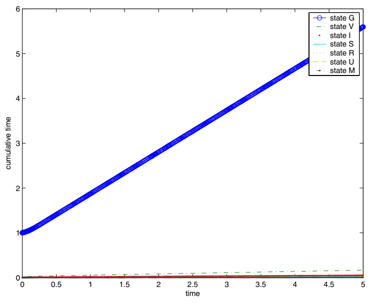
In Figure 9, we give an example of a poor system, which has a slow assessment and repair system. Compared with Figure 8, we can find that it takes a longer time for the system to reach steady states. The cumulative time of state G is not dominant. The system spends more time on damage assessment and repair.

7 Related Works

Despite that intrusion tolerance techniques, which gain impressive attention recently, are claimed to be able to enhance the system survivability, suitable and

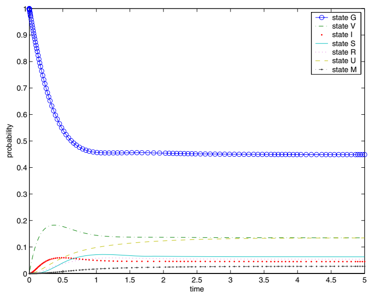


(a) transient probabilities of a good system

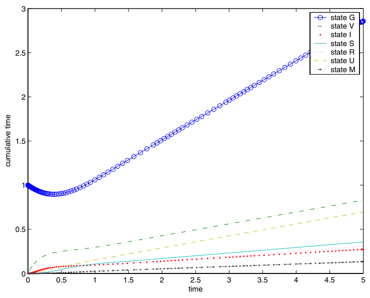


(b) cumulative time distribution of a good system

Fig. 8. Transient Behaviors of a good system



(a) transient probabilities of a poor system



(b) cumulative time distribution of a poor system

Fig. 9. Transient Behaviors of a poor system

precise measures to evaluate the survivability of an intrusion tolerant system are largely missed in the previous research. Most of the research in the literature report and discuss the survivable capability of their work from a qualitative point of view. Little research has proposed the quantitative evaluation metrics of survivability.

In [9] and [11], formal definitions of survivability are presented and compared with related concepts of reliability, availability, and dependability. [11] defined the survivability from several aspects and claimed that the big difference between reliability and survivability is that degraded services of survivable systems are acceptable to users, reliability assumes that the system is either available or not. However, the quantitative measurements of survivability and the level of degraded services are missing in that study.

The attacks and the response of an intrusion tolerant system are modeled as a random process in [5]. Stochastic modeling techniques are used to capture the attacker behavior as well as the system's response to a security intrusion. Quantitative security attributes of the system are proposed in the paper. Steady-state behaviors are used to analyze the security characterization. A security measure called the *mean time (or effort) to security failure* is proposed. However, "good guestimate" values of model parameters were used in their experiments. And the validation of their models is missing in their work.

Efforts for quantitative validation of security have usually been based on formal methods [12]. [13] shows that probabilistic validation through stochastic modeling is an attractive mechanism for evaluating intrusion tolerance. The authors use stochastic activity networks to quantitatively validate an intrusion-tolerant replication management system. Several measures defined on the model were proposed to study the survivability provided by the intrusion tolerant system. The impacts of system parameters variations are studied in that work.

Although several survivability models and corresponding measurements were proposed in the literature, they are limited in evaluating the security attributions of an intrusion tolerant database system. Zhang and Liu [14] take the first step towards delivering database services with information assurance guarantees. In particular, (a) the authors introduce the concept of Quality of Integrity Assurance(QoIA) services; (b) a data integrity model, which allows customers or applications to quantitatively specify their integrity requirements on the services that they want the database system to deliver, is proposed; and (c) the authors present an algorithm that can enable a database system to deliver a set of QoIA services without violating the integrity requirements specified by the customers on the set of services.

An online attack recovery system for work flow is proposed in [4]. The behaviors of the recovery system are analyzed based on a Continuous Time Markov Chain model. Both steady-state and transient behaviors are studied in that paper. Only 'NORMAL', 'SCAN', and 'RECOVERY' three categories of states are considered in the model. The deficiency of intrusion detection and damage propagation are not considered in that model.

In [15], we have done detailed, quantitative evaluation on the impact of intrusion detection deficiencies on the performance and survivability by running TPC-C benchmark. However, only some ad hoc survivability metrics were used. Systematic survivability model and measurements were not proposed in [15].

8 Conclusion

In this paper, we extend the classic availability model to a new survivability model. Comprehensive state transition approaches are applied to study the complex relationships among states and their transition structure encoding sequential response of intrusion tolerant database systems facing attacks. Mean Time to Attack (MTTA), Mean Time to Detection (MTTD), Mean Time to Marking (MTTM), and Mean Time to Repair (MTTR) are proposed as basic measures of survivability. Quantitative metrics integrity and rewarding-availability are defined to evaluate the survivability of intrusion tolerant database systems.

A real intrusion tolerant database system is established to conduct comprehensive experiments running TPC-C benchmark transactions to validate the state transition models we established. Experimental results show the validity of proposed survivability models. To further evaluate the security of ITDB, we have done an empirical survivability evaluation, where maximum-likelihood methods are applied to estimate the values of the parameters used in our state transition models. The impacts of existing system deficiencies and attack behaviors on the survivability are studied using quantitative measures we defined. Our evaluation results indicate that (1) ITDB can provide essential database services in the presence of attacks, and (2) maintain the desired essential survivability properties without being seriously affected by various system deficiencies and different attack intensity.

References

1. Trivedi, K.S.: Probability and statistics with reliability, queuing and computer science applications. John Wiley and Sons Ltd. (2002)
2. Liu, P.: Architectures for intrusion tolerant database systems. In: Proceedings of 18th Annual Computer Security Applications Conference (ACSAC 2002). (2002) 311–320
3. Liu, P., Jing, J., Luenam, P., Wang, Y., Li, L., Ingsriswang, S.: The design and implementation of a self-healing database system. Journal of Intelligent Information Systems (JIIS) **23**(3) (2004) 247–269
4. Yu, M., Liu, P., Zang, W.: Self-healing workflow systems under attacks. In: Proceedings of 24th International Conference on Distributed Computing Systems (ICDCS 2004). (2004) 418–4025
5. Madan, B.B., Goseva-Popstojanova, K., Vaidyanathan, K., Trivedi, K.S.: A method for modeling and quantifying the security attributes of intrusion tolerant systems. Performance Evaluation **56**(1-4) (2004) 167–186
6. Lippmann, R., Fried, D., Graf, I., Haines, J., Kendall, K., McClung, D., Weber, D., Webster, S., Wyschogrod, D., Cunningham, R., Zissman, M.: Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In: Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX). (2000) 12–26
7. Hu, Y., Panda, B.: A data mining approach for database intrusion detection. In: Proceedings of the 2004 ACM Symposium on Applied Computing (SAC). (2004) 711–716
8. Chen, W., Toueg, S., Aguilera, M.K.: On the quality of service of failure detectors. IEEE Transactions on Computers **51**(1) (2002) 13–32
9. Ellison, R.J., Fisher, D.A., Linger, R.C., Lipson, H.F., Longstaff, T.A., Mead, N.R.: Survivability: Protecting your critical systems. IEEE Internet Computing **3**(6) (1999) 55–63
10. TPC: Tpc-c benchmark. http://www.tpc.org/tpcc/ (2004)
11. Knight, J.C., Strunk, E.A., Sullivan, K.J.: Towards a rigorous definition of information system survivability. Volume 1. (2003) 78–89
12. Landwehr, C.E.: Formal models for computer security. ACM Computing Surveys **13**(3) (1981) 247–278

13. Singh, S., Cukier, M., Sanders, W.H.: Probabilistic validation of an intrusion-tolerant replication system. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN 2003). (2003) 615–624
14. Zhang, J., Liu, P.: Delivering services with integrity guarantees in survivable database systems. In: Proceedings of the 17th Annual Working Conference on Data and Application Security. (2003) 33–46
15. Wang, H., Liu, P., Li, L.: Evaluating the impact of intrusion detection deficiencies on the cost-effectiveness of attack recovery. In: Proceedings of 7th International Information Security Conference (ISC 2004). (2004) 146–157

A Formal Framework for Confidentiality-Preserving Refinement

Thomas Santen

Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany
santen@acm.org

Abstract. Based on a system model consisting of processes describing the machine, the honest users and the adversary, this paper introduces an abstract framework of refinement relations preserving existential confidentiality properties for nondeterministic, probabilistic systems. It allows a refinement step to trade functionality between the machine and its environment, thus shifting the conceptual boundary between machine and environment. A refinement also permits the realization to extend the observational means of an adversary. A confidentiality-preserving refinement relation is defined in terms of another, more basic relation that considers deterministic probabilistic processes. An instantiation with an entropy-based confidentiality property illustrates the use of this framework. The relationship to other concepts of secure refinement, in particular to reactive simulatability, is discussed.

1 Introduction

The paradigm of system and software development by stepwise refinement is a long standing one. Although rarely practiced rigorously, it provides the formal semantical justification for modern techniques such as behavioral subtyping for object-oriented inheritance [15], design by contract [21], and model-driven development. In its rigorous form, it has been applied, among others, in safety-critical applications [4].

The general idea of stepwise refinement is to first capture the essential requirements on a system in a concise model, the initial specification, that abstracts from all unnecessary detail and leaves room for subsequent design decisions. In a refinement step, two models, the *(abstract) specification* and the *(concrete) realization* are related by a preorder on models, the *refinement relation*. Compared to the specification, the refinement relation may admit to reduce nondeterminism, change the types of data, or replace atomic actions by sequences of "more primitive" actions. This process terminates with a completely refined model, the *implementation*, which is supposed to be easily transformable to a conventional program. A refinement relation should preserve the essential properties of the specification, be a preorder, and be compositional, which means that replacing a sub-specification by a realization in the context of a model yields a realization of that model, i.e., contexts are monotonic functions with respect to the refinement preorder.

Starting with [11], there is a vast body of research (e.g., [1, 7, 13]) investigating refinement relations that preserve *functional* properties such as deadlock freedom or the observational behavior of an abstract data type. The preservation theorems for those refinement relations are *universal* in the following sense: they guarantee that *any* realization refining a given specification has the same (suitably rephrased) properties as the specification.

Considering the development of secure systems, confidentiality poses particular problems. It is well-known that refinement relations which allow one to reduce nondeterminism [12] do not preserve confidentiality properties, such as possibilistic information flow properties, of nondeterministic specifications. Roscoe [24] called this the *refinement paradox*. Several approaches to deal with this deficiency of classical refinement relations have been proposed. Roscoe et al. [26] avoid the problem by requiring the adversary's view of the system to be deterministic. Similarly, Jürjens [14] distinguishes specification nondeterminism from implementation nondeterminism, and disallows specification nondeterminism whenever it influences the validity of a security property. Mantel [18] shows how refinement operators tailored for specific information flow properties can modify an intended realization such that the resulting realization preserves the given flow property. Ryan and Schneider [27] discuss the effects of nondeterminism on information flow properties in depth. They conceptually distinguish High nondeterminism and system nondeterminism to show where nondeterminism may influence information flow. That distinction somehow reflects the distinction between nondeterminism for specification purposes and the kind of nondeterminism induced by probabilistic choices at run-time.

Contributions. In the present paper, we take a different view to the problem. Accepting that the particular way of resolving nondeterminism influences the confidentiality properties of the resulting realization, we investigate refinement relations that preserve the *existence* if a secure implementation: they keep invariant the existence of a deterministic realization of the given model that satisfies a given confidentiality property.

We consider a system model including the machine[1] to be built as well as its environment. The environment consists of a model of the honest users and a model of the adversary. This allows one to model systems whose security depends on assumptions of user and adversary behavior, i.e., to make these assumptions explicit. This system model is an extension of the one we proposed before [10, 29]. It has some similarity with the one proposed by Backes et. al [3], which we discuss in Section 7.

Our models may use three kinds of choice: External choice, nondeterministic choice, and probabilistic choice. Jürjens' implementation nondeterminism basically is an abstraction of probabilistic choice. We consider it important to explicitly model the probabilistic behavior of systems because confidentiality is inherently probabilistic: a system keeps confidential which one of alternative system behaviors that produce the same observations for an adversary not by just producing the same observations, but their relative probabilities also must be approximately equal. Otherwise the adversary's risk of making a false guess would be (unacceptably) low. The underlying assumption of possibilistic models is that the (unknown) stochastic behavior of the system is such that it produces a sufficiently high risk for the adversary of guessing wrong. To include

[1] Following Jackson [32], we call the "system to be implemented" *machine* in order to distinguish it from the *system* comprising the machine *and* its environment.

probabilities in the analysis, however, does not mean that one needs to exactly know the stochastic behavior of the system: Realistic assumptions together with a robustness of the confidentiality properties that allows for a range of probabilities without destroying security are sufficient.

(Specification) nondeterminism, on the other hand, is important for abstract models, in which specifiers cannot or do not want to prematurely choose between alternative behavior. Furthermore, certain modeling operators in general introduce nondeterminism.

We propose an abstract notion of confidentiality preserving refinement. It builds on behavior refinement, which allows a realization to reduce nondeterminism and to change the data types of input / output data. The realization may also provide additional means for the adversary to observe the machine. The refinement relation is parameterized by another preorder, the *information flow refinement order*, which ensures the existential preservation of a given confidentiality property.

Since the data space and the means of observation may change, the preservation property refers to a *point of reference* describing the model to which the confidentiality property directly refers. In a sequence of refinement steps, the initial specification usually is the point of reference.

The system model and the refinement framework allow for different environments in the specification and the realization. Thus, *trading* of functionality between the environment and the machine can be accomplished in a refinement step.

The initial specification will usually be concise and the machine model may even be deterministic. Data refinements or trading can then produce nondeterminism in more detailed models, which will be resolved by implementation choices in subsequent refinement steps.

Finally, we instantiate the framework with a confidentiality property based on the entropy of classes of indistinguishable behavior, and with an information flow refinement order defined in terms of mutual information.

Overview. Section 2 sketches probabilistic CSP, the process calculus on which our framework is based. Section 3 introduces our general system model, consisting of processes describing the machine and its environment, as well as the adversary capabilities of observing the system. The structure of confidentiality properties is the topic of Section 4, and Section 5 presents the main result of this paper: the abstract definition of confidentiality-preserving refinement and the corresponding preservation theorem. Section 6 instantiates that framework for a specific confidentiality property. The reader may wish to browse this section first to aid understanding the abstract discussions in Sections 3 through 5. Section 7 discusses related work, before the conclusion gives some pointers to ongoing and future research.

2 Probabilistic Communicating Sequential Processes

To formally model the systems we reason about, we use the probabilistic extension PCSP of the process algebra CSP [25] which Morgan et al. [22] have proposed. We use PCSP because it integrates probabilistic choice with nondeterministic and external choice, and its semantics, in particular the semantics of probabilistic choice, is centered around the concept of refinement. This supports well our investigation of the relationship of refinement and confidentiality.

CSP. A *process* P produces sequences of *events*, called *traces*. An event $c.d$ consists of a channel name c and a data item d. Two processes can *synchronize* on a channel c by transmitting the same data d over c. If one process generates an event $c.d$ and the other generates an event $c.x$, where x is a variable, both processes exchange data when synchronizing on channel c: the value of x becomes d. The set of traces of P is $traces(P)$. The length of a trace t is $\#t$.

The process $e \rightarrow P$ first generates event e, and behaves like P afterwards. The process $P \,|[\, X \,]|\, Q$ is a parallel composition of P and Q synchronized on the channels in X: if P or Q generate events on channels not in X, then those events appear in an arbitrary order; if a process generates an event on a channel in X, it waits until the other process also generates an event on the same channel; if the data transmitted by both processes are equal (or can be made equal because an event contains a variable), then the parallel composition generates that event, otherwise the parallel composition deadlocks. The composition $P \Vert\!\!\rightarrow_X Q$ asynchronously transmits data from P to Q and synchronizes the processes on the remaining channels, such that the behavior of Q on X cannot influence P. The definition of $P \Vert\!\!\rightarrow_X Q$ involves a buffering process that collects events from P on X and forwards them to Q while blocking any flow of events from Q to P through X.

Although not uniformly definable in terms of the standard CSP operators, $P \Vert\!\!\rightarrow_X Q$ can be constructed for any given P, Q, and X.

In the notion of refinement we use, we are interested in changing data representations of the communicated data (*I/O refinement*), because many effects compromising confidentiality can be described by distinguishing data representations in an implementation that represent the same abstract data item (e.g., different representations of the same natural number). For a relation R on data, the process[2] $P[\![R]\!]_D$ is the process P where each data item a in events of P is replaced by a data item b that is in relation with a, i.e., $a\,R\,b$ holds.

The process $P \setminus X$ is distinguished from P by *hiding* the channels in $X \subseteq \alpha P$, where αP is the set of channels used by P. The traces of $P \setminus X$ are the traces of P where all events over channels in X are removed. The external choice $P \,\Box\, Q$ is the process that behaves like either P or Q, depending on the event that the environment offers. For a family of processes $P(x)$, the process $\bigsqcap P(x)$ nondeterministically behaves like one of the $P(x)$. The process $P \sqcap Q$ nondeterministically behaves like P or like Q.

There are several refinement relations for standard CSP. Most commonly, one uses the failures/divergences refinement. Informally, the process Q refines the process P, written $P \sqsubseteq Q$, if Q is more deterministic and less diverging than P. For details, see [25].

For $n \in \mathbb{N}$, the *finite approximation* $P \downarrow n$ of P behaves like P for the first n events and diverges afterwards. Any process P is characterized by its finite approximations. It

[2] The subscript D indicates that this variant of relational renaming does not change the channel names but only the communicated data.

is their least upper bound with respect to the refinement order: $P = \bigsqcup n : \mathbb{N} \bullet P \downarrow n$. A process F that diverges after n events is called a *finite* process.

The *cone* $P\uparrow$ of a process P is the set of all refinements of P, $P\uparrow = \{Q : CSP \mid P \sqsubseteq Q\}$.

Probabilistic CSP. Morgan et al. [22] extend standard CSP by a probabilistic choice operator: The process $P\,_p\!\oplus Q$ behaves like P with a probability of p, and it behaves like Q with a probability of $1-p$. This view of probabilistic processes does not appeal to the intuition that a process chooses particular behavior (a trace or a failure) probabilistically. It rather emphasizes that a probabilistic process behaves like a standard process with a certain probability. Although it may seem unfamiliar at first sight, this view leads to a smooth integration of probabilistic choice with the other operators of CSP, in particular with nondeterministic choice.

The semantics of PCSP relies on continuous evaluations over a Scott topology of the inductive partial ordering $(CSP, \sqsubseteq)$. We can only present the essential concepts relevant to our work here. See [22] for further detail.

The set of *probabilistic processes PCSP* is the set of continuous evaluations mapping "Scott-open" sets of standard processes to $[0,1]$ over the failure-divergences model of CSP under the refinement order, $(CSP, \sqsubseteq)$. Let P and Q be probabilistic processes in $PCSP$. The process Q *refines* P iff for all Scott-open $Y \subseteq CSP$: $P(Y) \leq Q(Y)$. Since standard processes can be embedded in PCSP and the refinement orders coincide, we write $P \sqsubseteq Q$ for PCSP refinement, too.

For a finite standard process F and a probabilistic process P, we write $F \precsim P$ for the probability that P is a member of $F\uparrow$. If $P \sqsubseteq Q$ then it also holds for all finite $F \in CSP$ that $F \precsim P \leq F \precsim Q$.

For processes P, Q in $PCSP$, and $p \in [0,1]$, the *probabilistic choice* of P and Q is defined for all Scott-open subsets Y of CSP:

$$(P\,_p\!\oplus Q)(Y) = p \cdot P(Y) + (1-p) \cdot Q(Y)$$

Because the cone of a finite process is Scott-open, the following relationship between the probability of refining a finite process and probabilistic choice holds. For P, Q in $PCSP$, finite F in CSP and probability p,

$$F \precsim P\,_p\!\oplus Q = p \cdot (F \precsim P) + (1-p) \cdot (F \precsim Q)$$

Furthermore, any non-recursive probabilistic process can be expressed as a probabilistic choice of finitely many standard processes, because probabilistic choice distributes through all (embedded) operators of CSP.

Finally, we remark that nondeterministic choice generalizes probabilistic choice (for any probability p) and external choice in PCSP, whereas probabilistic choice and external choice are not related by refinement.

$$P \sqcap Q \sqsubseteq \begin{cases} P\,_p\!\oplus Q \\ P \,\Box\, Q \end{cases}$$

The indexed probabilistic choice $\bigoplus_{i:I}^{\mathcal{P}} P_i$ canonically generalizes the binary operator for finite index sets I: this process chooses i – and thus P_i – with probability $\mathcal{P}(i)$.

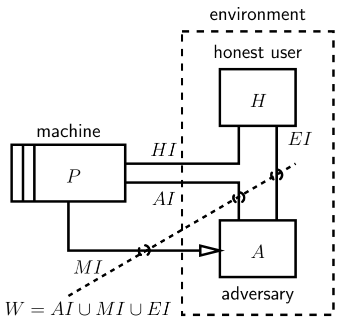


Fig. 1. A system consists of a machine and its environment

Probabilistic Linear Processes. We consider probabilistic confidentiality properties that refer to the probability of a process QE performing a trace t, i.e., the process QE is considered a random variable on traces. This is possible only if QE is deterministic, does not admit external choice, and if the length of the considered traces of QE is bounded by some natural number k. The latter is necessary to distinguish a trace t from a prefix s of t if QE may block after s. Then s and t refer to different probabilistic events.

To resolve nondeterministic and external choices, we consider the set $P^{\top}$ of all *maximal* refinements of P. The members Q of $P^{\top}$ are probabilistic deterministic, i.e., they are free of nondeterminism, but they still admit external choices. The latter are resolved by means of an environment process E that probabilistically resolves external choices of Q and thus serves as a scheduler [30, 6]. We call a process E achieving this in the k-approximation of the composition $Q \parallel_{\rightarrow W} E$ an *admissible* environment. Admissibility is characterized by the fact that $(Q \parallel_{\rightarrow W} E) \downarrow k$ is *probabilistic linear*, i.e., there is a probability function $\mathcal{P}_E$ such that

$$(Q \parallel_{\rightarrow W} E) \downarrow k = \bigoplus^{\mathcal{P}_E}_{t \in traces(Q) \downarrow k} \mathrm{Fin}_k(t)$$

where $\mathrm{Fin}_k(t)$ is the process producing the first k events of t and diverging afterwards. If the length of t is less than k then $\mathrm{Fin}_k(t)$ deadlocks after t.

The environment process E is admissible for an arbitrary process P if it is admissible for all $Q \in P^{\top}$. For $QE = (Q \parallel_{\rightarrow W} E) \downarrow k$ the probability $\mathcal{P}_E(t)$ is $\mathrm{Fin}_k(t) \sqsubseteq QE$. Therefore, we define

$$\mathrm{Pr}^k_{QE}(t) = (\mathrm{Fin}_k(t) \sqsubseteq QE)$$

This is the probability of QE producing exactly the first $\min\{k, \#t\}$ events of t.

3 System Model

A system consists of three PCSP processes, as shown in Figure 1: the machine P, the (honest) user environment H, and the adversary environment A. The machine synchronizes with the adversary via the channels in the (functional) *adversary interface* AI.

Additionally, the adversary can observe the machine on the channels in the *monitoring interface MI*, and it can interact with the honest users on the *environment interface EI*. The sets of channels AI, EI, and MI partition the channels of A. An *adversary model* (P,A,H,HI,AI,MI,EI,k) additionally determines a bound k on the length of system traces that are to be considered. The union of the adversary interfaces $W = AI \cup EI \cup MI$ is the *adversary window*. By convention, we use W (with suitable indexes) to denote adversary windows.

The set $\mathcal{E}_{P,k}^{EI,MI}(H,A)$ comprises all deterministic admissible environment processes.

$$\mathcal{E}_{P,k}^{EI,MI}(H,A) = \{H_d \,|[EI]|\, A_d \mid H_d \in H^{\top} \wedge A_d \in A^{\top} \\ \wedge\, H_d \,|[EI]|\, A_d \text{ admissible for } P, W, k\}$$

To make assertions about the probabilistic behavior of an adversary model means to consider the probabilistic linear processes $(Q \Vert\!\!\to_{MI} E) \downarrow k$ where $Q \in P^{\top}$ and $E \in \mathcal{E}_{P,k}^{EI,MI}(H,A)$. Those processes refine the system in its environment, and we call them the *variants* of the adversary model:

$$P \Vert\!\!\to_{MI} (H \,|[EI]|\, A) \sqsubseteq Q \Vert\!\!\to_{MI} E$$

An adversary model captures a model of the machine to be built together with assumptions on the behavior of the honest users and an adversary. The interfaces AI and EI allow the adversary to actively influence the machine and the honest users (permitting active attacks on the users). The user environment can also allow an adversary to compromise users during a system run.

The bound k not only reflects a technical necessity but also a realistic assumption: Associating time units with system events, it restricts the time an adversary can spend on attacks. It is a parameter of the concepts we introduce in the following.

It is possible to strengthen these concepts, requiring them to hold for all k and using an inductive argument to establish the required properties. However, we will restrict the presentation and consider a fixed k only.

4 Confidentiality Properties

This section discusses a common abstraction of the confidentiality properties of adversary models. In particular, it motivates the existential nature of those properties.

Basic Confidentiality Properties. The concept of indistinguishable traces is the foundation for defining confidentiality properties of adversary models. Given a set of channels W, two traces $s,t \in traces(P)$ of a process P are *indistinguishable by* W (denoted $s \equiv_W t$) if their projections to W are equal:

$$s \equiv_W t \Leftrightarrow s \upharpoonright W = t \upharpoonright W$$

where $s \upharpoonright W$ is the projection of s to the sequence of events on W.

Indistinguishability induces a partition on the trace set of a process. We are particularly interested in the traces up to the length of k. The set $J_W^{P,k}(o)$ contains the traces of P with a length of at most k that produce the observation o on W. The set $\mathrm{Obs}_W^k(P)$ comprises all observations P produces with traces that are no longer than k.

$$J_W^{P,k}(o) = \{t : traces(P) \mid t \upharpoonright W = o \wedge \#t \leqslant k\}$$
$$\mathrm{Obs}_W^k(P) = \{t \upharpoonright W \mid t \in traces(P) \wedge \#t \leqslant k\}$$

The traces that are indistinguishable by an adversary window W are the ones that an adversary cannot obviously distinguish. Given an observation o, the adversary does not know which member of $J_W^{P,k}(o)$ caused the observation (unless that set is a singleton).

In earlier work [28], we have discussed several confidentiality properties based on indistinguishability. *Possibilistic* confidentiality properties, such as the various information flow properties that Mantel [19] analyzes, basically require at least one alternative indistinguishable behavior to exist for any given one, according to the system design. They neither distinguish systems with respect to the number of alternative behaviors, nor with respect to the degree of evidence (in any suitable measure) an adversary might assign to the alternative behaviors in question. We are primarily interested in *probabilistic* confidentiality properties. These define the "degree of evidence" of alternative behaviors based on the probabilistic behavior of the system in a given environment. Therefore, we focus on predicates $\mathcal{CP}(QE, W, k)$ depending on a probabilistic linear process QE (a realization of the machine process in an admissible environment), an adversary window W and the length bound k. We call such a property a *basic confidentiality property*.

We do not further characterize basic confidentiality properties here. Section 6 discusses an example. In the following discussion of the structure of confidentiality properties, a predicate $\mathcal{CP}(QE, W, k)$ serves as a placeholder.

Structure of Confidentiality Properties. It is not obvious for an adversary model $(P, A, H, HI, AI, MI, EI, k)$ which refinements QE of a given machine P in an admissible environment $E \in \mathcal{E}_{P,k}^{EI,MI}(H, A)$ must satisfy $\mathcal{CP}(QE, W, k)$ for the adversary model to satisfy a confidentiality property based on $\mathcal{CP}(QE, W, k)$.

As already indicated in Section 1, the refinement paradox motivates the *existential* nature of confidentiality properties.

Since it has become known that possibilistic information flow properties are closure properties [19, 20], the observation that refinement does not preserve confidentiality in general is not so surprising anymore: refinement reduces nondeterminism and thus diminishes the set of traces, which is the definition of trace refinement. A closure property requires that, given a member of a set, certain other items are also members of that set. Therefore, a trace refinement, in general, does not preserve closure properties.

These considerations show that we cannot expect *all* refinements QE of a system in its environment to satisfy a given basic confidentiality property $\mathcal{CP}(QE, W, k)$ with respect to the adversary window W and the trace bound k, unless we can exclude "specification nondeterminism" in P. However, this is hardly possible in the current theory of probabilistic (and standard) CSP for two reasons. A technical reason is that hiding and data renaming almost inevitably introduce nondeterminism. Methodologically, the nondeterministic choice of CSP has an interpretation as "execution time nondeterminism", because it is *demonic* and must be considered to be resolved "after" all probabilistic and external choices. On the other hand, it is refined by probabilistic and external choice, as well. Thus, the definition of CSP refinement clearly considers nondeterminism as a means of postponing implementation decisions. From a methodological point of view, it is also necessary to allow P to contain "specification nondeterminism", because P actually *is* a specification and, as such, must provide ways of abstracting from design decisions including decisions on how the system chooses alternative behavior.

The environment, in contrast to the machine process, must be considered with all variations that the adversary model permits. Analyzing a system for security with respect to a *single* admissible realization $H_0 \,|[\, EI \,]|\, A_0$ of the user and adversary environment would yield a quite weak result. Instead, all $E \in \mathcal{E}_{P,k}^{EI,MI}(H,A)$ need to be considered for evaluating the security of a system.

This argument shows that, although we inevitably need to consider an environment process E describing user and adversary behavior to obtain a probabilistic analysis of security properties of a system, we must not restrict the analysis to one particular such process but we must carry out that analysis for all members of $\mathcal{E}_{P,k}^{EI,MI}(H,A)$. In particular, this allows an analysis to consider arbitrary adversary. Taking the chaotic process *Chaos* as the adversary environment models the most liberal assumption about the adversary behavior, because *Chaos* is refined by any other process.

We conclude that an adversary model satisfies a confidentiality property that is defined in terms of a basic confidentiality property $\mathcal{CP}$ if *there is* a probabilistic linear realization of the machine process that satisfies $\mathcal{CP}$ in all admissible environments. Therefore, a confidentiality property has the general form:

$$\exists\, Q : P^{\top} \bullet \forall\, E : \mathcal{E}_{P,k}^{EI,MI}(H,A) \bullet \textbf{let}\ QE = (Q \Downarrow_{MI} E) \downarrow k \bullet \mathcal{CP}(QE, W, k)$$

This definition avoids the refinement paradox, because it explicitly states that not necessarily all functionally correct realizations are supposed to be secure but that at least one realization needs to exist that is. It also avoids the misconception that a system will be secure in any working environment but makes the admissible working conditions and the constraints on the behavior of adversaries explicit.

Remark 1. Other "non-functional" requirements have a similar "existential" nature: To be adequate for a system with real-time performance requirements, for example, a model must admit a performing implementation, but not all functionally correct implementations of the model necessarily satisfy the real-time constraints.

5 Confidentiality Preserving Refinement

This section discusses a definition of refinement of adversary models that preserves a given confidentiality property. The refinement relation allows the realization to be more deterministic, to change the communicated data, to shift the responsibility to realize behavior from the environment to the machine, and to extend the adversary window, thus providing the adversary with new means of observation. The motivation[3] to consider

[3] Refer to [10] for a more detailed motivation.

these variation points lies in the fact that moving from an initial specification to an implementation, the concepts the adversary models need to consider necessarily include more detailed descriptions of the processed data and also more intricate ways of the adversary to observe the system. Furthermore, the interpretation of an adversary model differs depending on its role in a refinement: as a specification, an adversary model reflects what an adversary *is allowed* to observe (and to do); as a realization, an adversary model describes what an adversary *can* observe (or do). Therefore, the refinement relation needs to ensure that an adversary's abilities do not exceed his permissions: if the specification satisfies a confidentiality property then the realization must satisfy a similar confidentiality property that is "re-abstracted" to the data model of the first confidentiality property.

In the following, we first introduce behavior refinement. Then, we define a re-abstraction preorder on the variants of the specification and the realization adversary models. The definition of confidentiality preserving refinement (CPR) refers to another preorder, the information flow refinement order: CPR holds if the re-abstraction and the information flow refinement preorders coincide on the adversary models in question.

Behavior Refinement. Allowing the refining process to communicate different data than the refined process, behavior refinement generalizes PCSP refinement. Of course, the change of data must not be completely arbitrary but there must be a relation between the concrete and the abstract data that is compatible to PCSP refinement. A *retrieve relation* R maps the data of the concrete process Q to the data of the abstract process P, i.e., it is total on the data of Q and its range is in the data of P.

A retrieve relation R abstracts away the additional detail of the concrete data to "retrieve" the abstract data that the concrete data implements. The following definition of behavior refinement uses a retrieve relation to abstract the data of the refining process before comparing that "data abstracted" process to the refined process with PCSP refinement. With data renaming, we have a CSP operator at hand to perform the data abstraction.

Definition 1 (Behavior Refinement). *Let P and Q be probabilistic processes. Let R be a retrieve relation from Q to P. Then Q* behaviorally refines *P* via *R (written $P \sqsubseteq_R Q$), if $P \sqsubseteq Q[\![R]\!]_D$.*

Behavior refinement allows Q to resolve nondeterminism in P (as usual either by external or by probabilistic choice). Additionally, it offers *new* implementation freedom for Q if R maps several data items c_{i,k_i} of Q to the same abstract data item a_i of P. In particular, if P offers a probabilistic choice between several $e_1.a_i$ and $e_2.a_j$, then the refinement condition $P \sqsubseteq Q[\![R]\!]_D$ requires Q to produce $e_1.c_{i,k_i}$ and $e_2.c_{j,k_j}$ for *some* k_i and k_j with the same distribution as P, but it does not prescribe the choice of the k_i and k_j, which Q may choose nondeterministically.

Extending behavior refinement to adversary models, there are two points to clarify: first, how can the relationship between system process and environment change in a refinement; and second, how do the adversary windows relate?

The following Definition 2 allows a refinement to change the "responsibility" of the machine and its environment to establish certain behavior. It relates the abstract and concrete machines *in their respective environments*. It does not require that the concrete machine as such refines the abstract one (and the environment processes relate similarly).

A central objective of our investigation on confidentiality preserving refinement is to clarify the conditions under which the adversary's observational power may change securely under refinement. Definition 2 allows the refining adversary model to extend the adversary window, i.e., it requires $W_a \subseteq W_c$. The additional channels in W_c give the adversary means of observing the system that are not present in the abstract adversary model. The behavior refinement does not relate those means of observation to the abstract model, which the definition reflects by hiding $W_c - W_a$. Thus it allows the adversary to make arbitrary additional observations. In the rest of this section, we addresses the question whether those additional observations affect the security of the system.

Definition 2 (Behavior Refinement of Adversary Models). *Let two adversary models with identical bound k be given:* $\mathcal{A} = (P_a, A_a, H_a, HI_a, AI_a, MI_a, EI_a, k)$ *and* $\mathcal{C} = (P_c, A_c, H_c, HI_c, AI_c, MI_c, EI_c, k)$. *The realization* $\mathcal{C}$ behaviorally refines *the specification* $\mathcal{A}$ *via the retrieve relation* R_{ca} *(written* $\mathcal{A} \sqsubseteq_{R_{ca}} \mathcal{C}$*) if* $W_a \subseteq W_c$ *and*

$$P_a \Downarrow_{MI_a} (H_a \,|[EI_a]|\, A_a) \sqsubseteq_{R_{ca}} (P_c \Downarrow_{MI_c} (H_c \,|[EI_c]|\, A_c)) \setminus (W_c - W_a)$$

To refine a specification to an implementation in a stepwise fashion, any refinement relation must be a preorder, i.e., be reflexive and transitive for an appropriate choice of retrieve relations. Behavior refinement inherits these properties from PCSP refinement, i.e., $\mathcal{A} \sqsubseteq_{\mathrm{id}} \mathcal{A}$ and $\mathcal{A} \sqsubseteq_{R_{ba}} \mathcal{B} \wedge \mathcal{B} \sqsubseteq_{R_{cb}} \mathcal{C} \Rightarrow \mathcal{A} \sqsubseteq_{R_{cb} \fatsemi R_{ba}} \mathcal{C}$ hold.

Re-Abstraction. Basic confidentiality properties refer to the variants of adversary models, and CPR must place conditions on the "matching" variants of the specification and the realization in order to ensure preservation of the property. Re-abstraction relates the variants of the specification to the "data abstracted" variants of the realization. By definition, variants are probabilistic linear (before diverging after k events). This means that a variant of the specification cannot be refined further (up to k). Data renaming a variant of the realization, however, may introduce nondeterminism. Therefore, there may be several "matching" variants of the specification for a given variant of the realization. These are exactly the ones that the re-abstraction selects.

Definition 3 (Re-Abstracted Refinement). *Let* R_{ca} *be a retrieve relation from the data of* QE_c *to the data of* QE_a. *Let* W_a *and* W_c *be sets of channels of* QE_a *and* QE_c, *respectively, such that* $W_a \subseteq W_c$. *Then the re-abstracted refinement of* (QE_c, W_c) *by* (QE_a, W_a), *denoted* $(QE_c, W_c) \overset{\nwarrow}{\sqsupseteq}_{R_{ca}} (QE_a, W_a)$, *is defined by*

$$(QE_a, W_a) \overset{\nwarrow}{\sqsupseteq}_{R_{ca}} (QE_c, W_c) \Leftrightarrow QE_c \setminus (W_c - W_a)[\![R_{ca}]\!]_D \sqsubseteq QE_a$$

Similar to behavior refinement, re-abstraction is a preorder.

Information Flow Refinement. A behavioral refinement possibly refines the data which the processes communicate, and it may also change the adversary window. A basic confidentiality property $\mathcal{CP}$ refers to the data and the adversary window of the abstract model. To determine whether a refined adversary model satisfies the same confidentiality property, it is in general necessary to relate the concrete data and adversary window back to the abstract ones, to which $\mathcal{CP}$ originally refers. In a sequence of refinement steps, one usually wishes to relate back to the confidentiality property of the initial specification.

To capture this formally, we say that a *refined basic confidentiality property* $\mathcal{CP}_r(QE, W, W_r, R_r, k)$ refers to a *point of reference* consisting of an adversary window W_r and a retrieve relation R_r. A refined basic confidentiality property induces a simple one by the following equivalence:

$$\mathcal{CP}(QE, W, k) \Leftrightarrow \mathcal{CP}_r(QE, W, W, \mathrm{id}, k)$$

Re-abstraction relates the “matching” variants of the specification and the realization adversary models. The following concept of information flow refinement serves as an abstraction of the relationship that the matching variants must satisfy in order to preserve a given confidentiality property.

Definition 4 (Information Flow Refinement). *Let $\mathcal{CP}_r$ be a refined confidentiality property. A preorder $(QE_a, W_a) \preccurlyeq^k_{R_{ca}} (QE_c, W_c)$ on pairs of probabilistic linear processes and adversary windows is called an* information flow refinement *relation for $\mathcal{CP}_r$ with the point of reference (W_r, R_r) if it strengthens the re-abstraction preorder and it is sufficient to preserve $\mathcal{CP}_r$, i.e., for all adversary models $\mathcal{A} = (P_a, AI_a, W_a, k, H_a, A_a)$ and $\mathcal{C} = (P_c, AI_c, W_c, k, H_c, A_c)$ such that the domain of R_r comprises the data space of $\mathcal{A}$, and $\mathcal{A} \sqsubseteq_{R_{ca}} \mathcal{C}$ holds, the following is satisfied:*

$$\begin{aligned}
&\forall\, Q_a : P_a^\top;\, Q_c : P_c^\top;\; E_a : \mathcal{E}^{EI_a, MI_a}_{P_a,k}(H_a, A_a);\; E_c : \mathcal{E}^{EI_c, MI_c}_{P_c,k}(H_c, A_c) \bullet\\
&\quad \textbf{let}\ QE_a = Q_a \Downarrow_{MI_a} E_a;\; QE_c = Q_c \Downarrow_{MI_c} E_c \bullet\\
&\qquad\qquad (\,(QE_a, W_a) \preccurlyeq^k_{R_{ca}} (QE_c, W_c) \Rightarrow (QE_a, W_a) \overset{\nwarrow}{\sqsupseteq}_{R_{ca}} (QE_c, W_c)\,)\\
&\qquad\quad \wedge (\,(QE_a, W_a) \preccurlyeq^k_{R_{ca}} (QE_c, W_c) \wedge \mathcal{CP}_r(QE_a, W_a, W_r, R_r, k)\\
&\qquad\qquad\quad \Rightarrow \mathcal{CP}_r(QE_c, W_c, W_r, R_{ca} \fatsemi R_r, k)\,)
\end{aligned}$$

By definition, an information flow refinement relation is a subset of the re-abstraction relation. Usually, it makes sense only for variants that are related by re-abstraction. In the following, we will see that the crucial condition for confidentiality-preserving refinement requires that the reverse implication is true and the two preorders coincide on the variants of the adversary models in question.

CPR. Under which condition is a behavioral refinement of adversary models a confidentiality-preserving one? Due to the existential nature of confidentiality properties, it is not necessarily the case that a behavioral refinement admits a secure refinement at all. The realization might exclude all possible secure refinements even though the specification satisfies the confidentiality property, i.e., there is a variant of the specification satisfying the desired basic confidentiality property. The behavior refinement can only preserve confidentiality if there is a secure variant $\widehat{QE_a}$ of the specification that (PCSP-)

refines the re-abstracted realization. If this is the case, then we need to know that the re-abstracted variants of the realization matching $\widehat{QE_a}$ are secure, too. Confidentiality-preserving refinement guarantees the latter.

Definition 5 (Confidentiality-Preserving Refinement, CPR). *Let $\preccurlyeq$ be an information flow refinement relation. The adversary model $\mathcal{C}$ is a* confidentiality-preserving refinement (CPR) *of the adversary model $\mathcal{A}$ for $\preccurlyeq$ via the retrieve relation R_{ca} (written $\mathcal{A} \sqsubseteq^{\preccurlyeq}_{R_{ca}} \mathcal{C}$) if $\mathcal{A} \sqsubseteq_{R_{ca}} \mathcal{C}$ and the re-abstracted refinement of variants of $\mathcal{A}$ and $\mathcal{C}$ is sufficient for their information flow refinement:*

$$\begin{aligned}
&\forall\, Q_a : P_a^{\top};\ Q_c : P_c^{\top};\ E_a : \mathcal{E}^{EI_a,MI_a}_{P_a,k}(H_a, A_a);\ E_c : \mathcal{E}^{EI_c,MI_c}_{P_c,k}(H_c, A_c) \bullet \\
&\quad \textbf{let } QE_a = Q_a \Downarrow_{MI_a} E_a;\ QE_c = Q_c \Downarrow_{MI_c} E_c \bullet \\
&\qquad (QE_a, W_a) \mathrel{\overset{\nwarrow}{\sqsupseteq}_{R_{ca}}} (QE_c, W_c) \Rightarrow (QE_a, W_a) \preccurlyeq^{k}_{R_{ca}} (QE_c, W_c)
\end{aligned}$$

In conjunction with the first implication in Definition 4, the definition of CPR implies that (given $\mathcal{A} \sqsubseteq_{R_{ca}} \mathcal{C}$) $\mathcal{A} \sqsubseteq^{\preccurlyeq}_{R_{ca}} \mathcal{C}$ is equivalent to the identity of information flow refinement and re-abstraction on the variants of $\mathcal{A}$ and $\mathcal{C}$.

The following lemma establishes that CPR is a well-behaved refinement relation.

Lemma 1 (CPR is a Preorder). *For all adversary models $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, CPR satisfies $\mathcal{A} \sqsubseteq^{\preccurlyeq}_{\mathrm{id}} \mathcal{A}$ and $\mathcal{A} \sqsubseteq^{\preccurlyeq}_{R_{ba}} \mathcal{B} \wedge \mathcal{B} \sqsubseteq^{\preccurlyeq}_{R_{cb}} \mathcal{C} \Rightarrow \mathcal{A} \sqsubseteq^{\preccurlyeq}_{R_{cb} \fatsemi R_{ba}} \mathcal{C}$.*

Proposition 1 states the most important property of CPR, namely that it does indeed preserve confidentiality properties (with an appropriately adjusted point of reference). As indicated before, CPR cannot be expected to allow "secure" refinements only but it can establish that a behavior refinement whose re-abstraction admits an "abstractly secure" PCSP refinement preserves that security over data refinement and extension of adversary windows.

Proposition 1 (CPR preserves $\mathcal{CP}_r$). *Let $\mathcal{CP}_r$ be a refined basic confidentiality property with point of reference (W_r, R_{ar}). Let $\preccurlyeq$ be an information flow refinement property for $\mathcal{CP}_r$. If $\mathcal{A} \sqsubseteq^{\preccurlyeq}_{R_{ca}} \mathcal{C}$ then the following implication holds:*

$$\begin{aligned}
&\big(\exists\, Q_a : P_a^{\top} \bullet \forall\, E_a : \mathcal{E}^{EI_a,MI_a}_{P_a,k}(H_a, A_a) \bullet \\
&\qquad ((P_c \Downarrow_{MI_c} (H_c \,|[EI_c]|\, A_c)) \setminus (W_c - W_a))[\![R_{ca}]\!]_D \sqsubseteq Q_a \Downarrow_{MI_a} E_a \\
&\qquad\qquad \wedge\, \mathcal{CP}_r(Q_a \Downarrow_{MI_a} E_a, W_a, W_r, R_{ar}, k)\big) \\
&\qquad\qquad\qquad \Rightarrow \\
&\big(\exists\, Q_c : P_c^{\top} \bullet \forall\, E_c : \mathcal{E}^{EI_c,MI_c}_{P_c,k}(H_c, A_c) \bullet \mathcal{CP}_r(Q_c \Downarrow_{MI_c} E_c, W_c, W_r, R_{ca} \fatsemi R_{ar}, k)\big)
\end{aligned}$$

Remark 2. Carefully analyzing the constellation of the quantifiers in Proposition 1 suggests that the definition of information flow refinement might be too strong. For confidentiality preservation, it suffices indeed to require alternating universal and existential quantifiers like $\forall\, Q_a\, \exists\, Q_c\, \forall\, E_c\, \exists\, E_a \bullet \ldots$ in Definition 4. Unfortunately, the resulting definition of CPR is not transitive, because the required witnesses for the variant of the intermediate adversary model need not match.

6 Probabilistic Confidentiality: Ensured Entropy

This section serves to illustrate an instantiation of the framework discussed in the preceeding sections. It presents the probabilistic confidentiality property of *ensured entropy* [28]. Space limitations[4] do not permit to show the definitions in full detail or discuss the use of this property by way of an application example.

As mentioned in Section 4, classes $J_W^{P,k}(o)$ of indistinguishable traces are our starting point for defining confidentiality properties. Each variant of an adversary model induces a system of indistinguishability classes. Turning this fact into a requirement, one can propose a *mask* $\mathcal{M}$ that is a system of sets of traces such that the traces in each $M \in \mathcal{M}$ are indistinguishable. A variant QE (possibilistically) secures a mask $\mathcal{M}$ if its set $\mathcal{I}$ of indistinguishability classes *covers* $\mathcal{M}$, written $\mathcal{I} \Supset \mathcal{M}$:

$$\mathcal{I} \Supset \mathcal{M} \Leftrightarrow \forall M : \mathcal{M};\ I : \mathcal{I} \bullet M \cap I = \emptyset \vee M \subseteq I$$

If $\mathcal{I} \Supset \mathcal{M}$, the variant QE can produce all members of each $M \in \mathcal{M}$. Upon observing o at W, an adversary cannot conclude which member of $M \subseteq J_W^{QE,k}(o)$ caused that observation.

Ensured entropy extends this idea and requires that the indistinguishability classes of QE not only cover $\mathcal{M}$ but also that their entropy[5] (given the respective observation) exeeds a lower bound associated to the members of $\mathcal{M}$. As the entropy is a measure of uncertainty, this requirement puts a lower bound on the adversary's effort to infer the trace that caused an observation from that observation. Ensured entropy is weaker than, e.g., probabilistic noninterference [9] because it does not strictly prevent information flow from the machine to the adversary.

Given a mask $\mathcal{M}$ and a mapping $\mathcal{H} : \mathcal{M} \to \mathbb{R}^+$, the corresponding basic confidentiality property $\mathcal{CP}_{\mathcal{M}}(QE, W, k)$ is defined by

$$\begin{aligned} \mathcal{CP}_{\mathcal{M}}(QE, W, k) \Leftrightarrow & \{o : \mathrm{Obs}_W(QE) \bullet J_W^{QE,k}(o)\} \Supset (\mathcal{M} \downarrow k) \\ & \wedge \forall M : \mathcal{M} \bullet \forall o : \mathrm{Obs}_W(QE) \mid M \subseteq J_W^{QE,k}(o) \bullet \mathcal{H}(M) \leq H_W^k(QE|o) \end{aligned}$$

where $H_W^k(QE|o)$ is the entropy of the process QE given the observation o, i.e., the entropy of $J_W^{QE,k}(o)$ given o. $\mathcal{CP}_{\mathcal{M}}(QE, W, k)$ is well-defined because QE is a probabilistic linear process and, therefore, the probabilities of traces of QE can be determined. (We spare the reader the technical definition of the refined version of $\mathcal{CP}_{\mathcal{M}}(QE, W, k)$.)

Information Flow Refinement. The information flow refinement relation that preserves $\mathcal{CP}_{\mathcal{M}}$ is defined in terms of the conditional mutual information between the behavior of the specification variant QE_a and the observations of the realization variant QE_c given an observation o_a of the specification. Using the identifiers of processes and adversary windows to denote random variables for the respective processes and

[4] A companion paper presenting the details of what is sketched here is in preparation.

[5] For an explanation of entropy, mutual information and the other concepts of information theory, refer to any book on coding theory, such as [17].

their observations, the information flow refinement relation $\preccurlyeq^k_{R_{ca}}$ for $\mathcal{CP}_{\mathcal{M}}(QE, W, k)$ is defined by:

$$(QE_a, W_a) \preccurlyeq^k_{R_{ca}} (QE_c, W_c) \Leftrightarrow \big(QE_c \setminus (W_c - W_a)[\![R_{ca}]\!]_D = QE_a \\ \wedge I(QE_a;\ (QE_c, R_{ca}, W_c) \mid W_a = o_a) = 0 \big)$$

The mutual information[6] $I(QE_a;\ (QE_c, R_{ca}, W_c) \mid W_a = o_a)$ describes the difference of the entropy of the traces of QE_a producing the observation o_a on W_a, and of the entropy of the observations on W_c produced by traces of QE_c whose re-abstractions produce the abstract observation o_a.

It is relatively straightforward to show that this information flow refinement relation preserves $\mathcal{CP}_{\mathcal{M}}(QE, W, k)$, i.e., the entropy of re-abstracted indistinguishability classes is equal to the one of the corresponding specification classes.

The transitivity proof of the relation, however, is quite involved. It needs to use several independence relationships between observations and process behaviors at different levels of refinement.

Having established those lemmas, however, the framework of CPR delivers a theory of "refinement preserving the entropy of indistinguishable system behavior".

7 Related Work

The work presented here extends previous work [10, 29] on CPR. The general idea of an adversary window to model possible observations is already present there. To consider indistinguishability classes as the basis for definitions of confidentiality properties also is not new. Zakinthinos and Lee [31] call indistinguishability classes *low level equivalence sets* (LLES). They give a definition of a (possibilistic) security property as one that can be recast as a property holding for each indistinguishability class and show that several information flow properties can be defined as properties of those classes.

The definition of CPR in [10, 29] is a quite restrictive variant of ensured entropy: It basically requires the entropy of all indistinguishability classes of the realization to be maximal, but it does not relate the probabilistic properties of the specification and the realization. Furthermore, that early definition of CPR assumed that scheduling takes place at the "meta-level" and did not make the task of the environment as a scheduler explicit. Finally, it assumed a probabilistic extension of CSP but did not explicitly base on PCSP.

The relationship to other propositions of secure refinement [14, 18, 24] has been discussed in Section 1. To the best of our knowledge, Graham-Cumming [8] still is one of the few to address security issues in data refinement. But he does not consider I/O refinement as we do.

Lowe [16] also uses the idea of quantifying over the possible refinements of a specification similar to our variants of an adversary model. Furthermore, he quantifies information flow discretely, without referring to probabilities, and thus his work mediates between a possibilistic yes/no concept of information flow and one based on probabilistic information theory. In contrast to our view, he uses a pessimistic approximation and

[6] The exact definition is quite technical and cannot be presented here.

considers the worst case, i.e., maximal, flow of information produced by all variants. Our view is pessimistic on the environment but considers an optimistic view on the machine, because the implementors control the way machine nondeterminism is resolved.

The framework as it stands now has some strong similarities with the system model underlying reactive simulatability [23, 3], which addresses the cryptographically secure implementation of "ideal" cryptographic protocols by "real" ones using cryptographic algorithms [2]. Both system models explicitly distinguish the machine, the honest users, and the adversary, all of which can interact through designated communication channels. Like our adversary window, "forbidden" channels model means of the adversary to which honest users do not have access. The differences between the two approaches stem from the different purposes they are designed to serve: We aim at a stepwise development of an "ideal" system starting from a very abstract initial specification and ending at an implementation model that still abstracts from issues of computational complexity. The model of Backes, Pfitzmann and Waidner, in contrast, is designed to support the last transition from such an "ideal" implementation model to one that uses "real" algorithms. Therefore, it is asynchronous and deterministic. It has a very detailed step semantics that allows one to analyze computation and communication acts in a very detailed manner (including their computational complexity). The concept of reactive simulatability is used to compare an ideal with a real model. It essentially is a strong (probabilistic) bisimulation [30] that enforces cryptographic indistinguishability (not to be confused with our notion of indistinguishability) of the honest users' view of the system, while the adversary can change in the transition from "ideal" to "real". Thus, reactive simulatability is very well suited to analyze the question whether an implementation of a cryptographic protocol is correct. However, it is too restrictive to support stepwise refinement from a very abstract to a much more detailed system model. In particular, it insists that the user model is the same for both, ideal and real system. This also implies that trading functionality between the machine and its environment does not establish a valid simulation.

8 Conclusion

Our framework for CPR captures general conditions sufficient to preserve probabilistic confidentiality properties in behavior refinements of nondeterministic probabilistic systems. It takes the refinement paradox into account by considering existential confidentiality properties. Definitions 4 and 5 provide an abstraction that separates the issues of preserving a probabilistic (basic) property from the ones of preserving an existential one. Thus, it allows to investigate the relationship of different properties within the same framework.

The central definitions only rely on the fact that PCSP refinement is a compositional preorder, and that hiding may introduce nondeterminism. Any formalism coming with such a refinement order could replace PCSP in the framework.

The framework establishes the essential property of a refinement relation, namely that it is a preorder. Research on conditions of compositionality of CPR is still going on. However, because CPR is more liberal than, e.g., simulatability, a result as strong as the one for that relation [5] cannot be expected without additional side conditions.

To identify conditions of compositionality is one task of ongoing research, as is the representation of standard confidentiality properties such as probabilistic [9] noninterference. Furthermore, tool support is a very important issue. Here, one can build on established tools for standard CSP and combine those with verifiers for probabilistic calculi.

Finally, confidentiality is not the only property that standard behavior refinement does not preserve. Many properties such as real-time constraints and quality of service behave similarly under refinement. Therefore, there is hope to apply the present results to other application areas as well.

Acknowledgments. Thanks go to anonymous reviewers for constructive and informed comments.

References

[1] J.-R. Abrial. *The B-Book: Assigning programs to meanings*. Cambridge University Press, 1996.

[2] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003.

[3] M. Backes, B. Pfitzmann, and M. Waidner. Secure asynchronous reactive systems. IACR ePrint Archive, March 2004. Online available at `http://eprint.iacr.org/2004/082.ps`.

[4] P. Behm, P. Benoit, A. Faivre, and J.-M. Meynadier. Météor: A successful application of B in a large project. In J. Wing, J. Woodcock, and J. Davies, editors, *FM'99 – Formal Methods*, volume I of *LNCS 1708*, pages 369–387. Springer-Verlag, 1999.

[5] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.

[6] F. Ciesinski and M. Größer. On probabilistic computation tree logic. In C. Baier, B. R. Haverkort, H. Hermanns, J.-P. Katoen, and M. Siegle, editors, *Validation of Stochastic Systems: A Guide to Current Research*, LNCS 2925, pages 147 – 188. Springer-Verlag, 2004.

[7] J. Derrick and E. Boiten. *Refinement in Z and Object-Z*. Springer-Verlag, London, 2001.

[8] J. Graham-Cumming and J. W. Sanders. On the refinement of non-interference. In *9th IEEE Computer Security Foundations Workshop*, pages 35–42. IEEE Computer Society Press, 1991.

[9] J. W. Gray, III. Toward a mathematical foundation for information flow security. *Journal of Computer Security*, pages 255–294, 1992.

[10] M. Heisel, A. Pfitzmann, and T. Santen. Confidentiality-preserving refinement. In *14th IEEE Computer Security Foundations Workshop*, pages 295–305. IEEE Computer Society Press, 2001.

[11] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.

[12] J. Jacob. On the derivation of secure components. In *IEEE Symposium on Security and Privacy*, pages 242–247. IEEE Press, 1989.

[13] C. B. Jones. *Systematic Software Development using VDM*. Prentice Hall, 2nd edition, 1990.

[14] J. Jürjens. Secrecy-preserving refinement. In J. N. Oliveira and P. Zave, editors, *FME 2001: Formal Methods for Increasing Software Productivity*, LNCS 2021, pages 135–152. Springer-Verlag, 2001.

[15] B. Liskov and J. Wing. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):1811–1841, 1994.

[16] G. Lowe. Quantifying information flow. In *15th IEEE Computer Security Foundations Workshop*, pages 18–31. IEEE Computer Society, 2002.

[17] D. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[18] H. Mantel. Preserving information flow properties under refinement. In *IEEE Symposium on Security and Privacy*, pages 78–91. IEEE Computer Society Press, 2001.

[19] H. Mantel. *A Uniform Framework for the Formal Specification and Verification of Information Flow Security*. PhD thesis, Universität des Saarlandes, 2003.

[20] J. McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, 1996.

[21] B. Meyer. Applying "design by contract". *IEEE Computer*, pages 40–51, October 1992.

[22] C. Morgan, A. McIver, K. Seidel, and J. W. Sanders. Refinement-oriented probability for CSP. *Formal Aspects of Computing*, 8(6):617–647, 1996.

[23] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–201. IEEE Computer Society, 2001.

[24] A. W. Roscoe. CSP and determinism in security modelling. In *Proc. IEEE Symposium on Security and Privacy*, pages 114–127. IEEE Computer Society Press, 1995.

[25] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998.

[26] A. W. Roscoe, J. C. P. Woodcock, and L. Wulf. Non-interference through determinism. In D. Gollmann, editor, *European Symposiom on Research in Computer Security (ESORICS)*, LNCS 875, pages 33–53. Springer-Verlag, 1994.

[27] P. Y. A. Ryan and S. A. Schneider. Process algebra and non-interference. In *12th IEEE Computer Security Foundations Workshop*, pages 214–227. IEEE Computer Society, 1999.

[28] T. Santen. Probabilistic confidentiality properties based on indistinguishability. In H. Federrath, editor, *Proc. Sicherheit 2005 – Schutz und Zuverlässigkeit*, Lecture Notes in Informatics, pages 113–124. Gesellschaft für Informatik, 2005.

[29] T. Santen, M. Heisel, and A. Pfitzmann. Confidentiality-preserving refinement is compositional – sometimes. In D. Gollmann, G. Karjoth, and M. Waidner, editors, *Computer Security – ESORICS 2002*, LNCS 2502, pages 194–211. Springer-Verlag, 2002.

[30] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[31] A. Zakinthinos and E. S. Lee. A general theory of security properties. In *Proc. IEEE Symposium on Security and Privacy*, pages 94–102, 1997.

[32] P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, 1997.

Timing-Sensitive Information Flow Analysis for Synchronous Systems

Boris Köpf and David Basin

Information Security
ETH Zurich, Switzerland
{bkoepf, basin}@inf.ethz.ch

Abstract. Timing side channels are a serious threat to the security of cryptographic algorithms. This paper presents a novel method for the timing-sensitive analysis of information flow in synchronous hardware circuits. The method is based on a parameterized notion of confidentiality for finite transition systems that allows one to model information leakage in a fine-grained way. We present an efficient decision procedure for system security and apply it to discover timing leaks in nontrivial hardware implementations of cryptographic algorithms.

1 Introduction

Timing side channels are a serious threat to the security of cryptographic algorithms [4, 12, 18]. By analyzing the running times of algorithms such as RSA decryption, an attacker may be able to deduce information about the secret key used, possibly even recovering the key in its entirety. Several countermeasures against this threat have been proposed, including blinding and randomization techniques. While these techniques successfully defeat certain known attacks, it is difficult to argue their completeness, in the sense that they defeat all attacks that exploit timing information.

One systematic and complete countermeasure for preventing timing attacks is to ensure that the algorithms' running times are independent of the secrets processed. Agat [1] pursues this approach by giving a security type system for a simple imperative programming language. If a program can be assigned a security type, then its running times are independent of the secrets it computes with, and hence do not reveal this confidential information. This is shown by proving the soundness of the type system with respect to a semantic notion of secure information flow. Although this result (as well as other approaches that use programming language-based models [29, 22, 3]) provides an attractive analysis method, the timing model used is based on a high-level language and is therefore too simplistic. Indeed, if the timing behavior of the underlying hardware is not accurately modeled, it is unclear what is gained from such a formal analysis. Unfortunately, providing precise timing models for today's processors seems out of reach. Giving upper bounds for the real-time behavior of multi-purpose processors is already a daunting task [20] and is still not sufficient for proving the absence of timing leaks.

In this paper, we approach the problem on a different level of abstraction. We focus on special-purpose hardware implementations of cryptographic algorithms, which are particularly important in resource-critical application domains [24]. We develop a method for the information flow analysis of synchronous (clocked) hardware circuits. To this end, we define R_I/R_O-security, a parameterized and timing-sensitive notion of security for Mealy machines, in which each transition corresponds to one clock tick. R_I/R_O-security can be instantiated to standard noninterference definitions, but it can also be used to express that only partial information on each confidential input is revealed through the system's observable behavior. In system runs of arbitrary length, this partial information can accumulate. We show that the guarantees of R_I/R_O-security can be combined with assumptions on the environment to derive an upper bound on the number of distinguishable output behaviors, a measure for what an attacker may learn about the processed secrets. We develop efficient algorithms for deciding whether a finite-state system is R_I/R_O-secure. For deterministic systems, we reduce this to a reachability problem for a special kind of product automaton. In the nondeterministic case, we reduce this to a generalization of the Partition Refinement Problem. We also provide a compositionality result as a first step to scaling-up the analysis method to more complex designs.

Finally, we report on initial experimental results using our method. We have encoded our decision procedures in an off-the-shelf model checker and used it to discover subtle timing side channels in a textbook hardware implementation of a finite-field exponentiation algorithm. The synchronous hardware description language Gezel [25] provides the link between our analysis method and concrete hardware implementations. Namely, Gezel allows one to specify synchronous circuits in terms of automata, and it comes with a tool for translating the designs into a subset of the industrial-strength hardware description language Vhdl. The translation is *cycle-true*, which means that it preserves the timing behavior within the granularity of clock ticks. Moreover, the output is *synthesizeable*, i.e. it can be mapped to a physical implementation. Hence, the security guarantees obtained using our analysis method translate into guarantees for real-world hardware implementations.

Our main contributions are twofold. First, we extend well-studied notions of information flow security to a model that is appropriate for the analysis of timing side channels in hardware implementations. Second, we develop efficient algorithms for deciding whether a system is secure and we show that they can be practically applied to nontrivial circuits.

The remainder of this paper is structured as follows. In Section 2, we introduce our automaton model and define security. In Section 3, we develop reduction techniques and efficient algorithms for deciding whether an automaton has secure information flow. We report on experimental results in Section 4, before we present related work and draw conclusions in Sections 5 and 6.

2 The Scenario

2.1 Machine Model

We use Mealy machines as a model for hardware circuits that are synchronized by a global clock signal. We assume that one transition corresponds to one clock cycle and that, during each clock cycle, input signals are read and output signals generated. Furthermore, we assume input enabledness, that is, the machine can always react to every possible input. While hardware is typically designed to be deterministic, nondeterminism is useful too, for example, for modeling requirements, and hence we will keep our presentation general wherever possible. As there is no standard notion of a nondeterministic Mealy machine, we use the term *automaton with output.*

Definition 1. *An* automaton with output *is a 5-tuple $M = (S, \Sigma, \Gamma, \delta, s_0)$, where S is a finite set of states, Σ is a finite input alphabet, Γ is a finite output alphabet, $\delta \subseteq S \times \Sigma \times \Gamma \times S$ is a transition relation, and $s_0 \in S$ the initial state. We call M* deterministic *if for every $(s, a) \in S \times \Sigma$ there is at most one $(b, s') \in \Gamma \times S$ with $(s, a, b, s') \in \delta$.*

In a transition (s, a, b, s'), a denotes the input and b denotes the output. We will sometimes use the shorthand $\delta(s, a, b)$ to denote the set $\{s' \mid (s, a, b, s') \in \delta\}$. As noted above, we require the transition relation to be *total*, i.e. for all $s \in S$ and $a \in \Sigma$, there is at least one $b \in \Gamma$ and one $s' \in S$ with $(s, a, b, s') \in \delta$. We do not consider ϵ-transitions as they contradict the assumption that input and output are provided during each clock cycle. In the setting of hardware circuits, Σ and Γ will be of the form $\{0, 1\}^n$, for some $n \in \boldsymbol{N}$, and represent the values of all ingoing and outgoing signals.

2.2 Defining Security

We specify security with respect to an observer of the system. An observer is modeled in terms of its capabilities for distinguishing different system behaviors. If all system runs are indistinguishable, even when the system computes with different secret data, then the system is intuitively secure. Conversely, information may leak if the system shows distinguishable behavior while processing different secrets.

Distinguishing atomic inputs/outputs. The fact that two outputs $a, b \in \Gamma$ are indistinguishable is captured by an equivalence relation $R_O \subseteq \Gamma \times \Gamma$. We say that a and b are *observationally equivalent*, or simply *R_O-equivalent*, if and only if $a \; R_O \; b$. In other words, if the system outputs $x \in \Gamma$, an observer can only deduce the R_O-equivalence class $[x]$. Similarly, we use the equivalence relation $R_I \subseteq \Sigma \times \Sigma$ to model to what extent an observer can distinguish the input of the system.

In the following, Id_X denotes the identity on a set X and All_X denotes $X \times X$. For relations $R \subseteq \Gamma_1 \times \Gamma_1$ and $Q \subseteq \Gamma_2 \times \Gamma_2$, we overload notation and define $R \times Q \subseteq (\Gamma_1 \times \Gamma_2)^2$ as $(r_1, q_1) \; (R \times Q) \; (r_2, q_2)$ if and only if $r_1 \; R \; r_2$ and $q_1 \; Q \; q_2$.

Example 1. The relation $R_O = \mathrm{All}_\Gamma$ formalizes an observer who cannot distinguish between any two system outputs. In contrast, the relation $R_O = \mathrm{Id}_\Gamma$ models an observer who can determine the (singleton) Id_Γ-equivalence class of the output, or equivalently, who can see the entire output. We can also model more fine-grained capabilities. Consider, for example, $\Gamma = \Gamma_1 \times \Gamma_2$, with $\Gamma_1 = \{0,1\}^n$, and the predicate $\Psi_{\Gamma_1} = \{(a,b) \in \Gamma_1 \times \Gamma_1 \mid \|a\| = \|b\|\}$, where $\|x\|$ denotes the Hamming weight of x, i.e. the number of bits set to 1. The relation $\Psi_{\Gamma_1} \times \mathrm{Id}_{\Gamma_2}$ models that an observer can see the entire Γ_2-component of the output, but can only deduce the Hamming weight (determine the Ψ_{Γ_1}-equivalence class) of the Γ_1-component. $\diamond$

Expressing security. Two states of a system are *observationally equivalent* if every output from one state can be matched by an R_O-equivalent output from the other state whenever the corresponding inputs are R_I-equivalent. We call the observational equivalence of states *R_I/R_O-equivalence*, which is a *partial equivalence relation* (Per), i.e. symmetric and transitive, but not necessarily reflexive. If the initial state of a system is *not* observationally equivalent to itself, then running the system on R_I-equivalent input sequences may lead to observable differences in the system behavior. This constitutes a refinement of an observer's knowledge about the input (modeled by R_I), and thus is an information leak. If, on the other hand, the initial state is observationally equivalent to itself, then we say that the system is *R_I/R_O-secure.* The idea that security can be modeled as a system being observationally equivalent to itself is formalized in the Per model of secure information flow [23].

The next section gives a formal account of these ideas.

2.3 A Parameterized Notion of Observational Equivalence

For the systems under consideration, we model observational equivalence of states by using a parameterized notion of strong bisimulation. This will capture timing behavior, as every transition corresponds to a tick of the global clock, and strong bisimulation equivalence allows one to distinguish process behaviors that differ in the number of transitions leading to some output.

Definition 2 (R_I/R_O-Equivalence). *Let $M = (S, \Sigma, \Gamma, \delta, s_0)$ be an automaton with output, and let $R_I \subseteq \Sigma^2$ and $R_O \subseteq \Gamma^2$ be equivalence relations. We define $\simeq_{R_O}^{R_I}$ as the union of all symmetric and transitive relations $\mathcal{R}$ on S with the property that for all $s_1, s_2 \in S$:*

$$s_1 \, \mathcal{R} \, s_2 \Rightarrow \forall a_1, a_2 \in \Sigma.(a_1 \, R_I \, a_2 \Rightarrow \forall (s_1, a_1, o_1, s_1') \in \delta. \; \exists (s_2, a_2, o_2, s_2') \in \delta. \; s_1' \, \mathcal{R} \, s_2' \wedge o_1 \, R_O \, o_2). \tag{1}$$

*Two states $s_1, s_2 \in S$ are R_I/R_O-*equivalent *iff $s_1 \simeq_{R_O}^{R_I} s_2$.*

Definition 3 (R_I/R_O-Security). *Let $M = (S, \Sigma, \Gamma, \delta, s_0)$ be an automaton with output, and let $R_I \subseteq \Sigma^2$ and $R_O \subseteq \Gamma^2$ be equivalence relations. Then M is R_I/R_O-*secure *iff $s_0 \simeq_{R_O}^{R_I} s_0$.*

It is easy to see that $\simeq_{R_O}^{R_I}$ is a partial equivalence relation on S and that $\simeq_{R_O}^{R_I}$ itself satisfies Property (1) of Definition 2.

We will now give several instances of R_I/R_O-security. We first show how it encompasses a number of security notions from language-based information-flow (for an overview of this area, see [21]). Afterwards, we instantiate R_I/R_O-security to specify partial information flow, which will prove to be useful in our experiments.

In the following examples, we assume two security domains, *high* and *low*, and we restrict the flow of information from the high domain to the low domain. A common assumption in programming language-based approaches is that each variable is classified as either high or low and that an observer may only see the values of the low variables. In our setting, input and output signals take the role of variables and have high and low components. This intuition is reflected by assuming that $\Sigma = \Sigma_H \times \Sigma_L$, where Σ_H and Σ_L represent the values of all high and low input signals, respectively. Similarly, we assume that $\Gamma = \Gamma_H \times \Gamma_L$. The policy that no information flows from the high into the low domain can then be formalized in the framework of R_I/R_O-equivalence by choosing $R_I = \mathrm{All}_{\Sigma_H} \times \mathrm{Id}_{\Sigma_L}$ and $R_O = \mathrm{All}_{\Gamma_H} \times \mathrm{Id}_{\Gamma_L}$. When Σ is understood, we write Id_L as an abbreviation for Id_{Σ_L}. We abbreviate analogously for Γ, All_L and the high domain.

Example 2. In the deterministic case, $\simeq_{\mathrm{All}_H \times \mathrm{Id}_L}^{\mathrm{All}_H \times \mathrm{Id}_L}$ represents a notion of observational equivalence closely related to Agat's *Γ-bisimulation* [1]. In our model, every transition takes one time unit, while in Agat's approach the duration of each transition is given by a label representing the primitive operations of the underlying machine. ◇

Example 3. In the nondeterministic case, $\simeq_{\mathrm{All}_H \times \mathrm{Id}_L}^{\mathrm{All}_H \times \mathrm{Id}_L}$ represents a possibilistic notion of security similar to Volpano and Smith's *concurrent noninterference* [26], which has been used to model the security of multithreaded programs in the presence of a purely nondeterministic scheduler. Note that our definition is more restrictive with respect to timing, as it is based on strong bisimulation equivalence as opposed to the weak bisimulation-based concurrent noninterference. ◇

In addition to capturing variants of previously studied notions of security, R_I/R_O-equivalence allows one to express more fine-grained forms of information flow.

Example 4. Consider the binary predicate $\Psi_\Sigma = \{(a,b) \in \Sigma \times \Sigma \mid \|a\| = \|b\|\}$, where $\Sigma = \{0,1\}^n$ and $\|x\|$ denotes the Hamming weight of x. Suppose we have $s_0 \simeq_{\mathrm{Id}_\Gamma}^{\Psi} s_0$, where s_0 is the initial state of a deterministic system. Then the system shows the same behavior for each pair of (and hence, by transitivity of $\simeq_{\mathrm{Id}_\Gamma}^{\Psi}$, for all) input traces $a_1 \cdots a_m$ and $b_1 \cdots b_m$, where $\|a_i\| = \|b_i\|$ for every $i \in \{1, \ldots, m\}$. ◇

The converse of Example 4 is more subtle. An observable difference between two output traces of a deterministic system implies that the input traces' Hamming weight differs at some point in time. While the leakage of a single input's

Hamming weight might be acceptable, the leakage of the Hamming weight of all symbols in the input trace can be used to encode arbitrary information.

Example 5. Consider an automaton M with a single state s_0, alphabets $\Sigma_H = \Sigma_L = \Gamma = \{0,1\}$, and transitions $\{(s_0, (h,l), h, s_0) \mid h,l \in \{0,1\}\}$. M maps every high input h to the identical output. Still, M is $\Psi_H \times \mathrm{Id}_L/\mathrm{Id}_\Gamma$-secure. ◇

To assess how much can be learned by observing the behavior of a R_I/R_O-secure system, we need to consider the environment that provides it with high input.

2.4 Environment Behavior

In this section, we consider the interaction of a deterministic automaton $M = (S, \Sigma_H \times \Sigma_L, \Gamma, \delta, s_0)$ with an environment that provides it with high input. We will show how to combine security guarantees for M with restrictions on the environment to give bounds on the number of distinguishable behaviors. This is a useful measure for assessing the information leakage from the high to the low domain since, for a deterministic system and an arbitrary low input, variations in the output are necessarily due to variations in the high input. Thus, a greater number of distinguishable output traces means that more information about the high input is leaked.

We specify the *high environment* as a subset $E \subseteq \Sigma_H^*$ of high input traces of the form $\bigcup_{i=0}^{\infty} \circ_{j=1}^{i} A_j$, where $A_j \subseteq \Sigma_H$ represents the set of possible inputs from the high environment at time instant j and $\circ$ denotes word concatentation. Dually to the requirement that M is input enabled, we require the high environment to provide an input at every clock cycle. For an arbitrary trace $w \in \Sigma_L^*$, a high environment E, and $R_O \subseteq \Gamma \times \Gamma$, we denote the set of *distinguishable behaviors* by $\mathcal{B}_{M,R_O}(E,w)$. Concretely, for $(s,a,b,s') \in \delta$, we define $\lambda_{R_O}(s,a) = [b]$, where $[b]$ denotes the R_O-equivalence class of b. We canonically extend λ_{R_O} to a mapping from input traces to R_O^*-equivalence classes of output traces. Here, the relation R_O^* is defined as $\bigcup_{i=0}^{\infty} R_O^k$ where $a_1 \cdots a_k \; R_O^k \; b_1 \cdots b_k$ iff $a_i \; R_O \; b_i$, for all $i \in \{1,\ldots,k\}$. Now we can formally define $\mathcal{B}_{M,R_O}(E,w) = \{\lambda_{R_O}(s_0, \langle v,w\rangle) \mid v \in E,\ |w| = |v|\}$, where $|\cdot|$ is the length function and where $\langle v,w\rangle$ denotes the trace in $(\Sigma_H \times \Sigma_L)^*$ obtained by pairing corresponding elements of v and w.

Recall that if a $Q \times \mathrm{Id}_L/R_O$-secure deterministic system is provided with input from a high environment E in which all traces of the same length are Q^*-equivalent, then it will produce only R_O^*-equivalent output. That is, $|\mathcal{B}_{M,R_O}(E,w)| = 1$ for every $w \in \Sigma_L^*$. If we weaken the requirement that the input is always Q-equivalent, the number of distinguishable behaviors may increase. The next theorem gives an upper bound for this number.

Theorem 1. *Let $M = (S, \Sigma_H \times \Sigma_L, \Gamma, \delta, s_0)$ be a deterministic automaton with output, let $Q \subseteq \Sigma_H \times \Sigma_H$ and $R_O \subseteq \Gamma \times \Gamma$ be equivalence relations, and let $E = \bigcup_{i=0}^{\infty} \circ_{j=1}^{i} A_j \subseteq \Sigma_H^*$ be a high environment. If M is $(Q \times Id_L)/R_O$-secure, then for all $w \in \Sigma_L^*$ we have*

$$|\mathcal{B}_{M,R_O}(E,w)| \;\leq\; \Pi_{j=1}^{\infty} |A_j/_Q| \; .$$

Proof. It suffices to prove $|\{\lambda_{R_O}(s_0, \langle v, w\rangle) \mid v \in \circ_{j=1}^{k} A_j\}| \leq \Pi_{j=1}^{k}|A_j/_Q|$ for all $w \in \Sigma_L^k$, as taking the limit $k \to \infty$ then leads to the desired result. We define the mapping $\lambda'_w : \circ_{j=1}^{k} A_j/_{Q^k} \to \{\lambda_{R_O}(s_0, \langle v, w\rangle) \mid v \in \circ_{j=1}^{k} A_j\}$ by $\lambda'_w([u]) = \lambda_{R_O}(s_0, \langle u, w\rangle)$. λ'_w is well-defined since $\lambda_{R_O}(s_0, \langle v, w\rangle) = \lambda_{R_O}(s_0, \langle v', w\rangle)$ for all v, v' with $v\ Q^k\ v'$. Note that λ'_w is surjective and that the range of a function is of cardinality less or equal than its domain, hence $|\{\lambda_{R_O}(s_0, \langle v, w\rangle) \mid v \in \circ_{j=1}^{k} A_j\}| \leq |\circ_{j=1}^{k} A_j/_{Q^k}|$ holds. We conclude with $|\circ_{j=1}^{k} A_j/_{Q^k}| = \Pi_{j=1}^{k}|A_j/_Q|$.□

Note that the mapping λ'_w from the proof of Theorem 1 expresses the correpondence between equivalence classes of high input and output behaviors. The fact that its domain is independent of w shows that an attacker cannot learn more than the Q^*-equivalence class of a fixed high input, even if he runs the system with all possible low inputs.

Example 6. If a system is $\mathrm{All}_H \times \mathrm{Id}_L/\mathrm{All}_H \times \mathrm{Id}_L$-secure (see Example 3) and is provided with input from a high environment $E = \bigcup_{i=0}^{\infty} \circ_{j=1}^{i} A_j \subseteq \Sigma_H^*$, then the number of distinguishable output behaviors is $\Pi_{i=1}^{\infty}|A_j/_{\mathrm{All}_H}| = 1$. That is, the system can be securely operated in an arbitrary high environment. ◇

Example 7. Consider again the $\Psi_H \times \mathrm{Id}_L/\mathrm{Id}_\Gamma$-secure automaton M from Example 5. The number of possible system behaviors is unbounded, as $|\{0,1\}/_{\Psi_H}| = 2$ and $\Pi_{j=1}^{\infty}|\{0,1\}/_{\Psi_H}|$ diverges. This estimation is tight in the sense that an arbitrary amount of information can be leaked. ◇

Example 8. Consider an $\Psi_H \times Id_L/\mathrm{All}_H \times Id_L$-secure circuit in which the high component is initialized during the first clock tick and subsequent high input is 0. The environment here is given by $E = \bigcup_{i=0}^{\infty} \circ_{j=1}^{i} A_j$ where $A_1 = \Sigma_H$ and $A_j = \{0\}$ for $j > 1$. Then, for each low input w, the system shows at most $|\Sigma_H/_{\Psi_H}|$ distinguishable behaviors, each of which corresponds to one Ψ_H-equivalence class. This correspondence is given by the mapping λ'_w from the proof of Theorem 1. Thus at most the secret input's Hamming weight is leaked during execution. ◇

3 Deciding R_I/R_O-Equivalence

In this section, we reduce the question of deciding R_I/R_O-equivalence to tractable, well-understood problems and we analyze the complexity of the resulting algorithms. We start with the case when the automata are deterministic, which turns out to be very efficiently solvable.

3.1 Deterministic Case

We first reduce the problem of deciding the R_I/R_O-equivalence of states to a reachability problem for a special type of product automaton. This may seem surprising as, in general, information flow properties are properties of sets of traces rather than properties of individual traces [16]. The key idea behind our construction is that every trace of the product automaton corresponds to a pair of traces of the original system. Taking the transitivity of R_I/R_O-equivalence into account, it suffices to analyze each individual trace of the product automaton in order to establish R_I/R_O-security for the original system.

Definition 4. *Let $M_i = (S_i, \Sigma, \Gamma, \delta_i, s_{0,i})$, with $i \in \{1,2\}$, be deterministic automata with output, and let R_I and R_O be equivalence relations on Σ and Γ, respectively. Then $M_1 \times^{R_I}_{R_O} M_2$ is the automaton $(S_1 \times S_2, R_I, \{0,1\}, \delta', (s_{0,1}, s_{0,2}))$, where*

$$\delta' = \{((s_1,s_2),(a,b),\chi,(t_1,t_2)) \mid a\ R_I\ b \wedge (\chi = \mathit{if}\ c\ R_0\ d\ \mathit{then}\ 1\ \mathit{else}\ 0) \wedge (s_1,a,c,t_1) \in \delta_1 \ \wedge\ (s_2,b,d,t_2) \in \delta_2\}\,.$$

A *falsifying state* is a state with an outgoing transition labeled with 0. We now show that deciding observational equivalence of states is equivalent to determining whether a falsifying state can be reached in $M \times^{R_I}_{R_O} M$.

Theorem 2. *Let $M = (S, \Sigma, \Gamma, \delta, s_0)$ be a deterministic automaton with output, $R_I \subseteq \Sigma \times \Sigma$ and $R_O \subseteq \Gamma \times \Gamma$ equivalence relations, and let $s_1, s_2 \in S$. Then*

$$s_1 \simeq^{R_I}_{R_O} s_2 \Leftrightarrow \textit{no falsifying state is reachable from } (s_1,s_2) \textit{ in } M \times^{R_I}_{R_O} M\,.$$

Proof. ($\Rightarrow$) We show that no input $w \in (R_I)^*$ can trigger a transition labeled with 0. We proceed by induction on the length of w. The assertion is clear for $w = \epsilon$. Suppose now that $w = (a,b)w'$. As $s_1 \simeq^{R_I}_{R_O} s_2$ and δ is total and deterministic, there are unique transitions (s_1,a,c,t_1) and $(s_2,b,d,t_2) \in \delta$, with $t_1 \simeq^{R_I}_{R_O} t_2$ and $(c,d) \in R_O$. Hence $M \times^{R_I}_{R_O} M$ outputs 1 on this transition and we apply the induction hypothesis to (t_1,t_2) and w'.

($\Leftarrow$) We show that $\mathcal{Q} = \{(t_1,t_2) \mid (t_1,t_2)$ can be reached from $(s_1,s_2)\}$ fulfills (1) of Definition 2. Pick $(t_1,t_2) \in \mathcal{Q}$ and $(a,b) \in R_I$. Since δ is total and deterministic, there are unique transitions (t_1,a,c,t_1') and $(t_2,b,d,t_2') \in \delta$. Clearly, (t_1',t_2') can also be reached from (s_1,s_2) in $M \times^{R_I}_{R_O} M$ and, as no transition labeled with 0 can be triggered by assumption, $(c,d) \in R_O$ holds. Hence $\mathcal{Q}$ is contained in the union of all relations with (1) of Definition 2. □

This theorem justifies a simple decision procedure where we decide R_I/R_O-equivalence by searching the product automaton from Definition 4. We use breadth-first search, as it will find a shortest path to a falsifying state.

Corollary 1. *Let $M = (S, \Sigma, \Gamma, \delta, s_0)$ be a deterministic automaton with output, let $s_1, s_2 \in S$, and let $R_I \subseteq \Sigma$ and $R_O \subseteq \Gamma$ be equivalence relations. Then $s_1 \simeq^{R_I}_{R_O} s_2$ can be decided in time $\mathcal{O}(|S|^2|R_I|)$, given the product automaton $M \times^{R_I}_{R_O} M$.*

Proof. Breadth-first search can be implemented in time $\mathcal{O}(|V|+|E|)$ on a graph $G = (V,E)$. $M \times^{R_I}_{R_O} M$ has $|S|^2$ states and $|S|^2|R_I|$ transitions. This yields an $\mathcal{O}(|S|^2|R_I|)$ upper bound for the time complexity of deciding R_I/R_O-equivalence. □

3.2 Nondeterministic Case

A straightforward extension of the above reduction does not appear possible in the nondeterministic case. Instead, we use the Partition Refinement Problem [19] as a starting point for deciding R_I/R_O-equivalence.

A *partition* of a set S is a set $\pi = \{A_1, \dots, A_n\}$ of pairwise disjoint *blocks* with the property that $\bigcup_{i=1}^{n} A_i = S$. A *refinement* of a partition π is a partition π' such that every block of π' is contained in some block of π. A partition π can also be formalized in terms of an equivalence relation R_π, where the elements of π correspond to the equivalence classes of R_π. The Partition Refinement Problem is, given a partition π and a property P, to find the coarsest refinement π' of π such that π' fulfills P. This is equivalent to finding the greatest equivalence relation $R_{\pi'}$, with $R_{\pi'} \subseteq R_\pi$, such that $R_{\pi'}$ satisfies P.

Since R_I/R_O-equivalence is a partial equivalence relation, PER for short, we need to generalize the Partition Refinement Problem. The *Partial Partition Refinement Problem* is, given a partial equivalence relation R and a property P, to find the coarsest refinement R' of R, such that R' satisfies P. We next show that the problem of deciding R_I/R_O-equivalence can be cast as an instance of this problem. Then, following the ideas in [11], we compute this coarsest refinement as the maximal fixed point of a monotone mapping Φ.

The *domain* of a partial equivalence relation R is the set $\mathrm{dom}(R) = \{x \in S \mid x\, R\, x\}$ on which R is reflexive, and hence an equivalence relation. A *partial partition* of a set S is a pair $\langle \{A_1, \dots, A_n\}, C\rangle$, where the A_i are pairwise disjoint blocks with $\bigcup_{i=1}^{n} A_i \cup C = S$ and $\bigcup_{i=1}^{n} A_i \cap C = \emptyset$. There is a one-to-one correspondence between PERs R of a set S and partial partitions $\langle \{A_1, \dots, A_n\}, C\rangle$, where the A_i correspond to the equivalence classes of R, and $C = S \setminus \mathrm{dom}(R)$. As notation, we denote this correspondence as $\langle \{A_1, \dots, A_n\}, C\rangle \mathrel{\hat{=}} \langle R, C\rangle$. Let $\pi_1 = \langle \{A_1, \dots, A_n\}, C_1\rangle$ and $\pi_2 = \langle \{B_1, \dots, B_m\}, C_2\rangle$ be partial partitions of S. We define $\pi_1 \leq \pi_2$ to hold whenever $C_1 \supseteq C_2$ and if every block of π_1 is contained in some block of π_2. The relation $\leq$ is a partial order on the set of all partial partitions of a set S. In fact, it is also a lattice when we define the meet $\sqcap$ as $\langle \{A_1, \dots, A_n\}, C_1\rangle \sqcap \langle \{B_1, \dots, B_m\}, C_2\rangle = \langle \{E_{i,j}\}, C_1 \cup C_2\rangle$, with $E_{i,j} = A_i \cap B_j$ for $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, m\}$.

In the remainder of this subsection, let $M = (S, \Sigma, \Gamma, \delta, s_0)$ be a nondeterministic automaton with output, and let $R_I \subseteq \Sigma \times \Sigma$ and $R_O \subseteq \Gamma \times \Gamma$ be equivalence relations.

Definition 5 (R_I/R_O-partition). *A* R_I/R_O-partition of S *is a partial partition* $\langle \{A_1, \dots, A_n\}, C\rangle$ *of* S*, with*

$$\begin{aligned} &\forall i, j \in \{1, \dots, n\}.\ \forall s_1, s_2 \in A_i.\ \forall (a_1, a_2) \in R_I.\ \forall x \in \Gamma/_{R_O}. \\ &\quad \overline{\delta}(s_1, a_1, x) \cap A_j \neq \emptyset \ \Leftrightarrow\ \overline{\delta}(s_2, a_2, x) \cap A_j \neq \emptyset \ \wedge \\ &\quad \overline{\delta}(s_1, a_1, x) \cap C \ =\ \overline{\delta}(s_2, a_2, x) \cap C \ =\ \emptyset\,, \end{aligned} \tag{2}$$

where $\overline{\delta}(s, a, x)$ *denotes the set* $\bigcup_{c \in x} \delta(s, a, c)$*. A* R_I/R_O*-partition* π *of* S *is* maximal *if* $\pi \geq \pi'$ *holds for every* R_I/R_O*-partition* π' *of* S*.*

We adapt (2) of Definition 5 to a mapping on partial partitions whose fixed points are precisely the R_I/R_O-partitions of S. To this end, let $\pi = \langle\{A_1,\ldots,A_n\},C_1\rangle \hat{=} \langle R_1, C_1\rangle$ be a partial partition of S. We define $\Phi(\pi) := \langle R_2, S \setminus \mathrm{dom}\,(R_2)\rangle$, where $s_1\ R_2\ s_2$ if and only if

$$\begin{aligned} s_1\ R_1\ s_2 \ \wedge\ \forall j \in \{1,\ldots,n\}.\ \forall (a_1,a_2) \in R_I.\ \forall x \in \Gamma/_{R_O}. \\ \overline{\delta}(s_1,a_1,x) \cap A_j \neq \emptyset \ \Leftrightarrow\ \overline{\delta}(s_2,a_2,x) \cap A_j \neq \emptyset \ \wedge \\ \overline{\delta}(s_1,a_1,x) \cap C_1 \ =\ \overline{\delta}(s_2,a_2,x) \cap C_1 \ =\ \emptyset\,. \end{aligned}$$

Lemma 1. *Let $\langle R, C\rangle$ be a partial partition of the set of states S. Then the following are equivalent:*

1. *$\langle R, C\rangle$ is a fixed point of Φ.*
2. *$\langle R, C\rangle$ is a R_I/R_O-partition of S.*
3. *R satisfies (1) of Definition 2.*

The proof of Lemma 1 is given in Appendix A. From Lemma 1, it follows that the relation $\simeq_{R_O}^{R_I}$ is a maximal fixed point of the function Φ. In particular, $\simeq_{R_O}^{R_I}$ itself satisfies (1) of Definition 2 and is thus contained in every maximal fixed point of Φ. Conversely, as every fixed point of Φ satisfies Property (1), the maximal fixed point is contained in $\simeq_{R_O}^{R_I}$, the union of all such relations.

The following theorem gives a constructive way to derive maximal R_I/R_O-partitions.

Theorem 3. *There exists a unique maximal R_I/R_O-partition π^* of S, namely, $\pi^* = \Phi^n(\langle\{S\},\emptyset\rangle)$, for some $n \in \boldsymbol{N}$.*

Proof (Sketch). First observe that Φ is monotone with respect to $\leq$. Now since the set of partial partitions of S is a complete lattice, it follows from the Knaster-Tarski fixed point theorem that a unique maximal fixed point of Φ exists. By Lemma 1, this fixed point is also a maximal R_I/R_O-partition. The full proof details are given in Appendix A. □

Theorem 3 provides the basis of an efficient algorithm for deciding the R_I/R_O-equivalence of states.

Corollary 2. *For two states $s_1, s_2 \in S$ we can decide $s_1 \simeq_{R_O}^{R_I} s_2$ in time*

$$\mathcal{O}(|S|^4 \cdot |R_I| \cdot |\Gamma/_{R_O}|)\,,$$

under the assumption that $\overline{\delta}(s,a,x) = \bigcup_{c\in x} \delta(s,a,c)$ is given as an array indexed by $s \in S$, $a \in \Sigma$, and $x \in \Gamma/_{R_O}$.

Proof. It suffices to show that a single application of Φ can be computed in time $\mathcal{O}(|S|^3 \cdot |R_I| \cdot |\Gamma/_{R_O}|)$. Due to Theorem 3, a fixed point can be obtained by iteratively applying Φ. As $\Phi(\pi) \leq \pi$ for every partition π, this process terminates within at most $|S|$ applications.

We assume $S = \{s_1, \ldots, s_n\}$ and that the equivalence class of each state is given by a representative s_i with minimal i, and by a distinguished symbol $* \notin S$ if the state is outside the domain of the relation. For example, in the case of $\pi_\top = \langle \{S\}, \emptyset \rangle$, the canonical representative for every state is s_1. Suppose now we are given a partial partition $\pi = \langle R, C \rangle$ and we want to compute $\Phi(\pi) = \langle R', C' \rangle$. To decide whether two states s_i and s_j relate in R', we perform the following procedure: for all $(a_1, a_2) \in R_I$, and for all $x \in \Gamma/_{R_O}$, we compare the corresponding sets of R-equivalence classes of the target states. If all of the corresponding sets coincide, s_i and s_j are in the same R'-equivalence class. By iterating i stepwise from 1 to n, we perform this check for every $j \in \{1, \ldots, n\}$. Under this ordering, the canonical representative of the R'-equivalence class of each s_j is the s_i with minimal index such that equivalence of s_i and s_j can be established, and $*$ if there is no such s_i. In this way, each application of Φ can be computed in time $\mathcal{O}(|S|^3 \cdot |R_I| \cdot |\Gamma/_{R_O}|)$. $\square$

3.3 Compositionality

Compositionality is a prerequisite for scaling our analysis method to larger systems. In this section we use the example of sequential composition to show how the guarantees obtained from analyzing sub-circuits can be combined to a guarantee for the entire system. To this end, we first define an operator that connects the output signals of a machine M_1 to the input signals of a machine M_2. M_2's transition function is total, hence communication never blocks. This notion of composition models a sequential connection of two synchronous circuits with a common clock.

Definition 6. *Let $M_i = (S_i, \Sigma_i, \Gamma_i, \delta_i, s_{0,i})$, with $i \in \{1, 2\}$, be automata with output and let $\Gamma_1 \subseteq \Sigma_2$. Then $M_1 \cdot M_2$ is the automaton $(S_1 \times S_2, \Sigma_1, \Gamma_2, \delta', (s_{0,1}, s_{0,2}))$, where*

$$\delta' = \{((s_1, s_2), a, b, (t_1, t_2)) \mid \exists c \in \Gamma_1.\ (s_1, a, c, t_1) \in \delta_1 \wedge (s_2, c, b, t_2) \in \delta_2\}\,.$$

If the observational equivalence relation on the input alphabet of M_2 is coarser than the one on the output alphabet of M_1, then we can safely compose the two machines, as the following theorem shows.

Theorem 4. *Let $M_i = (S_i, \Sigma_i, \Gamma_i, \delta_i, s_{0,i})$, with $i \in \{1, 2\}$, be automata with output and let $R_I \subseteq \Sigma_1 \times \Sigma_1$, $R_O \subseteq \Gamma_1 \times \Gamma_1$, $Q_I \subseteq \Sigma_2 \times \Sigma_2$, and $Q_O \subseteq \Gamma_2 \times \Gamma_2$ be equivalence relations. Let $s_1, s_2 \in S_1$ and $t_1, t_2 \in S_2$. If $\Gamma_1 \subseteq \Sigma_2$, $R_O \subseteq Q_I$, $s_1 \simeq_{R_O}^{R_I} s_2$, and $t_1 \simeq_{Q_O}^{Q_I} t_2$, then*

$$(s_1, t_1) \simeq_{Q_O}^{R_I} (s_2, t_2) \text{ in } M_1 \cdot M_2\,.$$

The proof of Theorem 4 is given in Appendix A.

Example 9. Suppose $M_i = (S_i, \Sigma_i, \Gamma_i, \delta_i, s_{0,i})$, for $i \in \{1, 2\}$, are automata with output, $\Sigma_1 = \Sigma_H \times \Sigma_L$, and $\Gamma_1 = \Sigma_2 = \{0, 1\}^n$. If the output of M_1 is not distinguishable with respect to the Hamming weight, i.e. $s_{0,1} \simeq_{\Psi}^{\mathrm{All}_H \times \mathrm{Id}_L} s_{0,1}$, (where $\Psi = \{(a,b) \in \{0,1\}^n \mid \|a\| = \|b\|\}$) and if M_2 does not leak anything other than possibly the Hamming weight, i.e. $s_{0,2} \simeq_{\mathrm{Id}_{\Gamma_2}}^{\Psi} s_{0,2}$, then the composition $M_1 \cdot M_2$ does not leak any information, i.e. the initial state relates to itself under $\simeq_{\mathrm{Id}_L}^{\mathrm{All}_H \times \mathrm{Id}_{\Gamma_2}}$. $\diamond$

Analogous results hold for the parallel composition of two circuits.

4 Experimental Results

Below we report on two case studies: a simple circuit for bit-serial multiplication of nonnegative integers and a circuit for exponentiation in the field $\mathbb{F}_{2^k}$. Exponentiation over $\mathbb{F}_{2^k}$ is relevant, for example, in the generalized ElGamal encryption scheme, where decryption consists of one exponentiation and one multiplication step [17]. We implemented and tested both circuits in the hardware description language Gezel. Instead of implementing a search procedure by hand, we used the symbolic model checker Smv to automate the search on the product automaton from Definition 4.[1] Note that we translated the Gezel implementations to the input language of Smv by hand. However, the semantic gap between both languages is so small that an automated translation would be straightforward.

4.1 The Circuits

Bit-serial multiplication. For multiplying two natural numbers m and n bitwise, consider the representation $n = \Sigma_{i=0}^{k-1} n_i 2^i$, where n_i denotes the ith bit of n. The product $m \cdot \Sigma_{i=0}^{k-1} n_i 2^i$ can be expanded to

$$(\dots((n_{k-1} \cdot m) \cdot 2 + n_{k-2} \cdot m) \cdot 2 + \dots) \cdot 2 + n_0 \cdot m\,,$$

which can easily be turned into an algorithm: starting with $p = 0$, one iterates over all the bits of n, beginning with the most significant bit. If $n_i = 1$, one updates p by adding m and then doubling p's value. Alternatively, if $n_i = 0$, one updates p by just doubling its value. At the end of the loop, $p = m \cdot n$.

We implemented two versions of this algorithm. In the first version, the doubling and adding operations each take one clock cycle. Hence, the running time reflects the number of 1-bits in n. In the second version, we introduce a dummy step whenever no addition takes place. In this way, the running time is independent of the operands. In our Smv implementations, the input signals are called `hi_in` and `lo_in` and they are initialized during the first clock cycle with the values of n and m, respectively. Input values of subsequent cycles are ignored. We use two output signals: one for the result `p` and a flag `done`, which signals termination.

[1] The Gezel and Smv-code is given in the accompanying technical report [13].

Exponentiation in a finite field. We analyzed a hardware implementation of the finite field exponentiation algorithm from [6]. Basically, it consists of the following three building blocks:

1. To compute the *exponentiation of a field element* x with exponent $a = \Sigma_{i=0}^{n-1} a_i 2^i$, one iterates over all bits of the exponent
$$x^a = (\dots(((x^{a_{n-1}})^2 \cdot x^{a_{n-2}})^2 \cdot x^{a_{n-3}})^2 \cdot \dots)^2 \cdot x^{a_0} \,. \tag{3}$$
In finite fields, every element x is represented by the coefficients of a polynomial, and thus each square and each multiplication operation in Equation 3 is again implemented by a loop.
2. *Multiplication of polynomials* q and $x = \Sigma_{j=0}^{r-1} x_j T^j$ is computed using the expansion $(\dots((x_{r-1} \cdot q) \cdot T + x_{r-2} \cdot q) \cdot T + \dots) + x_0 \cdot q$ in a loop similar to the one for bit-serial multiplication.
3. At the bit level, *multiplication by* T of a polynomial represented by coefficients $s = (s_{r-1}, \dots, s_0)$ can be implemented as follows. If $s_{r-1} = 0$, left-shift s by one. If $s_{r-1} = 1$, left-shift s by one and XOR the result with the coefficients of the field polynomial.

In our Smv-implementation of this exponentiation algorithm, the input signals `hi_in` and `lo_in` are initialized during the first clock cycle with a and x, respectively. Input values of subsequent cycles are ignored. We use two output signals, `p` and `done`, to represent the result x^a and termination, respectively.

4.2 Security Properties

We analyzed the multiplication and the exponentiation circuits from Section 4.1 with respect to two different security properties.

Property 1. We specify that a circuit's running time is independent of the confidential part of the input, the input signal `hi_in`. Recall that, in the case of serial multiplication, `hi_in` and `lo_in` are initialized with the operands n and m, respectively. In the case of exponentiation, `hi_in` and `lo_in` are initialized with the exponent a and the basis x, respectively. For verifying that the execution time is independent of the high input, we are only interested in when the computation terminates, that is, when the flag `done` is set, and we ignore all other output. This is specified by the relation $\simeq^{\mathrm{All}_{\Sigma_H} \times \mathrm{Id}_{\Sigma_L}}_{\mathrm{All}_{\Gamma_H} \times \mathrm{Id}_{\Gamma_L}}$. Here, $\Sigma_H = \{0,1\}^k$ denotes the range of `hi_in`, and $\Sigma_L = \{0,1\}^l$ denotes the range of `lo_in`. The `done`-flag ranges over $\Gamma_L = \{0,1\}$, and Γ_H stands for all output that is not considered.

Figure 1 demonstrates how the product construction of Definition 4 can be encoded in a few lines of Smv-code. The system to be analyzed is a module that we call `circuit`, which we instantiate twice in line 4. Both instances, `sys1` and `sys2`, are provided with the same low input (as specified by Id_{Σ_L}), and are provided with all possible combinations of high inputs (as specified by All_{Σ_H}). This

```
1   MODULE main
2   VAR
3    lo,hi1,hi2 : array (SIZE-1)..0 of boolean;
4    sys1 : circuit; sys2 : circuit;
5   ASSIGN
6    sys1.lo_in:=lo; sys1.hi_in:=hi1;
7    sys2.lo_in:=lo; sys2.hi_in:=hi2;
8
9  SPEC !EF(!sys1.done=sys2.done)
```

Fig. 1. Product construction in SMV

is reflected in lines 6 and 7. In fact, all such input combinations are considered, as no assignments are made to the variables `lo`, `hi1`, and `hi2`.

Reachability of a falsifying state of the product automaton corresponds to a violation of the CTL-formula `!EF(!sys1.done=sys2.done)` in line 9. If we reach a state in which one instance's `done` flag is set before the other instance terminates, then we have found a falsifying state of the product automaton. In this case, SMV computes a counterexample, namely, two $\mathrm{All}_{\Sigma_H} \times \mathrm{Id}_{\Sigma_L}$-equivalent input sequences that lead to a distinguishable output.

Property 2. While it is easy to see that the running times of the multiplication and exponentiation algorithms depend on the input to the `hi_in`-signal, it is less clear what these dependencies are. We now specify and check a second property that formalizes that the running time only depends on the Hamming weight of the `hi_in` input (see also Example 4). That is, if the system is provided with two input sequences that are indistinguishable with respect to the Hamming weight of corresponding inputs to `hi_in`, then we require that the system has equivalent timing behavior. This is specified by the relation $\simeq^{\Psi \times Id_{\Sigma_L}}_{\mathrm{All}_{\Sigma_H} \times Id_{\Gamma_L}}$, where $\Psi = \{(a, b) \in \Sigma_H \times \Sigma_H \mid \|a\| = \|b\|\}$. Here again, $\Sigma_H = \{0, 1\}^k$ denotes the range of `hi_in`, and $\Sigma_L = \{0, 1\}^l$ denotes the range of `lo_in`. The `done`-flag ranges over $\Gamma_L = \{0, 1\}$, and Γ_H stands for all output that is not considered.

The SMV-implementation of Property 2 follows along the same lines as the implementation of Property 1. The only difference is that we modify the input to `hi_in` of `sys2` in line 7 of Figure 1 in the following way:

```
sys2.hi_in:=
case
  hi1[0]+...+hi1[SIZE-1]=hi2[0]+...+hi2[SIZE-1] : hi2;
  1 : hi1;
esac;
```

The variables `hi1` and `hi2` both take all possible values in their range. Only when their Hamming weight coincides is `sys2` fed with `hi2`. Otherwise its input is `hi1`. In this way, we ensure that the inputs to both instances of `circuit` always have the same Hamming weight and that all such combinations are considered.

4.3 Results

Security Analysis. The table in Figure 2 presents the results of our analysis. The first column corresponds to the serial multiplication algorithm where dummy steps are inserted to avoid timing leaks. The second column corresponds to the multiplication algorithm without dummy steps, and the third column contains the results for the finite-field exponentiation algorithm. The rows correspond to Properties 1 and 2 described in Section 4.2. An entry ✓ denotes that the model is secure with respect to the corresponding notion of security, whereas $\times$ denotes that this is not the case.

The first column reflects what was intended by inserting dummy computation steps into the design: the circuit's running time is independent of the input to the signal `hi_in`. In particular, as Example 6 shows, arbitrary input sequences do not lead to distinguishable behavior.

The second column shows that the running time of the multiplication algorithm without dummy computation depends on the input to the signal `hi_in`. However, if the implementation is only run on inputs with equal Hamming weight, then we cannot observe any differences between the running times. Example 8 shows that, if the high environment provides input only during the first clock cycle, no more than the Hamming weight of the input can be leaked. Note that this actually holds in an arbitrary environment, as the circuit ignores input during all but the first clock ticks.

The third column shows that the running time of the exponentiation algorithm depends on the input to the signal `hi_in`, which corresponds to the exponent. The result of the analysis with respect to inputs of equal Hamming weights is surprising. When only considering loop 1 (see Section 4.1), one might expect the same result as for serial multiplication. However, the second row states that this is not the case: even when provided with input of the same Hamming weight, the system shows differences in its running times. This means that information other than the Hamming weight can be leaked. We have not yet undertaken a precise characterization of this leak. The counterexample computed by SMV suggests that this might be nontrivial: the first difference between the sequences of states reached in both instances of `circuit` occurs after 20 steps, and distinguishable output is not produced until 36 steps.

Performance. We performed our experiments on a 2.4 GHz machine with 3 gigabytes of RAM. In the case of serial multiplication, we were able to analyze designs up to 10 bits per operand within one minute. In the case of exponentiation, we were able to analyze designs with up to 3 bits per operand within 2 minutes.[2] For larger bit-widths the running times increased notably. Note that these numbers were obtained by using SMV "out of the box", that is, without applying one of the many existing optimization techniques. We expect a significant performance gain by tailoring the search procedure to our spe-

[2] This corresponds to a state-space size of approximately 2^{52} for the product automaton.

	Multiplication (padded)	Multiplication	Exponentiation
$\simeq^{\mathrm{All}_H \times Id_L}_{\mathrm{All}_H \times Id_L}$	✓	×	×
$\simeq^{\Psi \times Id_L}_{\mathrm{All}_H \times Id_L}$	✓	✓	×

Fig. 2. Results of Analysis

cific problem instance, for example by adopting abstraction techniques for handling bit-vectors.

5 Related Work

Both timing-aware security definitions and decidability results exist in process algebraic settings, e.g., [7, 15], to name just a few. Their standard model of communication is event-based and differs significantly from our time-synchronous model. Likewise, security definitions for process algebras usually restrict the detection of secret events by low-level observers, while R_I/R_O-security aims at protecting a stream of confidential data. While formal connection between language-based and process algebraic approaches can be made [8], we focus on methods from language-based security as they are more directly related to our work.

Several authors use bisimulations to express timing-sensitive notions of secure information flow, e.g., [29, 1, 22]. The use of arbitrary equivalence relations for capturing partial information flow has been proposed in a timing-insensitive context [2, 9]. In this context, the notion of *independent composition* [2] is related to our product construction $\times^{R_I}_{R_O}$. R_I/R_O-security marries the timing-awareness of the bisimulation-based approaches with the accuracy of the parameterized approaches. In [10], a parameterized and timing-aware definition of secure information flow is given. However, it does not allow for input sequences of arbitrary length and it is unclear whether it can be efficiently checked. The idea of quantifying information by the number of distinguishable behaviors has been proposed by Lowe [15] as an over-approximation for Shannon's information-theoretic measure.

Programming language-based approaches to counter timing leaks usually assume infinite-state transition systems, which leads to undecidable analysis problems. One way to approximate undecidable security conditions is to use syntax-driven techniques, such as security type systems. Several security type systems for dealing with timing-sensitive notions of secure information flow for programming languages have been proposed [1, 29, 22, 14, 3]. We exploit the fact that the state spaces in our setting are finite to develop a method for efficiently deciding system security.

Tolstrup et al. [28] present an information flow analysis method for the hardware description language VHDL that does not consider timing issues. A recent follow-up paper [27] also incorporates timing and provides a type-system to approximate a semantic definition of security. Analyzing hardware on the level of VHDL has the advantage of being very concrete, but it also means that one has to deal with artifacts such as processes and δ-time. Our automata-based model is more abstract and it allows for a clean separation of program semantics and security definitions. Moreover, our approach has the advantage of an efficient decision procedure.

6 Conclusions and Outlook

The results presented in this paper are both theoretical and practical. On the theoretical side, we have developed a parametric notion of security for an automaton model for synchronous systems and have given algorithms and complexity bounds for its decision problem. In the deterministic case, we have derived quantitative bounds for the confidential information that a system may reveal to an attacker. On the practical side, we have shown that our definitions encompass a number of interesting security properties and applied our techniques to verify (or detect timing leaks in) nontrivial hardware implementations of cryptographic algorithms.

While the notion of R_I/R_O-security proposed appears to be a general and useful parametric notion of information flow, counting distinguishable behaviors provides only an approximate measure of the quantity of information that a system may leak. It should not be difficult though to incorporate probability distributions on the inputs to give more concrete, information-theoretic bounds, e.g. along the lines of [5].

Another area for future work concerns algorithms and abstractions that can help us manage both larger systems and those with infinite state-spaces. It would also be interesting to use our security notions as a starting point for techniques to automatically correct insecure systems.

References

1. J. Agat. Transforming out Timing Leaks. In *Proc. POPL '00*, pages 40–53.
2. G. Barthe, P. D'Argenio, and T. Rezk. Secure Information Flow by Self-Composition. In *Proc. CSFW '04*, pages 100–114.
3. G. Barthe, T. Rezk, and M. Warnier. Preventing Timing Leaks Through Transactional Branching Instructions. In *Proc. QAPL '05.*
4. D. Boneh and D. Brumley. Remote Timing Attacks are Practical. In *Proc. USENIX Security Symposium '03*, 2003.
5. D. Clark, S. Hunt, and P. Malacaria. Quantitative Information Flow, Relations and Polymorphic Types. *J. Log. Comput.*, 18(2):181–199, 2005.
6. M. Davio, J. P. Deschamps, and A. Thayse. *Digital Systems with Algorithm Implementation.* John Wiley & Sons, Inc., 1983.
7. R. Focardi, R. Gorrieri, and F. Martinelli. Information Flow Analysis in a Discrete-Time Process Algebra. In *Proc. CSFW '00*, pages 170–184.

8. R. Focardi, S. Rossi, and A. Sabelfeld. Bridging Language-Based and Process Calculi Security. In *Proc. FoSSaCS '05*, LNCS 3829, pages 299–315.
9. R. Giacobazzi and I. Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *Proc. POPL'04*, pages 186–197.
10. R. Giacobazzi and I. Mastroeni. Timed Abstract Non-Interference. In *Proc. FORMATS'05*, LNCS 3829, pages 289–303.
11. P. Kanellakis and S. Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. *Information and Computation*, 86:43–68, 1990.
12. P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proc. CRYPTO '96*, LNCS 1109, pages 104–113.
13. B. Köpf and D. Basin. Timing-Sensitive Information Flow Analysis for Synchronous Systems. Technical Report 526, ETH Zürich, 2006.
14. B. Köpf and H. Mantel. Eliminating implicit information leaks by transformational typing and unification. In *In Proc. FAST'05*, LNCS 3866, pages 42–62, 2006.
15. G. Lowe. Quantifying Information Flow. In *Proc. CSFW '02*, pages 18–31.
16. J. D. McLean. A General Theory of Composition for Trace Sets Closed under Selective Interleaving Functions. In *Proc. IEEE Symp. on Security and Privacy '94*, pages 79–93.
17. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
18. D. A. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: the Case of AES. In *Proc. CT-RSA '06*, LNCS 3860, pages 1–20.
19. R. Paige and R. E. Tarjan. Three Partition Refinement Algorithms. *SIAM J. Comput.*, 6(16):973–989, 1987.
20. P. Puschner and A. Burns. A Review of Worst-Case Execution-Time Analysis. *Real-Time Systems*, 18(2/3):115–128, 2000.
21. A. Sabelfeld and A. C. Myers. Language-based Information-Flow Security. *J. Selected Areas in Communication*, 21(1):5–19, 2003.
22. A. Sabelfeld and D. Sands. Probabilistic Noninterference for Multi-threaded Programs. In *Proc. CSFW '00*, pages 200–215.
23. A. Sabelfeld and D. Sands. A PER Model of Secure Information Flow in Sequential Programs. *Higher-Order and Symbolic Computation*, 14(1):59–91, 2001.
24. P. Schaumont and I. Verbauwhede. Domain-Specific Codesign for Embedded Security. *IEEE Computer*, 36(4):68–74, 2003.
25. P. Schaumont and I. Verbauwhede. The Descriptive Power of GEZEL. Technical report, 2005.
26. G. Smith and D. Volpano. Secure Information Flow in a Multi-Threaded Imperative Language. In *Proc. POPL '98*, pages 355–364.
27. T. Tolstrup and F. Nielson. Analyzing for Absence of Timing Leaks in VHDL. In *Proc. WITS '06 (to appear).*
28. T. Tolstrup, F. Nielson, and H. Nielson. Information Flow Analysis for VHDL. In *Proc. PaCT '05*, LNCS 3606, pages 79–98.
29. D. Volpano and G. Smith. Eliminating Covert Flows with Minimum Typings. In *Proc. CSFW '97*, pages 156–168.

A Proofs

In the following, let $M = (S, \Sigma, \Gamma, \delta, s_0)$ be a nondeterministic automaton with output, and let $R_I \subseteq \Sigma \times \Sigma$ and $R_O \subseteq \Gamma \times \Gamma$ be equivalence relations.

Lemma 1. *Let $\langle R, C\rangle$ be a partial partition of the set of states S. Then the following are equivalent:*

1. *$\langle R, C\rangle$ is a fixed point of Φ.*
2. *$\langle R, C\rangle$ is a R_I/R_O-partition of S.*
3. *R satisfies (1) of Definition 2.*

Proof. (*1. $\Rightarrow$ 2.*) The assertion follows by setting $R_1 = R_2 = R$ in the definition of Φ, and observing that two states s_1 and s_2 relate in R whenever they are contained in the same set A_i of the corresponding partial partition.

(*2. $\Rightarrow$ 3.*) Let $s_1\ R\ s_2$, $a_1\ R_I\ a_2$, and $(s_1, a_1, c_1, s_1') \in \delta$. As $\overline{\delta}(s_1, a_1, [c_1]) \cap C = \emptyset$, we have $s_1' \in \overline{\delta}(s_1, a_1, [c_1]) \cap A$ for some equivalence class A of R. By hypothesis, we also have $\overline{\delta}(s_2, a_2, [c_1]) \cap A \neq \emptyset$, and hence there is a transition $(s_2, a_2, c_2, s_2') \in \delta$ with $c_1\ R_O\ c_2$ and $s_1'\ R\ s_2'$.

(*3. $\Rightarrow$ 1.*) Let $\langle R, C\rangle \mathrel{\hat{=}} \langle\{A_1, \ldots, A_n\}, C\rangle$ and $\Phi(\langle R, C\rangle) = \langle R', C'\rangle$. It suffices to show that $R' = R$. The implication $R' \subseteq R$ follows directly from the definition of Φ. To show that $R \subseteq R'$, choose $s_1\ R\ s_2$ and $a_1\ R_I\ a_2$. If $\overline{\delta}(s_1, a_1, x) \cap A_j \neq \emptyset$, then there is a $(s_1, a_1, c_1, s_1') \in \delta$ with $c_1 \in x$ and $s_1' \in A_j$. As R satisfies (1) of Definition 2, there is also a $(s_2, a_2, c_2, s_2') \in \delta$, with $c_2 \in x$ and $s_2' \in A_j$. Hence $\overline{\delta}(s_2, a_2, x) \cap A_j \neq \emptyset$, and $R \subseteq R'$ follows. □

Theorem 3. *There exists a unique maximal R_I/R_O-partition π^* of S, namely, $\pi^* = \Phi^n(\langle\{S\}, \emptyset\rangle)$, for some $n \in \boldsymbol{N}$.*

Proof. To apply the Knaster-Tarski fixed-point theorem, it suffices to show that Φ is monotone. To this end, consider the partial partitions $\pi_1 = \langle\{A_1, \ldots, A_n\}, C_1\rangle \mathrel{\hat{=}} \langle Q_1, C_1\rangle$ and $\pi_2 = \langle\{B_1, \ldots, B_m\}, C_2\rangle \mathrel{\hat{=}} \langle Q_2, C_2\rangle$, where $\pi_1 \leq \pi_2$. Furthermore, let $\Phi(\pi_1) = \langle Q_1', C_1'\rangle$ and $\Phi(\pi_2) = \langle Q_2', C_2'\rangle$. We need to show that $s_1\ Q_1'\ s_2$ implies $s_1\ Q_2'\ s_2$. Assume $s_1\ Q_1'\ s_2$. By the definition of Φ, this implies $s_1\ Q_1\ s_2$, which implies $s_1\ Q_2\ s_2$. Furthermore, for all $(a_1, a_2) \in R_I$, and for all $x \in \Gamma/_{R_O}$, we have $\overline{\delta}(s_1, a_1, x) \cap C_1 = \overline{\delta}(s_2, a_2, x) \cap C_1 = \emptyset$. As $C_1 \supseteq C_2$, we also have $\overline{\delta}(s_1, a_1, x) \cap C_2 = \overline{\delta}(s_2, a_2, x) \cap C_2 = \emptyset$. Finally, let $(a_1, a_2) \in R_I$ and $x \in \Gamma/_{R_O}$, and suppose $s_1' \in \overline{\delta}(s_1, a_1, x) \cap B_i$. s_1' is also contained in some $A_j \subseteq B_i$, as otherwise this would contradict the assumption $\overline{\delta}(s_1, a_1, x) \cap C_1 = \emptyset$. Then, as $s_1\ Q_1'\ s_2$, we also have $\overline{\delta}(s_2, a_2, x) \cap A_j \neq \emptyset$. Hence we conclude $\overline{\delta}(s_2, a_2, x) \cap B_i \neq \emptyset$. The proof that $\overline{\delta}(s_2, a_2, x) \cap B_i \neq \emptyset$ implies $\overline{\delta}(s_1, a_1, x) \cap B_i \neq \emptyset$ follows along the same lines and concludes the proof of the monotonicity of Φ.

As S is finite, the lattice of partial partitions of S is also finite and hence complete. The Knaster-Tarski fixed-point theorem guarantees the existence of a unique maximal fixed point π^*. We have $\Phi(\pi) \leq \pi$ for every partial partition π of S, and hence iteratively applying Φ to $\pi_\top = \langle\{S\}, \emptyset\rangle$ leads to the fixed point $\pi^* = \Phi^n(\pi_\top)$ after a finite number of steps n. □

Theorem 4. *Let $M_i = (S_i, \Sigma_i, \Gamma_i, \delta_i, s_{0,i})$, with $i \in \{1, 2\}$, be automata with output and let $R_I \subseteq \Sigma_1 \times \Sigma_1$, $R_O \subseteq \Gamma_1 \times \Gamma_1$, $Q_I \subseteq \Sigma_2 \times \Sigma_2$, and $Q_O \subseteq \Gamma_2 \times \Gamma_2$*

be equivalence relations. Let $s_1, s_2 \in S_1$ and $t_1, t_2 \in S_2$. If $\Gamma_1 \subseteq \Sigma_2$, $R_O \subseteq Q_I$, $s_1 \simeq^{R_I}_{R_O} s_2$, and $t_1 \simeq^{Q_I}_{Q_O} t_2$, then

$$(s_1, t_1) \simeq^{R_I}_{Q_O} (s_2, t_2) \textit{ in } M_1 \cdot M_2 .$$

Proof. Since $s_1 \simeq^{R_I}_{R_O} s_2$ and $t_1 \simeq^{Q_I}_{Q_O} t_2$, there are relations $\mathcal{R}_1 \subseteq S_1 \times S_1$ and $\mathcal{R}_2 \subseteq S_2 \times S_2$ that satisfy Property (1) of Definition 2, where $(s_1, s_2) \in \mathcal{R}_1$ and $(t_1, t_2) \in \mathcal{R}_2$. It suffices to show that $\mathcal{R}_{1,2} := \{((s,t),(s',t') \mid |(s,s') \in \mathcal{R}_1 \wedge (t,t') \in \mathcal{R}_2\}$ also fulfills Property (1). To this end, let $((s,t),(s',t')) \in \mathcal{R}_{1,2}$ and let $(a,b) \in R_I$. Choose $((s,t),a,c,(p,q)) \in \delta'$, where δ' is the transition function of $M_1 \cdot M_2$. From the definition of δ', there is an $e \in \Gamma_1$ such that $(s,a,e,p) \in \delta_1$ and $(t,e,c,q) \in \delta_2$. As $\mathcal{R}_1$ satisfies (1) of Definition 2, there is a $(s',b,d,p') \in \delta_1$, with $(p,p') \in \mathcal{R}_1$ and $(e,d) \in R_O$. As $R_O \subseteq Q_I$ and $(t,t') \in \mathcal{R}_2$, there is a $(t',d,c',q') \in \delta_2$ with $(c,c') \in Q_O$ and $(q,q') \in \mathcal{R}_2$. From the definition of δ', we have $((s',t'),b,c',(p',q')) \in \delta'$ with $(c,c') \in Q_O$ and $((p,q),(p',q')) \in \mathcal{R}_{1,2}$, as required. □

HBAC: A Model for History-Based Access Control and Its Model Checking

Jing Wang, Yoshiaki Takata, and Hiroyuki Seki

Graduate School of Information Science, Nara Institute of Science and Technology
Takayama 8916–5, Ikoma, Nara 630–0192 Japan

Abstract. Stack inspection is now broadly used as dynamic access control infrastructure in such runtime environments as Java virtual machines and Common Language Runtime. However, stack inspection is not sufficient for security assurance since the stack does not retain security information on the invoked methods for which execution is finished. To solve this problem, access control models based on execution history have been proposed. This paper presents a formal model for programs with access control based on execution history, which are called HBAC programs. Their expressive power is shown to be strictly stronger than programs with stack inspection. It is also shown that the verification problem for HBAC programs is EXPTIME-complete, while the problem is solvable in polynomial time under a reasonable assumption. Finally, this paper presents a few optimization techniques used in the implementation of a verification tool for HBAC programs. The results of experiments show that the tool can verify practical HBAC programs within a reasonable time.

1 Introduction

Stack inspection is now broadly used as dynamic access control infrastructure in such runtime environments as Java virtual machines [15] and the Common Language Runtime. However, stack inspection is not sufficient for security assurance since the stack does not retain security information on the invoked methods for which execution is finished. To solve this problem, a few access control models have been proposed [1, 14, 22]. A common feature of these works is that the history of execution such as method invocation and resource access is used for access control, and the history is not always forgotten even if the surrounding method execution is completely finished. Schneider [22] defines an enforceable security policy as a prefix-closed nonempty set of event sequences and also defines security automata, which exactly recognize enforceable policies. Later Fong [14] introduces several subclasses of security automata and compares the expressive power of these subclasses. Fong defines shallow history automata with finite state space and shows that the class of policies recognized by shallow history automata is incomparable with stack inspection. Another novel approach is proposed by Abadi and Fournet [1]. As in stack inspection, a target system for access control is an object-oriented recursive program. A set of permissions is assigned statically (before runtime) to each method; current permissions are modified each time a method is invoked. Generally, current permissions depend on all the methods executed so far. This forms a contrast to access control based on stack inspection, which completely cancels the effect of the finished method execution. In [1], an implementation built on the top of C$^\sharp$ runtime environment is reported. However, formal verification methods for the model of [1] have not been investigated, except [2].

In this paper, we propose a formal model for Abadi-Fournet style access control called the History-Based Access Control program (HBAC program). An HBAC program is a directed graph where a node represents a program point and an edge represents a control flow. We also show that the expressive power of HBAC programs is stronger than programs with stack inspection. We also define the security verification problem for HBAC programs and show that it is solvable in deterministic polynomial time under a reasonable assumption while the problem is EXPTIME-complete in general. Finally, we propose a few optimization techniques used in verification of HBAC programs. Experimental results show that practical HBAC programs can be verified within reasonable time and space.

Related works. There have been some studies on the verification of history-based access control [2, 3, 4, 11, 16]. The program model proposed in [3, 4] is a call-by-value λ-calculus augmented with local policy defined as a regular language of events. For each function call, a new (but statically bound to the function) local policy is imposed in a nested way. They proposed a model checking algorithm for a given program and a global security property by reducing the problem to the traditional model checking problem for basic process algebra by removing duplicated local policies caused by recursive calls. The access control mechanism of [3, 4] is an extension of [22, 14] that differs from [1] and ours, i.e., their model do not have an explicit dynamic check on permissions. Another access control mechanism based on [22, 14], which simulates security automaton by inserting dynamic access control codes into a target program, was proposed in [11]. In their later work [16], a type system is used to guarantee that the rewritten program adheres to security policies. Their model also differs from [1] and ours, and they do not deal with model checking problems. The previous work most related to ours is [2] where a program model with explicit dynamic checks on permissions and grant/accept constructs is defined. They proposed a type system in the Volpano-Smith style [23] and show that a typesafe program has noninterference property. This property is important because one of the main purposes of access control is to avoid undesirable information flow. However, they ignore model checking problems, as in [11, 16]. Moreover, unlike our study, none of these works discuss the computational complexity needed for verification or optimization issues that are important for implementing a useful verification tool.

2 HBAC Program

We will define the syntax and operational semantics of an HBAC program, which resembles but is more general than the model in [18, 20]. An HBAC program is simply a control flow graph with three types of nodes: call, return and check. The graph is decomposed into methods, and each method is given a subset of permissions for access control called the *static permissions* of the method. A (local) state of a program is a pair $\langle n, C\rangle$ of the current program point n and a subset of permissions C called the *current permissions.* A (global) configuration is represented by a stack, which is a finite sequence of local states. A call node has two parameters, grant permissions and accept permissions. When a method is called from a configuration $\langle n_1, C_1\rangle : \ldots : \langle n_k, C_k\rangle$ with the current (call) node n_1 with grant permissions P_G and accept permissions P_A, a new local state $\langle m, C'\rangle$ is pushed onto the stack where m is the entry point of the callee method and C' is the updated current permission obtained by intersecting C_1 with the static permission of the callee method. Furthermore, P_G is temporarily added to the current permissions during the execution of the callee method and P_A is added to the current permissions when returned from the callee method. A check node tests whether the current permissions include a specified subset of permissions, and if not, the execution is aborted. For simplicity, we do not include an exception handling mechanism in our HBAC model, although it is not difficult to incorporate a throw-catch-style exception handling into the model and extend the model checking algorithm presented in Section 4, as was done in our previous work [19].

Formally, an HBAC program is a directed graph given by a 7-tuple $\pi = (NO, TG, CG, IS, IT, PRM, SP)$ where NO is a finite set of nodes, $TG \subseteq NO \times NO$ is a set of *transfer edges*, $CG \subseteq NO \times NO$ is a set of *call edges*, $IS : NO \to \{call[P_G, P_A] \mid P_G, P_A \subseteq PRM\} \cup \{check[P] \mid P \subseteq PRM\} \cup \{return\}$ is the labeling function for nodes, $IT \in NO$ is the *initial node*, which represents the entry point of the entire program, PRM is a finite set of *permissions*, and $SP : NO \to 2^{PRM}$ is the assignment of permissions to nodes. Each node $n \in NO$ corresponds to a program point, and NO is divided into three subsets by IS as follows:

- $IS(n) = call[P_G, P_A]$ where $P_G, P_A \subseteq PRM$. Node n is a *call node* that represents a method call. Parameters P_G and P_A are called *grant permissions* and *accept permissions*, respectively.
- $IS(n) = return$. Node n is a *return node* that represents the return from a callee method.
- $IS(n) = check[P]$ where $P \subseteq PRM$. Node n is a *check node* that represents a test for the current permissions. (The formal definition of current permissions is given later.) If current permissions include P as a subset, then the execution continues. Otherwise, it is aborted. For $p \in PRM$, $check[\{p\}]$ is abbreviated as $check[p]$.

A transfer edge (tg) represents a control flow within a method, and a call edge (cg) connects a method caller and a callee. In the figure, a solid arrow denotes a cg and a dotted arrow denotes a tg. A node that has an incoming edge without a source node denotes the initial node.

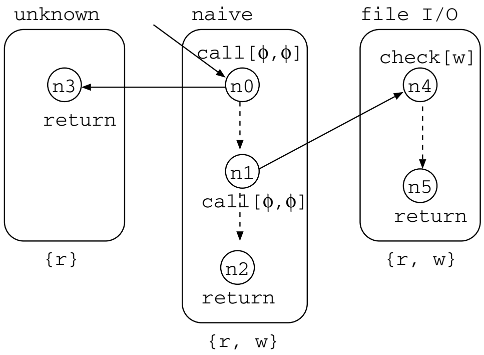


Fig. 1. An HBAC program

Example 1. Figure 1 is an example of an HBAC program π_1 with initial node n_0. There exists a tg from n_0 to n_1 (denoted as $n_0 \overset{TG}{\to} n_1$), which means control can move to n_1 just after the execution of n_0. Likewise, there exists a cg from n_1 to n_4 (denoted as $n_1 \overset{CG}{\to} n_4$). This means that if control reaches n_1, then control is further passed to n_4 by a method call. If control reaches n_5, it returns to n_1. □

For node n, $SP(n)$ specifies a subset of permissions that are assigned to n before runtime (*static permissions*). We assume that every node in the same method has the same static permissions, i.e.,

$$n \overset{TG}{\to} n' \Rightarrow SP(n) = SP(n').$$

Also, for every call node n such that $IS(n) = call[P_G, P_A]$, we require $P_G \subseteq SP(n)$ and $P_A \subseteq SP(n)$. In Fig 1, a method is represented by the set of nodes surrounded by a rectangle. A set beside the rectangle denotes the static permissions assigned to the nodes belonging to the method. For example, $SP(n_0) = SP(n_1) = SP(n_2) = \{r, w\}$ and $SP(n_3) = \{r\}$.

The description length of $\pi = (NO, TG, CG, IS, IT, PRM, SP)$ is defined as $\|\pi\| = |NO| \cdot |PRM| + |TG| + |CG|$. A state of π is a pair $\langle n, C\rangle$ of a node $n \in NO$ and a subset of permissions $C \subseteq PRM$. A *configuration* of π is a finite sequence of states, which is also called a *stack*. The concatenation of state sequences ξ_1 and ξ_2 is denoted as $\xi_1 : \xi_2$. The semantics of an HBAC program is defined by the transition relation $\to$ over the set of configurations, which is the least relation satisfying the following rules.

$$\frac{IS(n) = call[P_G, P_A],\ \ n \overset{CG}{\to} m}{\langle n, C\rangle : \xi \to \langle m, (C \cup P_G) \cap SP(m)\rangle : \langle n, C\rangle : \xi} \tag{1}$$

$$\frac{IS(m') = return,\ \ IS(n) = call[P_G, P_A],\ \ n \overset{TG}{\to} n'}{\langle m', C'\rangle : \langle n, C\rangle : \xi \to \langle n', C \cap (C' \cup P_A)\rangle : \xi} \tag{2}$$

$$\frac{IS(n) = check[P],\ \ P \subseteq C,\ \ n \overset{TG}{\to} n'}{\langle n, C\rangle : \xi \to \langle n', C\rangle : \xi} \tag{3}$$

Table 1. Modification of current permissions

	method call	return
(general case)	$(C \cup P_G) \cap SP(m)$	$C \cap (C' \cup P_A)$
$P_G = SP(n)$	$SP(n) \cap SP(m)$	$C \cap (C' \cup P_A)$
$P_G = \emptyset$	$C \cap SP(m)$	$C \cap (C' \cup P_A)$
$P_A = SP(n)$	$(C \cup P_G) \cap SP(m)$	C
$P_A = \emptyset$	$(C \cup P_G) \cap SP(m)$	$C \cap C'$
$P_A = \emptyset, P_G = \emptyset$	$C \cap SP(m)$	$C \cap C' (= C')$

For a configuration $\langle n_1, C_1\rangle : \ldots : \langle n_k, C_k\rangle$, the stack top is $\langle n_1, C_1\rangle$ where n_1 and C_1 are called the *current program point* and the *current permissions* of the configuration, respectively.

Rule (1) says that if control is at a call node n where $IS(n) = call[P_G, P_A]$ and there exists a cg $n \overset{CG}{\to} m$, then $\langle m, (C \cup P_G) \cap SP(m)\rangle$ can be pushed onto the stack. That is, when control reaches call node n, a method invocation can occur by passing control to m and the current permissions become $(C \cup P_G) \cap SP(m)$. Rule (2) concerns the return from the method. Assume a tg $n \overset{TG}{\to} n'$. If the current node is return node m' in the callee method, then the next current node can be n'. The current permissions become $C \cap (C' \cup P_A)$. Note that if there is no cg from call node n, then control cannot proceed beyond n since rule (1) cannot be applied to n. Similarly, if there is no tg from call node n, then control stops when it reaches a return node of the callee method. Although a program with such a node is pathological, for simplicity we make no syntactical restrictions on cgs and tgs.

We can easily show $C' \subseteq C \cup P_G$ whenever rule (2) can be applied to a configuration reachable from the initial configuration by induction on the rule application. The definition of current permissions for some special cases shown in Table 1 helps us understand why they are defined as in rules (1) and (2).

Finally, rule (3) says that if control reaches a check node n with $IS(n) = check[P]$ and a tg $n \overset{TG}{\to} n'$ and the current permissions include P, then the control can be passed to n'.

The trace set of π is defined as $[\![\pi]\!] = \{n_0 n_1 \ldots n_k \mid n_0 = IT,\ C_0 = SP(IT),\ \xi_0 = \varepsilon,\ \exists C_1, \ldots, C_k \subseteq PRM,\ \exists \xi_1, \ldots, \xi_k \in (NO \times 2^{PRM})^*,\ \langle n_i, C_i\rangle : \xi_i \to \langle n_{i+1}, C_{i+1}\rangle : \xi_{i+1} \text{ for } 0 \le i < k\}$, where ε denotes the empty sequence.

For a set S of sequences, let prefix(S) denote the set of all nonempty prefixes of sequences in S.

Example 2. We return to the HBAC program π_1 in Fig 1. When the method 'unknown' is called by n_0, the current permissions become $\{r, w\} \cap SP(n_3) = \{r, w\} \cap \{r\} = \{r\}$, since $IS(n_0) = call[\emptyset, \emptyset]$ (see Table 1). The test at node n_4 fails since $IS(n_4) = check[w]$ and the current permission $\{r\}$ does not include $\{w\}$. Summarizing,

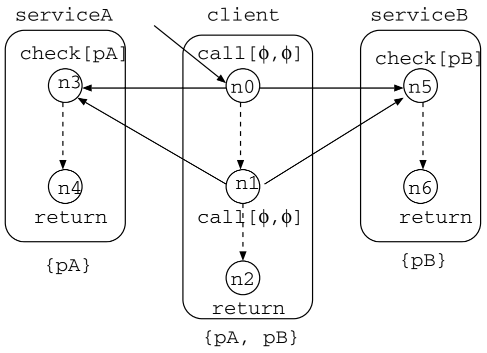


Fig. 2. Chinese wall policy

$$\begin{aligned}&\langle n_0, \{r, w\}\rangle \to \langle n_3, \{r\}\rangle : \langle n_0, \{r, w\}\rangle \to \langle n_1, \{r\}\rangle \\ &\to \langle n_4, \{r\}\rangle : \langle n_1, \{r\}\rangle \not\to \langle n_5, \{r\}\rangle : \langle n_1, \{r\}\rangle. \\ &[\![\pi_1]\!] = \{n_0, n_0n_3, n_0n_3n_1, n_0n_3n_1n_4\} = \text{prefix}(\{n_0n_3n_1n_4\}).\end{aligned}$$

Note that since no node exists with multiple outgoing tg or multiple outgoing cg (i.e., there is no nondeterminism) and no cycle exists in π_1, the trace set can be represented as the prefixes of a single sequence $n_0n_3n_1n_4$.

Consider the situation where method 'naive' calls 'unknown' and the latter method secretly changes the contents of a local variable of 'naive', say *fname*, to the name of a very critical file. Then 'naive' requests 'file I/O' to delete *fname* without knowing that the contents of *fname* have been changed. If 'file I/O' performs $check[w]$ before deleting the file, unintended file deletion can be avoided since the current permission does not include write permission $\{w\}$ as the effect of executing 'unknown.' As explained below, however, such access control cannot be realized by stack inspection.

Let π_2 be the HBAC program that is the same as π_1, except that $IS(n_0) = call[\emptyset, \{r, w\}]$. Since the accept permissions of n_0 are $\{r, w\}$,

$$\begin{aligned}&\langle n_0, \{r, w\}\rangle \to \langle n_3, \{r\}\rangle : \langle n_0, \{r, w\}\rangle \to \langle n_1, \{r, w\}\rangle \\ &\qquad \to \langle n_4, \{r, w\}\rangle : \langle n_1, \{r, w\}\rangle \to \langle n_5, \{r, w\}\rangle : \langle n_1, \{r, w\}\rangle.\end{aligned}$$

Similarly, let π_3 be the HBAC program that is the same as π_1, except that $IS(n_1) = call[\{r, w\}, \emptyset]$. Since the grant permissions of n_1 are $\{r, w\}$,

$$\begin{aligned}&\langle n_0, \{r, w\}\rangle \to \langle n_3, \{r\}\rangle : \langle n_0, \{r, w\}\rangle \to \langle n_1, \{r\}\rangle \\ &\qquad \to \langle n_4, \{r, w\}\rangle : \langle n_1, \{r\}\rangle \to \langle n_5, \{r, w\}\rangle : \langle n_1, \{r\}\rangle.\end{aligned}$$ □

Example 3. Chinese wall policy [5] is a policy such that a user has access permission to any resources, but once the user has accessed one of the resources, (s)he loses access permission to the resources belonging to competing parties. A simplified Chinese wall policy can be represented by program π_4 in Fig 2. If n_0 calls 'serviceA,' the current permissions lose permission p_B. Thus, if n_1

calls 'serviceB' afterward, the check at n_5 fails. The same situation occurs when 'serviceB' and 'serviceA' are called in this order. In fact,

$$[\![\pi_4]\!] = \text{prefix}(n_0 n_3 n_4 n_1 (n_3 n_4 n_2 + n_5) + n_0 n_5 n_6 n_1 (n_5 n_6 n_2 + n_3)),$$

where the argument of 'prefix' is specified by a regular expression and + denotes the union operator. □

3 Comparison with Stack Inspection

A program with Java stack inspection (abbreviated as SI program) can be represented by an 8-tuple $\pi = (NO, TG, CG, IS, IT, PRM, SP, PRV)$, where each component of π is identical to that of an HBAC program, except that label $IS(n)$ of each call node n is simply *call* without P_G and P_A, and a set of privileged nodes $PRV \subseteq NO$ is specified. The execution of check node $check[P]$ succeeds if (a) for every node n on the stack, $P \subseteq SP(n)$, or (b) there exists a node $n_0 \in PRV$ on the stack such that $P \subseteq SP(n_0)$ and for every later node n in the stack, $P \subseteq SP(n)$. By adopting an eager evaluation strategy, we can define the semantics of π by the following rules and rule (3) defined above (see [18, 20] for details).

$$\frac{IS(n) = call,\ n \overset{CG}{\rightarrow} m,\ n \notin PRV}{\langle n, C\rangle : \xi \rightarrow \langle m, C \cap SP(m)\rangle : \langle n, C\rangle : \xi} \tag{4}$$

$$\frac{IS(n) = call,\ n \overset{CG}{\rightarrow} m,\ n \in PRV}{\langle n, C\rangle : \xi \rightarrow \langle m, SP(n) \cap SP(m)\rangle : \langle n, C\rangle : \xi} \tag{5}$$

$$\frac{IS(m') = return,\ IS(n) = call,\ n \overset{TG}{\rightarrow} n'}{\langle m', C'\rangle : \langle n, C\rangle : \xi \rightarrow \langle n', C\rangle : \xi} \tag{6}$$

A program without check node is called a *basic program.* An HBAC (resp. SI) program π is an HBAC (resp. SI) *extension* of a basic program π_0 if π is obtained from π_0 by the following operations:

- Insert zero or more check nodes of HBAC (resp. SI) program into π_0;
- Add grant permissions and/or accept permissions to call nodes (in the case of HBAC extension); and
- Choose some nodes as privileged nodes (in the case of SI extension).

The formal definition of the extension is omitted due to space limitation. Let nc be a homomorphism over the set of nodes defined by $nc(n) = n$ for a call node and a return node n and $nc(n) = \varepsilon$ for a check node n. Let π_1 and π_2 be extensions of a basic program π_0. We say that π_1 and π_2 are *trace equivalent* if $nc([\![\pi_1]\!]) = nc([\![\pi_2]\!])$.

Comparing rules (4), (5), (6) with rules (1), (2), we can see that a non-privileged call node and a privileged node in an SI program can be simulated by $call[\emptyset, SP(n)]$ and $call[SP(n), SP(n)]$, respectively. This correspondence was informally described in [1]. However, the converse does not hold, as shown in the next example.

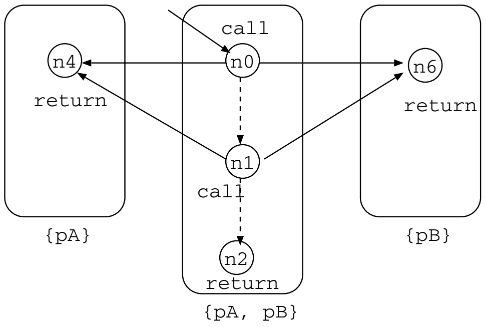


Fig. 3. A basic program

Example 4. Program π_4 in example 3 is an HBAC extension of basic program π_0 in Fig 3. Note that

$$nc([\![\pi_4]\!]) = \text{prefix}(n_0 n_4 n_1 n_4 n_2 + n_0 n_6 n_1 n_6 n_2).$$

There exists no SI extension π_{SI} of π_0 such that $nc([\![\pi_{\text{SI}}]\!]) = nc([\![\pi_4]\!])$. Informally, this is because the effect of executing 'serviceA' or 'serviceB' is canceled when control reaches n_1 in any SI extension of π_0. $\square$

Theorem 1. *For every basic program π_0 and every SI extension π of π_0, there exists an HBAC extension π' of π_0 that is trace equivalent to π. There exists a basic program π_0 and an HBAC extension π of π_0 such that there exists no SI extension π' of π_0 that is trace equivalent to π.* $\square$

4 Model Checking HBAC Program

In this section, we discuss the verification problem (or model checking problem) defined as follows:

Inputs: An (HBAC) program $\pi = (NO, \ldots)$ and a verification property $\psi \subseteq NO^*$.

Output: Does every trace in $[\![\pi]\!]$ satisfy ψ? (i.e., $[\![\pi]\!] \subseteq \psi$?)

Example 5. Consider the verification problem for program π_4 of example 3 and the verification property $\psi = (\Sigma - \{n_4\})^* + (\Sigma - \{n_6\})^*$, where $\Sigma = (n_0 + n_1 + \cdots + n_6)$. As explained in example 3, nodes n_4 and n_6 cannot be reached simultaneously in a single trace, and thus $[\![\pi_4]\!] \subseteq \psi$ holds. $\square$

Let M be any representation of a language such as an automaton and a grammar. The description length of M is denoted by $\|M\|$, and the language expressed by M is denoted by $L(M)$.

Lemma 1. *For an arbitrary HBAC program π, we can construct a context-free grammar (cfg) G such that $L(G) = [\![\pi]\!]$ and $\|G\| = O(\|\pi\| \cdot c^{|PRM|})$ $(c > 1)$.*

(Proof sketch) We define the set of nonterminal symbols of G as $(NO \times 2^{PRM}) \cup (NO \times 2^{PRM} \times 2^{PRM})$. A nonterminal symbol $\langle n, C\rangle \in NO \times 2^{PRM}$ derives every trace starting from node n with current permissions C. A nonterminal symbol $[n, C, C'] \in NO \times 2^{PRM} \times 2^{PRM}$ derives every trace starting from node n with current permissions C and ending with a return node with current permissions C'. In the following, let C, C', and C'' be arbitrary subsets of PRM. For each node n, G has the rule $\langle n, C\rangle \to n$. For each pair (n, m) of nodes such that $IS(n) = call[P_G, P_A]$ and $n \overset{CG}{\to} m$, G has the rule $\langle n, C\rangle \to n\,\langle m, P_1\rangle$, where $P_1 = (C \cup P_G) \cap SP(m)$. Moreover, for each node n' such that $n \overset{TG}{\to} n'$, G has the following rules:

$$\begin{aligned} \langle n, C\rangle &\to n[m, P_1, C']\langle n', P_2\rangle && (7)\\ [n, C, C''] &\to n[m, P_1, C'][n', P_2, C''] && (8)\\ P_2 &= C \cap (C' \cup P_A) \end{aligned}$$

For each pair (n, n') of nodes such that $IS(n) = check[P]$ and $n \overset{TG}{\to} n'$, if $P \subseteq C$, then G has the following rules:

$$\begin{aligned} \langle n, C\rangle &\to n\langle n', C\rangle\\ [n, C, C'] &\to n[n', C, C'] \end{aligned}$$

For each return node n, G has the rule $[n, C, C] \to n$. The start symbol of G is $\langle IT, SP(IT)\rangle$. □

Theorem 2. *Let π be an HBAC program and M be a finite automaton (fa). The verification problem for π and $\psi = \overline{L(M)}$ is solvable in deterministic $O(\|\pi\| \cdot c^{|PRM|} \cdot \|M\|^3)$ time ($c > 1$).*

(Proof sketch) By lemma 1, we can construct a cfg G such that $L(G) = [\![\pi]\!]$. Thus, the verification problem is equivalent to deciding whether $L(G) \cap L(M) = \emptyset$. The latter condition can be checked in $O(\|G\| \cdot \|M\|^3)$ time. □

Corollary 1. *The verification problem for π and $\psi = \overline{L(M)}$ is solvable in deterministic $O(\|\pi\|^2 \cdot \|M\|^3)$ time if $|PRM| = O(\log \|\pi\|)$.* □

The assumption that $|PRM| = O(\log \|\pi\|)$ is realistic since the number of permissions is usually not so large compared with the program size.

Let EXPTIME denote the class of decision problems solvable in deterministic $O(c^{p(n)})$ time for a constant c (> 1) and a polynomial p. The following theorem states that if the assumption that $|PRM| = O(\log \|\pi\|)$ does not hold, then the verification problem is EXPTIME-complete.

Theorem 3. *Let π be an HBAC program and M be an fa. The verification problem for π and $\psi = \overline{L(M)}$ is EXPTIME-complete.*

(Proof sketch) EXPTIME-hardness can be shown by a reduction from the membership problem for polynomial space-bounded alternating Turing machines [7].

5 Optimization of Model Checking Algorithm

From the proof of theorem 2, we obtain the following algorithm for solving the verification problem.

Algorithm 1. For a given HBAC program π and an fa M such that $\psi = \overline{L(M)}$, perform the following three steps in this order.

1. Construct a cfg G such that $L(G) = [\![\pi]\!]$ based on the proof of lemma 1.
2. Construct a cfg $\widehat{G}$ such that $L(\widehat{G}) = L(G) \cap L(M)$.
3. Decide whether $L(\widehat{G}) = \emptyset$.

The size of the G constructed in step 1 is exponential to $|PRM|$. In most cases, however, G contains useless rules. In this section, we describe techniques for avoiding the construction of useless rules so that we can greatly reduce verification time and space.

5.1 Basic Idea

The following is traditional algorithm for eliminating useless rules in a cfg [17]. Nonterminal symbol X is *generating* if there exists a derivation from X to some string of terminal symbols. X is *reachable* if a derivation exists from the start symbol of G to $\alpha X \beta$ for some α and β. A rule r is *useless* if r contains a symbol that is not generating or not reachable. The traditional algorithm finds set V of all the symbols that are generating and reachable and then removes all rules involving one or more symbols not in V. While this algorithm *eliminates* useless rules of a given cfg, we want to *avoid* constructing such rules in the cfg construction. From the definition of G in the proof of lemma 1, we can show the following lemma:

Lemma 2. *Let π be an HBAC program and G be the cfg constructed for π in step 1 of algorithm 1. For each $n \in NO$ and $C, C' \subseteq PRM$, $\langle n, C\rangle$ and $[n, C, C']$ are not reachable if $C \not\subseteq SP(n)$, and $[n, C, C']$ is not generating if $C' \not\subseteq C$.* □

By this lemma, we can avoid constructing rules involving $\langle n, C\rangle$ or $[n, C, C']$ such that $C \not\subseteq SP(n)$ or $C' \not\subseteq C$. However, the number of remaining rules is still exponential to $|PRM|$ in most cases, and thus we need further optimization.

5.2 Rules with Reachable Symbols

The following breadth-first search algorithm exactly constructs the rules involving only reachable symbols.

Algorithm 2

1. Let Q be an FIFO-queue (or simply, queue). Initialize Q to the queue only containing the start symbol $\langle IT, SP(IT)\rangle$.
2. Extract one symbol ($\langle n, C\rangle$ or $[n, C, C']$) from Q. Construct every rule whose left-hand side is the extracted symbol, if that rule has not been constructed. For example, if the extracted symbol is $[n, C, C']$ and $IS(n) = call[P_G, P_A]$, then construct $[n, C, C'] \to n[m, P_1, C''][n', P_2, C']$ where $P_1 = (C \cup P_G) \cap SP(m)$ and $P_2 = C \cap (C'' \cup P_A)$ for each m, n' and C'' such that $n \overset{CG}{\to} m$, $n \overset{TG}{\to} n'$ and $C'' \subseteq P_2$. Insert the nonterminal symbols contained in the right-hand side of the constructed rule into Q.
3. Repeat step 2 until Q becomes empty.

Since only a finite number of nonterminal symbols of the form $\langle n, C\rangle$ or $[n, C, C']$ exists, the algorithm always halts. Obviously, algorithm 2 constructs rule r of G if and only if r only contains reachable symbols. If the following conditions hold for some constant c, then the number of rules constructed by algorithm 2 is polynomial to $\|\pi\|$.

1. $|SP(n) \cap SP(m)| < c$ for each n in the main method (i.e., the method to which IT belongs) and each m such that $n \overset{CG}{\to} m$.
2. $|P_G(n)| < c$ for each call node n where $P_G(n)$ is the set of grant permissions of n.

The HBAC program of the Chinese wall policy in example 3 satisfies these conditions.

5.3 Precomputing Current Permissions

Some of the rules constructed in algorithm 2 may contain reachable but non-generating symbols. To avoid constructing such rules, we modify algorithm 2 as follows. Assume that a symbol $\langle n, C\rangle$ for a call node n is extracted from the queue in step 2 of algorithm 2. Instead of constructing a rule $\langle n, C\rangle \to n[m, P_1, C']\langle n', P_2\rangle$ for all $C' \subseteq P_1$ (rule (8) in the proof of lemma 1), the modified algorithm invokes the following procedure *Return-Permission* with actual parameters m and P_1. Then *Return-Permission* computes a set S of subsets of permissions such that $S \supseteq \{C' \mid [m, P_1, C'] \text{ is generating}\}$ through a depth-first search as well as constructing rules consisting of symbols that are generating and reachable from $[m, P_1, C']$ for some $C' \in S$. In the body of *Return-Permission*, $color[n, C]$ is a variable that indicates whether the pair (n, C) has been visited [9]. It is assumed that $color[n, C] = write$ for every pair (n, C) at the first time *Return-Permission* is invoked. If $color[n, C] = gray$, then (n, C) has been visited but computation for (n, C) is not completed. If $color[n, C] = black$, then computation for (n, C) has been completed. If $color[n, C] = gray$ holds in line 1 of *Return-Permission*(n, C), i.e., a cycle of method calls in the HBAC program is detected, then a conservative answer, the set of all the subsets of C, is returned (line 2).

```
$Return\text{-}Permission(n, C)$
1  if $color[n, C] = gray$          \a loop is found
2    then return $2^C$
3  if $color[n, C] = black$
4    then return $result[n, C]$     \return the results of previous calculation
5  $color(n, C) \leftarrow gray$
6  $result[n, C] \leftarrow \emptyset$
7  if $IS(n) = return$
8    then $construct\ a\ rule\ [n, C, C] \rightarrow n$
9         $result[n, C] \leftarrow result[n, C] \cup \{C\}$
10  if $IS(n) = check[P]\ and\ P \subseteq C$
11    then for $each\ n'\ such\ that\ n \overset{TG}{\rightarrow} n'$
12         do $R \leftarrow Return\text{-}Permission(n', C)$
13            for $each\ C' \in R$
14            do $construct\ a\ rule\ [n, C, C'] \rightarrow n[n', C, C']$
15               $result[n, C] \leftarrow result[n, C] \cup \{C'\}$
16  if $IS(n) = call[P_G, P_A]$
17    then for $each\ m\ such\ that\ n \overset{CG}{\rightarrow} m$
18         do $P_1 \leftarrow (C \cup P_G) \cap SP(m)$
19            $R_1 \leftarrow Return\text{-}Permission(m, P_1)$
20            for $each\ C' \in R_1$
21            do $P_2 \leftarrow C \cap (C' \cup P_A)$
22               for $each\ n'\ such\ that\ n \overset{TG}{\rightarrow} n'$
23               do $R_2 \leftarrow Return\text{-}Permission(n', P_2)$
24                  for $each\ C'' \in R_2$
25                  do $construct\ a\ rule\ [n, C, C''] \rightarrow [m, P_1, C'][n', P_2, C'']$
26                     $result[n, C] \leftarrow result[n, C] \cup \{C''\}$
27  $color(n, C) \leftarrow black$
28  return $result[n, C]$
```

5.4 Localizing the Precomputation

If a given HBAC program π is acyclic (as a directed graph with set of edges $CG \cup TG$), then the algorithm in 5.3 constructs a rule r of G if and only if r contains only generating and reachable symbols. On the other hand, assume that π contains a cycle. The algorithm may construct a rule that contains a reachable but nongenerating symbol for the following reason. If *Return-Permission*(m, P_1) is called recursively from line 19 of *Return-Permission* with $color[m, P_1] = gray$, then 2^{P_1} is substituted for R_1 and symbols $[m, P_1, C']$ with $C' \subseteq P_1$ are constructed in line 25, even though some of these symbols are nongenerating.

However, if $|P_1 - P_A|$ is small enough, then the number of different values $C \cap (C' \cup P_A)$ substituted for P_2 in line 21 is also small since $C \cap (C' \cup P_A) = (C \cap P_A) \cup (C \cap (C' - P_A))$, where $C' \subseteq P_1$ and thus the number of different values substituted for $C' - P_A$ is small. Especially if $P_A = SP(n)$, i.e., n simulates stack inspection, then $P_1 - P_A = \emptyset$, and only a single possible value of P_2 exists, which is C. Furthermore, C' does not directly affect $result[n, C]$ in line 26. Hence, by postponing the construction of a rule in line 25, we can modify line 20 as a **for** statement on every subset of $P_1 - P_A$ instead of every subset of P_1.

6 Experiments

To examine the optimization efficiency described in the previous section on practical HBAC programs, we implemented a verification tool and measured verification time in the following two examples.

- Chinese wall policy
  We extend program π_4 in example 3 to program $\pi_c(k)$ with $k+1$ methods $\{client, service_1, \ldots, service_k\}$ by replacing services A and B with k copies of serviceA. The set of static permissions of *client* is $\{p_1, p_2, \ldots, p_k\}$, and the one for $service_i$ $(1 \leq i \leq k)$ is $\{p_i\}$. We specify a verification property ψ for $\pi_c(k)$ as

  $$(N_1 \cup N_c)^* + (N_2 \cup N_c)^* + \cdots + (N_k \cup N_c)^*,$$

  where N_c is the set of the nodes of method *client* and N_i is the one for $service_i$. An HBAC program π satisfies ψ if and only if there is no trace of π containing nodes of two or more distinct service methods.
- Online banking system
  As mentioned in section 3, we can convert every SI program into an equivalent HBAC program. We define $\pi_o(k)$ as an HBAC program obtained from a sample SI program in [20], which models part of an integrated online banking system with k banks (Fig 4). Each bank serves its clients with a method for withdrawing money. Method *spender* is an agent of a reliable user that has static permissions $\{d_1, \ldots, d_k\}$, and *clyde* is an agent of an unreliable user without permission. They can access method $debit_i$ $(1 \leq i \leq k)$, which is a service provider of the i-th bank. Each $debit_i$ checks whether a user has permission d_i and performs privileged calls on $read_i$ and $write_i$. A verification property ψ is given the same as in [20]. That is, the negation of ψ is $\overline{\psi} = \Sigma^* N_{clyde} \Sigma^* N_{rw} \Sigma^*$, where Σ is the set of all nodes, N_{clyde} is the set of nodes in method *clyde*, and N_{rw} is the union of the sets of nodes of every $read_i$ and every $write_i$. An HBAC program π satisfies ψ if and only if control never reaches $read_i$ or $write_i$ after it once reaches a node in *clyde*.

Table 2 summarizes the results of the experiments. G is the cfg generated in step 1 of algorithm 1. M is a regular grammar such that $\psi = \overline{L(M)}$. Fig 5 shows computation time needed to verify $\pi_c(k)$ and $\pi_o(k)$. Without optimization, we could not verify $\pi_o(k)$ for any $k \geq 1$ and $\pi_c(k)$ for $k \geq 10$, because the number of the rules of G was exponential to k and the amount of memory was insufficient to store G. With full optimization, both the number of the rules of G and computation time were reduced to polynomial to k for $\pi_c(k)$ and $\pi_o(k)$. Especially, the computation time for $\pi_o(k)$ was linear to k. As in [20], we estimate

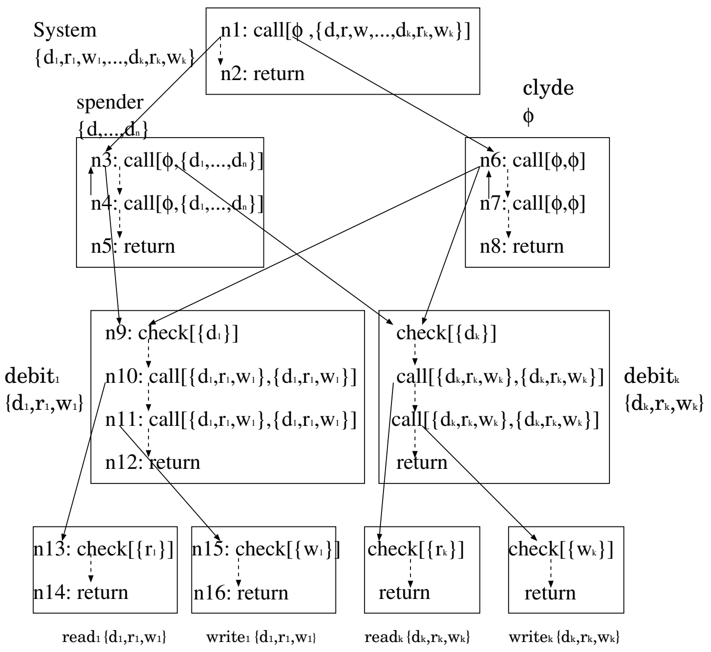


Fig. 4. Online banking system

Table 2. Verification profiles of sample programs

		$\pi_c(k)$						$\pi_o(k)$			
k		5	10	20	40	60	80	5	10	15	20
the number of permissions		5	10	20	40	60	80	15	30	45	60
the number of the rules of G	base [†]	1613									
	1[‡]	165	1103	2253	5753	10853		1866			
	1+2[‡]	81	211	621	2041	4261	7281	184	1316	33200	
	1+2+3 [‡]	81	211	621	2041	4261	7281	153	293	433	573
$\|M\|$		124	389	1369	5129	11289	20340	208	388	568	748
computation time(sec)	base	0.815									
	1	0.217	0.369	1.60	22.4	131		0.715			
	1+2	0.173	0.293	1.38	23.0	131	499	0.244	0.693	7.77	
	1+2+3	0.074	0.158	1.37	21.0	131	494	0.210	0.275	0.333	0.356
verification result		true	true	true	true	true	true	true	true	true	true

[†] base is the algorithm 1 modified based on lemma 2 (section 5.1).

[‡] optimizations 1 to 3 described in sections 5.2, 5.3, and 5.4, respectively.

that the number of permissions used in an ordinary network application is at most several tens, and the results suggest that the proposed verification method is feasible for practical programs.

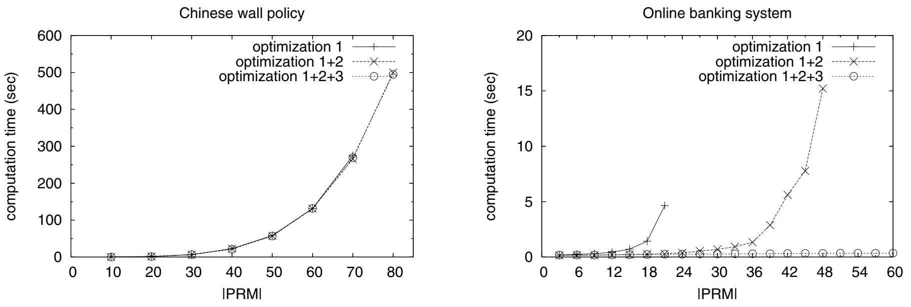


Fig. 5. Verification time for $\pi_c(k)$ and $\pi_o(k)$

7 Conclusion

In this paper, we presented a new model for dynamic access control based on execution history called HBAC programs. The expressive power of HBAC programs was examined, and the verification problem for HBAC programs was shown to be solvable. Although the complexity of the problem is EXPTIME-complete in general, our verification tool verified sample programs within a reasonable time.

Our program model is closely related to a class of infinite state systems called *pushdown systems* (abbreviated as PDS). Indeed, the behavior of an HBAC program can be modeled by a PDS with an exponential number of stack symbols. The decidability and complexity of LTL and CTL* model checking [8] for PDS are extensively studied in [10, 12]. Verification results conducted on a model checker for PDS are reported in [13]. Although the verification problem and the model checking algorithm in this paper are based on finite traces, we can extend the algorithm to infinite traces (and thus LTL) using ω-context-free grammars [6], and the time complexity of the algorithm is slightly better than the one needed when applying the algorithm in [10] to a PDS that models a given HBAC program. Namely, the former is proportional to $|Q_M|^3$, where $|Q_M|$ is the number of the states of a Büchi automaton M representing the negation of a verification property, while the latter is proportional to $|Q_M|^2|\Delta_M|$, where $|\Delta_M|$ is the number of the transitions of M. Note that the optimization described in section 5 can be applied not only when using our algorithm but also when using a model checker for PDS to verify an HBAC program.

Future work includes comparing the expressive power of various subclasses of security automata [14, 22] with that of HBAC programs.

References

1. M. Abadi and C. Fournet, "Access control based on execution history," Network & Distributed System Security Symp., pp.107–121, 2003.
2. A. Banerjee and D. A. Naumann, "History-based access control and secure information flow," CASSIS04, LNCS 3362, pp.27–48, 2004.

3. M. Bartoletti, P. Degano, and G. L. Ferrari, "History-based access control with local policies," 8th FOSSACS, LNCS 3441, pp.316–332, 2005.
4. M. Bartoletti, P. Degano, and G. L. Ferrari, "Enforcing secure service composition," IEEE 18th CSFW, pp.211-223, 2005.
5. D. F. C. Brewer and M. J. Nash, "The Chinese wall security policy," IEEE Security & Privacy, pp.206–214, 1989.
6. R. S. Cohen and A. Y. Gold, "Theory of ω-languages. I: Characterizations of ω-context-free languages," J. of Computer & System Science, 15, pp.169–184, 1977.
7. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer, "Alternation," J. of the ACM, 28, pp.114–133, 1981.
8. E. M. Clarke, Jr., O. Grumberg, and D. Peled, *Model Checking,* MIT Press, 2000.
9. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, MIT Press, 2003.
10. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon, "Efficient algorithms for model-checking pushdown systems," CAV2000, LNCS 1855, pp.232–247, 2000.
11. Ú. Erlingsson and F. B. Schneider, "IRM enforcement of Java stack inspection," IEEE Security & Privacy, pp.246–255, 2000.
12. J. Esparza, A. Kučera, and S. Schwoon, "Model-checking LTL with regular variations for pushdown systems," TACS01, LNCS 2215, pp.316–339, 2001.
13. J. Esparza and S. Schwoon, "A BDD-based model checker for recursive programs," CAV2001, LNCS 2102, pp.324–336, 2001.
14. P. W. Fong, "Access control by tracking shallow execution history," IEEE Security & Privacy, pp.43–55, 2004.
15. L. Gong, M. Mueller, H. Prafullchandra, and R. Schemers, "Going beyond the sandbox: An overview of the new security architecture in the Java^{TM} development kit 1.2," USENIX Symp. on Internet Technologies and Systems, pp.103–112, 1997.
16. K. W. Hamlen, G. Morrisett, and F. B. Schneider, "Certified in-lined reference monitoring on .NET," Cornell University Computing and Information Science Technical Report, TR2005-2003, 2005.
17. J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to automata theory, languages, and computation,* Addison Wesley, 2001.
18. T. Jensen, D. Le Métayer, and T. Thorn, "Verification of control flow based security properties," IEEE Security & Privacy, pp.89–103, 1999.
19. S. Kuninobu, Y. Takata, D. Taguchi, M. Nakae, and H. Seki, "A specification language for distributed policy control," 4th ICICS, LNCS 2513, pp.386-398, 2002.
20. N. Nitta, Y. Takata, and H. Seki, "An efficient security verification method for programs with stack inspection," 8th ACM Computer & Communications Security, pp.68–77, 2001.
21. A. Schaad, J. Moffett, and J. Jacob, "The role-based access control system of a European bank: a case study and discussion," 6th ACM Symp. on Access Control Models and Technologies, pp.3–9, 2001.
22. F. B. Schneider, "Enforceable security policies," ACM Trans. on Information & System Security, 3(1), pp.30–50, 2000.
23. D. Volpano and G. Smith, "A type-based approach to program security," TAPSOFT'97, LNCS 1214, pp.607–621, 1997.

From Coupling Relations to Mated Invariants for Checking Information Flow
(Extended Abstract)

David A. Naumann[⋆]

Stevens Institute of Technology, Hoboken NJ 07030 USA

Abstract. This paper investigates a technique for using automated program verifiers to check conformance with information flow policy, in particular for programs acting on shared, dynamically allocated mutable heap objects. The technique encompasses rich policies with forms of declassification and supports modular, invariant-based verification of object-oriented programs. The technique is based on the known idea of self-composition, whereby noninterference for a command is reduced to an ordinary partial correctness property of the command sequentially composed with a renamed copy of itself. The first contribution is to extend this technique to encompass heap objects, which is difficult because textual renaming is inapplicable. The second contribution is a systematic means to validate transformations on self-composed programs. Certain transformations are needed for effective use of existing automated program verifiers and they exploit conservative flow inference, e.g., from security type inference. Experiments with the technique using ESC/Java2 and Spec# verifiers are reported.

1 Introduction

Consider an imperative command S acting on variables with declaration Γ. For example, Γ could be x:int,y:int,z:bool and S could be $z := (y > 0); x := x + 1; y := x$. A standard notion of confidentiality policy is to label variables with levels from a partially ordered set, e.g., $\{low, high\}$ with $low \leq high$. This is interpreted to mean that information is only allowed to flow from one variable to another, say x to y, if the level of x is at most the level of y. Such a policy is of interest only under the *mandatory access control assumption* that a principal at level λ can directly read only variables with confidentiality label at or below λ. (Our results apply as well to the dual, integrity.)

This paper investigates a technique for using ordinary program verifiers to check conformance with policy, in particular for programs acting on shared, mutable heap objects and policies that specify flows with finer granularity than individual program variables. The intended application is to programs in Java and similar languages but the technique pertains to any program using pointer structure. The technique is known as *self-composition* [25,7,30,15]; it reduces security to an ordinary partial correctness property of the program composed with itself. The first contribution is a novel extension of this technique to encompass the heap. The second contribution is a systematic means

⋆ Supported in part by NSF grants CCR-0208984 and CCF-0429894.

to validate certain transformations on self-composed programs that are needed to make effective use of automated program verifiers to check security; this draws on work by Benton [9], Terauchi and Aiken [30]. The third contribution is to report on promising experiments with the ESC/Java2 [18] and Spec# [6] tools and to pose challenges for improvement of these tools.

To explain the main ideas we begin by considering the scenario above, with imperative commands acting on variables of primitive type. For the specific lattice $\{low, high\}$ we write $x \in vis$ to express that x is low. The formal notion that visible outputs reveal no information about secret inputs (high variables) is called *noninterference* [27] and is expressed in terms of two runs of the program:[1] If two initial states s and s' agree on variables in vis and t, t' are the final states from running the program on s and s' respectively, then t and t' also agree on variables in vis.

In program verification, a dashed identifier like s' is often used to refer to the final state corresponding to s; we do not use dashes that way, but rather to indicate a coupling relation.

Because the noninterference property involves two runs, it cannot simply be monitored at runtime or expressed directly as a pre/post specification. For static analysis, a popular approach is by means of a type system in which types include security labels [31,21,27,24,4,8]. The rules prevent, e.g., assignment to a low variable of an expression containing a high variable. Static analysis is useful to detect bugs and trojans; it does not prevent attacks that violate the abstractions embodied by the language semantics on which the analysis and definition of noninterference are based.

Self-composition. Besides type checking, another approach that has been explored is based on Hoare logic [14,10,11]. The condition "s and s' agree on visible variables" can be expressed by an assertion $x = x' \wedge y = y' \wedge \ldots$ with an equation $x = x'$ for each $x \in vis$, where x' is fresh variable. Such an assertion can be interpreted in a pair of states s, s', where x' is the value of x in s', or better still in a single state that assigns values to both x and x'. Now the property that S is noninterferent can be expressed using a renamed copy S' acting on the dashed variables. Add to the language a combinator $|$ so that $S|S'$ means parallel, independent execution of S and S'. Then S is noninterferent just if $S|S'$ takes initial states satisfying $x = x' \wedge y = y' \wedge \ldots$ to final states satisfying the same. For example, S at the beginning of the paper is noninterferent for $vis = \{x\}$ and for $vis = \{x, y\}$ but not for $vis = \{x, z\}$.

Two features of this formulation are interesting in terms of policy. Most importantly, the pre-post specification can be extended to allow partial releases at finer granularity than variables. For example, the equation $encrypt(k, secret) = encrypt(k', secret')$ in

[1] This paper focuses on termination-insensitive noninterference. Covert channels such as timing and power consumption are also ignored. The rationale is that such flows are harder to exploit as well as to prevent —our aim is a practical means to reduce the risk of trojans and bugs in production software. For further simplification in this paper, programs are deterministic, only initial and final states are observable, and the heap is unbounded. Pointer arithmetic is disallowed, just as it is in Java and its cousins (ignoring `hashcode`), since otherwise it is very hard to constrain information flow. On the other hand, the results make no assumption about the memory allocator, which may depend on the entire state; this entails some complications concerning but is a price worth paying for applicability to real systems.

a precondition would allow release of the encryption but not the secret plaintext. The policy with postcondition $z = z'$ and precondition $(y > 0) = (y' > 0)$ is satisfied by the S at the beginning of the paper. Preconditions can also condition secrecy of a variable on event history or permissions, or allow one but not both of two secrets to be released (e.g., [10,29,1,5]).

The second feature relevant to policy is that the formulation does not directly handle label lattices bigger than $\{low, high\}$. But it is well known that the noninterference property for a general lattice L can be reduced to a conjunction of properties, using just $\{low, high\}$ with *low* representing all the levels below a fixed level in L and *high* representing the rest. Henceforth we consider policy in the form of a set *vis* of low variable and field names.

This paper focuses on checking programs for conformance with given policy. Because only terminating computations are considered, and because S and S' act on disjoint variables, computations of $S|S'$ are the same as computations of the sequence $S;S'$ (and also $S';S$). We have arrived at the *self-composition* technique [7,30]: noninterference of S is reduced to an ordinary partial correctness property of $S;S'$ with respect to specifications over dashed and undashed copies of program variables.

Partial correctness is undecidable whereas the type-based approach is fast. But efficiency is gained at the cost of conservative abstraction; typical type systems are flow insensitive and may be very conservative in other ways, e.g., lack of arithmetic simplification. With a complete proof system, and at the cost of interactive proving, any noninterferent program can be proved so using self-composition. What is really interesting is the potential to use existing automated verification tools to check security of programs that are beyond the reach of a conventional type-based analysis. There are two significant obstacles to achieving this potential; overcoming them is the contribution of the paper.

Obstacles. To see the first obstacle, note first that there is reason to be optimistic about automation: pre-post specification of policy involves only simple equalities, not full functional correctness. But Terauchi and Aiken [30] point out that to verify a simple correctness judgement $\{x = x'\}S;S'\{y = y'\}$ requires —in some way or another depending on the verification method— to find an intermediate condition that holds at the semicolon. Suppose S involves a loop computing a value for y. The intermediate condition needs to describe the final state of y with sufficient accuracy that after S' it can be determined that $y = y'$. In the worst case this is nothing less than computing strongest postconditions. The weakest precondition for S' would do as well but is no easier to compute without a loop invariant being given. (And similarly for method calls.) This obstacle will reappear when we consider the second obstacle.

The second obstacle is due to dynamically allocated mutable objects. Note first that there is little practical motivation, or means, to assign labels to references themselves since upward flows can create useful low-high aliases and reference (pointer) literals are not available in most languages. Rather, field names and variables are labeled, as in Jif [21] and [4,8]. But noninterference needs to be based on a notion of indistinguishability taking into account that some but not all references are low visible. References to objects allocated in "high computations" (i.e., influenced by high branching conditions) must not flow to low variables and fields. References that are low-visible may differ between two runs, since the memory allocator may be forced to choose different addresses due to differing high allocations (unless we make unrealistic assumptions about the allocator as in some works). Suppose a state s takes the form $s = (h,r)$ where r is an assignment to variables as before and h is a partial function from references to objects (that map field names to values). Indistinguishability of (h,r) and (h',r') can be formalized in terms of a partial bijective relation on references, interpreted as a renaming between the visible references in $\mathsf{dom}\,h$ and those in $\mathsf{dom}\,h'$ (as in [4,8,22]).

The second obstacle is that self-composition requires a "renamed copy" of the heap —but objects are anonymous. To join (h,r) and (h',r') into a single state, r and r' are combined as a disjoint union $r \uplus r'$ as before. But how can h and h' be combined into a single heap? And how can S be transformed to an S' such that computations of $S;S'$ correspond to pairs of computations of S? —all in a way that can be expressed using assertions in a specification language like JML [19]. Our solution adds ghost fields to every object; these are used in assertions that express the partition of the heap into dashed and undashed parts as well as a partial bijection between them. Theorem 1 says that our solution is sound and complete (relative to the verification system). The theorem applies to relational properties in general, not just noninterference. The full significance of this pertains to data abstraction and is beyond the scope of this paper, but the importance of relations between arbitrary pairs S and S' should become apparent in the following paragraphs.

Theorem 1 says that S is noninterferent just if the corresponding "partial correctness judgement" (Hoare triple) is valid for $S;S'$ where S' is the dashed copy. The point is to use an off-the-shelf verifier to prove it. But our relations involve the ghost fields that encode a partial bijection; as usual with auxiliary variables, a proof will only be possible if the program is judiciously augmented with assignments to these variables. The assignments are only needed at points where visible objects are allocated: for $x := \mathbf{new}\ C$ in the original program S (where C is the name of the object's class), we need in S' to add assignments to fields of the object referenced by x' to link the two —after both have been allocated: $x := \mathbf{new}\ C; x' := \mathbf{new}\ C; Mate(x,x')$ where $Mate$ abbreviates the ghost assignments. But consider the following toy example where allocation occurs in a loop. The policy is that `secret` does not influence the result, which is an array of objects each with `val` field set to `x`.

```
Node[] m(int x, int secret) {
   Node[] m_result;  m_result= new Node[10];  int i= 0;
   while (i<10) { m_result[i]= new Node();  m_result[i].val= x; i++; }
   return m_result;         }
```

If two copies of the method body are sequentially composed, all the undashed objects have been allocated before any of the dashed ones are, so they cannot be paired up as required, at least not without additional reasoning about the postcondition of the first loop —the first obstacle reappears!

To overcome the first obstacle, Terauchi and Aiken [30] exploit that the sequence $S;S'$ has special structure. They give a transformation whereby S' is interleaved and partially merged with S so that equalities between undashed variables and their dashed counterparts are more easily tracked by an automated verifier. In particular, for a loop **while** E **do** S **od** with guard condition E known to be low, the two copies can be merged as **while** E **do** $S;S'$ **od** rather than **while** E **do** S **od**; **while** E' **do** S' **od**. (Example method `m` is shown self-composed in Fig. 3 and transformed in Fig. 2.) There are other optimizations, e.g., commuting independent commands and replacing $x := E; x' := E'$ by $x := E; x' := x$ in case E is an integer expression known to be low (and a mating version for reference types). The transformations depend on prior information and of course they must be proved sound. The prior information is itself a noninterference property (e.g., $E = E'$ modulo renaming of references, for the loop transformation), but it can be weaker than the ultimate policy to be checked. The idea is that a type based analysis is used first; if it fails to validate conformance with the desired policy, it may still determine that some expressions are low and this can be exploited to facilitate the self-composition technique.

Similar considerations apply to modular reasoning about method calls, for which thorough investigation is left to future work.

This paper formulates the transformations using *relational Hoare logic* as advocated by Benton [9]. The observation is that the contextual information on which many transformations depend (e.g., compiler optimizations) can be expressed as relational properties, typically partial equivalence relations, that are checked by various static analyses (including information flow typing). To adapt and extend Benton's logic from simple imperative programs to objects requires that renamings be incorporated and additional rules are needed. Type inference would be performed on the originaal program S, but the program to be transformed is the self-composed version $S; S'$, so an intricate embedding is needed to justify use of the transformed program to check security of S. This is Theorem 2, which is formulated semantically. Development of the requisite proof rules is left to future work.

Overview. Sect. 2 sketches the language for which our results are formalized, focusing on the model of state. Sect. 3 formalizes relational correctness judgements ("Hoare quadruples") and defines some important relations like indistinguishability. Noninterference for command S in context Γ is expressed as the relational correctness judgement $\Gamma|\Gamma \models S \sim S : \mathscr{R} \longrightarrow\!\!\!\triangleright\, \mathscr{S}$ where $\mathscr{R}$ and $\mathscr{S}$ are the precondition and postcondition expressing the security policy. (Notation adapted from [28,9].)

Sect. 4 gives the main definitions, which use ghost fields and local conditions to encode a pair of states as a single state, and thereby encode relations as predicates. Sect. 5 gives the first theorem: a program satisfies a relational correctness judgement just if the self-composed version satisfies the partial correctness judgement obtained by combining pairs of initial and final states. That is, $\Gamma|\Gamma \models S \sim S : \mathscr{R} \longrightarrow\!\!\!\triangleright\, \mathscr{S}$ is equivalent to validity of a partial correctness judgement $\Delta \models \{\mathscr{R}^1\}\ S; S'\ \{\mathscr{S}^1\}$ where context Δ is the combined state space, also written $\Gamma \uplus \Gamma'$, that declares both dashed and undashed copies of the variables. Here $\mathscr{R}^1$ and $\mathscr{S}^1$ are predicates on this state space that encode relations $\mathscr{R}$ and $\mathscr{S}$, and S' is the dashed copy of S.

Sect. 6 illustrates how the encoding of state pairs from Sect. 4 can be expressed as a formula in a specification language like JML [19] and it reports on encouraging experiments. Sect. 7 describes rules by which $S; S'$ can be transformed to a merged form S^* under the assumption of some weaker security property $\Gamma|\Gamma \models S \sim S : \mathscr{T} \longrightarrow\!\!\!\triangleright\, \mathscr{T}$ that would be obtained by type inference. The transformation itself is expressed in a form like $\Delta|\Delta \models S; S' \sim S^* : \mathscr{T} \longrightarrow\!\!\!\triangleright\, \mathscr{T}$ that says the two are equivalent under the assumption. The second theorem confirms that $\Delta \models \{\mathscr{R}^1\}\ S^*\ \{\mathscr{S}^1\}$ implies $\Delta \models \{\mathscr{R}^1\}\ S; S'\ \{\mathscr{S}^1\}$, thereby reducing the original security problem, $\Gamma|\Gamma \models S \sim S : \mathscr{R} \multimap\!\!\!\triangleright \mathscr{S}$, to verification of $\Delta \models \{\mathscr{R}^1\}\ S^*\ \{\mathscr{S}^1\}$.

Sect. 8 discusses related work and issues in modular specification of the "mating invariant" in JML and similar specification languages. An online version gives omitted definitions and proofs.

2 Programs, Semantics, and Partial Correctness Judgements

Our results pertain to languages like Java where objects are instances of named classes and the class of a reference can be tested (and cast). No arithmetic operations are applicable to references, except for equality test. One key lemma (Lemma 1), is proved by induction on syntax and thus requires a semantics for commands. The full version of the paper uses a language similar to that in [3]: a complete program is a class table, i.e., closed set of class declarations in which fields and methods can be mutually recursive. Commands include field update, assignment, control structures and local blocks dynamically bound method calls —essentially sequential Java. The main ideas and results only involve the semantic entities, in particular states and state transformers.

Programs are assumed to be type-correct. A command in context, written $\Gamma \vdash S$, denotes a state transformer of type $\Gamma \rightsquigarrow \Gamma$, i.e., a total function from initial states for Γ to $\bot$-lifted states for Γ. The improper state $\bot$ represents divergence and error. This denotational style of semantics is used primarily in order to support modular reasoning about method invocations. (The full version of the paper addresses modular, per-method verification as it is done in tools like ESC/Java2.)

A single syntactic category, "variable names", is used for field, parameter, and local variable names, with typical element x. The data types are given by $T ::= \mathsf{int} \mid \mathsf{bool} \mid C$ where C ranges over names of declared classes. A value of a class type C is either null or a reference to an allocated object of type C. Subtyping is the reflexive, transitive relation $\leq$ determined by the immediate superclass given in each class declaration.

States have no dangling pointers and every object's field and every local variable holds a value compatible with its type. To formalize these conditions and others, it is convenient to separate the type of an object from the state of its fields. A *ref context* is a finite partial function ρ that maps references to class names. The idea is that if $o \in \mathsf{dom}\,\rho$ then o is allocated and moreover o points to an object of type $\rho\, o$. We write $[\![T]\!]\rho$ for the set of values of type T in a state where ρ is the ref context. In case T is a primitive type, $[\![T]\!]\rho$ is a set of values, independent from ρ. But if T is a class C then $[\![C]\!]\rho$ is the set containing nil and all the allocated references $o \in \mathsf{dom}\,\rho$ with $\rho\, o \leq C$.

Given context Γ and ref context ρ, a *store for Γ in ρ* is an assignment r of values to variables, such if $x{:}T$ is in Γ then $r x$ is in $[\![T]\!]\rho$. Let $[\![\mathsf{Sto}\Gamma]\!]\rho$ be the set of stores for Γ in ρ. For instance, we write $\mathsf{fields}C$ for the variable context of declared and inherited fields of objects of exactly type C. Thus a store in $[\![\mathsf{Sto}(\mathsf{fields}C)]\!]\rho$ represents the state of a C-object. A *heap* for ρ is a function that maps each reference $o \in \mathsf{dom}\,\rho$ to an object of class $\rho\, o$, i.e., to an element of $[\![\mathsf{Sto}(\mathsf{fields}(\rho\, o))]\!]\rho$. Thus for $h \in \mathsf{Heap}\,\rho$ and $o \in \mathsf{dom}\,\rho$, the application $h o x$ (sometimes written $h o.x$ for clarity) denotes the value of field x of object o in h. A *pre-heap* is like a heap but with dangling pointers allowed. A *program state* for given context Γ is a triple (ρ, h, r) containing a ref context, a heap, and a store for Γ. The set of states for Γ is written $[\![\Gamma]\!]$ —note the absence of parameter ρ, since the ref context is part of the state. Finally, the meaning $[\![\Gamma \vdash S]\!]$ of a command S is a (total) function from $[\![\Gamma]\!]$ to $[\![\Gamma]\!] \cup \{\bot\}$.

Partial correctness judgements. It is usual for postconditions to be two-state predicates, i.e., to have some means to refer to the initial state and thereby relate initial and final values, e.g., "old" expressions in JML or auxiliary variables. We formalize auxiliaries in terms of indexed families of predicates. Suppose that $\mathscr{P}$ and $\mathscr{Q}$ are indexed families of predicates $\mathscr{P}_\tau \subseteq [\![\Gamma]\!]$, $\mathscr{Q}_\tau \subseteq [\![\Gamma]\!]$ for some Γ and with τ ranging over some set. Then define $\Gamma \models \{\mathscr{P}\}\ S\ \{\mathscr{Q}\}$ to mean $\forall \tau\ .\ \Gamma \models \{\mathscr{P}_\tau\}\ S\ \{\mathscr{Q}_\tau\}$. Here $\Gamma \models \{\mathscr{P}_\tau\}\ S\ \{\mathscr{Q}_\tau\}$ means $\forall t \in [\![\Gamma]\!]\ .\ \mathscr{P}_\tau t \wedge [\![\Gamma \vdash S]\!]t \neq \bot \Rightarrow \mathscr{Q}_\tau([\![\Gamma \vdash S]\!]t)$. In fact our only use of auxiliaries is to manipulate pointer renamings encoded in the state.

3 Coupling Relations and Relational Correctness Judgements

In this section we specify noninterference in terms of relational correctness judgements, where couplings involve bijective renaming of visible objects. Throughout the paper, we let τ and σ range over finite bijective relations on the (infinite) set of references. For such a relation we write $\tau{:}\rho \leftrightarrow \rho'$, and say τ is a *typed bijection from ρ to ρ'*, if and only if $\mathsf{dom}\,\tau \subseteq \mathsf{dom}\,\rho$, $\mathsf{rng}\,\tau \subseteq \mathsf{dom}\,\rho'$, and $\rho\, o = \rho'\, o'$ for all $(o,o') \in \tau$. Finally, τ is *total*, and called a *renaming*, if $\mathsf{dom}\,\tau = \mathsf{dom}\,\rho$ and $\mathsf{rng}\,\tau = \mathsf{dom}\,\rho'$. For states we write $\tau{:}s \leftrightarrow s'$ to abbreviate $\tau{:}\rho \leftrightarrow \rho'$ where $s = (\rho,h,r)$ and $s' = (\rho',h',r')$. Also $(o,o') \in \tau$ is often written as a curried application $\tau\, o\, o'$, that is, we confuse sets with characteristic functions.

For any Γ and Γ', an *indexed relation from Γ to Γ'* is a family, $\mathscr{R}$, indexed on typed bijections, such that $\mathscr{R}_\tau \subseteq [\![\Gamma]\!] \times [\![\Gamma']\!]$ for all τ and moreover if $\mathscr{R}_\tau(\rho,h,r)(\rho',h',r')$ then $\tau{:}\rho \leftrightarrow \rho'$. Note that we do not write $\mathscr{R}_\tau \subseteq [\![\Gamma]\!]\rho \times [\![\Gamma']\!]\rho'$ —we have not defined $[\![\Gamma]\!]\rho$. The first example is a kind of identity relation that takes into account that programs are insensitive to renaming, owing to the absence of address arithmetic. Indexed relations (on states) are usually defined in terms of a hierarchy of relations on simpler semantic objects —stores and values— and we reflect this in the notation.

$$\begin{array}{lll}
\mathsf{Id}_\tau \Gamma\,(\rho,h,r)\,(\rho',h',r') & \iff & (\tau{:}\rho \leftrightarrow \rho',\ \text{is total})\ \wedge\ \mathsf{Id}_\tau\,(\mathsf{Sto}\,\Gamma)\,r\,r' \\
 & & \wedge\ \forall (o,o') \in \tau\ .\ \mathsf{Id}_\tau\,(\mathsf{Sto}(\mathsf{fields}(\rho\, o)))\,(h\, o)\,(h'\, o') \\
\mathsf{Id}_\tau\,(\mathsf{Sto}\,\Gamma)\,r\,r' \iff \forall (x{:}T) \in \Gamma\ .\ \mathsf{Id}_\tau T\,(r\,x)\,(r'\,x) & & \\
\mathsf{Id}_\tau T\,v\,v' \iff v = v' \quad \text{for primitive type } T & & \\
\mathsf{Id}_\tau C\,v\,v' \iff (v = \mathsf{nil} = v') \vee \tau\, v\, v' \quad \text{for class } C & &
\end{array}$$

Strictly speaking, it is the function mapping τ to $\mathsf{Id}_\tau \Gamma$ that is an indexed relation (in this case, from Γ to Γ); but we indulge in harmless rearrangement of parameters for clarity.

To understand the third conjunct in the definition of $\mathsf{Id}_\tau \Gamma$, recall that $\mathsf{fields}\,C$ is the typing context for the fields declared and inherited in class C, and $\rho\, o$ is the class of reference o. So this conjunct uses the instantiation $\Gamma := \mathsf{fields}(\rho\, o)$ of the relation $\mathsf{Id}_\tau\,(\mathsf{Sto}\,\Gamma)$ for stores. Note that $\mathsf{Id}_\tau T \subseteq [\![T]\!]\rho \times [\![T]\!]\rho'$ and $\mathsf{Id}_\tau\,(\mathsf{Sto}\,\Gamma) \subseteq [\![\mathsf{Sto}\,\Gamma]\!]\rho \times [\![\mathsf{Sto}\,\Gamma]\!]\rho'$.

Although $\mathsf{Id}_\tau \Gamma$ requires τ to be total on the relevant ref contexts, the definitions of $\mathsf{Id}_\tau(\mathsf{Sto}\Gamma)$ and $\mathsf{Id}_\tau T$ make no such restriction on τ. This is exploited in the definition of relation Ind and others below which require τ to be from ρ to ρ' but not total.

The next relation is the simple form of indistinguishability with respect to a set *vis* of low fields and variables, used, e.g., in [4].

$$\begin{aligned}
\mathsf{Ind}_\tau^{vis}\Gamma(\rho,h,r)(\rho',h',r') &\iff (\tau{:}\rho\leftrightarrow\rho') \wedge \mathsf{Ind}_\tau^{vis}(\mathsf{Sto}\Gamma)\,r\,r' \\
&\qquad \wedge\ \forall(o,o')\in\tau\ .\ \mathsf{Ind}_\tau^{vis}(\mathsf{Sto}(\mathsf{fields}(\rho\, o)))\,(h o)\,(h' o') \\
\mathsf{Ind}_\tau^{vis}(\mathsf{Sto}\Gamma)\,r\,r' &\iff \forall(x{:}T)\in\Gamma\ .\ x\in vis \Rightarrow \mathsf{Id}_\tau T\,(r x)\,(r' x)
\end{aligned}$$

Note that $\mathsf{Ind}_\tau^{vis}\Gamma$ differs from $\mathsf{Id}_\tau\Gamma$ in that Ind_τ^{vis} does not require τ to be total. References in $\mathsf{dom}\,\tau$ or in $\mathsf{rng}\,\tau$ are forced to include all those that are visible to the low observer; this is because the definition of $\mathsf{Ind}_\tau^{vis}(\mathsf{Sto}\Gamma)$ requires the Id_τ relation to hold for all visible variables (and fields), which in turn requires that references in these variables are related by τ.

Whereas Id and Ind are from Γ to itself, we need similar relations from Γ to the dashed copy Γ'. (Recall that fields do not get renamed, only locals.)

$$\begin{aligned}
\mathsf{Idd}_\tau\Gamma(\rho,h,r)(\rho',h',r') &\iff (\tau{:}\rho\leftrightarrow\rho' \text{ is total}) \wedge \mathsf{Idd}_\tau(\mathsf{Sto}\Gamma)\,r\,r' \\
&\qquad \wedge\ \forall(o,o')\in\tau\ .\ \mathsf{Id}_\tau(\mathsf{Sto}(\mathsf{fields}(\rho\, o)))\,(h o)\,(h' o') \\
\mathsf{Idd}_\tau(\mathsf{Sto}\Gamma)\,r\,r' &\iff \forall(x{:}T)\in\Gamma\ .\ \mathsf{Id}_\tau T\,(r x)\,(r' x')
\end{aligned}$$

Note that relation Id suffices for the heap objects and for values; the only real difference is that for stores we need to compare $r x$ with $r' x'$, i.e., to impose $x = x'$. Similarly:

$$\begin{aligned}
\mathsf{Indd}_\tau^{vis}\Gamma(\rho,h,r)(\rho',h',r') &\iff (\tau{:}\rho\leftrightarrow\rho') \wedge \mathsf{Indd}_\tau^{vis}(\mathsf{Sto}\Gamma)\,r\,r' \\
&\qquad \wedge\ \forall(o,o')\in\tau\ .\ \mathsf{Ind}_\tau^{vis}(\mathsf{Sto}(\mathsf{fields}(\rho\, o)))\,(h o)\,(h' o') \\
\mathsf{Indd}_\tau^{vis}(\mathsf{Sto}\Gamma)\,r\,r' &\iff \forall(x{:}T)\in\Gamma\ .\ x\in vis \Rightarrow \mathsf{Id}_\tau T\,(r x)\,(r' x')
\end{aligned}$$

For heap objects, the fields are not renamed, so $\mathsf{Ind}_\tau^{vis}(\mathsf{Sto}(\mathsf{fields}(\rho\, o)))$ is used here.

Suppose S and S' are commands over Γ and Γ' respectively, and $\mathscr{R}$, $\mathscr{S}$ are indexed relations from Γ to Γ'. Here Γ' can be any variable context, not necessarily the dashed copy of Γ; even $\Gamma' = \Gamma$ is allowed. We define the *relational correctness judgement* $\Gamma|\Gamma' \models S \sim S' : \mathscr{R} \longrightarrow\!\!\triangleright\, \mathscr{S}$ to mean $\forall\tau\ .\ \Gamma|\Gamma' \models [\![\Gamma \vdash S]\!] \sim [\![\Gamma' \vdash S']\!] : \mathscr{R}_\tau \longrightarrow\!\!\triangleright\, \mathscr{S}_\tau$. This in turn is defined in the following.

Definition 1 (relational correctness judgement). Suppose f is in $\Gamma \rightsquigarrow \Gamma$, f' is in $\Gamma' \rightsquigarrow \Gamma'$, both $\mathscr{R}$ and $\mathscr{S}$ are indexed relations from Γ to Γ', and τ is a typed bijection. Define $\Gamma|\Gamma' \models f \sim f' : \mathscr{R}_\tau \longrightarrow\!\!\triangleright\, \mathscr{S}_\tau$ iff $\forall s, s'\ .\ \mathscr{R}_\tau s s' \wedge f s \neq \bot \wedge f' s' \neq \bot \Rightarrow \mathscr{S}_\tau (f s)(f' s')$.

For the languages of [3,4], one can prove that programs are insensitive to renaming in the following sense: For all $\Gamma \vdash S$ we have $\Gamma|\Gamma \models S \sim S : \mathsf{Id}\,\Gamma \longrightarrow\!\!\triangleright\, \mathsf{Id}\,\Gamma$.

One might expect to express noninterference for S and policy *vis* as the judgement $\Gamma|\Gamma \models S \sim S : \mathsf{Ind}^{vis} \longrightarrow\!\!\triangleright\, \mathsf{Ind}^{vis}$ but this does not take into account that newly allocated objects can exist in the final state. In fact a sensible formulation of policy is $\Gamma|\Gamma \models S \sim S : (\exists\tau\ .\ \mathsf{Ind}_\tau^{vis}) \longrightarrow\!\!\triangleright\, (\exists\tau\ .\ \mathsf{Ind}_\tau^{vis})$ which is attractive in that it eliminates the need for top level quantification over τ. But for modular checking of method calls, in particular

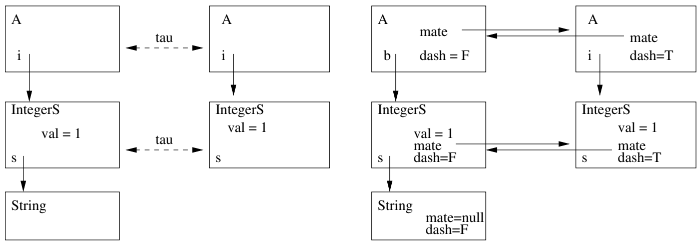


Fig. 1. On the left are two related heaps, encoded as one five-object heap on the right

to reason about the caller's store after a method call, it is important for the bijection supporting the final state to be an extension of the initial one. The property shown to be enforced by a type system in [4] is indeed this slightly stronger but more intricate property: $\Gamma|\Gamma \models S \sim S : \mathsf{Ind}^{vis}_{\tau} \longrightarrow\!\!\triangleright\, \exists\sigma \supseteq \tau\ .\ \mathsf{Ind}^{vis}_{\sigma}$ for all τ. (Here we write a logical quantifier but the meaning is a union $\cup_{\sigma\supseteq\tau}\mathsf{Ind}^{vis}_{\sigma}$.) That is, our main use of relational correctness judgements will instantiate $\mathscr{S}_{\tau}$ in Def. 1 by $\mathscr{S}_{\tau} := \exists\sigma\ .\ \sigma \supseteq \tau \wedge \mathscr{R}_{\sigma}$. In the self-composed version to be used by a verifier, the bijection is encoded in auxiliary state and the condition $\sigma \supseteq \tau$ comes for free because the ghost fields need never be updated after initialization.

It is convenient to express noninterference in terms of the renamed program. Let Γ' be the dashed copy of Γ and S' be the dashed copy of S. Then clearly we have $\Gamma|\Gamma \models S \sim S : \mathsf{Ind}^{vis} \longrightarrow\!\!\triangleright\, \mathsf{Ind}^{vis}$ if and only if $\Gamma|\Gamma' \models S \sim S' : \mathsf{Indd}^{vis} \longrightarrow\!\!\triangleright\, \mathsf{Indd}^{vis}$.

4 Ghost Mating: Encoding Relations as State Predicates

This section defines the encoding of two states as one. Class Object is assumed to declare ghost fields $\mathsf{dash} : \mathsf{bool}$ and $\mathsf{mate} : \mathsf{Object}$, so that they are present in all objects. None of the considered relations or programs should depend on these fields except through explicit use in the encoding.

Suppose ρ and ρ' are disjoint ref contexts, written $\rho\#\rho'$ (meaning $\mathsf{dom}\,\rho\#\mathsf{dom}\,\rho'$). Suppose we have typed bijection $\tau : \rho \leftrightarrow \rho'$, not necessarily total, and heaps $h \in \mathsf{Heap}\,\rho$, $h' \in \mathsf{Heap}\,\rho'$. We aim to encode a pair h, h' as a single heap k for $\rho \uplus \rho'$. The idea is that, in k, an object $o \in \mathsf{dom}\,\rho$ will have $k\,o.\mathsf{dash} = \mathsf{false}$ whereas an $o \in \mathsf{dom}\,\rho'$ will have $k\,o.\mathsf{dash} = \mathsf{true}$. Moreover, if $\tau\,o\,o'$ then we will have $k\,o.\mathsf{mate} = o'$ and $k\,o'.\mathsf{mate} = o$. This arrangement is formalized by conditions on k which can be expressed in formulas as $o.\mathsf{mate} \neq \mathsf{nil} \Rightarrow o.\mathsf{dash} = \neg o.\mathsf{mate}.\mathsf{dash} \wedge o.\mathsf{mate}.\mathsf{mate} = o$ and $o.x \neq \mathsf{nil} \Rightarrow o.\mathsf{dash} = o.x.\mathsf{dash}$ for every class type field $(x : C) \in \mathsf{fields}(\rho\,o)$, with $x \not\equiv \mathsf{mate}$. The right side of Figure 1 depicts a well mated heap. Given disjoint contexts Γ and Γ', a well mated state for $\Gamma \uplus \Gamma'$ is one where references in variables of Γ are to undashed objects (and Γ' to dashed). This notion is only used in the case that Γ' is the dashed copy of Γ.

Definition 2 (well mated state). Given disjoint contexts $\Gamma\#\Gamma'$, state $(\rho, h, r) \in [\![\Gamma \uplus \Gamma')]\!]$ is *well mated for* Γ *and* Γ' iff (a) h is well mated; (b) $r\,x = \mathsf{nil} \vee h(r\,x).\mathsf{dash} = \mathsf{false}$, for every x in $\mathsf{dom}\,\Gamma$ with $\Gamma\,x$ a class type; and (c) $r\,x' = \mathsf{nil} \vee h(r\,x').\mathsf{dash} = \mathsf{true}$, for every x' in $\mathsf{dom}\,\Gamma'$ with $\Gamma'\,x'$ a class type.

Note that there is no restriction on primitive values. We have $x \not\equiv \mathsf{mate}$ here because we assume field names are never reused as variable names.

Suppose that Γ is disjoint from Γ'. Given a typed bijection τ from ρ to ρ', we aim to combine states, say $(\rho,h,r) \in [\![\Gamma]\!]$ and $(\rho',h',r') \in [\![\Gamma']\!]$, into a single state that is well mated and reflects τ. We cannot assume $\rho\#\rho'$ —from initial states with disjoint heaps, running a pair of programs could lead to non-disjoint heaps (since we make no assumptions about the allocator). Instead, our construction includes a suitable renaming to make the heaps disjoint.

As a first step, function match is defined as follows. The idea is that for any $h \in \mathsf{Heap}\,\rho$, any $\tau : \rho \leftrightarrow \rho'$, and any boolean b, $\mathsf{match}(h,\tau,b)$ is a pre-heap where $o.\mathsf{dash} = b$ for every o and moreover if o is in the domain of τ then $o.\mathsf{mate}$ is a —dangling!— reference in accord with τ.

Now we define the combined state, $\mathsf{join}_\tau(\rho,h,r)(\rho',h',r')$, by the following steps. First, choose $\hat{\rho}$ and $\hat{\tau}$ such that $\hat{\rho}\#\rho$ and $\hat{\tau}$ is a renaming from ρ' to $\hat{\rho}$. Let $\hat{h}$ be the renaming of h' by $\hat{\tau}$ and *mutatis mutandis* for $\hat{r}$ and r'. Let $h_0 = \mathsf{match}(h,(\tau;\hat{\tau}),\mathsf{false})$ and also $\hat{h}_0 = \mathsf{match}(\hat{h},(\hat{\tau}^{-1};\tau^{-1}),\mathsf{true})$, writing ";" for relational composition. Finally, define $\mathsf{join}_\tau(\rho,h,r)(\rho',h',r') = ((\rho \uplus \hat{\rho}),\, h_0 \uplus \hat{h}_0,\, r \uplus \hat{r})$

Note that (ρ,h_0,r) is not quite an element of $[\![\Gamma]\!]$, because h_0 is only a pre-heap due to the dangling mate fields. For the same reason, $(\hat{\rho},\hat{h},\hat{r})$ is almost but not quite an element of $[\![\Gamma']\!]$. What matters is that if τ is a typed bijection from ρ to ρ' and $\Gamma\#\Gamma'$ then $\mathsf{join}_\tau(\rho,h,r)(\rho',h',r')$ is in $[\![\Gamma \uplus \Gamma']\!]$ and is well mated.

To partition a well mated heap into two, we first define $\mathsf{dsh}\ h = \{o \in \mathsf{dom}\,h \mid h\,o.\mathsf{dash} = \mathsf{true}\}$ and $\mathsf{undsh}\ h = \{o \in \mathsf{dom}\,h \mid h\,o.\mathsf{dash} = \mathsf{false}\}$. Roughly speaking, $h{\restriction}(\mathsf{dsh}\,h)$, i.e., h with its domain restricted to *include* only dashed objects, is in $\mathsf{Heap}(\rho{\restriction}(\mathsf{dsh}\,h))$. But in fact $h{\restriction}(\mathsf{dsh}\,h)$ may have dangling references in mate fields, so we define a function dematch that sets all mate fields to nil.

For splitting to invert joining we cannot just discard mates. If k in $\mathsf{Heap}\,\rho$ is well mated then we obtain typed bijection $\tau : (\rho{\restriction}(\mathsf{undsh}\,h)) \leftrightarrow (\rho{\restriction}(\mathsf{dsh}\,h))$ by

$$\tau\, o\, o' \iff k\,o.\mathsf{dash} = \mathsf{false} \wedge k\,o.\mathsf{mate} = o' \qquad (1)$$

Splitting and joining are mutually inverse, modulo renaming. The intricate details are omitted in this extended abstract. One consequence is that every well mated state in $[\![\Gamma \uplus \Gamma']\!]$ is equal, up to renaming, to one in the range of join. Even more, a well mated state that encodes via (1) a particular bijection τ is in the range of join_τ. Thus, for a relation that is insensitive to renaming, we can give a pointwise definition of a corresponding predicate.

Definition 3 (coupling relation to mated predicate). Given an indexed relation $\mathscr{R}$ from Γ to Γ' with $\Gamma\#\Gamma'$, define a predicate $\mathscr{R}^1_\tau \subseteq [\![\Gamma \uplus \Gamma']\!]$ by

$$\mathscr{R}^1_\tau\, t \iff \exists s,s' \,.\, t = \mathsf{join}_\tau s\, s' \wedge \mathscr{R}_\tau s\, s'$$

That is, t is in $\mathscr{R}^1_\tau$ iff t is well mated for τ and splits as some s,s' with $\mathscr{R}_\tau s\, s'$.

As an example, if a state with heap h satisfies $(Ind^{vis}_\tau)^1$ then for any o with $h\,o.\mathsf{mate} \neq \mathsf{nil}$ the visible primitive fields of $h\,o$ are equal to those of $h\,o.\mathsf{mate}$ and the visible class fields of $h\,o$ are mated. There is no constraints for field names not in *vis*.

5 Hoare Quadruples Reduced to Triples

This section shows that a relational correctness judgement for some coupling relation is equivalent to a partial correctness judgement for the corresponding predicate and the self-composed program. To begin, it is convenient to define $f \rhd f'$ which applies state transformer f and then f'. Suppose $\Gamma \# \Gamma'$, f is in $\Gamma \rightsquigarrow \Gamma$, and f' is in $\Gamma' \rightsquigarrow \Gamma'$. Define $f \rhd f'$ to be an element of $\Gamma \uplus \Gamma' \rightsquigarrow \Gamma \uplus \Gamma'$ as follows, where we partition the store as r, r' in accord with Γ, Γ'.

$$\begin{aligned}(f \rhd f')(\rho, h, r \uplus r') = \ & \mathsf{let}\ (\rho_0, h_0, r_0) = f(\rho, h, r)\ \mathsf{in} \\ & \mathsf{let}\ (\rho_1, h_1, r_1) = f'(\rho_0, h_0, r')\ \mathsf{in}\ (\rho_1, h_1, r_0 \uplus r_1)\end{aligned}$$

Our meta-notation "$\mathsf{let} - \mathsf{in}$" is $\bot$-strict, so $f \rhd f'$ returns $\bot$ if either f or f' does.

This notion is useful in case f acts only on the undashed part of the heap and f' on the dashed part, but the definition is more general. Note also that the domain $\Gamma \uplus \Gamma' \rightsquigarrow \Gamma \uplus \Gamma'$ includes state transformers that in no way respect the $\mathsf{dash}/\mathsf{mate}$ structure. In particular, well matedness of s does not imply the same for $(f \rhd f')s$. But we have the following.

Lemma 1. If $\Gamma \vdash S$, $\Gamma' \vdash S'$, and $\Gamma \# \Gamma'$ then $([\![\Gamma \vdash S]\!] \rhd [\![\Gamma' \vdash S']\!])s = [\![\Gamma \uplus \Gamma' \vdash S;S']\!]s$ for any well mated s. (Recall that we assume dash and mate do not occur in S or S'.)

A state transformer f is *independent from* $\mathit{mate}, \mathit{dash}$ provided it does not update these fields on initially existing objects or newly allocated objects and moreover $s \lfloor dm = t \lfloor dm \Rightarrow (f\,s) \lfloor dm = (f\,t) \lfloor dm$, where we abbreviate dm for $\mathsf{dash}, \mathsf{mate}$ and $\lfloor$ removes elements from a function's domain.

Lemma 2. Suppose $\Gamma \# \Gamma'$, f is in $\Gamma \rightsquigarrow \Gamma$, and f' is in $\Gamma' \rightsquigarrow \Gamma'$. (a) For any τ, u, s, s', if $u \lfloor dm = ((f \rhd f')(\mathsf{join}_\tau s s')) \lfloor dm$ and $u = \mathsf{join}_\sigma t t'$ for some σ, t, t' then $t = f\,s$ and $t' = f'\,s'$. (b) If f, f' are independent from $\mathsf{mate}, \mathsf{dash}$ then $((f \rhd f')(\mathsf{join}_\tau s s')) \lfloor dm$ is equivalent to $\mathsf{join}_\sigma\,(f\,s)(f'\,s')$ for some σ, up to renaming (i.e., related by Id).

To state the precise correspondence, in terms of states that include the dash and mate fields, we need to mask them as follows. Define $\hat{\mathscr{R}}^1_\tau$ by

$$\hat{\mathscr{R}}^1_\tau t \iff \exists u \,.\, u \lfloor dm = t \lfloor dm \wedge \mathscr{R}^1_\tau u$$

Theorem 1. Suppose $\Gamma \# \Gamma'$ and consider commands in context $\Gamma \vdash S$ and $\Gamma' \vdash S'$. Suppose $\mathscr{R}$ and $\mathscr{S}$ are indexed relations from Γ to Γ' that are insensitive to renaming. Then for any τ we have

$$\Gamma | \Gamma' \models S \sim S' : \mathscr{R}_\tau \mathrel{-\!\!\rhd} \exists \sigma \supseteq \tau \,.\, \mathscr{S}_\sigma \quad \text{iff} \quad \Gamma \uplus \Gamma' \models \{\hat{\mathscr{R}}^1_\tau\}\ S;S'\ \{\exists \sigma \,.\, \sigma \supseteq \tau \wedge \hat{\mathscr{S}}^1_\sigma\}$$

In particular, noninterference for a command S and policy vis is, by definition, the property that $\Gamma | \Gamma \models \mathscr{R}_\tau \sim S : S \mathrel{-\!\!\rhd} \exists \sigma \,.\, \sigma \supseteq \tau \wedge \hat{\mathscr{R}}_\sigma$ (for all τ) where $\mathscr{R}$ is $\mathsf{Ind}^{vis}_\tau \Gamma$. This is the same as the renamed version $\Gamma | \Gamma' \models \mathscr{R}_\tau \sim S : S' \mathrel{-\!\!\rhd} \exists \sigma \,.\, \sigma \supseteq \tau \wedge \hat{\mathscr{R}}_\sigma$ where $\mathscr{R}$ is $\mathsf{Indd}^{vis}_\tau \Gamma$. The Theorem reduces this to the triple $\Gamma \uplus \Gamma' \models \{\mathscr{R}^1_\tau\}\ S;S'\ \{\exists \sigma \,.\, \sigma \supseteq \tau \wedge \hat{\mathscr{R}}^1_\sigma\}$.

6 Experiments: Expressing Well Mating and Relations as Assertions

Sections 4 and 5 formulate the technique in semantic terms. A key feature of our encoding is that it requires no special instrumentation of program semantics but rather is expressed by first order conditions over auxiliary state. One can think of a number of variations on encoding; we describe one convenient pattern.

To express the self composed program using JML, a fresh method with two copies of the parameters is used. For non-static methods, the target object (`this`) needs to be made an explicit parameter so there can be two copies. Two copies of the result are needed; our encoding uses fields for this purpose. As a simple example, consider this method where the policy is that `secret` does not influence the result.

```
static int p(int x, int y, int secret) {
   x= secret; if (secret % 2 == 1) y=x * secret; else y= secret * secret;
   return y - secret * x;             }
```

For the self composed version, two fields and a new method `Pair_p` are added to the class, as follows (using $ for dash which is not legal in Java identifiers).

```
int p_result, p_result$; // new fields to hold the pair of results of p

/*@ requires x==x$ && y==y$;
  @ modifies p_result, p_result$;
  @ ensures p_result == p_result$;
  @*/
void Pair_p(int x, int y, int secret, int x$, int y$, int secret$) {
   x= secret; if (secret % 2 == 1) y=x * secret; else y= secret * secret;
   p_result= y - secret * x;
   x$= secret$; if(secret$ % 2==1) y$=x$*secret$;else y$=secret$*secret$;
   p_result$= y$ - secret$ * x$;                                    }
```

This is verified by ESC/Java2 (version 2.0a9) in 0.057sec; insecure versions are quickly rejected. Similar results for this and the other experiments were found using the Spec# tool. Note that ESC/Java2 is deliberately unsound in some ways, though not in ways that appear relevant to the present experiments.

Another experiment is to adapt the preceding method `p` by using a wrapper object for the result. For experiment we add the ghost fields explicitly where needed.

```
class Node {
   public int val;
  /*@ ghost public Node mate; */
  /*@ ghost public boolean dash; */   }
```

The variation using such a wrapper object verifies without difficulty, since the requisite ghost updates can be added following the second allocation.

As discussed in Sect. 1 and justified in Sect. 7 to follow, loops are most easily checked by applying an interleaving transformation for allocations that occur under low guards. For method `m` in Sect. 1 the self-composed version appears in Figure 2, where the transformation from **while** E **do** S **od**;**while** E' **do** S' **od** to **while** E **do** $S;S'$ **od**

```
Node[] m_result, m_result$; // new fields to hold the pair of results of m

  /*@ requires x==x$;  // policy
    // ordinary preconditions
    @ ensures m_result != null && m_result$ != null;
    @ ensures m_result.length==10 && m_result$.length==10;
    @ ensures (\forall int j; 0<=j&&j<10 ==> m_result[j]!=null);
    @ ensures (\forall int j; 0<=j&&j<10 ==> m_result$[j]!=null);
    // mating and policy
    @ ensures (\forall int j; 0<=j&&j<10 ==> m_result[j].mate==m_result$[j]);
    @ ensures (\forall int j; 0<=j&&j<10 ==> m_result$[j].mate==m_result[j]);
    @ ensures (\forall int j; 0<=j&&j<10 ==> !m_result[j].dash);
    @ ensures (\forall int j; 0<=j&&j<10 ==> m_result$[j].dash);
    @ ensures (\forall int j; 0<=j&&j<10 ==> m_result[j].val==m_result$[j].val);
    @ assignable m_result, m_result$, m_result[*], m_result$[*];
  */
  void Pair_m(int x, int secret, int x$, int secret$) {
      m_result= new Node[10];  m_result$= new Node[10];
      // **mating assignments for the arrays would go here**
      int i= 0;
      //@ maintaining ...
      while (i<10) {
         m_result[i]= new Node();  m_result$[i]= new Node();
         //@ set m_result[i].dash= false;         set m_result$[i].dash= true;
         //@ set m_result[i].mate= m_result$[i];  set m_result$[i].mate= m_result[i];
         m_result[i].val= x;  m_result$[i].val= m_result[i].val;
         i++;
      }                                                }
```

Fig. 2. Self-composed and transformed method `m` from Sect. 1, with JML annotation

has been applied. The self-composed version needs to be annotated with assignments to the ghost fields (written as JML comments with keyword `set`), at the point where the dashed copy has been allocated and is low-visible. Here we do not mate the arrays themselves, since JML doesn't allow ghost fields to be added to arrays, but we do mate their contents. This example verifies in 0.425sec (using the -loopSafe option for soundness) with the elipses replaced by an obvious invariant derived from the postcondition by a standard heuristic (replace constant 10 by variable i).

To illustrate that the transformation is not necessary in general, Fig. 3 shows the running example self-composed but not transformed. The mating assignments are all in the second loop body and this version verifies in 0.529sec. It works because the objects created by the first loop are easily referenced since they are in an array. But whereas the loop invariants needed for Fig. 2 are obtained from the postcondition by simply replacing constant 10 by index variable i, the version in Fig. 3 requires additional invariants (the only ones shown) expressing the absence of aliasing since the allocations are separated in the code. If instead of an array one considers a linked list or other linked structure, it is more difficult to state such invariants.

7 Transforming the Self-composed Command

Terauchi and Aiken propose an interleaving transformation like the one used in the preceding experiment and described in Sect. 1. They show it sound, but in the setting of a simpler language without objects. It depends on conservative analysis that could be obtained by type inference. Their formulation does not suggest an obvious way to extend the results to richer language features or policies. This section sketches how to

```
// Specification same as in previous version...
void Pair_m(int x, int secret, int x$, int secret$) {
    m_result= new Node[10];   m_result$= new Node[10];
    int i= 0;
    //@ maintaining ...(\forall int j,k; 0<=j&&j<k&&k<i ==> m_result[j]!=m_result[k]);
    while (i<10) {
       m_result[i]= new Node();
       m_result[i].val= x;
       i++;
    }
    i= 0;
    //@ maintaining ...(\forall int j,k; 0<=j&&j<k&&k<10 ==> m_result[j]!=m_result[k]);
    //@ maintaining (\forall int j,k; 0<=j&&j<10&&0<=k&&k<i ==> m_result[j]!=m_result$[k]);
    while (i<10) {
       m_result$[i]= new Node();
       m_result$[i].val= x$;
       //@ set m_result[i].dash= false;         set m_result$[i].dash= true;
       //@ set m_result[i].mate= m_result$[i];  set m_result$[i].mate= m_result[i];
       i++;
    }                                                 }
```

Fig. 3. Self-composed method m, not transformed

use relational correctness judgements to formulate the interface to the analysis as well as the transformations themselves. Indexing is elided for clarity.

Suppose the goal is to check the simple noninterference property $\Gamma|\Gamma \models S \sim S : \mathsf{Ind}^{vis} \longrightarrow\!\!\triangleright\, \mathsf{Ind}^{vis}$. After renaming the second copy, Theorem 1 tells us an equivalent partial correctness judgement $\Delta \models \{(\mathsf{Indd}^{vis})^1\}\ S;S'\ \{(\mathsf{Indd}^{vis})^1\}$ where Δ is $\Gamma \uplus \Gamma'$. Instead of directly verifying this, we want S^* such that $\Delta \models \{(\mathsf{Indd}^{vis})^1\}\ S^*\ \{(\mathsf{Indd}^{vis})^1\}$ implies $\Delta \models \{(\mathsf{Indd}^{vis})^1\}\ S;S'\ \{(\mathsf{Indd}^{vis})^1\}$. The requisite transformation can be expressed by relational correctness judgements: the relations express both the notion of equivalence (e.g., modulo renaming) and the conditions under which the transformation is valid (e.g., known initial values, or final values that aren't used). Here is an example judgement that transforms $x := y$ by renaming and exploiting a precondition:

$$x,y:\mathsf{int}\,|\,x',y':\mathsf{int} \models\ x := y\ \sim\ x' := 0\ : (y = 0 \wedge y = y') \longrightarrow\!\!\triangleright\, (x = x' \wedge y = y')$$

The situation of interest is complicated by the fact that the program $S;S'$ to be transformed already acts on two copies of Γ. The rule we need for loop transformation includes an antecedent of the form $\Delta|\Delta \models E \sim E' : \mathscr{R} \longrightarrow\!\!\triangleright \ldots$ where $\mathscr{R}$ expresses that the dashed and undashed copies of E have the same value.

To establish the antecedents, the idea of Terauchi and Aiken is to use type inference to find a less precise property of S, namely $\Gamma|\Gamma \models S \sim S : \mathsf{Ind}^V \longrightarrow\!\!\triangleright\, \mathsf{Ind}^V$ for some $V \subseteq vis$. Type inference would yields this property for all constituent parts including the loop guard E (if it is low; otherwise no transformation is needed). This is now lifted to Δ by a construction applicable to any context Γ: the cartesian square of a predicate, intersected with the identity. For any $\mathscr{P} \subseteq [\![\Gamma]\!]$, define $\mathscr{P} \otimes \mathscr{P} \subseteq [\![\Gamma]\!] \times [\![\Gamma]\!]$ by $(\mathscr{P} \otimes \mathscr{P})\, s\, s' \iff s = s' \wedge \mathscr{P}\, s$.

Taking $\mathscr{R}$ to be $\mathsf{Indd}^V \otimes \mathsf{Indd}^V$, the analysis-based transformations yield $\Delta|\Delta \models S;S' \sim S^* : \mathscr{R} \longrightarrow\!\!\triangleright\, \mathscr{R}$. Now the desired judgement $\Delta \models \{(\mathsf{Indd}^{vis})^1\}\ S;S'\ \{(\mathsf{Indd}^{vis})^1\}$ (which itself encodes a relation!) is lifted to the level of relations in the squared form $\Delta|\Delta \models S;S' \sim S;S' : (\mathsf{Indd}^{vis})^1 \otimes (\mathsf{Indd}^{vis})^1 \longrightarrow\!\!\triangleright\, (\mathsf{Indd}^{vis})^1 \otimes (\mathsf{Indd}^{vis})^1$. This can now be composed with the transformation $\Delta|\Delta \models S;S' \sim S^* : \mathscr{R} \longrightarrow\!\!\triangleright\, \mathscr{R}$ by general transitivity (that is: $\Gamma|\Gamma \models S0 \sim S1 : \mathscr{R}0 \longrightarrow\!\!\triangleright\, \mathscr{S}0$ and $\Gamma|\Gamma \models S1 \sim S2 : \mathscr{R}1 \longrightarrow\!\!\triangleright\, \mathscr{S}1$ imply $\Gamma|\Gamma \models S0 \sim S2 : (\mathscr{R}0;\mathscr{R}1) \longrightarrow\!\!\triangleright\, (\mathscr{S}0;\mathscr{S}1)$). The outcome is a judgement involving composed relations like $(\mathsf{Indd}^{V} \otimes \mathsf{Indd}^{V});((\mathsf{Indd}^{vis})^{1} \otimes (\mathsf{Indd}^{vis})^{1})$. For this process to be useful, the composed relations must boil down to the original noninterference property, which they do owing to a general result about the relational logic.

Theorem 2. *Let Δ abbreviate $\Gamma \uplus \Gamma'$. Suppose $\Delta|\Delta \models S \sim S^{*} : \mathscr{R} \longrightarrow\!\!\triangleright\, \mathscr{S}$ for some S and S^{*}, where $\mathscr{R}$ and $\mathscr{S}$ are symmetric. Let $\mathscr{P}$ and $\mathscr{Q}$ be predicates on Δ such that $\mathscr{R};(\mathscr{P} \otimes \mathscr{P}) = \mathscr{P} \otimes \mathscr{P}$ and $\mathscr{P} \otimes \mathscr{P} = (\mathscr{P} \otimes \mathscr{P});\mathscr{R}$ (and* mutatis mutandis *for $\mathscr{Q}$). Then $\Delta \models \{\mathscr{P}\}\ S^{*}\ \{\mathscr{Q}\}$ implies $\Delta \models \{\mathscr{P}\}\ S\ \{\mathscr{Q}\}$.*

This can be instantiated with $S := (S;S')$ with S' being a renamed copy of S; moreover $\mathscr{P}$ and $\mathscr{Q}$ encode the desired noninterference property based on Ind^{vis} and $\mathscr{R},\mathscr{S}$ encode the result of type based analysis, e.g., $\mathscr{P}$ is $\mathsf{Indd}^{vis} \otimes \mathsf{Indd}^{vis}$, $\mathscr{R}$ is $\mathsf{Indd}^{V} \otimes \mathsf{Indd}^{V}$ and $\mathscr{Q},\mathscr{S}$ correspond to $\mathscr{P},\mathscr{R}$ but with extended bijections as usual. In the situation described above with $V \subseteq vis$ we have $\mathsf{Indd}_{\tau}^{vis} \subseteq \mathsf{Indd}_{\tau}^{V}$ because larger vis is more restrictive. This yields the requisite absorption properties, e.g., $\mathscr{R};(\mathscr{P} \otimes \mathscr{P}) = \mathscr{P} \otimes \mathscr{P}$. So the Theorem justifies the use of transformations that are sound for Indd^{V} to prove noninterference with respect to *vis*.

8 Discussion

We defined a novel encoding to support self-composition in programs acting on the heap. The encoding is expressed in terms of auxiliary state, specifically ghost fields which are available in specification languages like JML. Theorem 1 says that a relational property is equivalent to a corresponding partial correctness property of a self-composed program. The notion of relational property is general enough to encompass rich declassification policies and also to be used to reason about program transformations. Theorem 2 justifies the use of transformations like those of Aiken and Terauchi [30] which are needed to make the self-composed version amenable to off-the-shelf program verifiers (automatic or interactive), in particular to bring allocations together by merging loops. Preliminary experiments indicate that the encoding and mechanical transformations work smoothly with extant tools.

For modular verification of commands with method calls, it is desirable to transform the program so that a duplicated pair of calls can be brought together and even replaced by a single invocation of a suitably transformed version of the method (like `pair_m` in the example). The transformed version can be proved to satisfy its specification using verification, if necessary, or by security type checking.

The fact that type checking can be used to justify transformations does not mean that the verification technique achieves nothing beyond what can be type checked. Transformation is not always necessary, as illustrated by Fig. 3; similarly, method calls need not be transformed if adequate functional specifications are available. The properties needed to justify transformations can themselves be proved by verification instead of type checking.

Related work. Self-composition is essentially an application of Reynolds' method for proving data refinements [25,13], but data refinement with general heaps has only recently been studied [3] and not yet using this method. Barthe, D'Argenio, and Rezk [7] develop a general theory of self-composition. They sketch a treatment of the heap using separation logic semantics; indistinguishability of abstract lists is used to avoid the issue of pointer renaming. Terauchi and Aiken [30] note that self-composition generalizes to general relational properties (of a single program, in their case). They introduce the transformations we studied in this paper and report good experimental results for deterministic simple imperative programs using the BLAST tool [17]. Correctness of the transformations is proved under reasonable assumptions typical of type systems [30]. But their formulation seems rather specific and it is unclear how to extend it to richer semantics, e.g., where equality may only be up to renaming.

Benton [9] develops relational Hoare logic for a deterministic imperative language and applies it to analysis-based compiler optimizations. Yang [32] develops a relational Separation Logic [26]. The secure flow logic of Amtoft and Banerjee [2] can be seen as a relational Hoare logic for the special case of noninterference; it is extended to heaps in [1] and applied to declassification in [5]. The focus of Amtoft et al. is automated static analysis; an abstract interpretation for the heap is presented in "logical form".

Darvas, Hähnle, and Sands [11] use the KeY tool for interactive verification of noninterference. It uses a dynamic logic for Java, which is more expressive than ordinary partial correctness assertions, allowing in particular existential quantification over weakest-precondition statements. For nondeterministic S, the self-composed version $S;S'$ does not capture relational properties, but they can be captured using the conjugate predicate transformer $\neg\mathsf{wlp}\neg$ [16]. This suggests it would be interesting to explore the use of dynamic logic for relational properties of nondeterministic programs.

Dufay, Felty, and Matwin [15] use the self-composition technique to check noninterference for data mining algorithms implemented in Java. They use the Krakatoa tool, based on the Coq theorem prover and using JML [19]. They extend JML with special notations to refer to the two copies of program state and extend Krakatoa to generate special verification conditions. The paper does not give much detail about the heap encoding. To prove that noninterference is enforced by their security type inference system for an ML-like language, Pottier and Simonet [24] extend the language with a pairing construct and semantics that encodes two runs of a program as one.

Future work. Although our small experiments worked without difficulty, there is an impediment to scaling up the idea. The mating condition appears in preconditions and postconditions of every method, so effectively it is an object invariant. But in general it needs to be fully ramified to all fields of all reachable objects. This is tricky because in languages like JML specifications must respect the visibility rules of the language and therefore cannot refer to "all" fields. One possibility is to define the `mate` predicate as a pure method, overridden in each class—it constrains the fields visible in that class, by invoking `mate` on class type fields and invoking `super.mate` for inherited ones. Care is needed, owing to cycles in the heap; moreover reasoning about pure methods in specifications is not well supported in current verifiers [12]. Another approach is to formulate mating in a decentralized way using explicit object invariants, which are a topic of active research on modular reasoning [20]. The mating invariant is incompatible with ownership-based invariants (or model fields) but it is compatible with friendship-based invariants [23]; friendship is slated to be added to Spec# and is available as an undocumented feature in the tool of Cees Pierik (www.cs.uu.nl/groups/IS/vft/).

Theorem 2 characterizes the useful transformations and some examples have been mentioned, but it remains to develop a full set of transformations. Benton's proof system could be extended to incorporate the heap and also method calls, and syntax added to manipulate the embeddings. The idea is to derive specialized transformations like those needed for self-composition from very powerful general rules that can be formulated in a relational setting.

Self-composition generalizes to simulations for data abstraction. In particular, we plan to investigate use of the encoding for establishing the antecedent in the representation independence property [3].

Acknowledgement. Thanks to Dustin Long for help with the ESC/Java experiments. Thanks to Dustin and to Mike Barnett for adapting the Java experiments to Spec# and trying them with the Boogie verifier. Insightful comments from reviewers significantly improved the presentation.

References

1. T. Amtoft, S. Bandhakavi, and A. Banerjee. A logic for information flow in object-oriented programs. In *ACM Symposium on Principles of Programming Languages (POPL)*, 2006.
2. T. Amtoft and A. Banerjee. Information flow analysis in logical form. In *Static Analysis Symposium (SAS)*, 2004.
3. A. Banerjee and D. A. Naumann. Ownership confinement ensures representation independence for object-oriented programs. *Journal of the ACM*, 52(6):894–960, Nov. 2005.
4. A. Banerjee and D. A. Naumann. Stack-based access control for secure information flow. *Journal of Functional Programming*, 15(2):131–177, 2005.
5. A. Banerjee and D. A. Naumann. A logical account of secure declassification (extended abstract). Submitted, 2006.
6. M. Barnett, K. R. M. Leino, and W. Schulte. The Spec# programming system: An overview. In G. Barthe, L. Burdy, M. Huisman, J.-L. Lanet, and T. Muntean, editors, *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices, International Workshop (CASSIS 2004), Revised Selected Papers*, volume 3362 of *LNCS*, pages 49–69. Springer, 2005.
7. G. Barthe, P. R. D'Argenio, and T. Rezk. Secure information flow by self-composition. In *IEEE Computer Security Foundations Workshop (CSFW)*, pages 100–114, 2004.
8. G. Barthe and T. Rezk. Non-interference for a JVM-like language. In M. Fähndrich, editor, *Proceedings of TLDI'05*, pages 103–112. ACM Press, 2005.
9. N. Benton. Simple relational correctness proofs for static analyses and program transformations. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 14–25, 2004.
10. E. S. Cohen. Information transmission in sequential programs. In A. K. J. Richard A. DeMillo, David P. Dobkin and R. J. Lipton, editors, *Foundations of Secure Computation*, pages 297–335. Academic Press, 1978.
11. A. Darvas, R. Hähnle, and D. Sands. A theorem proving approach to analysis of secure information flow. In D. Hutter and M. Ullmann, editors, *Proc. 2nd International Conference on Security in Pervasive Computing*, volume 3450 of *LNCS*, pages 193–209. Springer, 2005.

12. A. Darvas and P. Müller. Reasoning about method calls in JML specifications. In ECOOP workshop on *Formal Techniques for Java-like Programs*, 2005.
13. W.-P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge University Press, 1998.
14. D. E. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.
15. G. Dufay, A. Felty, and S. Matwin. Privacy-sensitive information flow with JML. In *Conference on Automated Deduction (CADE)*, 2005.
16. D. Gries. Data refinement and the tranform. In M. Broy, editor, *Program Design Calculi*. Springer, 1993. International Summer School at Marktoberdorf.
17. T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 58–70, 2002.
18. B. Jacobs and E. Poll. Java program verification at Nijmegen: Developments and perspective. Technical Report NIII-R0318, Computing Science Institute, University of Nijmegen, 2003. In *International Symposium on Software Security*, volume 3233, of *LNCS*, pages 134–153. Springer, 2003.
19. G. T. Leavens, Y. Cheon, C. Clifton, C. Ruby, and D. R. Cok. How the design of JML accommodates both runtime assertion checking and formal verification. In F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, editors, *Formal Methods for Components and Objects (FMCO 2002)*, volume 2852 of *LNCS*, pages 262–284. Springer, 2003.
20. P. Müller, A. Poetzsch-Heffter, and G. T. Leavens. Modular invariants for layered object structures. Technical Report 424, Department of Computer Science, ETH Zurich, 2004.
21. A. C. Myers. JFlow: Practical mostly-static information flow control. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 228–241, 1999.
22. D. A. Naumann. Verifying a secure information flow analyzer. In J. Hurd and T. Melham, editors, *18th International Conference on Theorem Proving in Higher Order Logics TPHOLS*, volume 3603 of *LNCS* pages 211–226. Springer, 2005.
23. D. A. Naumann and M. Barnett. Towards imperative modules: Reasoning about invariants and sharing of mutable state. To appear in *Theoretical Computer Science*, 2006.
24. F. Pottier and V. Simonet. Information flow inference for ML. *ACM Transactions on Programming Languages and Systems*, 25(1):117–158, Jan. 2003.
25. J. C. Reynolds. *The Craft of Programming*. Prentice-Hall, 1981.
26. J. C. Reynolds. Separation logic: a logic for shared mutable data structures. In *IEEE Logic in Computer Science (LICS)*, pages 55–74, 2002.
27. A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communications*, 21(1):5–19, Jan. 2003.
28. A. Sabelfeld and D. Sands. A per model of secure information flow in sequential programs. *Higher-order and Symbolic Computation*, 14(1):59–91, 2001.
29. A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *IEEE Computer Security Foundations Workshop (CSFW)*, 2005.
30. T. Terauchi and A. Aiken. Secure information flow as a safety problem. In *12th International Static Analysis Symposium (SAS)*, volume 3672 of *LNCS*, pages 352–367. Springer, 2005.
31. D. Volpano and G. Smith. A type-based approach to program security. In *Proceedings of TAPSOFT'97*, volume 1214 in *LNCS*, pages 607–621. Springer, 1997.
32. H. Yang. Relational separation logic. *Theoretical Comput. Sci.*, 2004. To appear.

A Linear Logic of Authorization and Knowledge[*]

Deepak Garg, Lujo Bauer, Kevin D. Bowers,
Frank Pfenning, and Michael K. Reiter

Carnegie Mellon University

Abstract. We propose a logic for specifying security policies at a very high level of abstraction. The logic accommodates the subjective nature of affirmations for authorization and knowledge without compromising the objective nature of logical inference. In order to accurately model consumable authorizations and resources, we construct our logic as a modal enrichment of linear logic. We show that the logic satisfies cut elimination, which is a proof-theoretic expression of its soundness. We also demonstrate that the logic is amenable to meta-reasoning about specifications expressed in it through several examples.

1 Introduction

In this paper we develop a logic for specifying security properties of distributed systems at a very high level of abstraction. One of the difficulties in this domain is that security specifications, by nature, depend on individuals' intent as well as their state of knowledge. In addition to logical inference regarding the *truth* of propositions, we therefore also need to reason with *affirmations* of principals (to express intent), and *knowledge* of principals. In addition, we often need to capture changes of state, such as transfer of money or goods, which is most easily expressed in linear logic. We therefore arrive at a linear logic with additional modal operators for affirmation and knowledge, indexed by principals. We believe the combination of linearity with modalities such as affirmation and knowledge is an original contribution of this paper with some new insights, such as how to model possession of consumable resources as linear knowledge, or single-use authorizations as linear affirmations.

We show that our logic satisfies cut elimination, which is a proof-theoretic expression of its soundness. Moreover, the cut elimination theorem shows that the various components of the logic are orthogonal and, for example, there is a coherent subsystem containing only affirmations and not knowledge. We illustrate our logic through two examples. The first concerns a student registration system and demonstrates how to represent use-once authorizations, but avoids the use of knowledge. The second is a specification of monetary instruments which employs affirmations expressing authorization and knowledge to model possession of resources. In both examples we show how to exploit the formal foundation

[*] This work is supported by grants N0014-04-1-0724 of the Office of Naval Research, DAAD19-02-1-0389 of the Army Research Office and CNS-0433540 of the National Science Foundation.

of our logic in order to reason about properties of specifications expressed in it. In the student registration example we show that various constraints, such as the maximal number of credits a student can sign up for, are respected. In the monetary examples, we verify balance conditions on the total amount of money in the bank or under the control of principals. These meta-theoretic analyses rely again on cut elimination and a somewhat deeper property, namely completeness of focusing.

While beyond the scope of the present paper, some prior work on proof-carrying authorization [7,8] suggests that fragments of our logic would be a suitable basis for policy enforcement in a distributed architecture.

We now present the logic in two steps, first reviewing affirmation as developed in [14] in the new context of linearity. We then add possession and knowledge, followed by a brief sketch of the meta-theory of our logic and the examples. We conclude with additional related work and future plans.

2 A Constructive Linear Logic of Affirmation

When designing a new specification logic, we have to consider the range of properties we would like to express. First, we realize that we need standard logical connectives such as conjunction and implication. These are concerned with the truth of propositions, which is therefore our first judgment, $A\ \mathsf{true}$. We also need reasoning from hypotheses, so our basic judgment form is a hypothetical judgment. One might say that this constitutes the objective part of the logic since everyone agrees on the laws of logical reasoning and therefore on the meaning of the connectives.

Second, we need a way for principals to express their intent, such as who may access some information they have. We call this judgment *affirmation*, written as $K\ \mathsf{affirms}\ A$ (principal K affirms proposition A). The affirmation of a proposition does not imply its truth, otherwise everyone could give themselves access to any resource simply by affirming this. For example, a principal may affirm that she has access to a certain file on the disk. This is an expression of her intent; in truth she will not have access to the file unless the security policy allows it. On the other hand, if A is true then all principals are willing to affirm A since we assume principals are rational and can verify evidence for A.

In an implementation, we imagine affirmations can be established in two ways: cryptographically via certificates containing A signed by K, and logically via a deduction proving that A is true. The combination of signed certificates and logical proofs is the foundation of proof-carrying authorization [6,7].

We would like to go a step further and allow affirmations that may be used only once. For example, a personal check for \$10 from K made out to L can be seen as an affirmation that K is prepared to pay L the sum of \$10. However, this affirmation can be used by L only once; L cannot be allowed to cash the check multiple times. In logical terms this means that the certificate is *linear*, and that our logic will be an enrichment of linear logic. Linear logic is characterized by a linear hypothetical judgment where each linear assumption must be used exactly once and therefore represents a consumable resource. Of course, we also still need assumptions whose use is unrestricted for reusable certificates. From the point of view of linear logic, these assumptions are of the form A valid, because their proof cannot depend on any linear resources.

Putting these observations together, we obtain the judgment forms

$$\begin{gathered} \Gamma; \Delta \Longrightarrow A \ \mathsf{true} \\ \Gamma; \Delta \Longrightarrow K \ \mathsf{affirms}\ A \end{gathered}$$

where Δ consists of linear (use-once) hypotheses A true and Γ consists of unrestricted (reusable) hypotheses A valid. It turns out that we do not need to explicitly consider conclusions of the form A valid or hypotheses of the form K affirms A because they can always be eliminated.

The first set of rules just captures the nature of the hypothetical judgments. Because A true or A valid can always be inferred from their position in the sequent, we will generally abbreviate these judgments to just A. The short-hand γ stands for judgments we consider on the right, which are either A true or K affirms A.

$$\frac{}{\Gamma; A \Longrightarrow A}(\mathtt{init}) \qquad \frac{\Gamma, A; \Delta, A \Longrightarrow \gamma}{\Gamma, A; \Delta \Longrightarrow \gamma}(\mathtt{copy})$$

Next, the rules pertaining to the affirmation judgment. Because we do not consider hypotheses of the form K affirms A, there is only one rule which states that any principal K is prepared to affirm any true proposition.

$$\frac{\Gamma; \Delta \Longrightarrow A}{\Gamma; \Delta \Longrightarrow K \ \mathsf{affirms}\ A}(\mathsf{affirms})$$

Next, we internalize affirmation as a propositional modal operator so that we can combine affirmations with logical connectives such as implication. We write $\langle K \rangle A$ for the proposition that internalizes K affirms A. In a sequent calculus connectives are characterized by left and right rules.

$$\frac{\Gamma; \Delta \Longrightarrow K \ \mathsf{affirms}\ A}{\Gamma; \Delta \Longrightarrow \langle K \rangle A}(\langle\rangle R) \qquad \frac{\Gamma; \Delta, A \Longrightarrow K \ \mathsf{affirms}\ C}{\Gamma; \Delta, \langle K \rangle A \Longrightarrow K \ \mathsf{affirms}\ C}(\langle\rangle L)$$

While the right rule is straightforward, the left rule is key to understanding the modal nature of affirmation. Observe that in order to apply the left rule to $\langle K \rangle A$, the succedent of the sequent must be an affirmation by the same principal K. This means we can move from the truth of $\langle K \rangle A$ to the truth of A, but only if we are reasoning about the affirmations of K. This captures that K is rational.

3 Possession and Knowledge

The next step is to introduce knowledge into our logic. We are not aware of any attempts to combine epistemic logic with linear logic, so we believe this to be a contribution of this paper of independent interest.

One assumption commonly made about knowledge is that it is monotonic: we may learn more, for example, by inference, but we do not forget. Then what is "linear knowledge"? Returning to the usual interpretation of linear logic, we consider a linear assumption A true as a *resource* that may be consumed in a proof. Then linear knowledge is nothing but *possession of a resource* that may be consumed in a proof. We therefore write K has A for the new judgment of possession which is linear. It turns out we can always eliminate possession in the succedent of a sequent, so there is only one judgmental rule.

$$\frac{\Gamma; \Delta, A \Longrightarrow \gamma}{\Gamma; \Delta, K\ \mathsf{has}\ A \Longrightarrow \gamma}(\mathsf{has})$$

Informally, it states that if K possesses A then A may be used a resource. We have to take care, however, to make sure that other principals cannot steal the resource.

We internalize possession as a proposition, $[K]A$, expressing that it is true that K possesses A. The hypothetical nature of sequents means that a proposition $[K]A$ in the succedent expresses *potential possession*, where a proof corresponds to a plan to achieve this position. For example, the sequent

$$\cdot; K\ \mathsf{has}\ (B \multimap A), K\ \mathsf{has}\ B \Longrightarrow [K]A$$

could be read as: *If K has a means to transform a resource B into a resource A, and K also has resource B, then it is true that K can obtain resource A.* Of course, resources B and $B \multimap A$ would be consumed in the process of obtaining A, since those resources are linear.

The right rule expresses that, in order to show that K could obtain resource A, we have to show that resource A can be obtained using *only* the resources in Γ and Δ that K possesses *and no others*. This requires a restriction operator that removes from a context Δ all assumptions that are not of the form K has A, and similarly for the unrestricted context. In comparison, the left rule for $[K]A$ is straightforward, just transforming the proposition to the corresponding judgment.

$$\frac{\Gamma|_K; \Delta|_K \Longrightarrow A}{\Gamma; \Delta|_K \Longrightarrow [K]A}([]R) \qquad \frac{\Gamma; \Delta, K\ \mathsf{has}\ A \Longrightarrow \gamma}{\Gamma; \Delta, [K]A \Longrightarrow \gamma}([]L)$$

The restriction operator on linear assumptions[1] is defined formally as

$$\begin{array}{ll} (\cdot)|_K & = \cdot \\ (\Delta, K\ \mathsf{has}\ A)|_K & = \Delta|_K, K\ \mathsf{has}\ A \\ (\Delta, L\ \mathsf{has}\ A)|_K & = \Delta|_K \quad \text{for } L \neq K \\ (\Delta, A\,\mathsf{true})|_K & = \Delta|_K \end{array}$$

This means in the $[]R$ rule above, the linear context in the premise and the conclusion must contain only assumptions of the form K has B for the given K but possibly distinct B.

[1] The corresponding operator on unrestricted assumptions Γ is given below.

We also need more traditional knowledge of propositions, subject to unrestricted reuse. We write this judgment as K knows A. By its nature, it belongs in the context Γ. Again, knowledge is only required in assumptions so we have only one rule pertaining directly to the judgment. It allows us to infer the truth of A given K's knowledge of A. Of course, the converse must be prohibited.

$$\frac{\Gamma, K \text{ knows } A; \Delta, A \Longrightarrow \gamma}{\Gamma, K \text{ knows } A; \Delta \Longrightarrow \gamma}(\mathsf{knows})$$

Internalizing knowledge in the same way as possession, keeping in mind its unrestricted nature, we obtain the following two rules.

$$\frac{\Gamma|_K; \cdot \Longrightarrow A}{\Gamma; \cdot \Longrightarrow [\![K]\!]A}([\![]\!]R) \qquad \frac{\Gamma, K \text{ knows } A; \Delta \Longrightarrow \gamma}{\Gamma; \Delta, [\![K]\!]A \Longrightarrow \gamma}([\![]\!]L)$$

The first expresses that K can obtain knowledge of A if it follows by logical reasoning from the knowledge that K already has and no other assumptions. Formally, restriction is defined as follows.

$$\begin{array}{ll}
(\cdot)|_K & = \cdot \\
(\Gamma, K \text{ knows } A)|_K & = \Gamma|_K, K \text{ knows } A \\
(\Gamma, L \text{ knows } A)|_K & = \Gamma|_K \quad \text{for } L \neq K \\
(\Gamma, A\,\text{valid})|_K & = \Gamma|_K
\end{array}$$

This concludes our introduction to the basic logic, omitting only the standard connectives, both linear and non-linear. A description of the these may be found in [11]. We assume that the first-order universal quantifier is included because we need it to encode the examples in section 5. All results of the next section extend to the first-order case easily.

4 Cut Elimination

In a sequent calculus the connectives are explained via their right and left rules. Since propositions are always decomposed in such rules when read from the conclusion to the premises, we are justified in saying that the meaning of propositions is determined by their proofs, but only if the underlying interpretation of the sequent as a hypothetical judgment is respected. This is the contents of two important theorems: the admissibility of cut and the identity principle. Admissibility of cut expresses that we can always eliminate an assumption $A\,\mathsf{true}$ if we can supply a proof of A true. The identity principle states that we only need the (`init`) rule $\Gamma; A \Longrightarrow A$ for the case where A is atomic. To prove these we need to state them in a more general form to account for the other judgments in our logic. Other properties, such as weakening and contraction for the unrestricted context are immediate and we don't state them explicitly.

Theorem 1 (Admissibility of cut).

1. *If* $\Gamma; \Delta \Longrightarrow A$ *and* $\Gamma; \Delta', A \Longrightarrow \gamma$ *then* $\Gamma; \Delta', \Delta \Longrightarrow \gamma$.
2. *If* $\Gamma; \cdot \Longrightarrow A$ *and* $\Gamma, A; \Delta' \Longrightarrow \gamma$ *then* $\Gamma; \Delta' \Longrightarrow \gamma$.
3. *If* $\Gamma; \Delta \Longrightarrow K \textsf{ affirms } A$ *and* $\Gamma; \Delta', A \Longrightarrow K \textsf{ affirms } C$ *then* $\Gamma; \Delta, \Delta' \Longrightarrow K \textsf{ affirms } C$.
4. *If* $\Gamma|_K; \Delta|_K \Longrightarrow A$ *and* $\Gamma; \Delta', K \textsf{ has } A \Longrightarrow \gamma$ *then* $\Gamma; \Delta', \Delta|_K \Longrightarrow \gamma$.
5. *If* $\Gamma|_K; \cdot \Longrightarrow A$ *and* $\Gamma, K \textsf{ knows } A; \Delta' \Longrightarrow \gamma$ *then* $\Gamma; \Delta' \Longrightarrow \gamma$.

Proof. By nested induction, first on the structure of the cut formula A and then on the size of the two given derivations, as in [18,11].

Theorem 2 (Identity). *In the sequent calculus where initial sequents are restricted to atomic propositions,* $\Gamma; A \Longrightarrow A$ *for any proposition* A.

Proof. By induction on the structure of A.

5 Examples and Reasoning About Policies

We present two examples of security policies expressed in our logic using linear authorization and knowledge. The first one uses linear authorizations to describe a university course registration system. In the second example, we use linear knowledge and authorizations to represent a system of monetary instruments like checks, promissory notes and bank accounts. We reason about interesting properties of these systems (like correctness with respect to a given specification) using the logic. Some of the methods used for reasoning can be generalized beyond these examples. We return to this point briefly at the end of the section.

5.1 Course Registration

This example describes a university registration system using linear authorizations. Some of the authorizations in this example may be replaced by linear possessions, but we do not do this to keep the example simple. We assume two main principals: a `calendar` which authorizes free time slots available for students, and a `registrar` who controls the entire registration process. The following table lists the predicates we use along with their intuitive meanings.

$\texttt{slot}(S, T)$	Student S is free during time slot T
$\texttt{credits}(S, R)$	Student S may register for R more credits in the semester
$\texttt{registered}(S, C, R, T)$	Student S is registered in course C for R credits in time slot T
$\texttt{seats}(C, N)$	There are N more seats available in course C
$\texttt{course}(C, R, T)$	Course C is worth R credits and runs in time slot T

We wish to enforce three conditions during registration:

1. No student registers for more than a stipulated number of credits.
2. A student does not register for two courses that use the same time slot.
3. A maximum registration limit for each course is respected.

In the logic, linear authorizations are represented as assumptions of the form $\langle K\rangle A$ in the linear context $\varDelta$. Authorizations meant for unrestricted use are represented as assumptions of the same form in the context $\varGamma$. In an implementation, these assumptions are substituted by certificates signed by the authorizing principals.

We assume that at the beginning of the semester a number of certificates are issued by the registrar and the calendar, i.e., we assume that a number of authorization assumptions are present in our context when we start reasoning. These are the following. For each student S there is one linear certificate of the form $\langle\texttt{registrar}\rangle\texttt{credits}(S, R)$ issued by the registrar. This certificate mentions the maximum number of credits R that the student is permitted to take during the semester. For each possible time slot T, each student S gets one certificate of the form $\langle\texttt{calendar}\rangle\texttt{slot}(S, T)$ from the calendar. This entitles the student to register for some course in time slot T.

For each course C, the registrar issues a linear certificate of the form $\langle\texttt{registrar}\rangle\texttt{seats}(C, N)$, that specifies the number of seats N in the course. The registrar also issues one *unrestricted* certificate for each course C. This certificate specifies the number of credits R the course is worth, and the time slot T in which it runs. It has the form $\langle\texttt{registrar}\rangle\texttt{course}(C, R, T)$.

Now we state the policy rule governing registration in courses. The rule is universally quantified over the terms S, N, R, R', C and T.

$$\begin{aligned}
\texttt{reg} : {} & \langle\texttt{registrar}\rangle\texttt{course}(C, R, T) \supset \\
& \langle\texttt{registrar}\rangle\texttt{seats}(C, N) \multimap \\
& \langle\texttt{registrar}\rangle\texttt{credits}(S, R') \multimap \\
& (N \geq 1) \supset (R' \geq R) \supset \\
& \langle\texttt{calendar}\rangle\texttt{slot}(S, T) \multimap \\
& \quad (\langle\texttt{registrar}\rangle\texttt{registered}(S, C, R, T) \otimes \\
& \quad \langle\texttt{registrar}\rangle\texttt{credits}(S, R' - R) \otimes \\
& \quad \langle\texttt{registrar}\rangle\texttt{seats}(C, N - 1))
\end{aligned}$$

Intuitively, this rule says the following: if course C, worth R credits and running in time slot T, has at least one seat available, and student S can register for at least R more credits during the semester and is free during time slot T, then S may register for the course C. The rule consumes the credential $\langle\texttt{registrar}\rangle\texttt{credits}(S, R')$, replacing it with a similar credential that decrements R' by the number R of credits that the course is worth. This enforces condition (1) above. The rule also consumes S's time slot credential corresponding to the course's time slot T to prevent her from registering in another course that runs in the same slot. This enforces condition (2). Condition (3) is enforced because the rule replaces the credential $\langle\texttt{registrar}\rangle\texttt{seats}(C, N)$ with $\langle\texttt{registrar}\rangle\texttt{seats}(C, N - 1)$. This reduces the number of seats available in the course by one.

Observe that there is no condition in the rule that represents *intent* of student S to register for course C. This is because we are interested in expressing only the security aspects of the system in the logic. If this rule were to be implemented, say using a protocol, it would be necessary to ensure that the rule is used only when student S is actually willing to register for the course C.

Atomicity in policy implementation. In any realistic implementation of the above policy, it is essential that the policy rule `reg` be used *atomically*, i.e., in any application of the rule, all its pre-conditions be satisfied simultaneously.[2] This is significant due to linearity, because proving some pre-conditions of the rule may utilize linear resources, and a partial application of the rule may result in a situation where linear resources are incorrectly consumed.

A simple example illustrates this point. Suppose a student S wishes to register for course C worth R credits in time slot T and the following hold: (a) the course has a seat (there is a certificate $\langle \texttt{registrar} \rangle \texttt{seats}(C, N)$, where $N \geq 1$), (b) the student S has sufficient number of available credits (she has a certificate $\langle \texttt{registrar} \rangle \texttt{credits}(S, R')$ where $R' \geq R$), and (c) S does *not* have the required time slot certificate $\langle \texttt{calendar} \rangle \texttt{slot}(S, T)$. If we were to permit non-atomic use of the policy rule, we could consume the linear certificates mentioned in (a) and (b) to conclude the following:

$$\begin{aligned} &\langle \texttt{calendar} \rangle \texttt{slot}(S, T) \multimap \\ &\quad (\langle \texttt{registrar} \rangle \texttt{registered}(S, C, R, T) \otimes \\ &\quad \langle \texttt{registrar} \rangle \texttt{credits}(S, R' - R) \otimes \\ &\quad \langle \texttt{registrar} \rangle \texttt{seats}(C, N - 1)) \end{aligned}$$

However, since the student does not have the required time slot certificate $\langle \texttt{calendar} \rangle \texttt{slot}(S, T)$, we cannot proceed any further. At this point, the system is stuck in an inconsistent state because two linear certificates mentioning the number of seats in course C and student S's available credits have been consumed. No student can register in course C now, and S cannot register for any other course. Thus it is essential that the policy rule above (and in general, any policy rule that uses linear resources), be enforced atomically in an implementation.

Atomicity in the logic. If we can enforce atomicity of policy rules at the level of the logic itself, we can reason more faithfully about the consequences of policies using the logic. In particular, we can prove useful invariance properties of the system as it evolves under a particular policy. Since atomicity is an artifact of the implementation, the method used to enforce it in the logic should not affect logical consequence or provability in the logic. One such method is focusing [5], which is a proof search technique that combines a number of inference steps atomically without affecting provability. A detailed description of focusing is beyond the scope of this paper. In the following we present it only to the extent that is appropriate for our purpose. We convert each policy rule into a derived inference rule, which we add to the logic. Atomicity is forced implicitly by the inference rule. For example, the rule `reg` can be converted to the following inference rule.

[2] The pre-conditions of a rule $A_1 \multimap \ldots A_n \multimap B$ are $A_1, \ldots, A_n$.

$$\frac{\begin{array}{c}\Gamma; \cdot \Longrightarrow \langle\texttt{registrar}\rangle\texttt{course}(C, R, T) \\ \Gamma; \Delta_1 \Longrightarrow \langle\texttt{registrar}\rangle\texttt{seats}(C, N) \\ \Gamma; \Delta_2 \Longrightarrow \langle\texttt{registrar}\rangle\texttt{credits}(S, R') \\ \Gamma; \cdot \Longrightarrow N \geq 1 \\ \Gamma; \cdot \Longrightarrow R' \geq R \\ \Gamma; \Delta_3 \Longrightarrow \langle\texttt{calendar}\rangle\texttt{slot}(S, T) \\ \begin{array}{l}\Gamma;\ \Delta_4, \langle\texttt{registrar}\rangle\texttt{registered}(S, C, R, T), \\ \quad \langle\texttt{registrar}\rangle\texttt{credits}(S, R' - R), \langle\texttt{registrar}\rangle\texttt{seats}(C, N-1)\end{array} \Longrightarrow \gamma\end{array}}{\Gamma; \Delta_1\Delta_2\Delta_3\Delta_4 \Longrightarrow \gamma}\ (\texttt{reg})$$

Read bottom up, the rule says that we can conclude γ using the linear resources $\Delta_1\Delta_2\Delta_3\Delta_4$, if we can prove the preconditions of the rule `reg` using Δ_1, Δ_2 and Δ_3, and use Δ_4 and the three authorizations produced by `reg` to prove γ. This is exactly the behavior of the rule `reg` if it were implemented atomically.

System states and steps. We now formalize a notion of system state and state transition (step) for our example. Once we have these definitions we can prove that certain properties of system states are invariant under steps. These properties can be used to establish that the policy satisfies the three conditions mentioned at the beginning of the example. Informally, a state of the system is a pair of contexts $\Gamma; \Delta$ that contains only those authorizations that are relevant to our example.

Definition 2 (State). *A state of the system is a pair of contexts $\Gamma; \Delta$, satisfying the following conditions.*

1. *All assumptions in Γ have the form $\langle\texttt{registrar}\rangle\texttt{course}(C, R, T)$.*
2. *All assumptions in Δ have one of the forms $\langle\texttt{registrar}\rangle\texttt{credits}(S, R)$, $\langle\texttt{calendar}\rangle\texttt{slot}(S, T)$, $\langle\texttt{registrar}\rangle\texttt{registered}(S, C, R, T)$ or $\langle\texttt{registrar}\rangle\texttt{seats}(C, N)$.*
3. *For each student S, Δ has exactly one assumption of the form $\langle\texttt{registrar}\rangle\texttt{credits}(S, R)$.*
4. *For each student S and each time slot T, there is at most one assumption of* one *of the following forms in Δ: $\langle\texttt{calendar}\rangle\texttt{slot}(S, T)$ and $\langle\texttt{registrar}\rangle\texttt{registered}(S, C, R, T)$.*
5. *For each course C, there is exactly one assumption of the form $\langle\texttt{registrar}\rangle\texttt{course}(C, R, T)$ in Γ and one assumption of the form $\langle\texttt{registrar}\rangle\texttt{seats}(C, N)$ in Δ.*
6. *For each student S and each course C, there is at most one assumption of the form $\langle\texttt{registrar}\rangle\texttt{registered}(S, C, R, T)$ in Δ.*

Next, we define a state change, or step of the system. This notion is closely related to a similar idea from multi-set rewriting [10].

Definition 3 (Step). *We say that the pair of contexts $\Gamma; \Delta$ steps to the pair $\Gamma'; \Delta'$ (written $\Gamma; \Delta \longrightarrow \Gamma'; \Delta'$), if there is a derivation of $\Gamma; \Delta \Longrightarrow \gamma$ from*

$\Gamma; \Delta' \Longrightarrow \gamma$ that is parametric *in the conclusion γ, i.e., there is a derivation of the following form that is correct for every γ.*

$$\begin{array}{c}\Gamma'; \Delta' \Longrightarrow \gamma \\ \vdots \\ \Gamma; \Delta \Longrightarrow \gamma\end{array}$$

By definition, $\longrightarrow$ is a transitive relation and $\longrightarrow^* = \longrightarrow$. The following lemma characterizes $\longrightarrow$ in terms of smaller inferences.

Lemma 1 (Characterization of steps). *Let $\Gamma; \Delta$ be a state, i.e., it satisfies all conditions in definition 2. Suppose $\Gamma; \Delta \longrightarrow \Gamma'; \Delta'$. Then $\Gamma = \Gamma'$ and the corresponding derivation from definition 3 must have the following form (for some $n \geq 0$).*

$$\begin{array}{c}\dfrac{\ldots \qquad \Gamma; \Delta' \Longrightarrow \gamma}{\Gamma; \Delta_n \Longrightarrow \gamma}\,(\mathtt{reg}) \\ \vdots \\ \dfrac{\ldots \qquad \Gamma; \Delta_1 \Longrightarrow \gamma}{\Gamma; \Delta \Longrightarrow \gamma}\,(\mathtt{reg})\end{array}$$

Further, the pairs $\Gamma; \Delta_i$ and $\Gamma; \Delta'$ are states of the system, i.e., they satisfy the conditions in definition 2.

Proof. The proof of this lemma follows by observing that if we reason backwards from the sequent $\Gamma; \Delta \Longrightarrow \gamma$, then the only rule that applies is `reg`. This is because γ is parametric (so no right rule applies). Further, we assumed that $\Gamma; \Delta$ is a state, and hence the only assumptions are of the form $\langle K \rangle A$, and so no left rule applies either because the form of γ is unknown. This argument can now be repeated. □

Lemma 1 provides us an induction principle for reasoning with steps. We can induct on the number of (`reg`) rules in the derivation mentioned in the lemma. Using this method we can prove the following correctness proposition.

Property 1 (Correctness of the policy). *Suppose $\Gamma; \Delta$ is a state and $\Gamma; \Delta \longrightarrow \Gamma; \Delta'$. Then the following hold.*

1. *For each student S, the sum of all credits R in authorizations of the form $\langle\mathtt{registrar}\rangle\mathtt{credits}(S, R)$ and $\langle\mathtt{registrar}\rangle\mathtt{registered}(S, C, R, T)$ in the context Δ is the same the corresponding sum in Δ'.*
2. *For every time slot T, and every student S, there is at most one authorization of the form $\langle\mathtt{registrar}\rangle\mathtt{registered}(S, C, R, T)$ in Δ and Δ'.*
3. *For each course C, the sum of N in the (unique) certificate of the form $\langle\mathtt{registrar}\rangle\mathtt{seats}(C, N)$ and the number of certificates of the form $\langle\mathtt{registrar}\rangle\mathtt{registered}(S, C, R, T)$ in the context Δ is the same as the corresponding sum in Δ'.*

Proof. (1) and (3) follow by induction on the number of (`reg`) rules in the derivation corresponding to $\Gamma; \Delta \longrightarrow \Gamma; \Delta'$ from lemma 1. (2) follows from the fact that $\Gamma; \Delta'$ must be a state (lemma 1) and that each state satisfies clause 4 of definition 2. □

Observe that if we start from a state $\Gamma; \Delta$ which has no assumptions of the form $\langle\texttt{registrar}\rangle\texttt{registered}(S, C, R, T)$, then the three statements of the above proposition imply the three correctness conditions mentioned at the beginning of the example for the state $\Gamma; \Delta'$. This formally proves that the policy implemented by the rule `reg` is correct with respect to those conditions.

5.2 Monetary Instruments

We describe a monetary system involving bank accounts, checks, promissory notes and tradeable items in our logic. In addition to linear authorizations, this example uses linear possessions to represent various monetary resources in the hands of principals. We use an approach similar to the previous example. First we describe the system in our logic using specific predicates and policy rules. Then we convert the policy rules to inference rules in order to force atomicity. Next, we define a notion of state and step for the system. Finally, we characterize steps in a manner analogous to lemma 1, and use this to prove two properties of the monetary system. The first property says that the total amount of money in the system remains unchanged as the system evolves. The second property says that the net assets of every principal remain constant.

We assume the existence of at least two principals, the `bank` and a credit company `cc`. We assume that every principal has an account in the bank. We represent account balances of principals as assumptions of linear possession: the assumption $\texttt{bank}\ \mathsf{has}\ \texttt{balance}(K, N)$ represents the fact that K has N dollars in her bank account. In particular, the bank maintains its own account. This is represented as $\texttt{bank}\ \mathsf{has}\ \texttt{balance}(\texttt{bank}, N)$.

A check for amount N is represented by the proposition $\texttt{check}(N)$.[3] A check is useful only when signed by a principal, who promises to pay the corresponding amount to its bearer, and possessed by some other principal who can use it. A check for N dollars signed by K, and possessed by K' is represented as $K'\ \mathsf{has}\ \langle K\rangle\texttt{check}(N)$. Observe the difference in the use of affirmation and knowledge here: K's signature on the check is represented using an affirmation, which corresponds realistically to the fact that the check is an intent of payment made by K, whereas the fact that K' holds the check is represented using linear possession.

A promissory note of amount N signed by principal K and possessed by principal K' is represented by the assumption $K'\ \mathsf{has}\ \langle K\rangle\texttt{iou}(N)$. The difference between a promissory note and check is that a check can be cashed at a bank whereas a promissory note cannot be. We assume that the credit company holds one promissory note of the form $\texttt{cc}\ \mathsf{has}\ \langle K\rangle\texttt{iou}(N)$, for each principal K. The

[3] In order to keep the representation simple, we do not write the beneficiary of the check as an explicit argument in the predicate `check`.

$$\begin{aligned}
\texttt{ax1} : {} & [L]\langle K\rangle\texttt{check}(N) \multimap [\texttt{bank}]\texttt{balance}(K, N_1) \multimap \\
& [\texttt{bank}]\texttt{balance}(L, N_2) \multimap (N_1 \geq N) \supset \\
& \quad ([\texttt{bank}]\texttt{balance}(K, N_1 - N) \otimes [\texttt{bank}]\texttt{balance}(L, N_2 + N))
\end{aligned}$$

$$\begin{aligned}
\texttt{ax2} : {} & [\texttt{bank}]\texttt{balance}(K, N_1) \multimap [\texttt{bank}]\texttt{balance}(\texttt{bank}, N_2) \multimap (N_1 \geq N) \supset \\
& \quad ([\texttt{bank}]\texttt{balance}(K, N_1 - N) \otimes \\
& \quad\; [K]\langle\texttt{bank}\rangle\texttt{check}(N) \otimes \\
& \quad\; [\texttt{bank}]\texttt{balance}(\texttt{bank}, N_2 + N))
\end{aligned}$$

$$\texttt{ax3} : [\texttt{cc}]\langle K\rangle\texttt{iou}(N) \multimap ([\texttt{cc}]\langle K\rangle\texttt{iou}(N + N') \;\otimes\; [K]\langle\texttt{cc}\rangle\texttt{check}(N'))$$

$$\texttt{ax4} : [L]\langle M\rangle\texttt{check}(N) \multimap [K]\texttt{item}(N) \multimap ([L]\texttt{item}(N) \;\otimes\; [K]\langle M\rangle\texttt{check}(N))$$

$$\begin{aligned}
\texttt{ax5} : {} & [K]\langle L\rangle\texttt{check}(N) \multimap [\texttt{cc}]\langle K\rangle\texttt{iou}(N') \multimap (N' \geq N) \supset \\
& \quad ([\texttt{cc}]\langle K\rangle\texttt{iou}(N' - N) \;\otimes\; [\texttt{cc}]\langle L\rangle\texttt{check}(N))
\end{aligned}$$

$$\texttt{ax6} : [K]\langle K\rangle\texttt{check}(N)$$

Fig. 1. Rules for the monetary system in section 5.2

amount N may be zero if K does not owe the credit company anything. Finally, we have tradeable items in the system. An item of value N possessed by K is represented as K has $\texttt{item}(N)$.

Various possible transactions in the system are represented by the rules ax1-ax6 shown in figure 1. These rules are universally quantified over terms in uppercase. (ax1) says that if a principal L has a check for amount N signed by K, she may take it to the bank and get it cashed. In the process, the bank increments L's balance by N and decrements K's balance by the same amount. (ax2) permits a principal K having account balance at least N to obtain a banker's check for that amount. The bank transfers the amount N from K's account to its own and gives K a signed check for the same amount. (ax3) says that the credit company is willing to sign checks for principals by taking promissory notes from them.

If a principal K possesses an item worth N, she can sell it to L, provided L can produce a check for the same amount. This is represented by (ax4). The check moves from L to K during the transaction. (ax5) permits a principal K to pay the credit company using a check. (ax6) says that any principal K may sign a check for any amount N.

As with our last example, we convert each of these policy rules to an inference rule, which we add to our logic. For brevity, we omit an explicit description of the rules. We proceed to define the notion of state and step for our system. In this example there are no unrestricted resources; therefore we omit the context Γ from our definitions.

Definition 4 (State). *A state of the system is a context Δ satisfying the following conditions.*

1. *Δ contains assumptions of the following forms only: (a)* `bank has balance` (K,N)*, (b)* `cc has` $\langle K\rangle$`iou`(N)*, (c)* K `has` $\langle L\rangle$`check`(N)*, and (d)* K `has` `item`(N)*.*
2. *For each principal K, there is exactly one assumption of each of the following forms in Δ:* `bank has balance`(K,N)*, and* `cc has` $\langle K\rangle$`iou`(N)*.*

The definition of a transition step is similar to that in the previous example.

Definition 5 (Step). *We say that the context Δ steps to Δ' (written $\Delta \longrightarrow \Delta'$) if there is a derivation of $\cdot;\Delta \Longrightarrow \gamma$ from the assumption $\cdot;\Delta' \Longrightarrow \gamma$ that is parametric in the conclusion γ.*

Now we state a characterization lemma for steps, similar to lemma 1.

Lemma 2 (Characterization of steps). *Suppose Δ is a state, i.e., it satisfies the conditions in definition 4. Let $\Delta \longrightarrow \Delta'$. Then the corresponding derivation from definition 5 has the following form, where each rule marked $(*)$ is an inference rule derived from one of the policy rules* (`ax1`)-(`ax6`)*.*

$$\dfrac{\dots \qquad \cdot;\Delta' \Longrightarrow \gamma}{\cdot;\Delta_n \Longrightarrow \gamma}\,(*)$$

$$\vdots$$

$$\dfrac{\dots \qquad \cdot;\Delta_1 \Longrightarrow \gamma}{\cdot;\Delta \Longrightarrow \gamma}\,(*)$$

Further, Δ_i and Δ' are states of the system, i.e., they satisfy the conditions in definition 4.

As before, this lemma gives us an induction principle for steps. Using that we can prove the property shown below. The first statement in the property says that the total amount of money in the system (as measured by the sum of the bank balances of all principals) remains constant. The second statement says that the net assets of every principal remain the same as the system evolves.

Property 2 (Consistency). *Let Δ be a state, and $\Delta \longrightarrow \Delta'$. Then the following hold.*

1. *The sum of all N such that* `bank has balance`(K,N) *exists in Δ, is the same as the corresponding sum for Δ'.*
2. *For each principal K, the sum of amounts N in all assumptions of the form* `bank has balance`(K,N)*,* K `has` $\langle L\rangle$`check`(N) *and* K `has item`(N) minus *the sum of amounts N in assumptions of the form L* `has` $\langle K\rangle$`check`(N) *and* `cc has` $\langle K\rangle$`iou`(N) *is the same for both Δ and Δ'.*

Proof. By induction on the number of rules marked $(*)$ in the derivation corresponding to $\Delta \longrightarrow \Delta'$ from lemma 2. □

Generic description of atomicity and steps. In reasoning about the policies expressed in the two examples above, we used a number of similar concepts. In each case we had: (a) inference rules derived from policy rules to force atomicity, (b) notion of state, (c) notion of step, and (d) correctness conditions that are invariant across steps (properties 1 and 2). Of these, the definition of state and correctness are specific to a given policy and hard to generalize, but we can describe atomicity and step for all policies expressible in the logic in a generic manner by building a logic programming language based on our logic. We discuss this briefly.

Our enforcement of atomicity using inference rules is based on focusing. The notion of step relates quite naturally to the idea of forward chaining from proof search. Focusing and forward chaining can be combined systematically to build a logic programming language based on our logic. In such a language, proof search would proceed through interleaving phases of goal directed backward search and forward chaining. For policies expressed in the language, the notions of atomicity and step arise from the semantics of proof search. Technically, such a language requires a new lax modality to separate the two phases of proof search. Prior experience with the language LolliMon [17] suggests that this is feasible and can be implemented in practice.

6 Related Work

Our logic combines three major concepts: affirmation, knowledge, and linearity. As far as we are aware, this is the first time that a linear logic of affirmation and knowledge has been proposed and investigated. From the security perspective, affirmations are used for authorization, knowledge to specify the intended flow of information, linear affirmations for use-once credentials, and linear knowledge for possession of consumable resources. In prior work, two of the authors have developed the (non-linear) logic of affirmations for authorization [14], which is a small fragment of what is presented here. The use of linearity for single-use credentials together with an enforcement mechanism was first proposed by four of the authors [8], but the underlying logic was not fully developed and its properties not investigated. Furthermore, this logic was lacking a treatment of possession and knowledge.

The study of authorization by logical means was initiated by Abadi et al. [4]. Their logic was classical, presented in an axiomatic style and studied through a Kripke semantics. No proof-theory or meta-theoretic properties like cut elimination were described. Subsequently, a number of authorization logics have been studied and implemented [13,9,16,15,12,19]. None of these logics is linear or provides proof-theoretic explanation of the logical operators. Further pointers on this line of work can be found in a survey by Abadi [2], who has also recently reformulated a (non-linear) constructive authorization logic [1] based on DCC [3] which is quite similar to a fragment of the logic given here.

The concepts of possession and knowledge are related to the K-operator from epistemic logics (see [20] for a survey of epistemic logics), which describes knowledge held by individuals and is similar to our operator $[\![K]\!]A$. As far as we are aware, the study of epistemic logics and its operators has been restricted almost exclusively to the classical setting. Simultaneous use of knowledge and linearity described in this paper to represent resources held by principals also appears to be new.

We believe that the proof-carrying authorization (PCA) [6,7] architecture can be extended to implement policies expressed in some fragments of the proposed logic. The main new problem is enforcement of single use and atomicity of operations. Prior work in this direction indicates that contract signing protocols can be used for doing this effectively [8]. We do not believe it possible to implement the entire logic in a proof-carrying architecture because a proof of authorization may depend on private knowledge held by individuals, and verifying such a proof could result in a breach of security. Therefore, in order to effectively implement this logic using PCA, we have to restrict ourselves to certain fragments, for example, where authorizations made by any principal do not depend on knowledge held by others.

7 Conclusion

We have presented a new constructive linear logic that develops the logical concepts of affirmation and knowledge from judgmental principles. The logic yields a clean proof theory and an analysis of meanings of the connectives from proof rules. We have shown that the logic satisfies cut elimination and demonstrated that policies expressed in the logic are amenable to meta-theoretic analysis regarding their correctness. We believe that this logic is a good foundation for expressing policies involving linear authorizations and resources held by principals.

There are several avenues for future work besides those mentioned with the related work. One is to construct and implement a logic programming language based on this logic, as mentioned at the end of section 5. Another avenue for future work is to study non-interference properties for the logic, along the lines of [14]. These properties permit administrators to explore the consequences of their policies by showing that certain forms of assumptions cannot affect provability of certain other forms of conclusions. A third direction is to extend the logic with explicit constructs for distribution of knowledge on physical sites to reason about networked security systems at a slightly lower level of abstraction.

References

1. Martín Abadi. Personal communication.
2. Martín Abadi. Logic in access control. In *Proceedings of the 18th Annual Symposium on Logic in Computer Science (LICS'03)*, pages 228–233, Ottawa, Canada, June 2003. IEEE Computer Society Press.
3. Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In *Conference Record of the 26th Sympoisum on Principles Of Programming Languages (POPL'99)*, pages 147–160, San Antonio, Texas, January 1999. ACM Press.

4. Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, October 1993.
5. Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.
6. Andrew W. Appel and Edward W. Felten. Proof-carrying authentication. In G. Tsudik, editor, *Proceedings of the 6th Conference on Computer and Communications Security*, pages 52–62, Singapore, November 1999. ACM Press.
7. Lujo Bauer. *Access Control for the Web via Proof-Carrying Authorization.* PhD thesis, Princeton University, November 2003.
8. Lujo Bauer, Kevin D. Bowers, Frank Pfenning, and Michael K. Reiter. Consumable credentials in logic-based access control. Technical Report CMU-CYLAB-06-002, Carnegie Mellon University, February 2006.
9. Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. *ACM Trans. Inf. Syst. Secur.*, 6(1):71–127, 2003.
10. Stefano Bistarelli, Iliano Cervesato, Gabriele Lenzini, and Fabio Martinelli. Relating Multiset Rewriting and Process Algebras for Security Protocol Analysis. *Journal of Computer Security*, 13:3–47, February 2005.
11. Bor-Yuh Evan Chang, Kaustuv Chaudhuri, and Frank Pfenning. A judgmental analysis of linear logic. Submitted. Extended version available as Technical Report CMU-CS-03-131R, December 2003.
12. Jason Crampton, George Loizou, and Greg O' Shea. A logic of access control. *The Computer Journal*, 44(1):137–149, 2001.
13. John DeTreville. Binder, a logic-based security language. In M.Abadi and S.Bellovin, editors, *Proceedings of the 2002 Symposium on Security and Privacy (S&P'02)*, pages 105–113, Berkeley, California, May 2002. IEEE Computer Society Press.
14. Deepak Garg and Frank Pfenning. Non-interference in constructive authorization logic. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW 19)*. IEEE Computer Society Press, 2006. To appear.
15. Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Trans. Inf. Syst. Secur.*, 6(1):128–171, 2003.
16. Ninghui Li and John C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *PADL '03: Proceedings of the 5th International Symposium on Practical Aspects of Declarative Languages*, pages 58–73. Springer-Verlag, 2003.
17. Pablo López, Frank Pfenning, Jeff Polakow, and Kevin Watkins. Monadic concurrent linear logic programming. In *Proceedings of the 7th International Symposium on Principles and Practice of Declarative Programming (PPDP'05)*, Lisbon, Portugal, 2005.
18. Frank Pfenning. Structural cut elimination I. Intuitionistic and classical logic. *Information and Computation*, 157(1/2):84–141, March 2000.
19. Harald Rueß and Natarajan Shankar. Introducing Cyberlogic. In *Proceedings of the 3rd Annual High Confidence Software and Systems Conference*, Baltimore, Maryland, April 2003.
20. W. van der Hoek and R. Verbrugge. Epistemic logic: A survey. *Game Theory and Applications*, 8:53–94, 2002.

Prêt à Voter with Re-encryption Mixes

P Y A Ryan[1] and S A Schneider[2]

[1] University of Newcastle
[2] University of Surrey

Abstract. We present a number of enhancements to the voter verifiable election scheme Prêt à Voter [CRS05]. Firstly, we propose a mechanism for the distributed construction by a set of independent clerks of the ballot forms. This construction leads to proto-ballot forms with the candidate list encrypted and ensures that only a collusion of all the clerks could determine the cryptographic seeds or the onion/candidate list association. This eliminates the need to trust a single authority to keep this information secret. Furthermore, it allows the on-demand decryption and printing of the ballot forms, so eliminating chain of custody issues and the chain voting style attacks against encrypted receipt schemes identified in [RP05].

The ballot forms proposed here use ElGamal randomised encryption so enabling the use of re-encryption mixes for the anonymising tabulation phase in place of the decryption mixes. This has a number of advantages over the RSA decryption mixes used previously: tolerance against failure of any of the mix tellers, full mixing of terms over the Z_p^* space and enabling the mixes and audits to be fully independently rerun if necessary.

1 Introduction

The Prêt à Voter scheme, presented in [CRS05], is a cryptographic voting scheme that enables voter-verifiability: at the time of casting their vote, voters are provided with an encrypted receipt. They can then check, via a secure Web Bulletin Board (WBB), that their receipt is accurately included in a robust anonymising mix process. Various checking mechanisms serve to detect any corruption in any phase of this process: encryption of the vote, recording and transmission of the encrypted ballot receipt and the decryptions of the votes. Full details can be found in [CRS05]. Henceforth we will refer to this version of the scheme as Prêt à Voter'05.

Prêt à Voter seeks to achieve the goals of accuracy and ballot secrecy with minimal trust in the system: software, hardware, officials. Assurance is achieved through a high degree of transparency and we thus verify the correctness of the election rather that attempting to verify the system.

This scheme has the benefit of providing a very simple and familiar voter experience, but certain vulnerabilities and trust assumptions have been identified, see [RP05]. In this paper we present a number of enhancements designed to counter these threats and eliminate the need for these trust assumptions.

The construction of the ballot forms presented here also enables the use of re-encryption mixes in the anonymising/tabulation phase. This also provides a number of advantages over the RSA/decryption mixes of Prêt à Voter'05.

The structure of the paper is as follows: in the next section we give the key elements of Prêt à Voter'05. Section 3 summarises some of the threats to and trust assumptions needed in Prêt à Voter'05. Section 4 presents the distributed construction of encrypted ballot forms. Sections 5 and 6 describe how these forms can be used in the vote casting process. Section 7 describes the use of this construction for re-encryption mixes during the anonymising and tabulation phase. Sections 8 and 9 describe the new auditing procedures required for the new ElGamal style ballot forms. Sections 10 and 11 discuss some further extensions to deal with more general voting methods and remote voting.

2 Outline of Prêt à Voter 2005

We now present an overview of the Prêt à Voter voter-verifiable scheme. Voters select at random a ballot form, an example of which is shown in Figure 1.

Obelix	
Asterix	
Panoramix	
Idefix	
	$7rJ94K$

Fig. 1. Prêt à Voter ballot form

In the booth, the voter makes her selection in the usual way by placing a cross in the right hand column against the candidate of choice, or, in the case of a Single Transferable Vote (STV) system for example, they mark their ranking against the candidates. Once the selection has been marked, the left hand strip is detached and discarded. The remaining right hand strip now constitutes the receipt, as shown in Figure 2.

X
$7rJ94K$

Fig. 2. Prêt à Voter ballot receipt

The voter now exits the booth and casts their vote in the presence of an official. The ballot receipt is placed under an optical reader or similar device that records the random value at the bottom of the strip and an index value indicating the cell into which the X was marked. The receipt is digitally signed and franked and the voter now retains this as their receipt.

Possession of a receipt might appear to open up the possibility of coercion or vote-buying. However, the candidate lists on the ballot forms are independently randomised for each ballot form. Thus, with the left hand strip removed, the right hand strip alone does not indicate which way the vote was cast.

The cryptographic value printed on the bottom of the receipt, the 'onion', is the key to extraction of the vote. Buried cryptographically in this value is the information needed to reconstruct the candidate list shown on the left hand strip. This information is encrypted under the secret keys shared by a number of tellers. Thus, only the tellers acting in concert are able to reconstruct the candidate order and so interpret the vote value encoded on the receipt.

Once the election has closed, all the receipts are transmitted to a central tabulation server which posts them to a secure WBB. This is an append-only, publicly visible facility. Only the tabulation server, and later the tellers, can write to this and, once written, anything posted to it will remain unchanged. Voters can visit this WBB and confirm that their receipt appears correctly.

After a suitable period, the tellers take over and perform a robust, anonymising, decryption mix on the batch of posted receipts. Various approaches can be used to ensure that the tellers perform the decryptions correctly. Details of this can be found in [CRS05].

Prêt à Voter'05 proposes an Authority responsible for the generation of the entropy for the crypto seeds and prior printing of the ballot forms. Random auditing, by independent organisations, of the forms before, during and after the election serve to detect any attempt by the the Authority to pass off incorrectly formed ballot forms. Later in this paper we propose an alternative approach using on-demand creation and printing of forms and post-auditing.

The Prêt à Voter'05 approach has the advantage of simplicity and results in a very simple and familiar experience for the voters: they simply register, collect a form, mark their selection in the booth and then cast the form.

For full details of the mechanisms used in the 2005 version of the scheme to detect any malfunction or misbehaviour by the devices or processes that comprise the scheme, see [CRS05]. The construction of the ballot forms used here calls for rather different monitoring and auditing mechanisms that we detail later.

3 Threats and Trust Models

The simplicity of the original scheme, in particular the use of a single authority and the pre-printing and pre-auditing of the ballot forms, comes at a certain cost: various trust assumptions need to be made. In this section we briefly recall the threats and assumptions of Prêt à Voter'05 identified in [RP05].

3.1 The Need to Trust the Authority for Confidentiality

In Prêt à Voter 2005, a single entity creates the ballot forms. Whilst it is not necessary to trust this entity from the point of view of accuracy, it is necessary to trust it not to leak the ballot form information. Clearly, if the Authority were to leak this information, the scheme would become susceptible to coercion or vote buying.

3.2 Chain of Custody

Just as we need to trust the Authority not to leak ballot form information, we also need to assume that mechanisms are in place to ensure that none of this information is leaked during storage and distribution. Various counter-measures are possible: for example, ballot forms could be kept in sealed envelopes to be revealed only by the voters in the booth. Alternatively, a scratch card style mechanism along the lines suggested in [RP05] could be used to conceal the onion value until the voter reveals it at the time of vote casting. The ballot forms would also need to be stored and distributed in locked, sealed boxes. All of these counter-measures are rather procedural in nature and so require various trust assumptions.

3.3 Chain Voting

Conventional, pen and paper elections may be vulnerable to a style form of vote buying known as chain voting. The UK system in particular is vulnerable. Here, the ballot forms are a controlled resource: on entering the polling station, the voter is registered and marked off on the electoral roll. They are given a ballot form which they take to the booth, mark and then cast in the ballot box. In principle, officials should observe the voters casting their form.

The attack works as follows: the coercer smuggles a blank ballot form out of the polling station. The controls on the distribution of the forms should make this a little tricky, but in practise there are many ways it could be achieved. Having marked the form for the candidate of their choice, the coercer intercepts a voter as they enter the polling station. The voter is told that if, when they exit the polling station, they hand a fresh, blank form back to the coercer they will receive an reward. The attack can now proceed inductively until a voter decides to cry foul. Note that, once initialised, the controls on the ballot forms works in the coercer's favour: if the voter emerges from the polling station with a blank form, it is a strong indication that they did indeed cast the marked form they were given by the coercer.

3.4 Kleptographic Channels

A further, rather subtle vulnerability can occur where a single entity is responsible for creating cryptographic variables: kleptographic attacks as described in [YY96]. The possible relevance of such attacks to cryptographic voting schemes is described in [M. 06]. The idea is that the entity may carefully choose the values of the crypto variables in order to leak information to a colluding party.

In the case of Prêt à Voter, the Authority might choose the seed values in such a way that an agreed, keyed cryptographic hash of the onion value indicates the candidate order. Clearly this may require quite a bit of searching and computation to find suitable values. Note however that such an attack could pass unnoticed: the distribution of seed values would look perfectly random to anyone ignorant of the cryptographic hash function.

4 Distributed Generation of Encrypted Ballot Forms

Many of the above attacks stem from the fact that a single entity is able to determine, in the sense of being able both to know and to control, the seed values. We now present a mechanism for the distributed generation of the seed values and ballot forms. Throughout, we will use ElGamal encryption rather than RSA as used in Prêt à Voter'05 and we will work in Z_p^*, p a (large) prime.

An analogous construction is possible for the distributed creation of the RSA, layered onions of Prêt à Voter'05. However, as we want to introduce re-encryption mixes at the tabulation stage, we present the construction for ElGamal encryption here. We note also that the term *onion* is a slight misnomer where ElGamal terms are used but we will retain it here for historical reasons.

The ballot forms will be generated by a set of l clerks in such a way that each contributes to the entropy of the crypto seed and this remains encrypted throughout. Consequently the candidate list, which is derived from the seed, remains concealed and all the clerks would have to collude to determine the seeds values.

We assume a set of decryption tellers who hold the key shares for a threshold ElGamal primitive with public key: (p, α, β_T). These will act much as the tellers of the original scheme and will be responsible for the final decryption stage after the anonymising, re-encryption mix phase. Details of the anonymising and decryption/tabulation phases will be given in section 7.

We also assume a set of Registrars with threshold secret key shares corresponding to the public key: (p, α, β_R). These public keys are known to the Clerks and are used in the construction of the ballot forms.

An initial clerk C_0 generates a batch of initial seeds s_i^0. These seeds are drawn randomly from a binomial distribution centred around 0 with standard deviation σ. σ would probably be chosen to be of order n, the number of candidates.

From these, C_0 generates a batch of pairs of "entangled" onions by encrypting each s_i^0, actually in the form $\gamma^{-s_i^0}$, under the Registrar key and the Teller key:

$$(\{\gamma^{-s_i^0}\}_{PK_R}, \{\gamma^{-s_i^0}\}_{PK_T}).$$

Expressed as ElGamal encryptions these have the form:

$$(\alpha^{x_i^0}, \beta_R^{x_i^0}.\gamma^{-s_i^0}), (\alpha^{y_i^0}, \beta_T^{y_i^0}.\gamma^{-s_i^0})$$

for fresh random values x_i^0, y_i^0 drawn from Z_p^*.

Notice that, for convenience later, we have encrypted the value $\gamma^{-s_i^0}$ for some generator γ of Z_p^* rather than encrypting s_i^0 directly. The reason for this will become apparent shortly.

The remaining $l-1$ Clerks now perform re-encryption mixes and transformations on this batch of onion pairs. Each Clerk takes the batch of pairs output by the previous Clerk and performs a combined re-encryption along with an injection of fresh entropy into the seed values. For each pair of onions, the same entropy is injected into the seed value of both onions to ensure that these values continue to match for each pair.

More precisely, for each pair of the batch, the jth Clerk C_j generates a new, random values $\bar{x}, \bar{y}$ and $\bar{s}$ and performs the following mix/transformation on each onion pair of the batch:

$$\begin{gathered}
\{(\alpha^{x_i^{j-1}}, {\beta_R}^{x_i^{j-1}}.\gamma^{-s_i^{j-1}}), (\alpha^{y_i^{j-1}}, {\beta_T}^{y_i^{j-1}}.\gamma^{-s_i^{j-1}})\} \\
\downarrow \\
\{(\alpha^{x_i^{j-1}}.\alpha^{\bar{x}_i^j}, {\beta_R}^{x_i^{j-1}}.{\beta_R}^{\bar{x}_i^j}.\gamma^{-s_i^{j-1}}.\gamma^{-\bar{s}_i^j}), (\alpha^{y_i^{j-1}}.\alpha^{\bar{y}_i^j}, {\beta_R}^{y_i^{j-1}}.{\beta_R}^{\bar{y}_i^j}.\gamma^{-s_i^{j-1}}.\gamma^{-\bar{s}_i^j})\} \\
\downarrow \\
\{(\alpha^{(x_i^{j-1}+\bar{x}_i^j)}, {\beta_R}^{(x_i^{j-1}+\ \bar{x}_i^j)}.\gamma^{-(s_i^{j-1}+\bar{s}_i^j)}), (\alpha^{(y_i^{j-1}+\bar{y}_i^j)}, {\beta_R}^{(y_i^{j-1}+\ \bar{y}_i^j)}.\gamma^{-(s_i^{j-1}+\bar{s}_i^j)})\} \\
\downarrow \\
\{(\alpha^{x_i^j}, {\beta_R}^{x_i^j}.\gamma^{-s_i^j}), (\alpha^{y_i^j}, {\beta_T}^{y_i^j}.\gamma^{-s_i^j})\}
\end{gathered}$$

where

$$\begin{aligned}
x_i^j &= x_i^{j-1} + \bar{x}_i^j \\
y_i^j &= y_i^{j-1} + \bar{y}_i^j \\
s_i^j &= s_i^{j-1} + \bar{s}_i^j
\end{aligned}$$

The $\bar{x}, \bar{y}$ denote fresh random values drawn from from Z_p^* generated by the Clerk during the mix. Similarly the $\bar{s}$ values are freshly created random values except that these are again chosen randomly and independently with a binomial distribution mean 0 and standard deviation σ. Having transformed each onion pair in this way, the Clerk C_j then performs a secret shuffle on the batch and outputs the result to the next Clerk, C_{j+1}.

Thus, each Clerk performs a re-encryption mix along with the injection of further entropy into the seed values $\bar{s}$.

So the final output after $l-1$ mixes is a batch of pairs of onions of the form: $\{\{(\alpha^{x_i}, \beta_R^{x_i}.\gamma^{-s_i}), (\alpha^{y_i}, \beta_T^{y_i}.\gamma^{-s_i})\}$ where:

$$x_i = x_i^l \ , \ y_i = y_i^l \ , \ s_i = s_i^l$$

thus:

$$x_i = \Sigma_{i=1}^l \bar{x}^i$$

etc.

The final s_i values will have binomial distribution mean 0 and standard deviation $\sigma\surd(l)$.

We will refer to the first onion as the "Registrar onion" or "booth onion" and the second onion as the "Teller onion".

For each pair, assuming correct behaviour of the clerks, the s values in the two onions should match. We'll discuss mechanisms to detect corruption of the forms later. As the seed values, and hence the candidate orders, remain encrypted, none of clerks knows the seed values and only if they all acted in collusion could they determine the seed values. These "proto-ballot form" can now be stored and distributed in encrypted form, thus avoiding the chain of custody problems mentioned above. The seed values can now be revealed on demand by a threshold set of the Registrars.

5 On-Demand Creation of Ballot Forms

The above construction of the proto-ballot forms means that the ballot form material can be stored and distributed in encrypted form. Once registered at the polling station, voters are assigned at random one of these forms:

$onion_L$	$onion_R$

The voter proceeds to the booth in which they find a device that reads the left-hand onion. In the simplest case, the secret key to decrypt the left-hand onions could be held in the devices in the booths. Thus, the left hand onion could be decrypted in the booth, the seed value s revealed and the candidate order π derived as some agreed function of s. If lodging the keys in a single device is considered rather fragile, the left-hand onion could be encrypted under a threshold key held by a number of registrars. The onions could be transmitted to these registrars and a threshold set of these would then decrypt the onions and return the seed to the booth device.

The candidate list can now be printed by the device in the booth to give a standard Prêt à Voter ballot form:

Obelix	
Asterix	
Panoramix	
Idefix	
$onion_L$	$onion_R$

As an additional precaution, the left-hand onion might be separately destroyed.

The point of the paired onions is now clear: we arrange for the booth device to see only the left hand onion and so it will not know the association of the candidate list with the right hand, teller onion that will appear on the receipt. Various mechanisms are possible to ensure that the booth device does not see the right-hand onion. The scratch strip mechanism could be invoked here again for example: the right-hand onion would be covered by a scratch strip that would only be removed at the time of casting, or even at some time after casting. The voter only really needs to reveal the teller onion when they come to check their receipt on the WBB.

Strictly speaking, the lth clerk in collusion with the booth device could form the candidate list/onion association. Elaborations of the scheme to counter the threat of such collusion attacks are the subject of ongoing research.

6 Supervised Casting of a Ballot

The voter in the booth now has a "conventional" Prêt à Voter style ballot form with the candidate list and the associated right hand (teller) onion. His vote can now be cast in the usual way by marking an X against the candidate of their choice. The left hand strip is detached and discarded and the voter leaves the booth and casts their vote in the presence of an official exactly as described previously. Their receipt is recorded digitally as (r, onion), where r is the index value indicating the position of the X.

The receipt can be digitally signed and franked at this point to counter any receipt faking attacks.

Once the election has closed, copies of the digitised receipts will be posted to the WBB exactly as before and the voters can visit this and assure themselves that their receipt has been correctly registered. In addition to this, a Verified Encrypted Paper Audit Trail mechanism could be deployed: at the time of casting, an extra paper copy of the receipt is made and retained in a sealed audit box. This can be used to independently check the correspondence with the receipts posted to the WBB.

7 Re-encryption/Tabulation Mixes

Our construction leads to ElGamal onions which appear to be well suited to being put through re-encryption mixes. However, the form of the ballot receipts means that this is not quite straightforward: in addition to the onion term we have the index value, in the clear as it were. An obvious approach would be to send the receipt terms through the mix re-encrypting the onions whilst leaving the index values unchanged. The problem with this is that an adversary is able to partition the mix according to the index values. There may be situations in which this is acceptable, for example large elections in which the number of voters vastly exceeds the number of voting options. In general it seems rather unsatisfactory.

A more satisfactory solution, at least for the case of a simple selection of one candidate from the list, is described in this section. We will discuss how to achieve full mixing in the more general case in section 10.

In this case we restrict ourselves to just cyclic shifts from the base ordering of the candidate list from a base ordering. For single candidate choice elections, this is sufficient to ensure that the receipts do not reveal the voter's selection. For more general styles of election, in which for example voters are required to indicate a ranking of the candidates, we of course need to allow full permutations of the candidate list. Indeed, even in the case of single selection elections, it is preferable to allow full permutations in order to eliminate any possibility of a systematic corruption of votes. For this moment we discuss the approach of simple cyclic shifts.

Let s_i be the shift of the candidate list for the ith ballot form. We can absorb the index value r into the onion:

$$(\alpha^y, \beta_T^y.\gamma^{r-s_i})$$

This gives a pure ElGamal term and the value $r - s_i$ taken modulo n indicates the voter's the original candidate choice in the base ordering. These ElGamal terms can now be sent through a conventional re-encryption mix by a set of mix tellers, see for example [JJR02]. These mix tellers do not hold any secret keys but read in a batch of ElGamal terms from the WBB, re-encrypt each of them and then post the resulting terms in random order to the WBB. After an appropriate number of such anonymising re-encryption mixes, (a threshold set of) the decryption tellers take over to extract the plaintext values.

Thus, in contrast to the decryption mixes uses previously, the anonymising and decrypting phases are separated out in re-encryption mixes.

This will yield decrypted terms of the form:

$$\gamma^{r-s_i} \ (mod\, p).$$

Now we have to extract the values $r - s_i \ (mod\, n)$ to recover the original votes. The difficulty is that $r - s_i$ is the discrete log of γ^{r-s_i} in Z_p^* so in general, if the seed values had been drawn randomly from Z_p^*, computing this would be intractable. However, we have set things up so that the s values are drawn from a binomial distribution so we can search the space very efficiently. We could, for example, generate a look-up table for the logs out to some multiple of $\sigma\surd(l)$. Occasionally we will have an outlier that will require some search beyond the range of the look-up table.

7.1 Coercion Resistance and Plausible Deniability

The point of using a binomial distribution for the seed value is to ensure plausible deniability or coercion resistance whilst at the same time avoiding the discrete log problem. An alternative approach would be to bound the possible seed values generated by the clerks to lie in some fixed range, between $-M$ and $+M$ say. This would have the problem that occasionally we would hit situations in which final decrypted $r - s$ values would take on extreme values, e.g., $r - s = -M$. In this case, an adversary could deduce that r must have equalled 0 and so be able to link this vote value back to a subset of the receipts, i.e., receipts with the index value 0.

Using a distribution avoids such "edge effects" whilst avoiding our having to compute arbitrary discrete logs in Z_p^*. Arguably, the adversary would be able to assign a non-flat probability distribution to the possible r values, but as long as no values of r can ever be eliminated, plausible deniability will be maintained.

We should also observe that even if it were possible to link a vote back to a particular index value, this would not typically violate ballot secrecy unless this it so happened that this identified a unique receipt, i.e., there happened to be only one receipt with this r value.

8 Auditing the Ballot Forms

The mechanisms described above allow for the distributed generation of ballot forms and just-in-time decryption of the candidate list and printing of the ballot forms. This has clear advantages in terms of removing the need to trust a single entity to keep the ballot form information secret and avoiding chain of custody issues. On the other hand, it means that we can no longer use the random pre-auditing of pre-printed ballot forms as suggested in [CRS05]. Consequently, we must introduce alternative techniques to detect and deter any corruption or malfunction in the creation of the ballot forms.

A possible approach, in the supervised context at least, is to incorporate the two sided ballot form mechanism suggested in [Rya06] and re-introduce a cut-and-choose mechanism into the voter protocol. Here, a ballot form would be assigned two independent, entangled pairs of onions. One printed on one side of the form, the other on the flip side. In the booth, on each side, the left hand onion would be decrypted and the corresponding candidate list printed in the left hand column. The result is two independent ballot forms, one printed on each side, as illustrated in Figure 3.

Obelix		
Asterix		
Panoramix		
Idefix		
	$7rJ94K$	

Side 1

Panoramix		
Idefix		
Obelix		
Asterix		
	$Yu78gf$	

Side 2

Fig. 3. Prêt à Voter ballot form

Figure 3 shows the two sides of such a dual ballot form. These two sides should be thought of as rotated around a vertical axis. Note that each side has an independent randomization of the candidate order along with the corresponding cryptographic values. Thus each side carries an independent Prêt à Voter ballot form.

The voter uses only one side to encode their vote and makes an arbitrary choice between the sides. Suppose that the voter in this case chooses what we are referring to as side 2 and wants to cast a vote for Idefix. They place an X against Idefix on side 2 and then destroy the left hand strip that shows the candidate order for side 2. This results in a ballot receipt of the form shown in Figure 4.

Obelix	
Asterix	
Panoramix	
Idefix	
	$7rJ94K$

auditable side
(remaining part of Side 1)

	▨
X	▨
	▨
	▨
$Yu78gf$	▨

vote encoding side
(remaining part of Side 2)

Fig. 4. Both sides of a Prêt à Voter ballot receipt

These two sides should be thought of as being rotated around a vertical axis with respect to each other. Thus the shaded, third column of side 1 would oppose the candidate list of side 2.

The voter makes a random choice of which side to use to cast their vote and made their mark on the middle column against their candidate of choice and leave the flip, unselected side blank. The left hand column of the selected side is destroyed, and so the blank column of the flip side is destroyed. This results in a receipt on which the candidate list for the chosen side has been destroyed, whilst the ballot form on the slip, unselected side is intact, i.e., still has the onion value and candidate list. The information on both sides would now be recorded when the ballot is cast and posted to the WBB.

This flip side can now be audited and checked to ensure that the candidate list printed by the booth correctly corresponds to the onion value. Such checks could be performed immediately at the time of casting to detect any problems as soon as possible. Additionally, checks could be performed on the posted values.

In addition to such post-auditing of the dual ballot forms, we can do some pre-auditing of the committed onions pairs. This would help pick up any malfunctions or corruption in the preparation of the proto-forms at an early stage.

9 Auditing the Anonymising Mixes

In order to detect any malfunction or corruption by the mix tellers, we can again use the Partial Random Checking approach of [JJR02]. Here the checks on audited links will be slightly different: rather than revealing the seed information for the layer in question, the teller is required to reveal the re-randomisation value used to e-encrypt the select link. Auditing of the decryption tellers is quite straightforward as we don't need any further mixing at this stage (the anonymising mixes will be enough to ensure ballot secrecy). The correctness of the decryptions can thus be directly checked by simply encrypting the final values with the public keys and checking that these agree with the initial terms.

10 Handling Full Permutations and STV Style Elections

In order to deal with full permutations of the candidate list it is not immediately clear how to generalise the approach of section 7. As mentioned, one possibility is to leave the index values unchanged through the mixes. This might be acceptable in some situations but is clearly not satisfactory in general.

One solution is simply to have one onion for each candidate position. For a single candidate selection the ballot receipt would in effect simply be the onion value against the chosen candidate. This feels rather inelegant and inefficient in terms of multiplying up the number of onions required.

For a ranked voting method, in which the voters are required to place a rank against each candidate, a ballot receipt would now comprise n pairs of rank value and onion. Each of these pairs could be put through the mix separately with the rank value unchanged (allowing the adversary to partition the mix according to the rank values seems not to matter). This approach works fine as long as the voting method does not require a voters rankings to be kept grouped for tabulation, as with STV for example.

11 Remote Voting with Prêt à Voter

The encrypted ballot forms proposed here would appear to be adaptable to remote voting. We could for example, use a protocol like that described in [ZMSR04], to transform left-hand onions encrypted under the registrars' public key to terms encrypted under an individual voter's public key. The protocol of [ZMSR04] achieves this without having to reveal the underlying plaintext (seed) in the process. A pair of such ballot forms could be supplied to each voter in order to mimic the cut-and-choose mechanism described above. Details of such protocols are the subject of ongoing research.

Any remote voting scheme must face problems of coercion. A possible approach to counter such threats is the use *capabilities* as proposed in [JCJ02]. The possibility of using such a mechanism in conjunction with Prêt à Voter 2005 was explored in [CM05]. Voters are supplied with capabilities that are essentially encryptions of a nonce and a *valid* string. Votes are cast along with a capability and these go through the mix alongside the ballot terms. They emerge from the mix decrypted. A valid capability will decrypt to a valid plaintext. The validity or otherwise of the capability is not apparent until it is decrypted. As a consequence, a voter who is being observed whilst casting their vote has the possibility of deliberately and surreptitiously corrupting their capability. As long as the voter has some window of unobserved access to system he can cast his vote with his valid capability.

12 Conclusions

We have proposed some extensions to Prêt à Voter 2005 to counter vulnerabilities identified previously:

– Authority knowledge of ballot form crypto variables.
– Chain of custody threats.
– Chain voting attacks.
– Kleptographic channels.

The new version of the scheme counters these threats by enabling the distributed construction of encrypted ballot forms by a set of clerks. As a result, only a collusion of all the clerks could determine the cryptographic seed values. This eliminates the need to trust a single entity to keep this material secret and prevents Kleptographic attacks.

Our construction results in ballot forms in which the cryptographic seed values remain encrypted and can be decrypted on demand. Thus, the ballot forms with the candidate ordering can be created and printed in the booth, so eliminating chain of custody and chain voting threats.

The new construction uses ElGamal encryption and so is better suited to using re-encryption mixes for the anonymising/tabulation phase. Earlier work on robust ElGamal mixes may be found in [JJ01, Nef01, GJJS04]. The rather special representation of the ballot receipt in Prêt à Voter, index value plus cryptographic onion, means that it is not entirely straightforward to send such terms through a re-encryption mix. We have shown how, for single candidate selection and cyclic shifts of the candidate list at least, the ballot receipts can be transformed into pure ElGamal terms and so are adapted to re-encryption mixes. We have indicated how the approach may be generalised to deal with alternative electoral methods.

This version of the scheme is, we believe, technically superior to the 2005 version in that it requires less trust assumptions and is more robust against a number of threats. On the other hand, from a socio-technical point of view, it may have certain disadvantages. The voter experience is a little more complex, in particular the need for the cut-and-choose element on the voter protocol, which could have usability implications as well as opening up possibilities of "social engineering" style attacks, [KSW05]. Thus, it is possible that, for some situations like general elections perhaps, in evaluating the trade-off between the trust assumptions of Prêt à Voter 2005 and the usability issues of this scheme, the former might be deemed more acceptable.

Acknowledgements

The authors would firstly like to thank Paul Syverson and Ron Rivest for suggesting adapting Prêt à Voter to work with re-encryption mixes. We would also like to thank James Heather for many helpful discussions, as well as Ran Cannetti, Michael Clarkson, Rob Delicata, Joshua Guttman, Markus Jakobsson, and Thea Peacock. Finally, we are grateful to Paul Syverson and the anonymous referees for their comments on this paper.

References

[CM05] M. Clarkson and A. Myers. Coercion-resistant remote voting using decryption mixes. In *Workshop on Frontiers of Electronic Elections*, 2005.

[CRS05] D. Chaum, P.Y.A. Ryan, and S. Schneider. A practical, voter-verifiable election scheme. In *European Symposium on Research in Computer Security*, number 3679 in Lecture Notes in Computer Science. Springer-Verlag, 2005.

[GJJS04] P. Golle, M. Jakobsson, A. Juels, and P. Syverson. Universal re-encryption for mixnets. In *Proceedings of the 2004 RSA Conference, Cryptographers' track*, San Francisco, USA, February 2004.

[JCJ02] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. To appear, 2002.

[JJ01] M. Jakobsson and A. Juels. An optimally robust hybrid mix network (extended abstract). In *Proceedings of Principles of Distributed Computing — PODC'01*. ACM Press, 2001.

[JJR02] A. Jakobsson, A. Juels, and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX Security '02*, 2002.

[KSW05] C. Karlof, N. Sastry, and D. Wagner. Cryptographic voting protocols: A systems perspective. In *USENIX Security Symposium*, number 3444 in Lecture Notes in Computer Science, pages 186–200. Springer-Verlag, 2005.

[M. 06] M. Gogolewski et al. Kleptographic attacks on e-election schemes. In *International Conference on Emerging trends in Information and Communication Security*, 2006. http://www.nesc.ac.uk/talks/639/Day2/workshop-slides2.pdf.

[Nef01] C. A. Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM Conference on Computer and Communications Security — CCS 2001*, November 2001.

[RP05] P.Y.A. Ryan and T. Peacock. Prêt à voter: a systems perspective. Technical Report CS-TR-929, University of Newcastle upon Tyne, 2005.

[Rya06] P.Y.A. Ryan. Putting the human back in voting protocols. In *Fourteenth International Workshop on Security Protocols*, Lecture Notes in Computer Science. Springer-Verlag, 2006. To appear.

[YY96] A. Young and M. Yung. The dark side of black-box cryptography, or: Should we trust capstone? In *Crypto'96*, Lecture Notes in Computer Science, pages 89–103. Springer-Verlag, 1996.

[ZMSR04] L. Zhou, M. Marsh, F. Schneider, and A. Redz. Distributed blinding for elgamal re-encryption. Technical Report TR 2004-1920, Cornell University, January 2004.

Secure Key-Updating for Lazy Revocation

Michael Backes[1], Christian Cachin[2], and Alina Oprea[3]

[1] Computer Science Department, Saarland University, Saarbruecken, Germany
backes@cs.uni-sb.de
[2] IBM Research, Zurich Research Laboratory, Rüschlikon, Switzerland
cca@zurich.ibm.com
[3] Dept. of Computer Science, Carnegie Mellon University, USA
alina@cs.cmu.edu

Abstract. We consider the problem of efficient key management and user revocation in cryptographic file systems that allow shared access to files. A performance-efficient solution to user revocation in such systems is lazy revocation, a method that delays the re-encryption of a file until the next write to that file. We formalize the notion of key-updating schemes for lazy revocation, an abstraction to manage cryptographic keys in file systems with lazy revocation, and give a security definition for such schemes. We give two composition methods that combine two secure key-updating schemes into a new secure scheme that permits a larger number of user revocations. We prove the security of two slightly modified existing constructions and propose a novel binary tree construction that is also provably secure in our model. Finally, we give a systematic analysis of the computational and communication complexity of the three constructions and show that the novel construction improves the previously known constructions.

1 Introduction

The recent trend of storing large amounts of data on high-speed, dedicated storage-area networks (SANs) stimulates flexible methods for information sharing, but also raises new security concerns. As the networked storage devices are subject to attacks, protecting the confidentiality of stored data is highly desirable in such an environment. Several cryptographic file systems have been designed for this purpose [15], [28], [23], [17], but practical solutions for efficient key management and user revocation still need to be developed further.

We consider cryptographic file systems that allow shared access to stored information and that use untrusted storage devices. In such systems, we can aggregate files into sets such that access permissions and ownership are managed at the level of these sets. The users who have access to the files in a set form a group, managed by the owner of the files, or the *group owner*. Initially, the same cryptographic key can be used to encrypt all files in a set, but upon revocation of a user from the group, the key needs to be changed to prevent access of revoked users to the files. The group owner generates and distributes this new key to the users in the group. There are two options for handling user revocation, *active* and *lazy* revocation, which differ in the way that users are revoked from a group. With active revocation, all files in a set are immediately re-encrypted with the new encryption key. The amount of work caused by a single revocation with this method might, however, be prohibitive for large sets of files. With the alternative method of lazy revocation, re-encryption of a file is delayed until the next write to that file and, thus, users do not experience disruptions in the operation of the file system caused by the immediate re-encryption of all files protected by the same revoked key. In systems adopting lazy revocation, the files in a set might be encrypted with different keys. Storing and distributing these keys becomes more difficult than in systems using active revocation.

In this paper, we address the problem of efficient key management in cryptographic file systems with lazy revocation. An immediate solution to this problem, adopted by the first cryptographic file systems using delayed re-encryption [15], is to store all keys for the files in a set at the group owner. However, we are interested in more efficient methods, in which the number of stored keys is not proportional to the number of revocations. We formalize the notion of *key-updating schemes for lazy revocation* and give a rigorous security definition. In our model, a *center* (e.g., the group owner) initially generates some state information, which takes the role of the master secret key. The center state is updated at every revocation. We call the period of time between two revocations a *time interval*. Upon a user request, the center uses its current local state to derive a *user key* and gives that to the user. From the user key of some time interval, a user must be able to extract the key for any previous time interval efficiently. Security for key-updating schemes requires that any polynomial-time adversary with access to the user key for a particular time interval does not obtain any information about the keys for future time intervals. The keys generated by our key-updating schemes can be used with a symmetric encryption algorithm to encrypt files for confidentiality or with a message-authentication code to authenticate files for integrity protection. Independently and concurrently to our work[1] Fu, Kamara, and Kohno [16] have also formalized key-updating schemes.

We describe two generic composition methods that combine two secure key updating schemes into a new scheme in which the number of time intervals is either the sum or the product of the number of time intervals of the initial schemes. Additionally, we investigate three constructions of key-updating schemes. The first scheme uses a chain of pseudorandom generator applications and is related to existing methods using one-way hash chains. It has constant update cost for the center, but the complexity of the user-key derivation is linear in the total number of time intervals. The second scheme can be based on arbitrary trapdoor permutations and generalizes the key rotation construction of the Plutus file system [23]. It has constant update and user-key derivation times, but the update algorithm uses a relatively expensive public-key operation. These two constructions require that the total number T of time intervals is polynomial in the security parameter. Our third scheme uses a novel construction. It relies on a tree to derive the keys at the leaves from the master key at the root. The tree can be seen as resulting from the iterative application of the additive composition method and supports a practically unbounded number of time intervals. The binary-tree construction balances the tradeoff between the center-state update and user-key derivation algorithms (both of them have logarithmic complexity in T), at the expense of increasing the sizes of the user key and center state by a logarithmic factor in T.

[1] A preliminary version of this paper appears as [6].

The rest of the paper is organized as follows. In Section 2 we give the definition of security for key-updating schemes. In Section 3, we introduce the additive and multiplicative composition methods for secure key-updating schemes. The three constructions and proofs for their security are presented in Section 4. A systematic analysis of the computational and communication complexities of the three constructions is given in Section 5, and an experimental evaluation is presented in Section 6. We compare our scheme to related work in Section 7.

2 Formalizing Key-Updating Schemes

2.1 Definition of Key-Updating Schemes

In our model, we divide time into intervals, not necessarily of fixed length, and each time interval is associated with a new key that can be used in a symmetric-key cryptographic algorithm. In a key-updating scheme, the center generates initial state information that is updated at each time interval, and from which the center can derive a user key. The user key for interval t permits a user to derive the keys of previous time intervals (k_i for $i \leq t$), but it should not give any information about keys of future time intervals (k_i for $i > t$).

We formalize key-updating schemes using the approach of modern cryptography and denote the security parameter by κ. For simplicity, we assume that all the keys are bit strings of length κ. The number of time intervals and the security parameter are given as input to the initialization algorithm.

Definition 1 (Key-Updating Schemes). *A key-updating scheme consists of four deterministic polynomial time algorithms* $\mathsf{KU} = (\mathsf{Init}, \mathsf{Update}, \mathsf{Derive}, \mathsf{Extract})$ *with the following properties:*

- *The initialization algorithm,* Init, *takes as input the* security parameter 1^κ, *the* number of time intervals T *and a* random seed $s \in \{0,1\}^{l(\kappa)}$ *for a polynomial* $l(\kappa)$, *and outputs a bit string* S_0, *called the initial center* state.
- *The key update algorithm,* Update, *takes as input the current* time interval $0 \leq t \leq T-1$, *the current center* state S_t, *and outputs the center* state S_{t+1} *for the next time interval.*
- *The user key derivation algorithm,* Derive, *is given as input a* time interval $1 \leq t \leq T$ *and the center* state S_t, *and outputs the* user key M_t. *The user key can be used to derive all keys* k_i *for* $1 \leq i \leq t$.
- *The key extraction algorithm,* Extract, *is executed by the user and takes as input a* time interval $1 \leq t \leq T$, *the* user key M_t *for interval* t *as received from the center, and a* target time interval i *with* $1 \leq i \leq t$. *The algorithm outputs the* key k_i *for interval* i.

W.l.o.g., we assume that the Update algorithm is run at least once after the Init algorithm, before any user keys can be derived. The first time the Update algorithm is run, it is given as input time interval $t = 0$. User keys and keys are associated with the time intervals between 1 and T.

2.2 Applications to Cryptographic File Systems

In a cryptographic file system adopting lazy revocation, the re-encryption of a file after a revocation is delayed until the next write to that file. Similarly to the Plutus file system, files can be divided into sets based on their access permissions, such that all files in a set have the same permissions. Initially, all files in a set can be encrypted with the same key. We assume that file owners are responsible for the generation and distribution of keys to the authorized users, so file owners take the role of the center in our model of key-updating schemes.

When a user is revoked from the group of users having access to the set of files, the file owner runs the Update algorithm generating a new state and advancing the time interval. The file owner then runs Derive and the new user key is distributed to all the users that have now access permissions to the files. A user writing a file uses the encryption key for the latest time interval, which can be efficiently extracted from the latest user key. To decrypt a file, a user needs to know the version of the key that was used to encrypt it, and extract the appropriate encryption key from the user key. The key version with which each file is encrypted might, for example, be stored in the file i-node on the file server.

Assuming that the integrity of files is protected with a message-authentication code (MAC), key-updating schemes can also be used to manage symmetric keys for authentication. To guarantee independence of the keys used for confidentiality and integrity, different instances of key-updating schemes have to be used for encryption and authentication.

2.3 Security of Key-Updating Schemes

The definition of security for key-updating schemes requires that a polynomial-time adversary with access to the user key for a time interval t is not able to derive any information about the keys for the next time interval. The definition we give here is related to the definition of forward-secure pseudorandom generators given by Bellare and Yee [8]. Formally, consider a probabilistic polynomial-time adversary $\mathcal{A} = (\mathcal{A}_\mathcal{U}, \mathcal{A}_\mathcal{G})$ that participates in the following experiment:

Initialization: The initial center state is generated with the Init algorithm.

Key updating: The adversary adaptively picks a time interval t such that $0 \leq t \leq T-1$ as follows. Starting with $t = 0, 1, \ldots$, algorithm $\mathcal{A}_\mathcal{U}$ is given the user keys M_t for all consecutive time intervals until $\mathcal{A}_\mathcal{U}$ decides to output stop or t becomes equal to $T-1$. We require that $\mathcal{A}_\mathcal{U}$, a probabilistic polynomial-time algorithm, outputs stop at least once before halting. $\mathcal{A}_\mathcal{U}$ also outputs some additional information $z \in \{0,1\}^*$ that is given as input to algorithm $\mathcal{A}_\mathcal{G}$.

Challenge: A challenge for the adversary is generated, which is either the key for time interval $t+1$ generated with the Update, Derive and Extract algorithms, or a random bit string of length κ.

Guess: $\mathcal{A}_\mathcal{G}$ takes the challenge and z as inputs and outputs a bit b.

The key-updating scheme is secure if the advantage of the adversary of distinguishing between the properly generated key for time interval $t+1$ and the random key is only negligibly larger than $\frac{1}{2}$. More formally, the definition of a secure key-updating scheme is the following:

Definition 2 (Security of Key-Updating Schemes). *Let* $\mathsf{KU} = (\mathsf{Init}, \mathsf{Update}, \mathsf{Derive}, \mathsf{Extract})$ *be a key-updating scheme and* $\mathcal{A}$ *a polynomial-time adversary algorithm that participates in one of the two experiments defined in Figure 1.*

```
$\mathsf{Exp}^{\texttt{sku-0}}_{\mathsf{KU},\mathcal{A}}(1^\kappa, T)$
    $S_0 \leftarrow \mathsf{Init}(1^\kappa, T)$
    $t \leftarrow 0$
    $(d, z) \leftarrow \mathcal{A}_\mathcal{U}(t, \perp, \perp)$
    $\mathsf{while}(d \neq \mathsf{stop})$ and $(t < T - 1)$
        $t \leftarrow t + 1$
        $S_t \leftarrow \mathsf{Update}(t-1, S_{t-1})$
        $M_t \leftarrow \mathsf{Derive}(t, S_t)$
        $(d, z) \leftarrow \mathcal{A}_\mathcal{U}(t, M_t, z)$
    $S_{t+1} \leftarrow \mathsf{Update}(t, S_t)$
    $M_{t+1} \leftarrow \mathsf{Derive}(t+1, S_{t+1})$
    $k_{t+1} \leftarrow \mathsf{Extract}(t+1, M_{t+1}, t+1)$
    $b \leftarrow \mathcal{A}_\mathcal{G}(k_{t+1}, z)$
    return $b$
```

```
$\mathsf{Exp}^{\texttt{sku-1}}_{\mathsf{KU},\mathcal{A}}(1^\kappa, T)$
    $S_0 \leftarrow \mathsf{Init}(1^\kappa, T)$
    $t \leftarrow 0$
    $(d, z) \leftarrow \mathcal{A}_\mathcal{U}(t, \perp, \perp)$
    $\mathsf{while}(d \neq \mathsf{stop})$ and $(t < T - 1)$
        $t \leftarrow t + 1$
        $S_t \leftarrow \mathsf{Update}(t-1, S_{t-1})$
        $M_t \leftarrow \mathsf{Derive}(t, S_t)$
        $(d, z) \leftarrow \mathcal{A}_\mathcal{U}(t, M_t, z)$
    $k_{t+1} \leftarrow_R \{0,1\}^\kappa$
    $b \leftarrow \mathcal{A}_\mathcal{G}(k_{t+1}, z)$
    return $b$
```

Fig. 1. Experiments defining the security of key-updating schemes

The advantage of the adversary $\mathcal{A} = (\mathcal{A}_\mathcal{U}, \mathcal{A}_\mathcal{G})$ *for* KU *is defined as*

$$\mathsf{Adv}^{\texttt{sku}}_{\mathsf{KU},\mathcal{A}}(\kappa, T) = \left|\Pr\left[\mathsf{Exp}^{\texttt{sku-1}}_{\mathsf{KU},\mathcal{A}}(1^\kappa, T) = 1\right] - \Pr\left[\mathsf{Exp}^{\texttt{sku-0}}_{\mathsf{KU},\mathcal{A}}(1^\kappa, T) = 1\right]\right|.$$

Without loss of generality, we can relate the success probability of adversary $\mathcal{A}$ *of distinguishing between the two experiments and its advantage as*

$$\Pr[\mathcal{A} \ \textit{succeeds}] = \frac{1}{2}\left[1 + \mathsf{Adv}^{\texttt{sku}}_{\mathsf{KU},\mathcal{A}}(\kappa, T)\right]. \tag{1}$$

The key-updating scheme KU *is secure if for all polynomial-time adversaries* $\mathcal{A}$ *and all* T*, there exists a negligible function* ϵ *such that* $\mathsf{Adv}^{\texttt{sku}}_{\mathsf{KU},\mathcal{A}}(\kappa, T) = \epsilon(\kappa)$.

Remark. The security notion we have defined is equivalent to a seemingly stronger security definition, in which the adversary can choose the challenge time interval t^* with the restriction that t^* is greater than the time interval at which the adversary outputs stop and that t^* is polynomial in the security parameter. This second security definition guarantees, intuitively, that the adversary is not gaining any information about the keys of any future time intervals after it outputs stop.

3 Composition of Key-Updating Schemes

Let $\mathsf{KU}_1 = (\mathsf{Init}_1, \mathsf{Update}_1, \mathsf{Derive}_1, \mathsf{Extract}_1)$ and $\mathsf{KU}_2 = (\mathsf{Init}_2, \mathsf{Update}_2, \mathsf{Derive}_2, \mathsf{Extract}_2)$ be two secure key-updating schemes using the same security parameter κ with T_1 and T_2 time intervals, respectively. In this section, we show how to combine the two schemes into a secure key-updating scheme $\mathsf{KU} = (\mathsf{Init}, \mathsf{Update}, \mathsf{Derive}, \mathsf{Extract})$, which is either the additive or multiplicative composition of the two schemes with $T = T_1 + T_2$ and $T = T_1 \cdot T_2$ time intervals, respectively. Similar generic composition methods have been given previously for forward-secure signature schemes [26].

For simplicity, we assume the length of the random seed in the Init algorithm of the scheme KU to be κ for both composition methods. Let $G : \{0,1\}^\kappa \rightarrow \{0,1\}^{l_1(\kappa)+l_2(\kappa)}$ be a pseudorandom generator; it can be used to expand a random seed of length κ into two random bit strings of length $l_1(\kappa)$ and $l_2(\kappa)$, respectively, as needed for Init_1 and Init_2. We write $G(s) = G_1(s)\|G_2(s)$ with $|G_1(s)| = l_1(\kappa)$ and $|G_2(s)| = l_2(\kappa)$ for $s \in \{0,1\}^\kappa$.

3.1 Additive Composition

The additive composition of two key-updating schemes uses the keys generated by the first scheme for the first T_1 time intervals and the keys generated by the second scheme for the subsequent T_2 time intervals. The user key for the first T_1 intervals in KU is the same as that of scheme KU_1 for the same interval. For an interval t greater than T_1, the user key includes both the user key for interval $t - T_1$ of scheme KU_2, and the user key for interval T_1 of scheme KU_1. The details of the additive composition method are described in Figure 2. The security of the composition operation is analyzed in the following theorem, whose proof is given in the full version of this paper [6].

Theorem 1. *Suppose that* $\mathsf{KU}_1 = (\mathsf{Init}_1, \mathsf{Update}_1, \mathsf{Derive}_1, \mathsf{Extract}_1)$ *and* $\mathsf{KU}_2 = (\mathsf{Init}_2, \mathsf{Update}_2, \mathsf{Derive}_2, \mathsf{Extract}_2)$ *are two secure key-updating schemes with* T_1 *and* T_2 *time intervals, respectively, and that G is a pseudorandom generator as above. Then* $\mathsf{KU} = (\mathsf{Init}, \mathsf{Update}, \mathsf{Derive}, \mathsf{Extract})$ *described in Figure 2 denoted as* $\mathsf{KU}_1 \oplus \mathsf{KU}_2$ *is a secure key-updating scheme with* $T_1 + T_2$ *time intervals.*

```
Init($1^\kappa, T, s$)
    $S_0^1 \leftarrow \mathsf{Init}_1(1^\kappa, T_1, G_1(s))$
    $S_0^2 \leftarrow \mathsf{Init}_2(1^\kappa, T_2, G_2(s))$
    return $(S_0^1, S_0^2)$
```

```
Derive($t, (S_t^1, S_t^2)$)
    if $t < T_1$
        $M_t^1 \leftarrow \mathsf{Derive}_1(t, S_t^1)$
        $M_t^2 \leftarrow \perp$
    else
        $M_t^1 \leftarrow \mathsf{Derive}_1(T_1, S_t^1)$
        $M_t^2 \leftarrow \mathsf{Derive}_2(t - T_1, S_t^2)$
    return$(M_t^1, M_t^2)$
```

```
Update($t, (S_t^1, S_t^2)$)
    if $t < T_1$
        $S_{t+1}^1 \leftarrow \mathsf{Update}_1(t, S_t^1)$
        $S_{t+1}^2 \leftarrow S_t^2$
    else
        $S_{t+1}^1 \leftarrow S_t^1$
        $S_{t+1}^2 \leftarrow \mathsf{Update}_2(t - T_1, S_t^2)$
    return $(S_{t+1}^1, S_{t+1}^2)$
```

```
Extract($t, (M_t^1, M_t^2), i$)
    if $i > T_1$
        $k_i \leftarrow \mathsf{Extract}_2(t - T_1, M_t^2, i - T_1)$
    else
        if $t < T_1$
            $k_i \leftarrow \mathsf{Extract}_1(t, M_t^1, i)$
        else
            $k_i \leftarrow \mathsf{Extract}_1(T_1, M_t^1, i)$
    return $k_i$
```

Fig. 2. The additive composition of KU_1 and KU_2

Extended Additive Composition. It is immediate to extend the additive composition to construct a new scheme with $T_1 + T_2 + 1$ time intervals. The idea is to use the first scheme for the keys of the first T_1 intervals, the second scheme for the keys of the next T_2 intervals, and the seed s as the key for the $(T_1 + T_2 + 1)$-th interval. By revealing the seed s as the user key at interval $T_1 + T_2 + 1$, all previous keys of KU_1 and KU_2 can be derived. This idea will be useful in our later construction of a binary tree key-updating scheme. We call this composition method *extended additive composition.*

3.2 Multiplicative Composition

The idea behind the multiplicative composition operation is to use every key of the first scheme to seed an instance of the second scheme. Thus, for each one of the T_1 time intervals of the first scheme, we generate an instance of the second scheme with T_2 time intervals.

We denote a time interval t for $1 \leq t \leq T_1 \cdot T_2$ of scheme KU as a pair $t = <i, j>$, where i and j are such that $t = (i-1)T_2 + j$ for $1 \leq i \leq T_1$ and $1 \leq j \leq T_2$. The Update algorithm is run initially for time interval $t = 0$, which will be expressed as $<0, 0>$. The user key for a time interval $t = <i, j>$ includes both the user key for time interval $i - 1$ of scheme KU_1 and the user key for time interval j of scheme KU_2. A user receiving $M_{<i,j>}$ can extract the key for any time interval $<m, n> \leq <i, j>$ by first extracting the key K for time interval m of KU_1 (this step needs to be performed only if $m < i$), then using K to derive the initial state of the m-th instance of the scheme KU_2, and finally, deriving the key $k_{<m,n>}$. The details of the multiplicative composition method are shown in Figure 3.

```
$\mathsf{Init}(1^\kappa, T, s)$
    $S_0^1 \leftarrow \mathsf{Init}_1(1^\kappa, T_1, G_1(s))$
    $S_1^1 \leftarrow \mathsf{Update}_1(0, S_0^1)$
    $k_1^1 \leftarrow \mathsf{Extract}_1(1, \mathsf{Derive}_1(1, S_1^1), 1)$
    $S_0^2 \leftarrow \mathsf{Init}_2(1^\kappa, T_2, G_2(k_1^1))$
    return $(\perp, S_0^1, S_0^2)$
```

```
$\mathsf{Derive}(<i, j>, (S_{i-1}^1, S_i^1, S_j^2))$
    if $i > 1$
        $M_{i-1}^1 \leftarrow \mathsf{Derive}_1(i-1, S_{i-1}^1)$
    else
        $M_{i-1}^1 \leftarrow \perp$
    $M_j^2 \leftarrow \mathsf{Derive}_2(j, S_j^2)$
    return $(M_{i-1}^1, M_j^2)$
```

```
$\mathsf{Update}(<i, j>, (S_{i-1}^1, S_i^1, S_j^2))$
    if $j = T_2$
        $S_{i+1}^1 \leftarrow \mathsf{Update}_1(i, S_i^1)$
        $k_{i+1}^1 \leftarrow \mathsf{Extract}_1(i+1,$
                    $\mathsf{Derive}_1(i+1, S_{i+1}^1), i+1)$
        $S_0^2 \leftarrow \mathsf{Init}_2(1^\kappa, T_2, G_2(k_{i+1}^1))$
        $S_1^2 \leftarrow \mathsf{Update}_2(0, S_0^2)$
        return $(S_i^1, S_{i+1}^1, S_1^2)$
    else
        $S_{j+1}^2 \leftarrow \mathsf{Update}_2(j, S_j^2)$
        return $(S_{i-1}^1, S_i^1, S_{j+1}^2)$
```

```
$\mathsf{Extract}(<i, j>, (M_{i-1}^1, M_j^2), <m, n>)$
    if $i = m$
        $k_{<m,n>} \leftarrow \mathsf{Extract}_2(j, M_j^2, m)$
    else
        $K \leftarrow \mathsf{Extract}_1(i-1, M_{i-1}^1, m)$
        $S_0^2 \leftarrow \mathsf{Init}_2(1^\kappa, T_2, G_2(K))$
        $k_{<m,n>} \leftarrow \mathsf{Extract}_2(T_2, S_0^2, n)$
    return $k_{<m,n>}$
```

Fig. 3. The multiplicative composition of KU_1 and KU_2

The security of the multiplicative composition method is analyzed in the following theorem, whose proof is given in the full version of this paper [6].

Theorem 2. *Suppose that* $\mathsf{KU}_1 = (\mathsf{Init}_1, \mathsf{Update}_1, \mathsf{Derive}_1, \mathsf{Extract}_1)$ *and* $\mathsf{KU}_2 = (\mathsf{Init}_2, \mathsf{Update}_2, \mathsf{Derive}_2, \mathsf{Extract}_2)$ *are two secure key-updating schemes with* T_1 *and* T_2 *time intervals, respectively, and that G is a pseudorandom generator as above. Then* $\mathsf{KU} = (\mathsf{Init}, \mathsf{Update}, \mathsf{Derive}, \mathsf{Extract})$ *described in Figure 3 denoted as* $\mathsf{KU}_1 \otimes \mathsf{KU}_2$ *is a secure key-updating scheme with* $T_1 \cdot T_2$ *time intervals.*

4 Constructions

In this section, we describe three constructions of key-updating schemes with different complexity and communication tradeoffs. The first two constructions are based on previously proposed methods [23,16]. We give cryptographic proofs that demonstrate the security of the existing constructions after some subtle modifications. Additionally, we propose a third construction that is more efficient than the known schemes. It uses a binary tree to derive the user keys and is also provably secure in our model.

4.1 Chaining Construction (CKU)

In this construction, the center generates an initial random seed of length κ and applies a pseudorandom generator iteratively i times to obtain the key for time interval $T - i$, for $1 \leq i \leq T - 1$. This construction is inspired by a folklore method using a hash chain for deriving the keys. A construction based on a hash chain can be proven secure if the hash function h is modeled as a random oracle. To obtain a provably secure scheme in the standard model, we replace the hash function with a pseudorandom generator.

Let $G : \{0,1\}^\kappa \rightarrow \{0,1\}^{2\kappa}$ be a pseudorandom generator. We write $G(s) = G_1(s) \| G_2(s)$ with $|G_1(s)| = |G_2(s)| = \kappa$ for $s \in \{0,1\}^\kappa$. The algorithms of the chaining construction, called CKU, are:

- $\mathsf{Init}(1^\kappa, T, s)$ generates a random seed s_0 of length κ from s and outputs $S_0 = s_0$.
- $\mathsf{Update}(t, S_t)$ copies the state S_t into S_{t+1}.
- $\mathsf{Derive}(t, S_t)$ and $\mathsf{Extract}(t, M_t, i)$ are given in Figure 4.

```
Derive($t, S_t$)
    $B_{T+1} \leftarrow S_t$
    for $i = T$ downto $t$
        $(B_i, k_i) \leftarrow G(B_{i+1})$
    return $(B_t, k_t)$
```

```
Extract($t, M_t, i$)
    $(B_t, k_t) \leftarrow M_t$
    for $j = t - 1$ downto $i$
        $(B_j, k_j) \leftarrow G(B_{j+1})$
    return $k_i$
```

Fig. 4. The $\mathsf{Derive}(t, S_t)$ and $\mathsf{Extract}(t, M_t, i)$ algorithms of the chaining construction

This construction has constant center-state size and linear cost for the user-key derivation algorithm. An alternative construction with linear center-state size and constant user-key derivation is to precompute all the keys k_i and user keys M_i, for $1 \leq i \leq T$ in the Init algorithm and store all of them in the initial center state S_0. The security of the chaining construction is given be the following theorem, whose proof is in the full version of this paper [6].

Theorem 3. *Given a pseudorandom generator G,* CKU *is a secure key-updating scheme.*

4.2 Trapdoor Permutation Construction (TDKU)

In this construction, the center picks an initial random state that is updated at each time interval by applying the inverse of a trapdoor permutation. The trapdoor is known only to the center, but a user, given the state at a certain moment, can apply the permutation iteratively to generate all previous states. The key for a time interval is generated by applying a hash function, modeled as a random oracle, to the current state. This idea underlies the key rotation mechanism of the Plutus file system [23], with the difference that Plutus uses the output of an RSA trapdoor permutation directly for the encryption key. We could not prove the security of this scheme in our model for key-updating schemes, even when the trapdoor permutation is not arbitrary, but instantiated with the RSA permutation.

This construction has the advantage that knowledge of the total number of time intervals is not needed in advance; on the other hand, its security can only be proved in the random oracle model. Let a family of trapdoor permutations be given such that the domain size of the permutations with security parameter κ is $l(\kappa)$, for some polynomial l. Let $h : \{0,1\}^{l(\kappa)} \rightarrow \{0,1\}^{\kappa}$ be a hash function modeled as a random oracle. The detailed construction of the trapdoor permutation scheme, called TDKU, is presented below:

- Init$(1^{\kappa}, T, s)$ generates a random $s_0 \leftarrow_R \{0,1\}^{l(\kappa)}$ and a trapdoor permutation $f : \{0,1\}^{l(\kappa)} \rightarrow \{0,1\}^{l(\kappa)}$ with trapdoor τ from seed s using a pseudorandom generator. Then it outputs $S_0 = (s_0, f, \tau)$.
- Update(t, S_t) with $S_t = (s_t, f, \tau)$ computes $s_{t+1} = f^{-1}(s_t)$ and outputs $S_{t+1} = (s_{t+1}, f, \tau)$.
- Derive(t, S_t) outputs $M_t \leftarrow (s_t, f)$.
- Extract(t, M_t, i) applies the permutation iteratively $t - i$ times to generate state $s_i = f^{t-i}(M_t)$ and then outputs $h(s_i)$.

The security of this construction is given be the following theorem, whose proof is in the full version of this paper [6].

Theorem 4. *Given a family of trapdoor permutations and a hash function h,* TDKU *is a secure key-updating scheme in the random oracle model.*

4.3 Tree Construction (TreeKU)

In the two schemes above, at least one of the algorithms Update, Derive and Extract has worst-case complexity linear in the total number of time intervals. We present a tree construction based on ideas of Canetti, Halevi and Katz [10] with constant complexity for the Derive algorithm and logarithmic worst-case complexity in the number of time intervals for the Update and Extract algorithms. Moreover, the amortized complexity of the Update algorithm is constant. In this construction, the user key size is increased by at most a logarithmic factor in T compared to the user key size of the two constructions described above.

Our tree-based key-updating scheme, called TreeKU, generates keys using a complete binary tree with T nodes, assuming that $T = 2^d - 1$ for some $d \in \mathbb{Z}$. Each node in the tree is associated with a time interval between 1 and T, a unique label in $\{0,1\}^*$, a *tree-key* in $\{0,1\}^\kappa$ and an *external key* in $\{0,1\}^\kappa$ such that:

1. Time intervals are assigned to tree nodes using post-order tree traversal, i.e., a node corresponds to interval i if it is the i-th node in the post-order traversal of the tree. We refer to the node associated with interval t as node t.
2. We define a function label that maps node t with $1 \le t \le T$ to its label in $\{0,1\}^*$ as follows. The root of the tree is labeled by the empty string ε, and the left and right children of a node with label ℓ are labeled by $\ell\|0$ and by $\ell\|1$, respectively. The parent of a node with label ℓ is denoted by $\mathsf{parent}(\ell)$, thus $\mathsf{parent}(\ell\|0) = \mathsf{parent}(\ell\|1) = \ell$. We denote the length of a label ℓ by $|\ell|$.
3. The tree-key for the root node is chosen at random. The tree-keys for the two children of an internal node in the tree are derived from the tree-key of the parent node using a pseudorandom generator $G : \{0,1\}^\kappa \rightarrow \{0,1\}^{2\kappa}$. For an input $s \in \{0,1\}^\kappa$, we write $G(s) = G_1(s)\|G_2(s)$ with $|G_1(s)| = |G_2(s)| = \kappa$. If the tree-key for the internal node with label ℓ is denoted u_ℓ, then the tree-keys for its left and right children are $u_{\ell\|0} = G_1(u_\ell)$ and $u_{\ell\|1} = G_2(u_\ell)$, respectively. This implies that once the tree-key for a node is revealed, then the tree-keys of its children can be computed, but knowing the tree-keys of both children of a node does not reveal any information about the tree-key of the node.
4. The external key of a node t is the key k_t output by the scheme to the application for interval t. For a node t with tree-key $u_{\mathsf{label}(t)}$, the external key k_t is obtained by computing $F_{u_{\mathsf{label}(t)}}(1)$, where $F_u(b) = F(u,b)$ and $F : \{0,1\}^\kappa \times \{0,1\} \rightarrow \{0,1\}^\kappa$ is a pseudorandom function on bits.

We describe the four algorithms of the binary tree key-updating scheme:

- $\mathsf{Init}(1^\kappa, T, s)$ generates the tree-key for the root node randomly, $u_T \leftarrow_R \{0,1\}^\kappa$, using seed s, and outputs $S_0 = (\{(\varepsilon, u_T)\}, \emptyset)$.
- $\mathsf{Update}(t, S_t)$ updates the state $S_t = (P_t, L_t)$ to the next center state $S_{t+1} = (P_{t+1}, L_{t+1})$. The center state for interval t consists of two sets: P_t that contains pairs of (label, tree-key) for all nodes on the path from the root to node t (including node t), and L_t that contains label/tree-key pairs for all left siblings of the nodes in P_t that are not in P_t.
  We use several functions in the description of the Update algorithm. For a label ℓ and a set A of label/tree-key pairs, we define a function $\mathsf{searchkey}(\ell, A)$ that outputs a tree-key u for which $(\ell, u) \in A$, if the label exists in the set, and $\perp$ otherwise. Given a label ℓ and a set of label/tree-key pairs A, function $\mathsf{rightsib}(\ell, A)$ returns the label and the tree-key of the right sibling of the node with label ℓ, and, similarly, function $\mathsf{leftsib}(\ell, A)$ returns the label and the tree-key of the left sibling of the node with label ℓ (assuming the labels and tree-keys of the siblings are in A). The function $\mathsf{leftkeys}$ is given as input a label/tree-key pair of a node and returns all label/tree-key pairs of the left-most nodes in the subtree rooted at the input node, including label and tree-key of the input node.

```
Update($t$, $(P_t, L_t)$)
    if $t = 0$
        $P_1 \leftarrow$ leftkeys($\varepsilon, u_T$)                /* $P_1$ contains the label/tree-key pairs of all the left-most nodes */
        $L_1 \leftarrow \emptyset$                                     /* the set of left siblings is empty */
    else
        $\ell_t \leftarrow$ label($t$)                                 /* compute the label of node $t$ */
        $u_t \leftarrow$ searchkey($\ell_t, P_t$)                      /* compute the tree-key of node $t$ */
        if $\ell_t$ ends in 0                                         /* $t$ is the left child of its parent */
            $(\ell_s, u_s) \leftarrow$ rightsib($\ell_t, P_t$)          /* compute the label/tree-key pair of the right sibling of $t$ */
            $P_{t+1} \leftarrow P_t \setminus \{(\ell_t, u_t)\} \cup$ leftkeys($\ell_s, u_s$) /* update the label/tree-key pairs in $P_{t+1}$ */
            $L_{t+1} \leftarrow L_t \cup \{(\ell_t, u_t)\}$            /* add the label/tree-key pair of $t$ to set of left siblings for $t + 1$ */
        else                                                          /* $t$ is the right child of its parent */
            $(\ell_s, u_s) \leftarrow$ leftsib($\ell_t, L_t$)           /* compute the label/tree-key pair of the left sibling of $t$ */
            $P_{t+1} \leftarrow P_t \setminus \{(\ell_t, u_t)\}$       /* remove label/tree-key pair of $t$ from $P_{t+1}$ */
            $L_{t+1} \leftarrow L_t \setminus \{(\ell_s, u_s)\}$       /* remove label/tree-key pair of left sibling of $t$ from $L_{t+1}$ */
    return $(P_{t+1}, L_{t+1})$

leftkeys($\ell, u$)
    $A \leftarrow \emptyset$                                           /* initialize set $A$ with the empty set */
    while $|\ell| \leq d$                                             /* advance to the left until we reach a leaf */
        $A \leftarrow A \cup \{(\ell, u)\}$                            /* add the label and tree-key of the current node in $A$ */
        $\ell \leftarrow \ell \| 0$                                    /* move to left child of the node with label $p$ */
        $u \leftarrow G_1(u)$                                          /* compute the tree-key of the left child */
    return $A$
```

Fig. 5. The Update(t, (P_t, L_t)) algorithm

The code for the Update and leftkeys algorithms is given in Figure 5. We omit the details of functions searchkey, rightsib and leftsib. The Update algorithm distinguishes three cases:

1. If $t = 0$, the Update algorithm computes the label/tree-key pairs of all left-most nodes in the complete tree using function leftkeys and stores them in P_1. The set L_1 is empty in this case, as nodes in P_1 do not have left siblings.
2. If t is the left child of its parent, the successor of node t in post-order traversal is the left-most node in the subtree rooted at the right sibling t' of node t. P_{t+1} contains all label/tree-key pairs in P_t except the tuple for node t, and, in addition, all label/tree-key pairs for the left-most nodes in the subtree rooted at t', which are computed by leftkeys. The set of left siblings L_{t+1} contains all label/tree-key pairs from L_t and, in addition, the label/tree-key pair for node t.
3. If t is the right child of its parent, node $t + 1$ is its parent, so P_{t+1} contains all label/tree-key pairs from P_t except the tuple for node t, and L_{t+1} contains all the label/tree-key pairs in L_t except the pair for the left sibling of node t.

- Algorithm Derive(t, (P_t, L_t)) outputs the user tree-key M_t, which is the minimum information needed to generate the set of tree-keys $\{u_i : i \leq t\}$. Since the tree-key of any node reveals the tree-keys for all nodes in the subtree rooted at that node, M_t consists of the label/tree-key pairs for the left siblings (if any) of all nodes on the path from the root to the parent of node t and the label/tree-key pair of node t. This information has already been pre-computed such that one can set $M_t \leftarrow \{(\text{label}(t), u_t)\} \cup L_t$.
- Algorithm Extract(t, M_t, i) first finds the maximum predecessor of node i in post-order traversal whose label/tree-key pair is included in the user tree-key M_t. Then it computes the tree-keys for all nodes on the path from that predecessor to node i. The external key k_i is derived from the tree-key u_i as $k_i = F_{u_i}(1)$ using the pseudorandom function. The algorithm is in Figure 6.

```
Extract($t$, $M_t$, $i$)
    $\ell_1 \dots \ell_s \leftarrow$ label($i$)                    /* the label of $i$ has length $s$ */
    $v \leftarrow s$
    $\ell \leftarrow \ell_1 \dots \ell_v$
    while $v > 0$ and searchkey($\ell$, $M_t$) $= \perp$   /* find a predecessor of $i$ that is in $M_t$ */
        $v \leftarrow v - 1$
        $\ell \leftarrow \ell_1 \dots \ell_v$
    for $j = v + 1$ to $s$                         /* compute tree-keys of all nodes on path from predecessor to $i$ */
        $u_{\ell_1 \dots \ell_j} \leftarrow G_{\ell_j}(u_{\ell_1 \dots \ell_{j-1}})$
    $k_{\ell_1 \dots \ell_s} \leftarrow F_{u_{\ell_1 \dots \ell_s}}(1)$               /* return external key of node $i$ */
    return $k_{\ell_1 \dots \ell_s}$
```

Fig. 6. The Extract(t, M_t, i) algorithm

Analysis of Complexity. The worst-case complexity of the cryptographic operations used in the Update and Extract algorithms is logarithmic in the number of time intervals, and that of Derive is constant. However, it is easy to see that the key for each node is computed exactly once if T updates are executed. This implies that the total cost of all update operations is T pseudorandom-function applications, so the amortized cost per update is constant. The size of the center state and the user key is proportional to the height of the binary tree, so the worst-case space complexity is $\mathcal{O}(\kappa \log_2 T)$ bits.

The security of the tree construction is given be the following theorem, whose proof is in the full version of this paper [6].

Theorem 5. *Given a pseudorandom generator G and a pseudorandom function F,* TreeKU *is a secure key-updating scheme.*

An Incremental Tree Construction. We can construct an incremental tree scheme using ideas from the generic forward-secure signature scheme of Malkin, Micciancio, and Miner [26]. The incremental scheme does not require the total number of time intervals to be known in advance.

Let TreeKU(i) be the binary tree construction with $2^i - 1$ nodes. Then the incremental tree scheme is obtained by additively composing binary tree schemes with increasing number of intervals: TreeKU$(1) \oplus$ TreeKU$(2) \oplus$ TreeKU$(3) \oplus \dots$. The keys generated by the tree scheme TreeKU(i) correspond to the time intervals between $2^i - i$ and $2^{i+1} - i - 2$ in the incremental scheme. Once the intervals of the tree scheme TreeKU(i) are exhausted, an instance of TreeKU$(i + 1)$ is generated, if needed.

In addition to allowing a practically unbounded number of time intervals, this construction has the property that the complexity of the Update, Derive and Extract algorithms and the size of the center state and user key depend on the number of past time intervals. Below we perform a detailed analysis of the cost of the scheme for an interval t that belongs to TreeKU(i) with $2^i - i \le t \le 2^{i+1} - i - 2$:

1. The center state includes all the root keys of the previous $i - 1$ trees and the center state for node t in TreeKU(i). In the worst-case, this equals $(i - 1) + (2i - 1) = 3i - 2 = 3\lceil \log_2(t) \rceil - 2$ tree-keys. Similarly, the user key for interval t includes the user key of node t as in scheme TreeKU(i) and the root keys of the previous $i - 1$ trees, in total $(i - 1) + (i - 1) = 2i - 2 = 2\lceil \log_2(t) \rceil - 2$ tree-keys. It follows that the space complexity of the center state and the user key for interval t is $\mathcal{O}(\kappa \log_2(t))$ bits.

2. The cost of both Update and Extract algorithms is at most $i = \lceil \log_2(t) \rceil$ applications of the pseudorandom generator. The cost of Derive is constant, as in the tree construction.

5 Performance of the Constructions

In this section we analyze the time complexity of the cryptographic operations and the space complexity of the center and the user for the three proposed constructions. Recall that all schemes generate keys of length κ. In analyzing the time complexity of the algorithms, we specify what kind of operations we measure and distinguish public-key operations (PK op.) from pseudorandom generator applications (PRG op.) because PK operations are typically much more expensive than PRG applications. We omit the time complexity of the Init algorithm, as it involves only the pseudorandom generator for all schemes except for the trapdoor permutation scheme, in which Init also generates the trapdoor permutation. The space complexities are measured in bits. Table 1 shows the details for a given number T of time intervals.

Table 1. Worst-case time and space complexities of the constructions for T time intervals. *Note: the amortized complexity of Update(t, S_t) in the binary tree scheme is constant.

	CKU	TDKU	TreeKU
Update(t, S_t) time	0	1 PK op.	$\mathcal{O}(\log_2 T)$ PRG op.*
Derive(t, S_t) time	$T - t$ PRG op.	const.	$O(\log_2 T)$
Extract(t, M_t, i) time	$t - i$ PRG op.	$t - i$ PK op.	$\mathcal{O}(\log_2 T)$ PRG op.
Center state size	κ	poly(κ)	$\mathcal{O}(\kappa \log_2 T)$
User key size	κ	κ	$\mathcal{O}(\kappa \log_2 T)$

In the chaining scheme CKU, the Update algorithm takes no work, but the Extract and the Derive algorithms take linear work in the number of time intervals. On the other hand, the trapdoor permutation scheme TDKU has efficient user-key derivation, which involves only a copy operation, but the complexity of the Update algorithm is one application of the trapdoor permutation inverse and that of the Extract(t, M_t, i) algorithm is $t - i$ applications of the trapdoor permutation. The tree-based scheme TreeKU balances the tradeoffs between the complexity of the three algorithms, taking logarithmic work in the number of time intervals for all three algorithms in the worst-case. The Derive algorithm involves only $\mathcal{O}(\log_2 T)$ copy operations, and Update and Extract algorithms involve $\mathcal{O}(\log_2 T)$ PRG operations. This comes at the cost of increasing the center-state and user-key sizes to $\mathcal{O}(\kappa \log_2 T)$. Note that the amortized cost of the Update algorithm in the binary tree construction is constant.

As the chaining and the trapdoor permutation schemes have worst-case complexities linear in T for at least one algorithm, both of them require the number of time intervals to be rather small. In contrast, the binary tree construction can be used for a practically unbounded number of time intervals.

In an application in which the number of time intervals in not known in advance, the incremental tree scheme can be used. Its space and time complexities only depend on the number of past revocations and not on the total number of revocations supported. The incremental tree construction is an interesting example of an additive composition of tree constructions with different number of intervals. Furthermore, our additive and multiplicative composition methods allow the construction of new schemes starting from the basic three constructions described in Section 4.

6 Experimental Evaluation

We have implemented the chaining, trapdoor, and tree constructions for 128-bit keys. We have used the 128-bit AES block cipher to implement the pseudorandom generator G as $G(s) = AES_s(0^{128})||AES_s(1^{128})$ with $|s| = 128$ for the CKU and TreeKU constructions of Sections 4.1 and 4.3. In construction TDKU from Section 4.2, we have used the RSA permutation with a bit length of 1024 and public exponent 3 and the SHA-1 hash function as the random oracle h.

We performed the following experiment. For a fixed total number of revocations T, the center first initializes the key-updating scheme. Then, the steps below are repeated for $t = 1, \ldots, T$:

- The center runs the Update and Derive algorithms to simulate one revocation.
- Given the user key for interval t, the user runs the Extract algorithm to obtain the key k_1 for the *first* time interval.

Note that the time to extract the key for the first interval is larger than the extraction time for any other interval between 1 and t in all three constructions. Hence, the extraction time for the first interval represents a worst-case measure. We measured the performance using four metrics: the maximum and average Update and Derive time for the center (over the T revocations), and the maximum and average Extract time for clients to compute the key for the first time interval (from one of the T time intervals). We ran our experiments on a 2.4 GHz Intel Xeon processor machine, running Linux 2.6. Our unoptimized implementation was written in C++ using gcc 3.2.1.

The results are presented in Figures 7, 8, and 9, respectively. The graphs show the measured time as a function of the total number of revocations T, which ranges from 2^8 to 2^{25} depending on the scheme. Note that the both axis are logarithmic and that the vertical axis differs for the three constructions. In the chaining construction, the cost of

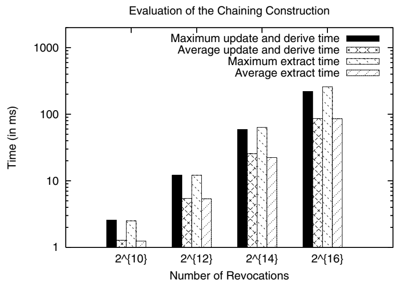


Fig. 7. Evaluation of the chaining scheme

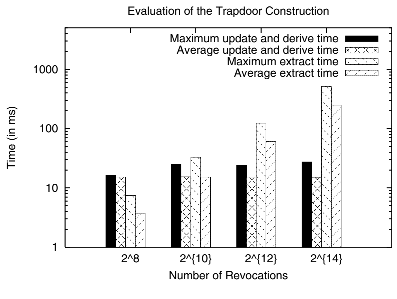


Fig. 8. Evaluation of the trapdoor scheme

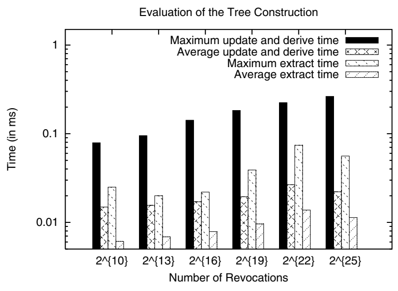


Fig. 9. Evaluation of the tree scheme

both the center and client computation increases linearly with the total number of revocations, as expected. In the trapdoor permutation construction, the center time is always constant, but the extraction time grows linearly with the total number of revocations. In the tree construction, all four metrics have a logarithmic dependence on the total number of revocations. We observe that the tree construction performs several orders of magnitude better than the other schemes.

Table 2 gives a direct comparison of the constructions in an experiment with 1024 revocations as above. It contains also the timing measurements for the first 1024 revocations in the tree construction where the upper bound T on number of revocations was set to a much larger value. This makes it possible to relate the tree construction to the trapdoor permutation scheme, which has no fixed upper bound on the number of revocations. It is evident that the tree scheme performs much better than the other schemes, even with a bound on the number of revocations that allows a practically unlimited number of them.

Table 2. Evaluation of the three constructions for 1024 revocations

Scheme	T	Maximum Time Update+Derive (ms)	Average Time Update+Derive (ms)	Maximum Time Extract (ms)	Average Time Extract (ms)
Chaining	1024	2.57	1.28	2.5	1.24
Trapdoor	1024	25.07	15.36	32.96	15.25
Tree	1024	0.079	0.015	0.025	0.006
Tree	2^{16}	0.142	0.015	0.018	0.0076
Tree	2^{25}	0.199	0.015	0.02	0.01

The space usage for $T = 1024$ is as follows. The center state is 16 bytes for the chaining construction, 384 bytes for the trapdoor construction, and at most 328 bytes for the tree scheme. The size of the user key is 32 bytes for the chaining construction, 128 bytes for the trapdoor construction, and at most 172 bytes for the tree scheme. In general, for the tree scheme with depth d, the center state takes at most $(2d - 1)(16 + d/8)$ bytes, containing $2d-1$ key value/key label pairs, assuming 16-byte keys and d-bit labels. The user key size is at most d key/label pairs, which take $d(16 + d/8)$ bytes.

In summary, we note that the performance of the tree scheme is superior to the others. The chaining construction has the smallest space requirements, but its computation cost becomes prohibitive for large T. The trapdoor construction has sligthly smaller space requirements than the tree scheme, but these savings are very small compared to the additional computational overhead.

7 Related Work

Time-Evolving Cryptography. The notion of secure key-updating schemes is closely related to forward- and backward-secure cryptographic primitives. Indeed, a secure key-updating scheme is forward-secure as defined originally by Anderson [4], in the sense that it maintains security in the time intervals following a key exposure. However, this is the opposite of the forward security notion formalized by Bellare and Miner [7] and used in subsequent work. Here we use the term forward security to refer to the latter notion.

Time-evolving cryptography protects a cryptographic primitive against key exposure by dividing the time into intervals and using a different secret key for every time interval. Forward-secure primitives protect past uses of the secret key: if a device holding all keys is compromised, the attacker can not have access to past keys. In the case of forward-secure signatures, the attacker can not generate past signatures on behalf of the user, and in the case of forward-secure encryption, the attacker can not decrypt old ciphertexts. There exist many efficient constructions of forward-secure signatures [7,2,21] and several generic constructions [24,26]. Bellare and Yee [8] analyze forward-secure private-key cryptographic primitives (forward-secure pseudorandom generators, message authentication codes and symmetric encryption) and Canetti, Halevi and Katz [10] construct the first forward-secure public-key encryption scheme.

Forward security has been combined with backward security in models that protect both the past and future time intervals, called key-insulated [13,14] and intrusion-resilient models [22,12]. In both models, there is a center that interacts with the user in the key update protocol. The basic key insulation model assumes that the center is trusted and the user is compromised in at most t time intervals and guarantees that the adversary does not gain information about the keys for the intervals the user is not compromised. A variant of this model, called strong key insulation, allows the compromise of the center as well. Intrusion-resilience tolerates arbitrarily many break-ins into both the center and the user, as long as the break-ins do not occur in the same time interval. The relation between forward-secure, key-insulated and intrusion-resilient signatures has been analyzed by Malkin, Obana and Yung [27]. A survey of forward-secure cryptography is given by Itkis [20].

Re-keying, i.e., deriving new secret keys periodically from a master secret key, is a standard method used by many applications. It has been formalized by Abdalla and Bellare [1]. The notion of key-updating schemes that we define is closely related to re-keying schemes, with the difference that in our model, we have the additional requirement of being able to derive past keys efficiently.

Multicast Key Distribution. In key distribution schemes for multicast, a group controller distributes a group encryption key to all users in a multicast group. The group of users is dynamic and each join or leave event requires the change of the encryption key. The goal is to achieve both forward and backward security. In contrast, in our model of key-updating schemes users should be able to derive past encryption keys efficiently.

A common key distribution model for multicast is that of *key graphs*, introduced by Wong et al. [32] and used subsequently in many constructions [30], [29], [19],[18]. In these schemes, each user knows its own secret key and, in addition, a subset of secret keys used to generate the group encryption key and to perform fast update operations. The relation between users and keys is modeled in a directed acyclic graphs, in which the source nodes are the users, intermediary nodes are keys and the unique sink node is the group encryption key. A path from a user node to the group key contains all the keys known to that user. The complexity and communication cost of key update operations is optimal for tree structures [31], and in this case it is logarithmic in the number of users in the multicast group. We also use trees for generating keys, but our approach is different in considering the nodes of the tree to be only keys, and not users. We obtain logarithmic update cost in the number of revocations, not in the number of users in the group.

Key Management in Cryptographic Storage Systems. Early cryptographic file systems [9,11] did not address key management. Cepheus [15] is the first cryptographic file system that considers sharing of files and introduces the idea of lazy revocation for improving performance. However, key management in Cepheus is centralized by using a trusted key server for key distribution. More recent cryptographic file systems, such as Oceanstore [25] and Plutus [23], acknowledge the benefit of decentralized key distribution and propose that key management is handled by file owners themselves. For efficient operation, Plutus adopts a lazy revocation model and uses a key-updating scheme based on RSA, as described in Section 4.2.

Farsite [3], SNAD [28] and SiRiUS [17] use public-key cryptography for key management. The group encryption key is encrypted with the public keys of all group members and these lockboxes are stored on the storage servers. This approach simplifies key management, but the key storage per group is proportional to the number of users in the group. Neither of these systems addresses efficient user revocation.

Independently and concurrently to our work Fu, Kamara, and Kohno [16] have proposed a cryptographic definition for key-updating schemes, which they call *key regression schemes*. Key regression schemes are, in principle, equivalent to key-updating schemes. Their work formalizes three key regression schemes: two constructions, one using a hash function and one using a pseudo-random permutation, are essentially equivalent to our chaining construction, and an RSA-based construction originating in Plutus, which is equivalent to our trapdoor-permutation construction. Our composition methods and the tree-based construction are novel contributions that go beyond their work.

8 Conclusions

Motivated by the practical problem of efficient key management for cryptographic file systems that adopt lazy revocation, we define formally the notion of key-updating schemes for lazy revocation and its security. In addition, we give two methods for additive and multiplicative composition of two secure key-updating scheme into a new scheme which can handle a larger number of user revocations, while preserving security. We also prove the security of two slightly modified existing constructions and propose a new construction, the binary-tree scheme, that balances the tradeoffs of the existing constructions. Finally, we provide a systematic analysis of the computational and communication complexities of the three constructions.

We can extend the definition of key-updating schemes to support user keys for interval t, from which only keys of the time intervals between i and t can be extracted, for any $1 \leq i \leq t$. This is useful in a model in which users joining the group at a later time interval should not have access to past information. The extension can be incorporated in the tree construction without additional cost, but the chaining and trapdoor permutation constructions do not work in this model because the user key reveals all previous keys.

In a companion paper [5], we show how to extend secure key-updating schemes to cryptosystems with lazy revocation, and introduce the notions of symmetric encryption, message-authentication codes, and signature schemes with lazy revocation. Furthermore, we demonstrate that using these cryptosystems in some existing distributed cryptographic file systems improves their efficiency and security.

References

1. M. Abdalla and M. Bellare, "Increasing the lifetime of a key: A comparitive analysis of the security of rekeying techniques," in *Proc. Asiacrypt 2000*, vol. 1976 of *Lecture Notes in Computer Science*, pp. 546–559, Springer-Verlag, 2000.
2. M. Abdalla and L. Reyzin, "A new forward-secure digital signature scheme," in *Proc. Asiacrypt 2000*, vol. 1976 of *Lecture Notes in Computer Science*, pp. 116–129, Springer-Verlag, 2000.
3. A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, "FARSITE: Federated, available, and reliable storage for an incompletely trusted environment," in *Proc. 5th Symposium on Operating System Design and Implementation (OSDI)*, Usenix, 2002.
4. R. Anderson, "Two remarks on public-key cryptology," Technical Report UCAM-CL-TR-549, University of Cambridge, 2002.
5. M. Backes, C. Cachin, and A. Oprea, "Lazy revocation in cryptographic file systems," in *Proc. 3rd Intl. IEEE Security in Storage Workhsop (SISW)*, 2005.
6. M. Backes, C. Cachin, and A. Oprea, "Secure key-updating for lazy revocation," Research Report RZ 3627, IBM Research, Aug. 2005. Appears also as Cryptology ePrint Archive, Report 2005/334.
7. M. Bellare and S. Miner, "A forward-secure digital signature scheme," in *Proc. Crypto 1999*, vol. 1666 of *Lecture Notes in Computer Science*, pp. 431–448, Springer-Verlag, 1999.
8. M. Bellare and B. Yee, "Forward-security in private-key cryptography," in *Proc. CT-RSA 2003*, vol. 2612 of *Lecture Notes in Computer Science*, pp. 1–18, Springer-Verlag, 2003.
9. M. Blaze, "A cryptographic file system for Unix," in *Proc. First ACM Conference on Computer and Communication Security (CCS)*, pp. 9–16, 1993.
10. R. Canetti, S. Halevi, and J. Katz, "A forward-secure public-key encryption scheme," in *Proc. Eurocrypt 2003*, vol. 2656 of *Lecture Notes in Computer Science*, pp. 255–271, Springer-Verlag, 2003.
11. G. Cattaneo, L. Catuogno, A. D. Sorbo, and P. Persiano, "The design and implementation of a transparent cryptographic file system for Unix," in *Proc. USENIX Annual Technical Conference 2001, Freenix Track*, pp. 199–212, 2001.

12. Y. Dodis, M. Franklin, J. Katz, A. Miyaji, and M. Yung, "Intrusion-resilient public-key encryption," in *Proc. CT-RSA 2003*, vol. 2612 of *Lecture Notes in Computer Science*, pp. 19–32, Springer-Verlag, 2003.
13. Y. Dodis, J. Katz, S. Xu, and M. Yung, "Key insulated public-key cryptosystems," in *Proc. Eurocrypt 2002*, vol. 2332 of *Lecture Notes in Computer Science*, pp. 65–82, Springer-Verlag, 2002.
14. Y. Dodis, J. Katz, and M. Yung, "Strong key-insulated signature schemes," in *Proc. Workshop of Public Key Cryptography (PKC)*, vol. 2567 of *Lecture Notes in Computer Science*, pp. 130–144, Springer-Verlag, 2002.
15. K. Fu, "Group sharing and random access in cryptographic storage file systems," Master's thesis, Massachusetts Institute of Technology, 1999.
16. K. Fu, S. Kamaram, and T. Kohno, "Key regression: Enabling efficient key distribution for secure distributed storage," in *Proc. Network and Distributed Systems Security Symposium (NDSS 2006)*, 2006.
17. E. Goh, H. Shacham, N. Modadugu, and D. Boneh, "SiRiUS: Securing remote untrusted storage," in *Proc. Network and Distributed Systems Security Symposium (NDSS 2003)*, pp. 131–145, 2003.
18. M. T. Goodrich, J. Z. Sun, and R. Tamassia, "Efficient tree-based revocation in groups of low-state devices," in *Proc. Crypto 2004*, vol. 3152 of *Lecture Notes in Computer Science*, pp. 511–522, Springer-Verlag, 2004.
19. J. Goshi and R. E. Ladner, "Algorithms for dynamic multicast key distribution trees," in *Proc. 22nd Symposium on Principles of Distributed Computing (PODC)*, pp. 243–251, ACM, 2003.
20. G. Itkis, "Forward security, adaptive cryptography: Time evolution." Survey, available from `http://www.cs.bu.edu/fac/itkis/pap/forward-secure-survey.pdf`.
21. G. Itkis and L. Reyzin, "Forward-secure signatures with optimal signing and verifying," in *Proc. Crypto 2001*, vol. 2139 of *Lecture Notes in Computer Science*, pp. 332–354, Springer-Verlag, 2001.
22. G. Itkis and L. Reyzin, "SiBIR: Signer-base intrusion-resilient signatures," in *Proc. Crypto 2002*, vol. 2442 of *Lecture Notes in Computer Science*, pp. 499–514, Springer-Verlag, 2002.
23. M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable secure file sharing on untrusted storage," in *Proc. 2nd USENIX Conference on File and Storage Technologies (FAST)*, 2003.
24. H. Krawczyk, "Simple forward-secure signatures from any signature scheme," in *Proc. 7th ACM Conference on Computer and Communication Security (CCS)*, pp. 108–115, 2000.
25. J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," in *Proc. 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 190–201, ACM, 2000.
26. T. Malkin, D. Micciancio, and S. Miner, "Efficient generic forward-secure signatures with an unbounded number of time periods," in *Proc. Eurocrypt 2002*, vol. 2332 of *Lecture Notes in Computer Science*, pp. 400–417, Springer-Verlag, 2002.
27. T. Malkin, S. Obana, and M. Yung, "The hierarchy of key evolving signatures and a characterization of proxy signatures," in *Proc. Eurocrypt 2004*, vol. 3027 of *Lecture Notes in Computer Science*, pp. 306–322, Springer-Verlag, 2004.
28. E. Miller, D. Long, W. Freeman, and B. Reed, "Strong security for distributed file systems," in *Proc. the First USENIX Conference on File and Storage Technologies (FAST)*, 2002.
29. O. Rodeh, K. Birman, and D. Dolev, "Using AVL trees for fault tolerant group key management," *International Journal on Information Security*, vol. 1, no. 2, pp. 84–99, 2001.

30. A. T. Sherman and D. A. McGrew, "Key establishment in large dynamic groups using one-way function trees," *IEEE Transactions on Software Engineering*, vol. 29, no. 5, pp. 444–458, 2003.
31. R. Tamassia and N. Triandopoulos, "Computational bounds on hierarchical data processing with applications to information security," in *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, 2005.
32. C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graphs," *IEEE/ACM Transactions on Networking*, vol. 8, no. 1, pp. 16–30, 2000.

Key Derivation Algorithms for Monotone Access Structures in Cryptographic File Systems

Mudhakar Srivatsa and Ling Liu

College of Computing, Georgia Institute of Technology
{mudhakar, lingliu}@cc.gatech.edu

Abstract. Advances in networking technologies have triggered the "storage as a service" (SAS) model. The SAS model allows content providers to leverage hardware and software solutions provided by the storage service providers (SSPs), without having to develop them on their own, thereby freeing them to concentrate on their core business. The SAS model is faced with at least two important security issues: (i) How to maintain the confidentiality and integrity of files stored at the SSPs? (ii) How to efficiently support flexible access control policies on the file system? The former problem is handled using a cryptographic file system, while the later problem is largely unexplored. In this paper, we propose secure, efficient and scalable key management algorithms to support monotone access structures on large file systems. We use key derivation algorithms to ensure that a user who is authorized to access a file, can efficiently derive the file's encryption key. However, it is computationally infeasible for a user to guess the encryption keys for those files that she is not authorized to access. We present concrete algorithms to efficiently and scaleably support a discretionary access control model (DAC) and handle dynamic access control updates & revocations. We also present a prototype implementation of our proposal on a distributed file system. A trace driven evaluation of our prototype shows that our algorithms meet the security requirements while incurring a low performance overhead on the file system.

1 Introduction

The widespread availability of networks, such as the Internet, has prompted a proliferation of both stationary and mobile devices capable of sharing and accessing data across networks spanning multiple administrative domains. Today, efficient data storage is vital for almost every scientific, academic, or business organization. Advances in the networking technologies have triggered the "storage as a service" (SAS) model [15][13]. The SAS model allows organizations to leverage hardware and software solutions provided by third party storage service providers (SSPs), thereby freeing them to concentrate on their core business. The SAS model decouples physical storage from file management issues such as access control and thus allows the file system to scale to a large number of users, files, and organizations. However, from the perspective of the organization (content owner), the SAS model should address at least two important security issues: (i) How to maintain the confidentiality & integrity of files stored at the SSPs? (ii) How to securely and efficiently support flexible access control policies on the file system?

Cryptographic File Systems. Cryptographic file systems address the first problem. These file systems essentially maintain the confidentiality and integrity of the file data by storing it in an encrypted format at the SSPs. With the advent of high speed hardware for encrypting and decrypting data, the overhead in a cryptographic file system due to file encryption and decryption is affordably small. Examples of cryptographic file systems include CFS [4], TCFS [7], CryptFS [29] and NCryptFS [28]. Examples of wide-area distributed cryptographic file systems include Farsite [2] and cooperative file system [8].

Access Control. Access control in a cryptographic file system translates into a secure key management problem. Cryptographic access control [12] is achieved by distributing a file's encryption key to only those users that are authorized to access that file. A read/write access to the files stored at the SSP is granted to all principals, but only those who know the key are able to decrypt the file data. However, there is an inherent tension between the cost of key management and the flexibility of the access control policies. At one extreme the access control matrix is highly flexible and can thus encode arbitrary access control policies (static). An access control matrix [17] is $(0, 1)$ matrix $M_{U \times F}$, where U is a set of users and F is a set of files and M_{uf} = 1 if and only if user u can access file f. Implementing cryptographic access control would require one key for every element M_{uf} such that M_{uf} = 1. This makes key management a challenging performance and scalability problem in a large file system wherein, access permissions may be dynamically granted and revoked.

Our Approach. The access control matrix representation of the access control rules does not scale well with the number of users and the number of files in the system. A very common strategy is to impose an access structure on the access control policies. An access structure, as the name indicates, imposes a structure on the access control policies. Given an access structure, can we perform efficient and scalable key management without compromising the access control policies in the file system? We propose to use an access structure to build a key derivation algorithm. The key derivation algorithm uses a *much smaller* set of keys, but gives the same effect as having one key for every M_{uf} = 1. The key derivation algorithm guarantees that a user u can use its small set of keys to efficiently derive the key for any file f if and only if M_{uf} = 1.

Monotone Access Structure. In this paper we consider access control policies based on monotone access structures. Most large enterprises, academic institutions, and military organizations allow users to be categorized into user groups. For example, let $\{g_1$, g_2, $g_3\}$ denote a set of three user groups. A user u can be a member of one or more groups denoted by G_u. Access control policies are expressed as monotone Boolean expressions on user groups. For example, a file f may be tagged with a monotone B_f = g_1 $\wedge$ (g_2 $\vee$ g_3). This would imply that a user u can access file f if and only if it belongs to group g_1 and either one of the groups g_2 or g_3. For example, if G_{u_1} = $\{g_1$, $g_3\}$ and G_{u_2} = $\{g_2$, $g_3\}$, then user u_1 can access file f, but not user u_2. Monotone access structures are a common place in role-based access control models [25]. In the RBAC model, each role (say, a physician or a pharmacist) is associated with a set of credentials. Files are associated with a monotone Boolean expression B_f on credentials. A role r can access a file f if and only if the credentials for role r satisfies the monotone B_f.

Our Contribution. In this paper, we propose secure, efficient and scalable key management algorithms that support monotone access structures in a cryptographic file system. (i) *Number of Keys:* We ensure that each user needs to maintain only a small number of keys. It suffices for a user to maintain only one key per group that the user belongs to. For example, a user u with $G_u = \{g_1, g_2\}$ needs to maintain keys one key corresponding to group g_1 and group g_2. (ii) *Efficient Key Derivation:* It is computationally easy for a user to derive the keys for all files that she is authorized to access. For example, a user who has the key for groups g_1 and g_2 can easily derive the encryption key for a file f with $B_f = g_1 \wedge (g_2 \vee g_3)$. (iii) *Secure Key Derivation:* It is computationally infeasible for a user to derive the key for any file that she is not authorized to access. For example, for a user who has the keys only for groups g_2 and g_3, it is infeasible to guess the encryption key for a file f with $B_f = g_1 \wedge (g_2 \vee g_3)$. (iv) *Discretionary Access Control (collusion resistance):* It is computationally infeasible for two or more colluding users to guess the encryption key for a file that none of them are independently authorized to access. For example, two colluding users u_1 with $G_{u_1} = \{g_1\}$ and u_2 with $G_{u_2} = \{g_3\}$ should not able to guess the encryption key for a file f with $B_f = g_1 \wedge (g_2 \vee g_3)$. (v) *Revocation:* Our key management algorithms support dynamic revocations of a user's group membership through cryptographic leases. A lease permits a user u to be a member of some group g from time a to time b. Our algorithms allow the lease duration (a, b) to be highly fine grained (say, to a millisecond precision).

Paper Outline. The following sections of this paper are organized as follows. Section 2 describes the SAS model and monotone access structures in detail. Section 3 presents a detailed design and analysis of our key management algorithms for implementing discretionary access control using monotone access structures in cryptographic file systems. Technical report [26] sketches an implementation of our key management algorithms on a distributed file system followed by trace-driven evaluation in Section 4. Finally, we present related work in Section 5 followed by a conclusion in Section 6.

2 Preliminaries

In this section, we present an overview of the SAS model. We explicitly specify the roles played by the three key players in the SAS architecture: content provider, storage service provider, and users. We also formally describe the notion of user groups and the properties of monotone access structures on user groups.

2.1 SAS Model

The SAS model comprises of three entities: the content provider, the storage service provider and the users.

Storage Service Providers (SSPs). Large SSPs like IBM and HP use high speed storage area networks (SANs) to provide large and fast storage solutions for multiple organizations. The content provider encrypts files before storing them at a SSP. The SSP serves only encrypted data to the users. The content provider does not trust the SSP with the confidentiality and integrity of file data. However, the SSP is trusted to perform read and write operations on the encrypted files. For performance reasons, each file is divided into multiple data blocks that are encrypted separately. An encrypted data block is the smallest granularity of data that can be read or written by a user or a content provider.

Content Provider. The content provider is responsible for secure, efficient and scalable key management. We assume that there is a secure channel between the group key management service and the users. This channel is used by the content provider to distribute keys to the users. We assume that the channel between the content provider & the SSP and that between the users & the SSP could be untrusted. An adversary would be able to eavesdrop or corrupt data sent on these untrusted channels. The content provider also includes a file key server. The users interact with the file key server to derive the encryption keys for the files they are authorized to access. The channel between the user and the file key server may be untrusted. In the following sections of this paper, we present an efficient, scalable and secure design for the file key server.

Users. We use an honest-but-curious model for the users. Content providers authorize users to access certain files by securely distributing appropriate keys to them. Let $K(f)$ denote the encryption key used to encrypt file f. If a user u is authorized to access file f, then we assume that the user u would neither distribute the key $K(f)$ nor the contents of the file f to an unauthorized user. However, a user u' who is not authorized to access file f would be curious to know the file's contents. We assume that unauthorized users may collude with one another and with the SSP. Unauthorized users may eavesdrop or corrupt the channel between an authorized user and the SSP. We use a discretionary access control (DAC) model to formally study collusions amongst users. Under the DAC model the set of files that is accessible to two colluding users u_1 and u_2 should be no more than the union of the set of files accessible to the user u_1 and the user u_2. Equivalently, if a file f is accessible neither to user u_1 nor to user u_2 then it should remain inaccessible even when the users u_1 and u_2 collude with one another.

2.2 Monotone Access Structures

In this section, we describe monotone access structures based access control policies. Our access control policies allow files to be tagged with monotone Boolean expressions on user groups. Let $G = \{g_1, g_2, \cdots, g_s\}$ denote a set of s user groups. A user may be a member of one or more user groups. Each file f is associated with a monotone Boolean expression B_f. For example, $B_f = g_1 \wedge (g_2 \vee g_3)$ would mean that the file f is accessible by a user u if and only if u is a member of group g_1 and a member of either group g_2 or group g_3.

We require that the Boolean expression B_f be a *monotone*. This assumption has several consequences: (i) Let G_u denote the set of groups to which user u belongs. Let $B_f(G_u)$ denotes $B_f(g_1, g_2, \cdots, g_s)$ where $g_i = 1$ if the group $g_i \in G_u$ and $g_i = 0$ otherwise. For two users u and v if $G_u \subseteq G_v$ then $B_f(G_u) \Rightarrow B_f(G_v)$. (ii) Let us suppose that a user u is authorized to access a set of files F. If the user u were to obtain membership to additional groups, it does not deny u access to any file $f \in F$ (monotone property). (iii) For all files f, B_f can be expressed using only $\wedge$ and $\vee$ operators (without the NOT ($\sim$) operator) [18]. (iv) Access control policies specified using monotone Boolean expressions are easily tractable. Let G_u denote the set of groups to which user

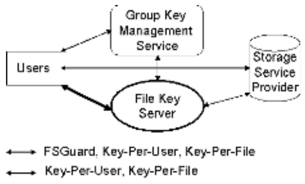

Group Key Management Service
Users
Storage Service Provider
File Key Server
FSGuard, Key-Per-User, Key-Per-File
Key-Per-User, Key-Per-File

Fig. 1. FSGuard Architecture

Table 1. Comparison of Key Management Algorithms

	File Access	B_f Update	G_u Update	Num keys per User	Storage Overhead
FSGuard	1cpu + 1net	-	-	4	1disk
Key-per-User	-	10^3cpu + 10net	10^6cpu + 10^4net	1	20disk
Key-per-File	-	10^3cpu + 10net	10^6cpu + 10^4net	10^4	1disk

u belongs. Then, one can efficiently determine whether the user u can access a file f by evaluating the Boolean expression $B_f(G_u)$. Note that evaluating a Boolean expression on a given input can be accomplished in $O(|B_f|)$ time, where $|B_f|$ denotes the size of the Boolean expression B_f.

3 User Groups

3.1 Overview

Figure 1 shows the entities involved in our design. The core component of our design is the file key server. We use the file key server for securely, efficiently and scaleably managing the file encryption keys. A high level description of our key management algorithm is as follows. Each file f is encrypted with a key $K(f)$. The key $K(f)$ is encrypted with a key encryption key $KEK(f)$. The encrypted file is stored at the SSP. The content owner stores the key encryption keys in the trusted file key server in a *compressed format*. The key server can use the stored information to efficiently derive the key encryption keys on the fly (Section 3.4) and distributes a secure transformation of the $KEKs$ to the users. A transformation on $KEK(f)$ is secure if the transformed version can be made publicly available (to all users and the SSP) without compromising the access control guarantees of the file system (Sections 3.2 and 3.3). We handle dynamic revocations of file accesses to users using a novel authorization key tree [26]. For comparison purposes, we describe two simple key management algorithms in this section: key-per-user and key-per-file.

Key-per-User. The key-per-user approach associates a secret key $K(u)$ with user u. For any file f, the key server determines the set of users that are permitted to access file f based on the group membership of user u and the monotone B_f. For all users u that can access file f, the key server stores $E_{K(u)}(KEK(f))$ along with the attributes of file f at the SSP. Note that $E_K(x)$ denotes a symmetric key encryption of input x using an encryption algorithm E (like DES [10] or AES [21]) and key K. However, such an implementation does not scale with the number of files and users in the system since the key server has to store and maintain updates on $KEK(f)$ for all f, B_f for all f, $K(u)$ for all u, and G_u for all u. For example, if G_u changes for any u, the key server needs to inspect all the files in the system before determining the set of files to which the user u's access needs to be granted or revoked. For all files f, whose access is either granted to user u, the key server has to add $E_{K(u)}(KEK(f)$ to its attribute. For all files f, whose access is revoked to user u, the key server has to update $KEK(f)$ (to say,

$KEK'(f)$); the key server has to add $E_{K(u')}(KEK'(f))$ for all other users u' that are allowed to access file f.

Key-per-File. The second approach is the key-per-file approach. This approach associates a key $K(f)$ with file f. For each user u, the key server determines the set of files that the user is permitted to access based on the group membership of the user u and the monotone B_f. We use the key server to distribute $KEK(f)$ to all users that are permitted to access the file f. We use a group key management protocol [20] to update $KEK(f)$ as the set of users permitted to access file f varies. However, the key-per-file approach also suffers from similar scalability problems as the key-per-user approach.

FSGuard. In this paper, we present our key management algorithms for implementing discretionary access control using monotone access structures in a cryptographic file system. As shown in Figure 1 our approach (FSGuard) does not require any communication between the key server & the group key management service and the key server & the SSP. Table 1 shows a rough cost comparison between our approach and other approaches. Our approach incurs a small processing (cpu) and networking (net) overhead for file accesses. The key-per-user and key-per-file approach incurs several orders of magnitude higher cost for updating a file's access control expression B_f and updating a user's group membership G_u. The average number of keys maintained by one user in key-per-file approach is several orders of magnitude larger than our approach and the key-per-user approach. The storage overhead at the SSP in the key-per-user approach is at least one order of magnitude larger than our approach and the key-per-file approach.

3.2 Basic Construction

In this section, we present a basic construction for building a secure transformation. Recall that a transformation on $KEK(f)$ is secure if the transformed version can be made publicly available (to all users and the SSP) without compromising the access control guarantees of the file system. The basic construction assumes that users do not collude with one another and that the access control policies are static with respect to time. Further, the basic construction incurs a heavy communication cost between the key server and the group key management service. We remove these restrictions in later Sections 3.3 and 3.4.

The key idea behind the basic construction is to transform the $KEK(f)$ such that a user u can reconstruct $KEK(f)$ if and only if the user u satisfies the condition B_f. Our construction is based on generalized secret sharing scheme presented in [3]. We assume that all keys are 128-bits long and all integer arithmetic is performed in a 128-bit integer domain (modulo 2^{128}). We use $K(g)$ to denote the group key for group g. When a user u joins group g, it gets the group key $K(g)$ from the group key management service via a secure channel. In this section, we assume a non-collusive setting: a user u knows $K(g)$ if and only if user u is a member of group g. We extend our algorithm to permit collusions in Section 3.3.

Given a monotone Boolean expression B_f we mark the literals in the expression as follows. The i^{th} occurrence of a literal g in the expression B_f is marked as g^i. For example, $B_f = (g_1 \vee g_2) \wedge (g_2 \vee g_3) \wedge (g_3 \vee g_4)$ is marked as $(g_1^1 \vee g_2^1) \wedge (g_2^2 \vee$

$g_3^1) \wedge (g_3^2 \vee g_4^1)$. The key server published $T(KEK(f), B_f)$, where the transformation function T is recursively defined as follows:

$$\begin{aligned}
T(x, A_1 \wedge A_2) &= T(x_1, A_1) \cup T(x_2, A_2) \text{ such that } x_1 + x_2 = x \\
T(x, A_1 \vee A_2) &= T(x, A_1) \cup T(x, A_2) \\
T(x, g^i) &= x + H_{salt}(K(g), i)
\end{aligned}$$

The symbols A_1 and A_2 denote arbitrary monotone Boolean expressions. The $\cup$ denotes the union operator and $+$ denotes the modular addition operator on a 128-bit integer domain. For the Boolean $\wedge$ operator, we chose x_1 and x_2 randomly such that x_1 + x_2 = x. Observe that knowing only x_1 or x_2 does not give any information about x = x_1 + x_2. Note that H denotes a keyed pseudo-random function (PRF) (like HMAC-MD5 or HMAC-SHA1 [16]). The *salt* value is randomly chosen per file and is stored at the SSP along with the rest of the file f's attributes. The *salt* value is used as the key for the PRF H. The above construction can be easily extended to cases where the function T takes more than two arguments:

$$\begin{aligned}
T(x, \bigwedge_{i=1}^{n} A_i) &= \bigcup_{i=1}^{n} T(x_i, A_i) \text{ such that } \sum_{i=1}^{n} x_i = x \\
T(x, \bigvee_{i=1}^{n} A_i) &= \bigcup_{i=1}^{n} T(x, A_i) \\
T(x, g^i) &= x + H_{salt}(K(g), i)
\end{aligned}$$

Theorem 1. *The transformation T described in Section 3.2 secure in the absence of collusions amongst malicious users.*

Drawbacks. While the basic construction presents a secure transformation T, it has several drawbacks. First, the basic construction does not tolerate collusions among users. A collusion between two users u_1 and u_2 may result in *unauthorized privilege escalation*. For example, let us say that u_1 is a member of group g_1 and u_2 is a member of group g_2. By colluding with one another, users u_1 and u_2 would be able to access a file f with B_f = $g_1 \wedge g_2$, thereby violating the discretionary access control (DAC) model. Recall that in a DAC model, the set of files that is accessible to two colluding users u_1 and u_2 should be no more than the union of the set of files accessible to the user u_1 and the user u_2. Second, the key server needs to know $KEK(f)$ and B_f for all files in the system. In a static setting, wherein $KEK(f)$ and B_f do not change with time, this incurs heavy storage costs at the key server. In a dynamic setting, this incurs heavy communication, synchronization and consistency maintenance costs in addition to the storage cost. Note that in a dynamic setting, the key server has to maintain up to date information on $KEK(f)$ and B_f for all files in the system.

3.3 Collusion Resistant Construction

In this section, we present techniques to tolerate malicious collusions between users. The key problem with our basic construction (Section 3.2) is that the authorization information given to a user u_3 that belongs to both groups g_1 and g_2 (namely, $K(g_1)$

and $K(g_2)$) is simply the union of the authorization information given to a user u_1 that belongs to group g_1 (namely, $K(g_1)$) and to a user u_2 that belongs to group g_2 (namely, $K(g_2)$). We propose that when an user u joins a group g, it gets two pieces of authorization information $K(g)$ and $K(u, g)$. The key $K(u, g)$ binds user u to group g. However, using randomly chosen values for $K(u, g)$ does not scale with the number of users, since the group key management service and our key server would have to maintain potentially $|U| * |G|$ keys, where $|U|$ is the number of users and $|G|$ is the number of groups. We propose to mitigate this problem by choosing $K(u, g)$ pseudo-randomly. We derive $K(u, g)$ as $K(u, g) = H_{MK}(u, g)$, where MK is the master key shared between the group management service and the key server. For notational simplicity, we overload u and g to denote the u's user identifier and g's group identifier respectively.

Now, we modify the recursive definition of the transformation T described in Section 3.2 as follows:

$$T(x, u, \bigwedge_{i=1}^{n} A_i) = \bigcup_{i=1}^{n} T(x_i, u, A_i) \text{ such that } \sum_{i=1}^{n} x_i = x$$

$$T(x, u, \bigvee_{i=1}^{n} A_i) = \bigcup_{i=1}^{n} T(x, u, A_i)$$

$$T(x, u, g^i) = x + H_{salt}(K(u, g), i)$$

Theorem 2. *The transformation T described in Section 3.3 is secure and collusion resistant.*

3.4 Key Encryption Keys

We have so far described techniques to securely transform and distribute key encryption keys. However, a major scalability bottleneck still remains in the system. The key server needs to know $KEK(f)$ and B_f for all files in the file system. This incurs not only heavy storage costs, but also incurs heavy communication costs to maintain the consistency (up to date) of $KEK(f)$ and B_f. In this section, we propose to circumvent this problem as follows. We propose to derive $KEK(f)$ as a function of B_f. Hence, when a user u requests for $T(KEK(f), u, B_f)$, the key server first computes $KEK(f)$ as a function of B_f. Then, it uses the derived value for $KEK(f)$ to construct the $T(KEK(f), u, B_f)$ as described in Section 3.3. In the following portions of this section, we present a technique to derive $KEK(f)$ from B_f. Our technique maintains the semantic equivalence of monotone Boolean expressions, that is, for any two equivalent but non-identical representations of a monotone Boolean function B_f and B'_f, $KEK(B_f) = KEK(B'_f)$.

Preprocessing. Given a monotone Boolean expression B_f we normalize it as follows. We express B_f in a minimal conjunctive normal form (CNF) as $B_f = C_1 \wedge C_2 \cdots \wedge C_n$. $C_1 \wedge C_2 \cdots \wedge C_n$ is a minimal expression of B_f if for no $1 \leq i, j \leq n$ and $i \neq j$, $C_i \Rightarrow C_j$. Note that a monotone Boolean expression in its minimal form is unique up to a permutation on the clauses and permutation of literals within a clause. If not, let us suppose that $B_f = C_1 \wedge C_2 \cdots \wedge C_n = C'_1 \wedge C'_2 \cdots \wedge C'_{n'}$ be two distinct minimal

CNF representations of B_f. Then, there exists C_i such that $C_i \neq C'_j$ for all $1 \leq j \leq n'$. Setting all the literals in C_i to `false` sets the expression B_f to `false`. Hence, for $C'_1 \wedge C'_2 \cdots \wedge C'_{n'}$ to be an equivalent representation, there has to exist C'_j such that the literals in C'_j is a proper subset of the literals in C_i. Then, setting all the literals in C'_j to `false` sets B_f to `false`. Hence, for $C_1 \wedge C_2 \cdots \wedge C_n$ to be an equivalent representation, there has to exist $C_{i'}$ $(i \neq i')$ such that the literals in $C_{i'}$ is a proper subset of the literals in C'_j. Hence, the literals in $C_{i'}$ is a proper subset of the literals in C_i, that is, $C_{i'} \Rightarrow C_i$ $(i \neq i')$. This contradicts the fact that $C_1 \wedge C_2 \cdots \wedge C_n$ is a minimal CNF representation of the monotone Boolean expression B_f. We normalize the representation of each clause C_i as $g_{i_1} \vee g_{i_2} \vee \cdots \vee g_{i_m}$ such that $i_j < i_{j+1}$ for all $1 \leq j < m$.

Deriving $KEK(f)$. We compute $KEK(f)$ recursively as follows:

$$KC(C_i) = H_{MK}(i_1, i_2, \cdots, i_m) \text{ where } C_i = g_{i_1} \vee g_{i_2} \vee \cdots \vee g_{i_m} \text{ and } i_1 < i_2 < \cdots < i_m$$
$$KEK(B_f) = H_{MK}(KC(C_1) \oplus KC(C_2) \oplus \cdots \oplus KC(C_n)) \text{ where } B_f = C_1 \wedge C_2 \wedge \cdots \wedge C_n$$
$$KEK(f) = H_{MK}(KEK(B_f), salt)$$

Note that MK is a master key used by the key server. The $salt$ value is an auxiliary attribute associated with the file f. The PRF H is neither commutative nor associative; hence, we impose an arbitrary total order on groups using their group number. The $\oplus$ operator is both commutative and associative; hence, the order of the clauses in B_f does not affect $KEK(B_f)$. Hence, given any two equivalent representations of a monotone Boolean function $B_f = B'_f$, our algorithm computes the same key encryption key.

Security Analysis. It is easy to see that a user u who is authorized to access file f can easily recover $KEK(f)$ from $T(KEK(f), u, B_f)$. Let us suppose that a user u is not authorized to access file f. The user can present incorrect inputs since, the inputs are not authenticated by the key server. Recall that the key server accepts three inputs $salt$, u and B_f. Let us suppose that a user u sends an incorrect input u'. By the property of the secure transformation function T (Section 3.3), the user u cannot guess $KEK(f)$ from $T(KEK(f), u', B_f)$. Even if the users u and u' were to collude, we have shown in Section 3.3 that they can obtain $KEK(f)$ if and only if either u or u' is indeed authorized to access the file f. Let us suppose that a user u sends an incorrect input B'_f. By the description of our key derivation algorithm in this Section, using an incorrect B'_f results in an incorrect $KEK'(f)$. Indeed the properties of the PRF H ensures that the user u cannot guess $KEK(f)$ from $KEK'(f)$. The same argument also applies if the user u were to send an incorrect input $salt'$. Hence, given one or more outputs from the key server, a user u can construct $KEK(f)$ if and only if the user u is authorized to access the file f, that is, $B_f(G_u)$ = `true`.

The key server exports only one interface that accepts the file's $salt$, u and B_f as inputs and returns a secure transformation of $KEK(f)$, namely, $T(KEK(f), u, B_f)$ as output. The key server does not have to interact with either the group key management service to maintain $KEK(f)$ and B_f for all files f or G_u and $K(u, g)$ for all users u and groups g. This large minimizes the storage costs, communication costs, synchronization and consistency management costs in a dynamic setting and largely improves the scalability of the key server.

Table 2. Parameters

Parameter	Default	Description
nf	10^7	number of files
nu	1000	number of users
ng	32	number of groups
nug	zipf(1, 10)	number of groups per user
nc	zipf(2, 4)	number of clauses in B_f
nl	zipf(2, 4)	number of literals per clause
δt	1	time granularity (seconds)

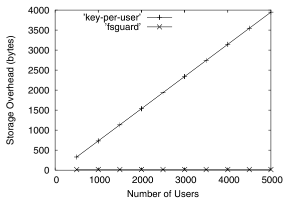


Fig. 2. Storage Overhead

4 Evaluation

In this section, we present a concrete evaluation of our prototype implementation. We ran our prototype implementation on eight machines (550 MHz Intel Pentium III Xeon processor running RedHat Linux 9.0) connected via a high speed 100 Mbps LAN. We used six machines to operate as the file servers, one machines to operate as the client, and one machine operates as the key server.

We compare our approach with two other approaches: key-per-user and key-per-file approach (see Section 3.1). We evaluate the performance of our proposal using four performance metrics: number of keys per user, storage cost at SSP, communication cost for various file system operations (file access, file's access control expression update, user's group membership update), and computation cost for various file system operations (file access, file's access control expression update, user's group membership update). We perform trace driven evaluations using the SPECsfs workload generator [1] of our approach to study the scalability of the key server and the performance overhead of our approach on a cryptographic file system. We used a synthetic file system with 10 million files, 1000 users, and 32 user groups. We assume that the group popularity follows a Zipf distribution [24], that is, the number of users that are a member of group i ($1 \leq i \leq 32$) is proportional to $\frac{1}{i}$. We assume that the number of clauses in any monotone B_f follows a Zipf distribution between 2 to 4 and the number of literals per clause follows a Zipf distribution between 2 to 4. Table 2 summarizes our main file system parameters.

4.1 Storage, Computation and Communication Costs

Number of Keys per User. In our first experiment, we measure the average number of keys maintained by a user using the three approaches. As the number of keys per user increases, so does the cost of managing those keys. Also, requiring a user to maintain a large number of keys increases the risk of one more keys being lost or accidentally leaked to an adversary. The key-per-user approach requires the user to store only one key. Our approach requires the user to store one key per group; we found that the average number of keys per user was 3.78.

The key-per-file approach requires the user to store one key per file that it is permitted to access. The average number of keys per user in this case is about $4.2 * 10^5$. [20][22] propose techniques to cluster files (termed file groups) based on their similarity. One can cluster files based on their access control expression B_f: all files in a cluster have identical (equivalent) B_f. We found that amongst 10 million files, there were $1.3 * 10^5$ unique

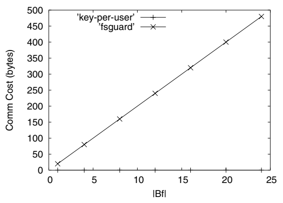


Fig. 3. Communication Cost: File Access

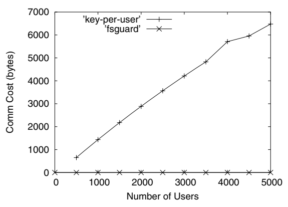


Fig. 4. Communication Cost: File Access Control Expression Update

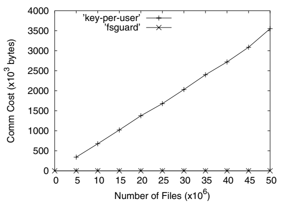


Fig. 5. Communication Cost: User Group Membership Update

monotones B_f: hence, we had $1.3 * 10^5$ file clusters with 1-337 files per cluster. We found that even with the clustering mechanism, the number of keys per user was about $2.3 * 10^4$. Because of the practical infeasibility of the key-per-file approach, the rest of our experiments focus exclusively on the key-per-user approach and our proposal.

Storage Cost as SSP. In our second experiment, we study the storage overhead at the SSP for storing additional file attributes. The key-per-user approach requires that we store $E_{K(u)}(K(f))$ per file block for all users u that is permitted to access the file f. Our approach stores only attribute $E_{KEK(f)}(K(f))$ (16 Bytes). Under the default settings described in Table 2, we found that the average number of users that can access a file was 45.7. Hence, each file block (8 KB) stored on the SSP the key-per-user approach incurs about 45.7 * 16 Bytes = 731.2 Bytes overhead (8.9%), while our approach (`fsguard`) incurs only a 16 Byte overhead (0.2%). As the number of users increase, the size of attributes stored with a file increases. Figure 2 shows the average size of a file's attribute as the number of users varies. Observe that as the numbers of users become 5000, the attribute size is about 4 KB. Using 8 KB file blocks, at least 50% of the storage space on the SSP would be expended on storing file attributes.

Communication Cost. In our third experiment, we measure the communication cost for three important operations: file access (read/write), update on a file's access control expression, and update on a user's group memberships.

File Access (read/write). A file access in the key-per-user approach does not involve any interaction between the user and the key server. The user fetches the file block and $E_{K(u)}(K(f))$ from the SSP and performs read/write operations on the block. On the other hand, file access in our approach requires the user to interact with the key server if the file encryption key $K(f)$ is not available in the user's local key cache. Observe that the communication cost between the user and the key server is $O(|B_f|)$, where $|B_f|$ denotes the number of literals in the monotone expression B_f. For example, $|(g_1 \vee g_2) \wedge (g_1 \vee g_3)| = 4$. Figure 3 shows the communication cost between the user and the key server for different values of $|B_f|$. Observe that even for complex (large) monotones, the communication cost is about a few hundred bytes.

File Access Control Expression Update. Let B_f and B'_f denote the old the new access control expression for file f. In the key-per-user approach, the key server has to determine the set of users U and U' whose group membership satisfies the expression

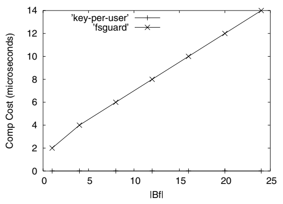


Fig. 6. Computation Cost: File Access

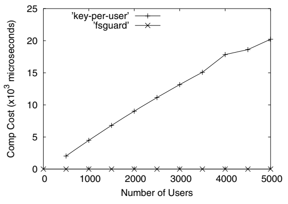


Fig. 7. Computation Cost: File Access Control Expression Update

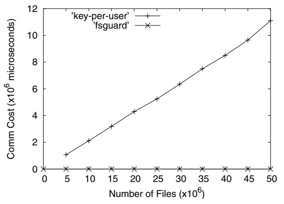


Fig. 8. Computation Cost: User Group Membership Update

B_f and B'_f respectively. For all $u \in U' - U$, the key server has to add $E_{K(u)}(K(f))$ to the file f's attribute. For all $u \in U - U'$, the key server has to remove $E_{K(u)}(K(f))$ from the file f's attribute. On the next write operation on file f, the key server needs to update $K(f)$ to a new key $K'(f)$ and consequently add $E_{K(u)}(K'(f))$ for all $u \in U'$ as attributes of file f. Note that the old attributes $E_{K(u)}(K(f))$ for all $u \in U$ can be deleted by the SSP. Using our approach, an update to the file's access control expression does not incur any communication cost. Recall that the interface exported by the key server operates on B_f rather than f itself. Figure 4 shows the communication cost between the key server and the SSP as nu, the number of users vary. Observe that as the number of users increase, the communication cost on the key server increases. This largely limits the scalability of the key server with the number of users in the file system. Observe from Figures 3 and 4 that an update on a file's access control expression costs about 1000 times the cost of a file access incurred by our approach.

User Group Membership Update. Let us suppose that a user u's group membership changed from G to G'. In the key-per-user approach, the key server has to determine the set of files F and F' whose access control expression is satisfied by group membership G and G' respectively. For all files $f \in F' - F$, the key server has to add $E_{K(u)}(K(f))$ to the file f's attribute. For all files $f \in F - F'$, the key server has to remove $E_{K(u)}(K(f))$ from the file f's attribute. On the next write operation on any file $f \in F - F'$, the key server needs to update $K(f)$ to a new key $K'(f)$. Consequently the key server has to add $E_{K(u')}(K'(f))$ as an attribute for the file f for all users u' that can access file f. Using our approach, an addition to a user's group membership requires an interaction with the group key management service. Revocation of a group membership does not require any communication using our algorithm in [26]. Figure 5 shows the communication cost as nf, the number of files vary. Using the key-per-user approach, the communication cost incurred in updating one user's group membership grows linearly with the number of files in the system and is of the order of several megabytes. This largely limits the scalability of the key server with the number of files in the system. Observe from Figures 3 and 5 that an update on a user's group membership costs about million times the cost of a file access incurred by our approach.

Computation Cost. In our fourth experiment, we measure the computation cost for three important operations: file access (read/write), update on a file's access control expression, and update on a user's group memberships. The computation cost is divided between the key server and the user. We computation cost is expressed in seconds as measured using a 550 MHz Intel Pentium III Xeon processor running RedHat Linux 9.0. Figures 6, 7 and 8 shows the computation cost at the client and the key server for the three operations listed above. Similar to the communication cost, our approach incurs computation cost only for file read/write operations. Further, this computation cost is incurred only if the file's key is not available in the user's cache. The key-per-user approach imposes heavy computation cost when a file's access control expression is updated or when a user's group membership is updated. An update on a file's access control expression costs about 1000 times the cost of a file access incurred by our approach; an update on a user's group membership costs about million times the cost of a file access incurred by our approach.

5 Related Work

Advances in the networking technologies have triggered several networking services such as: 'software as a service' also referred to as the application service provider (ASP) model [14], 'database as a service' (DAS) [11] that permits organizations to outsource their DBMS requirements, and 'storage as service' (SAS) model. The SAS model inherits all the advantages of the ASP model, indeed even more, given that a large number of organizations have their own storage systems. This model allows organizations to leverage hardware and software solutions provided by the service providers, without having to develop them on their own, thereby freeing them to concentrate on their core businesses. However, implementing flexible access control mechanisms and protecting the confidentiality from a storage service provider (SSP) has been a critical problem in the SAS model.

Cryptographic file systems like CFS [4], TCFS [7], CryptFS [29], NCryptFS [28], Farsite [2], StegFS [19], cryptographic disk driver [9] and cooperative file system [8] permit the file data to be kept confidential from the SSP. These file systems were designed with the goal of data confidentiality, while balancing scalability, performance and convenience. However, these systems were not designed with the goal of supporting flexible access control policies.

Cryptographic access control [12] make it possible for one to rely exclusively on cryptography to ensure confidentiality and integrity of data stored in the system. Data are encrypted as the applications store them on a server, which means that the storage system only manages encrypted data. Read/Write access to the physical storage device is granted to all principals (only those who know the key are able to decrypt the data). Cryptographic access control has been deployed to maintain secrecy in group key management protocols [27][5][6][23]. However, the access control policies that could be specified using cryptographic access control mechanisms were naive and inflexible. In this paper we have proposed techniques to implement monotone structure based access control policies in a cryptographic file system.

6 Conclusion

In this paper we have presented – secure, efficient and scalable mechanisms to enforce discretionary access control using monotone access structures on a cryptographic file system. We have presented key derivation algorithms that guarantee that a user who is authorized to access a file, can efficiently derive the file's encryption key; while, it is computationally infeasible for a user to guess the encryption keys associated with the files that she is not authorized to access. We have also presented concrete algorithms to support dynamic access control updates & revocations. We have also presented a prototype implementation of our proposal on a distributed file system. A cost based evaluation of our system showed that our approach incurs lower key management, storage, communication and computation cost when compared to the key-per-user and key-per-file approach. A trace driven evaluation of our prototype showed that our algorithms meet the security requirements while preserving the performance and scalability of the file system.

Acknowledgements. This research is partially supported by NSF CNS CCR, NSF ITR, DoE SciDAC, CERCS Research Grant, IBM Faculty Award, IBM SUR grant, and HP Equipment Grant.

References

1. SPEC SFS (system file server) benchmark. http://www.spec.org/osg/sfs97r1.
2. A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: Federated, available and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th International Symposium on OSDI*, 2002.
3. J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *Proceedings of CRYPTO*, 1988.
4. M. Blaze. A cryptographic file system for unix. In *Proceedings of ACM CCS*, 1993.
5. R. Canetti, J. Garay, G. Itkis, and D. Micciancio. Multicast security: A taxonomy and some efficient constructions. In *Proceedings of the IEEE INFOCOM, Vol. 2, 708-716*, 1999.
6. R. Canetti, T. Malkin, and K. Nissim. Efficient communication-storage tradeoffs for multicast encryption. In *Advances in Cryptology - EUROCRYPT. J. Stem, Ed. Lecture Notes in Computer Science, vol. 1599, Springer Verlag, pp: 459-474*, 1999.
7. G. Cattaneo, L. Catuogno, A. D. Sorbo, and P. Persiano. The design and implementation of transparent cryptographic file system for unix. In *Proceedings of Annual USENIX Technical Conference*, 2001.
8. F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM SOSP*, October 2001.
9. R. Dowdeswell and J. Ioannidis. The cryptographic disk driver. In *Proceedings of Annual USENIX Technical Conference*, 2003.
10. FIPS. Data encryption standard (DES). http://www.itl.nist.gov/fipspubs/fip46-2.htm.
11. H. Hacigumus, B. Iyer, and S. Mehrotra. Providing database as a service. In *Proceedings of 18th IEEE ICDE*, 2002.
12. A. Harrington and C. Jensen. Cryptographic access control in a distributed file system. In *Proceedings of the 8th ACM SACMAT*, 2003.
13. HP. Data center services. http://h20219.www2.hp.com/services/cache/114078-0-0-225-121.aspx.
14. IBM. Application service provider business model. http://www.redbooks.ibm.com/abstracts/sg246053.html.

15. IBM. IBM datacenter scalable offering. http://www-03.ibm.com/servers/eserver/xseries/windows/datacenter/scalable.html.
16. H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. http://www.faqs.org/rfcs/rfc2104.html.
17. B. Lampson. Protection. In *Proceedings of the 5th Princeton Symposium on Information Sceinces and Systems, pp: 437-443*, 1971.
18. Mathpages. Generating monotone boolean functions. http://www.mathpages.com/home/kmath094.htm.
19. A. D. McDonald and M. G. Kuhn. Stegfs: A steganographic file system for linux. In *Information Hiding, pp: 462-477*, 1999.
20. S. Mittra. Iolus: A framework for scalable secure multicasting. In *Proceedings of ACM SIGCOMM*, 1997.
21. NIST. AES: Advanced encryption standard. http://csrc.nist.gov/CryptoToolkit/aes/.
22. L. Opyrchal and A. Prakash. Secure distribution of events in content-based publish subscribe system. In *Proceedings of the 10th USENIX Security Symposium*, 2001.
23. A. Perrig, D. Song, and J. D. Tygar. ELK: A new protocol for efficient large group key distribution. In *Proceedings of IEEE Symposium on Security and Privacy*, 2001.
24. C. roadknight, I. Marshall, and D. Vearer. File popularity characterization. In *Proceedings of the 2nd Workshop on Internet Server Performance*, 1999.
25. R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. In *IEEE Computer, Vol. 29, No. 2*, 1996.
26. M. Srivatsa and L. Liu. Key derivation algorithms for monotone access structures in large file systems. Technical report, College of Computing, Georgia Tech, 2006.
27. C. K. Wong, M. G. Gouda, and S. S. Lam. Secure group communications using key graphs. In *IEEE/ACM Transactions on Networking: 8, 1(Feb), 16-30*, 2000.
28. C. P. Wright, M. C. Martino, and E. Zadok. Ncryptfs: A secure and convinient cryptographic file system. In *Proceedings of Annual USENIX Technical Conference*, 2003.
29. E. Zadok, I. Badulescu, and A. Shender. Cryptfs: A stackable vnode level encryption file system. Technical Report CUCS-021-98, Columbia University, 1998.

Cryptographically Sound Security Proofs for Basic and Public-Key Kerberos[⋆]

M. Backes[1], I. Cervesato[2], A.D. Jaggard[3], A. Scedrov[4], and J.-K. Tsay[4]

[1] Saarland University
backes@cs.uni-sb.de
[2] Deductive Solutions
iliano@deductivesolutions.com
[3] Tulane University
adj@math.tulane.edu
[4] University of Pennsylvania
{scedrov, jetsay}@math.upenn.edu

Abstract. We present a computational analysis of basic Kerberos and Kerberos with public-key authentication (PKINIT) in which we consider authentication and key secrecy properties. Our proofs rely on the Dolev-Yao style model of Backes, Pfitzmann and Waidner, which allows for mapping results obtained symbolically within this model to cryptographically sound proofs if certain assumptions are met. This is the most complex fragment of an industrial protocol that has yet been verified at the computational level. Considering a recently fixed version of PKINIT, we extend symbolic correctness results we previously attained in the Dolev-Yao model to cryptographically sound results in the computational model.

1 Introduction

Cryptographic protocols have traditionally been verified in one of two ways: the first, known as the Dolev-Yao or symbolic approach, abstracts cryptographic concepts into an algebra of symbolic messages [25]; the second, known as the computational or cryptographic approach, retains the concrete view of messages as bitstrings and cryptographic operations as algorithmic mappings between bitstrings, while drawing security definitions from complexity theory [16,26,27].

⋆ Backes was partially supported by the German Research Foundation (DFG) under grant 3194/1-1. Cervesato was partially supported by ONR under Grant N00014-01-1-0795. Jaggard was partially supported by NSF Grants DMS-0239996 and CNS-0429689, and by ONR Grant N00014-05-1-0818. Scedrov was partially supported by OSD/ONR CIP/SW URI "Software Quality and Infrastructure Protection for Diffuse Computing" through ONR Grant N00014-01-1-0795 and OSD/ONR CIP/SW URI "Trustworthy Infrastructure, Mechanisms, and Experimentation for Diffuse Computing" through ONR Grant N00014-04-1-0725. Additional support from NSF Grants CNS-0429689 and CNS-0524059. Tsay was partially supported by ONR Grant N00014-01-1-0795 and NSF grant CNS-0429689.

While proofs in the computational approach (with its much more comprehensive adversary model) entail stronger security guarantees, verification methods based on the Dolev-Yao abstraction have become efficient and robust enough to tackle large commercial protocols, often even automatically [1,15,18,34].

Kerberos, a widely deployed protocol that allows a user to authenticate herself to multiple end servers based on a single login, constitutes one of the most important examples that have been formally analyzed so far within the Dolev-Yao approach. Kerberos 4, which was then the prevalent version, was verified using the Isabelle theorem prover [15]. The currently predominant version, Kerberos 5 [39], has been extensively analyzed using the Dolev-Yao approach.This analysis of Kerberos 5 showed that: a detailed specification of the core protocol enjoys the expected authentication and secrecy properties except for some relatively innocuous anomalies [18]; "cross-realm" authentication in Kerberos is correct when compared against its specification but has weaknesses in practice [21]; and discovered a serious attack against the then-current specification of the public-key extension (PKINIT) of Kerberos [20]. The discovery of the attack on PKINIT led to an immediate correction of the specification and a security bulletin and patch for Microsoft Windows [36].

The proofs for both Kerberos 5 as well as the fixes to PKINIT are restricted to the Dolev-Yao approach, and currently there does not exist a theorem which allows for carrying the results of existing proofs of Kerberos over to the cryptographic domain with its much more comprehensive adversary. Thus, despite the extensive research dedicated to the Kerberos protocol, and despite its tremendous importance in practice, it is still an open question whether an actual implementation of Kerberos based on provably secure cryptographic primitives is secure under cryptographic security definitions. We close this gap (at least partially) by providing the first security proof of the core aspects of the Kerberos protocol in the computational approach. More precisely, we show that core parts of Kerberos 5 are secure against arbitrary active attacks if the Dolev-Yao-based abstraction of the employed cryptography is implemented with actual cryptographic primitives that satisfy the commonly accepted security notions under active attacks, e.g., IND-CCA2 for public-key encryption.

Obviously, establishing a proof in the computational approach presupposes dealing with cryptographic details such as computational restrictions and error probabilities, hence one naturally assumes that our proof heavily relies on complexity theory and is far out of scope of current proof tools. However, our proof is not performed from scratch in the cryptographic setting, but based on the Dolev-Yao style model of Backes, Pfitzmann, and Waidner [8,12,13] (called the *BPW model* henceforth), which provides cryptographically faithful symbolic abstractions of cryptographic primitives, i.e., the abstractions can be securely implemented using actual cryptography. Thus our proof itself is symbolic in nature, but refers to primitives from the BPW model. Kerberos is the largest and most complex protocol whose cryptographic security has so far been inferred from a proof in this Dolev-Yao style approach. Earlier proofs in this approach were mainly for small examples of primarily academic interest, e.g., the Needham-Schroeder-Lowe, the Otway-Rees, and the Yahalom protocols [4,7,11]; some similar work has been done on industrial protocols, e.g., [29], although none that are as complex as Kerberos. We furthermore analyze the recently fixed version of PKINIT and derive computational guarantees for it from a symbolic proof based on the BPW model. Finally we also draw some lessons learned in the process, which highlight areas where to focus research in order to simplify the verification of large commercial protocols with computational security guarantees. In particular it would be desirable to devise suitable proof techniques based on the BPW model for splitting large protocols into smaller pieces which can then be analyzed modularly while still retaining the strong link between the Dolev-Yao and the computational approach.

1.1 Related Work

Early work on linking Dolev-Yao models and cryptography [2,3,28] only considered passive attacks, and therefore cannot make general statements about protocols. A cryptographic justification for a Dolev-Yao model in the sense of simulatibility [40], i.e., under active attacks and within arbitrary surrounding interactive protocols, was first given in [12] with extensions in [8,13]. Based on that Dolev-Yao model, the well-known Needham-Schroeder-Lowe, Otway-Rees, and Yahalom protocols were proved secure in [4,7,11]. All these protocols are considerably simpler than Kerberos, which we analyze in this paper, and arguably of much more limited practical interest. Some work has been done on industrial protocols, such as 802.11i [29], although Kerberos is still a much more complex protocol.

Laud [33] has presented a cryptographic underpinning for a Dolev-Yao model of symmetric encryption under active attacks. His work is directly connected with a formal proof tool, but it is specific to certain confidentiality properties and protocol classes. Herzog et al. [30] and Micciancio and Warinschi [35] have also given a cryptographic underpinning under active attacks. Their results are narrower than those in [12] since they are specific for public-key encryption and certain protocol classes, but consider slightly simpler real implementations. Cortier and Warinschi [22] have shown that symbolically secret nonces are also computationally secret, i.e., indistinguishable from a fresh random value given the view of a cryptographic adversary. Backes and Pfitzmann [9] and Canetti and Herzog [19] have established new symbolic criteria for proving a key cryptographically secret. We stress that none of this work is comprehensive enough to infer computational security guarantees of Kerberos based on an existing symbolic proof; either they are missing suitable cryptographic primitives or rely on slightly changed symbolic abstractions, e.g., as in [12].

Finally, there is also work on formulating syntactic calculi for dealing with probability and polynomial-time considerations and encoding them into proof tools, in particular [17,23,32,37]. This is orthogonal to the work of justifying Dolev-Yao models.

1.2 Structure of the Paper

We start in Sect. 2 with a review of Kerberos and its public-key extension PKINIT. In Sect. 3, we recall the Dolev-Yao style model of Backes, Pfitzmann, and Waidner (e.g., [6,8,13,14]), and apply it to the specification of Kerberos 5 and Public-key Kerberos (i.e., Kerberos with PKINIT). Section 4 proves security results for these protocols and lift them to the computational level. Finally, Sect. 5 summarizes this effort and outlines areas of future work.

2 Kerberos 5 and Its Public-Key Extension

The Kerberos protocol [38,39] allows a legitimate user to log on to her terminal once a day (typically) and then transparently access all the networked resources she needs for the rest of that day. Each time she wants to, e.g., retrieve a file from a remote server, a Kerberos client running on her behalf securely handles the required authentication. The client acts behind the scenes, without any user intervention.

Kerberos comprises three subprotocols: the initial round of authentication, in which the client obtains a credential that might be good for a full day; the second round of authentication, in which she presents her first credential in order to obtain a short-term credential (five-minute lifetime) to use a particular network service; and the client's interaction with the network service, in which she presents her short-term credential in order to negotiate access to the service.

In the core specification of Kerberos 5 [39], all three subprotocols use symmetric (shared-key) cryptography. Since the initial specification of Kerberos 5, the protocol has been extended by the definition of an alternate first round which uses asymmetric (public-key) cryptography. This new subprotocol, called PKINIT [31], may be used in two modes: "public-key encryption mode" and "Diffie-Hellman (DH) mode." In recent work [20], we showed that there was an attack against the then-current draft specification of PKINIT when public-key encryption mode was used and then symbolically proved the security of the specification as it was revised in response to our attack. Here we study both basic Kerberos (without PKINIT) and the public-key mode of PKINIT as it was revised to prevent our attack.

Kerberos Basics. The client process—usually acting for a human user—interacts with three other types of principals when using Kerberos 5 (with or without PKINIT). The client's goal is to be able to authenticate herself to various application servers (e.g., email, file, and print servers). This is done by obtaining a "ticket-granting ticket" (TGT) from a "Kerberos Authentication Server" (KAS) and then presenting this to a "Ticket-Granting Server" (TGS) in order to obtain a "service ticket" (ST), the credential that the client uses to authenticate herself to the application server. A TGT might be valid for a day, and may be used to obtain several STs for many different application servers from the TGS, while a single ST is valid for a few minutes (although it may be used repeatedly) and is used for a single application server. The KAS and the TGS are together known as the "Key Distribution Center" (KDC).

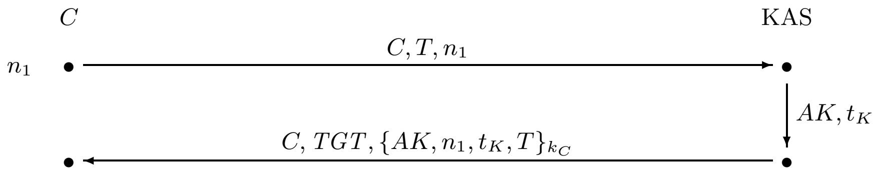


Fig. 1. Message Flow in the Traditional AS Exchange, where $TGT = \{AK, C, t_K\}_{k_T}$

The client's interactions with the KAS, TGS, and application servers are called the Authentication Service (AS), Ticket-Granting (TG), and Client-Server (CS) exchanges, respectively. We will describe the AS exchange separately for basic Kerberos and PKINIT; as PKINIT does not modify the other subprotocols, we only need to describe them once.

The Traditional AS Exchange. The abstract structure of the AS exchange is given in Fig. 1. A client C generates a fresh nonce n_1 and sends it, together with her own name and the name T of the TGS for whom she desires a TGT, to the KAS K. This message is called the AS_REQ message [39]. The KAS responds by generating a fresh authentication key AK for use between the client and the TGS and sending an AS_REP message to the client. Within this message, AK is sent back to the client in the encrypted message component $\{AK, n_1, t_K, T\}_{k_C}$; this also contains the nonce from the AS_REQ, the KAS's local time t_K, and the name of the TGS for whom the TGT was generated. (The AK and t_K to the right of the figure illustrate that these values are new between the two messages.) This component is encrypted under a long-term key k_C shared between C and the KAS; this key is usually derived from the user's password. This is the only time that this key is used in a standard Kerberos run because later exchanges use freshly generated keys. AK is also included in the ticket-granting ticket sent alongside the message encrypted for the client. The TGT consists of AK, C, t_K, where t_K is K's local time, encrypted under a long-term key k_T shared between the KAS and the TGS named in the request. The computational model we use here does not support timestamps, so we will treat these as nonces; this does not compromise our analysis as the timestamps that we include here are used like nonces. Once the client has received this reply, she may undertake the Ticket-Granting exchange.

It should be noted that the actual AS exchange, as well as the other exchanges in Kerberos, is more complex than the abstract view given here. We refer the reader to [39] for the complete specification of Kerberos 5, [31] for the specification of PKINIT, and [18] for a formalization of Kerberos at an intermediate level of detail.

The AS Exchange in PKINIT. PKINIT [31] is an extension to Kerberos 5 that uses public key cryptography to avoid shared secrets between a client and KAS; it modifies the AS exchange but not other parts of the basic Kerberos 5

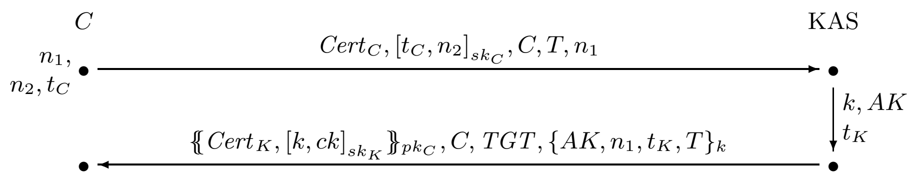

C
KAS
n_1, n_2, t_C
$Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1$
k, AK, t_K
$\{\!|Cert_K, [k, ck]_{sk_K}|\!\}_{pk_C}, C, TGT, \{AK, n_1, t_K, T\}_k$

Fig. 2. Message flow in the fixed version of PKINIT, where $TGT = \{AK, C, t_K\}_{k_T}$

protocol. The long-term shared key (k_C) in the traditional AS exchange is typically derived from a password, which limits the strength of the authentication to the user's ability to choose and remember good passwords; PKINIT does not use k_C and thus avoids this problem. Furthermore, if a public key infrastructure (PKI) is already in place, PKINIT allows network administrators to use it rather than expending additional effort to manage users' long-term keys as in traditional Kerberos.

In PKINIT, the client C and the KAS possess independent public/secret key pairs, (pk_C, sk_C) and (pk_K, sk_K), respectively. Certificate sets $Cert_C$ and $Cert_K$ issued by a PKI independent from Kerberos are used to testify of the binding between each principal and her purported public key. This simplifies administration as authentication decisions can now be made based on the trust the KDC holds in just a few known certification authorities within the PKI, rather than keys individually shared with each client (local policies can, however, still be installed for user-by-user authentication). Dictionary attacks are defeated as user-chosen passwords are replaced with automatically generated asymmetric keys.

PKINIT resembles the basic AS exchange in that the KAS generates a fresh key AK for the client and TGS to use, and then the KAS transmits AK and the TGT to the client. In public-key encryption mode, attacked and fixed in [20] and now analyzed here, the key pairs are used for both signature and encryption. The latter is designed to (indirectly) protect the confidentiality of AK, while the former ensures its integrity.

Figure 2 illustrates the AS exchange when the fixed version (which defends against the attack of [20]) of PKINIT is used. Here we use $[m]_{sk}$ for the digital signature of message m with secret key sk, $\{\!|m|\!\}_{pk}$ for the encryption of m with the public key pk, and $\{m\}_k$ for the encryption of m with the symmetric key k.

The first line of Fig. 2 shows our formalization of the AS_REQ message that a client C sends to a KAS K when using PKINIT. The last part of the message—C, T, n_1—is exactly as in the traditional AS_REQ. The new data added by PKINIT are the client's certificates $Cert_C$ and her signature (with her secret key sk_C) over a timestamp t_C and another nonce n_2.

The second line in Fig. 2 shows our formalization of K's response, which is more complex than in basic Kerberos. The last part of the message—C, TGT, $\{AK, n_1, t_K, T\}_k$—is very similar to K's reply in basic Kerberos; the difference is that the symmetric key k protecting AK is now freshly generated by K and is not a long-term shared key. Because k is freshly generated for the reply, it

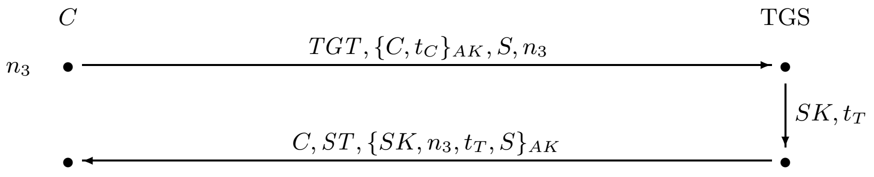

C
TGS
$TGT, \{C, t_C\}_{AK}, S, n_3$
n_3
SK, t_T
$C, ST, \{SK, n_3, t_T, S\}_{AK}$

Fig. 3. Message flow in the TGS exchange, where $TGT = \{AK, C, t_K\}_{k_T}$ and $ST = \{SK, C, t_T\}_{k_S}$

must be communicated to C before she can learn AK. PKINIT does this by adding the message $\{\!\{Cert_K, [k, ck]_{sk_K}\}\!\}_{pk_C}$. This contains K's certificates and his signature, using his secret key sk_K, over k and a keyed hash ck ('checksum' in the language of [39]) taken over the entire request from C using the key k; all of this is encrypted under C's public key pk_C. The keyed hash ck binds this response to the client's request and was added in response to the attack we discovered and reported in [20].

The Later Exchanges. After the client C has obtained the key AK and the TGT, either through the basic AS exchange or the PKINIT AS exchange, she then initiates the TGS exchange, shown in Fig. 3. The first line shows our formalization of the client's request, called a TGS_REQ message; it contains the TGT (which is opaque to the client), an *authenticator* $\{C, t_C\}_{AK}$, the name of the server S for which C desires a service ticket, and C's local time. Once the TGS receives this message, he decrypts the TGT to learn AK and uses this to decrypt the authenticator. Assuming his local policies for granting a service ticket are satisfied (while we do not model these here, they might include whether the request is sufficiently fresh), the TGS produces a fresh key SK for C and S to share and sends this back to the client in a TGS_REP message. The form of this message is essentially the same as the AS_REP message from the KAS to C: it contains a ticket (now the service ticket, or ST, $\{SK, C, t_T\}_{k_S}$ instead of the TGT) encrypted for the next server (now S instead of T) and encrypted data for C (now encrypted under AK instead of k_C).

Finally, after using the AS exchange to obtain the key SK and the ST, the client may use the CS exchange to authenticate herself to the end server. Figure 4 shows this exchange, including the optional reply from the server that authenticates this server to the client. The client C starts by sending a message (AP_REQ) that is similar to the TGS_REQ message of the previous round: in contains the (service) ticket and an authenticator ($\{C, t'_C\}_{SK}$) that is encrypted under the key contained in the ST. The server S simply responds with an AP_REP message $\{t'_C\}_{SK}$ containing the timestamp from the authenticator encrypted under the key from the ST.

Attack on PKINIT. The attack that we found against the then-current specification of PKINIT was reported in [20]. This attack was possible because, at

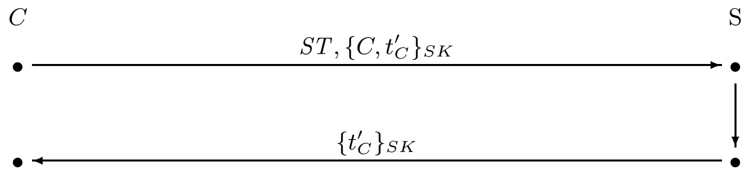


Fig. 4. Message flow in the CS exchange, where $ST = \{SK, C, t_T\}_{k_S}$

the time, the reply from the KAS to the client contained $[k, n_2]_{sk_K}$ in place of $[k, ck]_{sk_K}$. In particular, the KAS did not sign any data that depended upon the client's name. This allowed an attacker to copy a message from C to the KAS, use this data in her own request to the KAS, read the reply from the KAS, and then send this reply to C as though it was generated by the KAS for C (instead of for the attacker). The effect of this attack was that the attacker could impersonate the later servers (TGS and application servers) to the client, or she could let the client continue the authentication process while the attacker gains knowledge of all new keys shared by the client and various servers. In the latter variation, the client would be authenticated as the attacker and not as C.

Security Properties. We now summarize the security properties that we prove here at the symbolic level for both basic Kerberos and Kerberos with PKINIT; the implications on the computational level are discussed in the subsequent sections. We have proved similar properties in symbolic terms using a formalization in MSR for basic Kerberos [18] and for the AS exchange when PKINIT is used [20]. The first property we prove here concerns the secrecy of keys, a notion that is captured formally as Def. 1 in Sect. 4. This property may be summarized as follows.

Property 1 (Key secrecy). For any honest client C and honest server S, if the TGS T generates a symmetric key SK for C and S to use (in the CS-exchange), then the intruder does not learn the key SK.

The second property we study here concerns entity authentication, formalized as Def. 2 in Sect. 4. This property may be summarized as follows.

Property 2 (Authentication properties).

i. If a server S completes a run of Kerberos, apparently with C, then earlier: C started the protocol with some KAS to get a ticket-granting ticket and then requested a service ticket from some TGS.
ii. If a client C completes a run of Kerberos, apparently with server S, then S sent a valid AP_REP message to C.

Theorem 1 below shows that these properties hold for our symbolic formalizations of basic and public-key Kerberos in the BPW model. Theorem 2 shows that the authentication property holds as well for cryptographic implementations of these protocols if provably secure primitives are used; the standard cryptographic definition of key secrecy however turns out not to hold for cryptographic implementations of Kerberos, which we further investigate below. Because authentication can be shown to hold for Kerberos with PKINIT, it follows that at the level of cryptographic implementation, the fixed specification of PKINIT does indeed defend against the attack reported in [20].

3 The BPW Model

3.1 Review of the BPW Model

The BPW model introduced in [14] offers a deterministic Dolev-Yao style formalism of cryptographic protocols with commands for a vast range of cryptographic operations such as public-key, symmetric encryption/decryption, generation and verification of digital signatures as well as message authentication codes, and nonce generation. Every protocol participant is assigned a machine (an I/O automaton), which is connected to the machines of other protocol participants and which executes the protocol for its user by interacting with the other machines (see Fig. 5). In this reactive scenario, semantics is based on state, i.e., of who already knows which terms. The state is here represented by an abstract "database" and handles to its entries: Each entry (denoted $D[j]$) of the database has a type (e.g., "signature") and pointers to its arguments (e.g., "private key" and "message"). This corresponds to the way Dolev-Yao terms are represented. Furthermore, each entry in the abstract database also comes with handles to participants who have access to that entry. These handles determine the state. The BPW model does not allow cheating: only if a participant has a handle to the entry $D[j]$ itself or to the right entries that could produce a handle to $D[j]$ can the participant learn the term stored in $D[j]$. For instance, if the BPW model receives a command, e.g., from a user machine, to encrypt a message m with key k, then it makes a new abstract database entry for the ciphertext with a handle to the participant that sent the command and pointers to the message and the key as arguments; only if a participant has handles to the ciphertext and also to the key can the participant ask for decryption. Furthermore, if the BPW model receives the same encryption command a second time then it will generate a new (different) entry for the ciphertext. This meets the fact that secure encryption schemes are necessarily probabilistic. Entries are made known to other participants by a send command, which adds handles to the entry.

The BPW model is based on a detailed model of asynchronous reactive systems introduced in [40] and is represented as a deterministic machine $\mathsf{TH}_{\mathcal{H}}$ (also an I/O automaton), called *trusted host*, where $\mathcal{H} \subset \{1, \ldots, n\}$ denotes the set of honest participants out of all m participants. This machine executes the commands from the user machines, in particular including the commands for cryptographic operations. A *system* consists of several possible *structures.* A structure consists of a set $\hat{M}$ of connected correct user machines and a subset S of the free

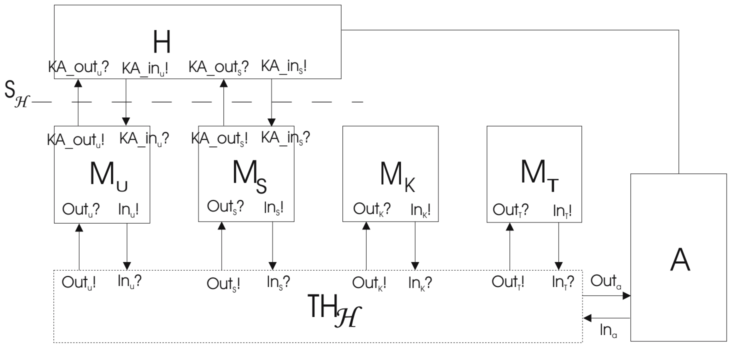


Fig. 5. Overview of the Kerberos symbolic system

ports, i.e., S is the user interface of honest users. In order to analyze the security of a structure $(\hat{M}, S)$, an arbitrary probabilistic polynomial-time *user* machine H is connected to the user interface S and a polynomial-time *adversary* machine A is connected to all the other ports and H. This completes a structure into a *configuration* of the system (see Fig. 5). The machine H represents all users. A configuration is a runnable system, i.e., for each security parameter k, which determines the input lengths (including the key length), one gets a well-defined probability space of *runs*. To guarantee that the system is polynomially bounded in the security parameter, the BPW model maintains length functions on the entries of the abstract database. The *view* of H in a run is the restriction to all inputs and outputs that H sees at the ports it connects to, together with its internal states. Formally one defines the view $view_{conf}(\mathsf{H})$ of H for a configuration *conf* to be a family of random variables X_k where k denotes the security parameter. For a given security parameter k, X_k maps runs of the configuration to a view of H.

Corresponding to the BPW model, there exists a cryptographic implementation of the BPW model and a computational system, in which honest participants also operate via handles on cryptographic objects. However, the objects are now bitstrings representing real cryptographic keys, ciphertexts, etc., acted upon by interactive polynomial-time Turing machines (instead of the symbolic machines and the trusted host). The implementation of the commands now uses provably secure cryptographic primitives according to standard cryptographic definitions (with small additions like type tagging and additional randomization). In [8,12,13,14] it was established that the cryptographic implementation of the BPW model is *at least as secure as* the BPW model, meaning that whatever an active adversary can do in the implementation can also be achieved by another adversary in the BPW model, or the underlying cryptography can be broken. More formally, a system Sys_1 being at least as secure as another system Sys_2 means that for all probabilistic polynomial-time user H, for all probabilistic polynomial-time adversary A_1 and for every computational structure $(\hat{M}_1, \mathsf{S}) \in Sys_1$, there exists a polynomial-time adversary A_2 on a corresponding symbolic structure $(\hat{M}_2, \mathsf{S}) \in Sys_2$ such that the view of H is computationally indistinguishable in both configurations. This captures the cryptographic notion of *reactive simulatability.*

3.2 Public-Key Kerberos in the BPW Model

We now model the Kerberos protocol in the framework of [14] using the BPW model. We write ":=" for deterministic assignment, "=" for testing for equality and "$\leftarrow$" for probabilistic assignment.

The descriptions of the symbolic systems of Kerberos 5 and PKINIT are very similar, with the difference that the user machines follow different algorithms for the two protocols. We denote Kerberos with PKINIT by "PK," and basic Kerberos by "K5." If we let Kerb$\in${PK, K5} then, as described in Sect. 3.1, for each user $u \in \{1, \ldots, n\}$ there is a *protocol machine* $\mathrm{M}_u^{\mathrm{Kerb}}$ which executes the protocol for u. There are also protocol machines for the KAS K and the TGT T, denoted by $\mathrm{M}_K^{\mathrm{Kerb}}$ and $\mathrm{M}_T^{\mathrm{Kerb}}$. Furthermore, if $S_1, \ldots, S_l$ are the servers in T's 'realm[1]', then there are server machines $\mathrm{M}_S^{\mathrm{Kerb}}$ for $S \in \{S_1, \ldots, S_l\}$. Each user machine is connected to the user via ports: A port for outputs to the user and a port for inputs from the user, labeled $\mathrm{KA\_out}_u!$ and $\mathrm{KA\_in}_u?$, respectively ("KA" for"Key sharing and Authentication"). The ports for the server machines are labeled similarly (see Fig. 5).

The behavior of the protocol machines is described in detail in [5]. In the following, we comment on two algorithms of PKINIT (Fig. 6 and Fig. 7) . If, for instance, a protocol machine $\mathrm{M}_u^{\mathrm{PK}}$ receives a message (new_prot, PK, K, T) at $\mathrm{KA\_in}_u?$ then it will execute Algorithm 1A (Fig. 6) to start a protocol run. We give a description below. The state of the protocol machine $\mathrm{M}_u^{\mathrm{Kerb}}$ consists of the bitstring u and the sets $Nonce_u$, $Nonce2_u$, $TGTicket$, and $Session\_KeysS_u$, in which $\mathrm{M}_u^{\mathrm{Kerb}}$ stores nonces, ticket-granting tickets, and the session keys for server S, respectively. This is the information a client needs to remember during a protocol run.

Only the machines of honest users $u \in \{1, \ldots, n\}$ and honest servers $S \in \{S_1, \ldots, S_l\}$ will be present in the protocol run, in addition to the machines for K and T. The others are subsumed in the adversary. We denote by $\mathcal{H} \subset \{1, \ldots, n,$ $K,$ $T,$ $S_1, \ldots, S_l\}$ the honest participants, i.e., for $v \in \mathcal{H}$ the machine $\mathrm{M}_v^{\mathrm{Kerb}}$ is guaranteed to run correctly. And we assume that KAS K and TGS T are always honest, i.e., $K, T \in \mathcal{H}$.

Furthermore, given a set $\mathcal{H}$ of honest participants, with $\{K, T\} \subset \mathcal{H} \subset \{1, \ldots, n,$ $K,$ $T,$ $S_1, \ldots, S_l\}$ the user interface of public-key Kerberos will be the set $S_{\mathcal{H}} := \{\mathrm{KA\_out}_u!, \mathrm{KA\_in}_u? \mid u \in \mathcal{H} \setminus \{K, T\}\}$. The symbolic system is the set $Sys^{\mathrm{Kerb,\ symb}} := \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}})\}$. Note that, since we are working in an asynchronous system, we are replacing protocol timestamps by arbitrary messages that we assume are known to the participants generating the timestamps (e.g. nonces). All algorithms should immediately abort if a command to the BPW model yields an error, e.g., if a decryption request fails.

[1] I.e., administrative domain; we do not consider cross-realm authentication here, although it has been analyzed symbolically in [21].

Notation. The entries of the database D are all of the form *(ind, type, arg,* $hnd_{u_1}, \ldots, hnd_{u_m}, hnd_a$*, len)*, where $\mathcal{H} = \{u_1, \ldots, u_m\}$. We denote by $\downarrow$ an error element available to all ranges and domains of all functions and algorithms. So, e.g., $hnd_a = \downarrow$ means the adversary does not have a handle to the entry. For entries $x \in D$, the *index* $x.ind \in \mathcal{INDS}$ consecutively numbers all entries in D. The set $\mathcal{INDS}$ is isomorphic to $\mathbb{N}$ and is used to distinguish index arguments. We write $D[i]$ for the selection $D[ind = i]$, i.e., it is used as a primary key attribute of the database. The entry $x.type \in typeset = \{$auth, cert, enc, nonce, list, pke, pkse, sig, ske, skse,$\}$ identifies the type of x. Here ske/pke is a private/public key pair and skse is a symmetric key which comes with a 'public' key pkse. This "public key identifier" pkse cannot be used for any cryptographic operation but works as a pointer to skse instead (see [7] for a more detailed explanation) . The entry $x.arg = (a_1, \ldots, a_j)$ is a possibly empty list of arguments. Many values a_i are in $\mathcal{INDS}$. $x.hnd_u \in \mathcal{HNDS} \cup \{\downarrow\}$ for $u \in \mathcal{H} \cup \{a\}$ are handles by which u knows this entry. We always use a superscript "*hnd*" for handles. $x.len \in \mathbb{N}_0$ denotes the "length" of the entry; it is computed by applying length functions (mentioned in Sect. 3.1).

Initially, D is empty. $\mathrm{TH}_{\mathcal{H}}$ has a counter $size \in \mathcal{INDS}$ for the current size of D. For the handle attributes, it has counters $currhnd_u$ initially 0. First we need to add the symmetric keys shared exclusively by K and T, S and T. Public-key Kerberos uses certificates; therefore, in this case all users need to know the public key for certificate authorities and have their own public-key certificates signed by a certificate authority. For simplicity we use only one certificate authority CA. Therefore, we add to D an entry for the public key of CA with handles to all users (i.e., to all user machines). For every user we add an entry for the certificate of that user signed by the certificate authority with a handle to the user (machine). In the case of Kerberos 5, we are adding entries for the key k_u shared exclusively by K and u, for all users u.

Example of Algorithms. Due to space constraints we are only going to examine PKINIT (Fig. 2) and explain the steps of its Algorithms 1A and 2 (Fig. 6 and Fig. 7) which are more complex than the algorithms in Kerberos 5. For details on the definition of the used commands see [8,13,14]. For readability of the figures, we noted on the right (in curly brackets) to which terms in the more commonly used Dolev-Yao notation the terms in the algorithms correspond ($\approx$).

Protocol start of PKINIT. In order to start a new PKINIT protocol, user u inputs (new_prot, PK, K, T) at port KA_in$_u$?. Upon such an input, $\mathrm{M}_u^{\mathrm{PK}}$ runs Algorithm 1A (Fig. 6) which prepares and sends the AS_REQ to K using the BPW model. $\mathrm{M}_u^{\mathrm{PK}}$ generates symbolic nonces in steps 1A.1 and 1A.2 by sending the command gen_nonce(). In step 1A.3 the command list(_,_) concatenates t_u and $n_{u,2}$ into a new list that is signed in step 1A.4 with u's private key. Since we are working in an asynchronous system, the timestamp t_u is approximated by some arbitrary message (e.g., by a nonce). The command store(_) in step 1A.5–6 makes entries in the database for the names of u and T. Handles for the names u and T are returned, which are added to a list in the next step. $\mathrm{M}_u^{\mathrm{PK}}$ stores

A) Input:$(\text{new\_prot}, \text{PK}, K, T)$ at $KA\_\text{in}_u?$.

```
1. $n_{u,1}t_u^{hnd} \leftarrow \text{gen\_nonce}()$
2. $n_{u,2}^{hnd} \leftarrow \text{gen\_nonce}()$
3. $l^{hnd} \leftarrow \text{list}(t_u^{hnd}, n_{u,2}^{hnd})$                                   $\{l \approx (t_C, n_2)\}$
4. $s^{hnd} \leftarrow \text{sign}(ske_u^{hnd}, l^{hnd})$                                   $\{s \approx [t_C, n_2]_{sk_C}\}$
5. $u^{hnd} \leftarrow \text{store}(u)$
6. $T^{hnd} \leftarrow \text{store}(T)$
7. $m_1^{hnd} \leftarrow \text{list}(cert_u^{hnd}, s^{hnd}, u^{hnd}, T^{hnd}, n_{u,1}^{hnd})$     $\{m_1 \approx Cert_C, [t_C, n_2]_{sk_C}, C, T, n_1\}$
8. $Nonce_u := Nonce_u \cup \{(n_{u,1}^{hnd}, m_1^{hnd}, K)\}$
9. $\text{send\_i}(K, m_1^{hnd})$
```

B) Input:$(\text{continue\_prot}, \text{PK}, T, S, AK^{hnd})$ at $\text{KS\_in}_u?$ for $S \in \{S_1, ..., S_l\}$

```
1. if $(\nexists (TGT^{hnd}, AK^{hnd}, T) \in TGTicket_u)$ then
2.     Abort
3. end if
4. $z^{hnd} \leftarrow \text{list}(u^{hnd}, t_u^{hnd})$                                   $\{z \approx C, t_C\}$
5. $auth^{hnd} \leftarrow \text{sym\_encrypt}(AK^{hnd}, z^{hnd})$                           $\{auth \approx \{C, t_C\}_{AK}\}$
6. $n_{u,3}^{hnd} \leftarrow \text{gen\_nonce}()$
7. $Nonce2_u := Nonce2_u \cup \{n_{u,3}^{hnd}, T, S)\}$
8. $m_2^{hnd} \leftarrow \text{list}(TGT^{hnd}, auth^{hnd}, S^{hnd}, n_{u,3}^{hnd})$           $\{m_2 \approx TGT, \{C, t_C\}_{AK}, S, n_3\}$
9. $\text{send\_i}(T, m_2^{hnd})$
```

Fig. 6. Algorithm 1 of Public-key Kerberos: Evaluation of inputs from the user (starting the AS and TG exchanges)

information in the set $Nonce_u$, which it will need later in the protocol to verify the message authentication code sent by K. In step 1A.8 $Nonce_u$ is updated. Finally, in step 1A.9 the AS_REQ is sent over an insecure ("i" for insecure) channel.

Behavior of the KAS K in PKINIT. Upon input (v, K, i, m^{hnd}) at port $\text{out}_K?$ with $v \in \{1, .., n\}$, the machine M_K^{PK} runs Algorithm 2 (Fig. 7) which first checks if the message m is a valid AS_REQ and then prepares and sends the corresponding AS_REP. In order to verify that the input is a possible AS_REQ, the types of the input message m's components are checked in steps 2.1–2.5. The command retrieve(x_i^{hnd}) in step 2.3 returns the bitstring of the entry $D[hnd_u = x_i^{hnd}]$. Next the machine verifies the received certificate x_1 of v by checking the signature of the certificate authority CA (steps 2.6–2.10). Then the machine extracts the public key pke_v out of v's certificate with the command pk_of_cert(_) and uses this public key to verify the signature x_2 received in the AS_REQ (steps 2.11–2.16). In steps 2.17–2.21 the types of the message components of the signed message y_1 are checked, as well as the freshness of the nonce y_{12} by comparison to nonces stored in $Nonce3_K$. If the nonce is fresh then it will be stored in the set $Nonce3_K$ in step 2.23 for freshness checks in future protocol runs. Finally, in steps 2.24–2.36 M_K^{PK} generates two symmetric keys k and AK, composes the AS_REP, and sends it to v over an insecure channel.

Input:(v, K, i, m^{hnd}) at $\text{out}_K?$ with $v \in \{1, ..., n\}$.

```
 1. $x_i^{hnd} \leftarrow \text{list\_proj}(m^{hnd}, i)$ for $i = 1, ..., 5$
 2. $type_i \leftarrow \text{get\_type}(x_i^{hnd})$ for $i = 1, 2, 5$    $\{x_1 \approx Cert_C, x_2 \approx [t_C, n_2]_{sk_C}, x_5 \approx n_1\}$
 3. $x_i \leftarrow \text{retrieve}(x_i^{hnd})$ for $i = 3, 4$    $\{x_3 \approx C, x_4 \approx T\}$
 4. if $(type_1 \neq \text{cert}) \vee (type_2 \neq \text{sig}) \vee (type_5 \neq \text{Nonce}) \vee (x_3 \neq v) \vee (x_4 \neq T)$ then
 5.     Abort
 6. end if
 7. $v^{hnd} \leftarrow \text{store}(v)$
 8. $b \leftarrow \text{verify\_cert}(pke_{CA}^{hnd}, x_1^{hnd}, v^{hnd})$
 9. if $b = false$ then
10.     Abort
11. end if
12. $pke_v^{hnd} \leftarrow \text{pk\_of\_cert}(pke_{CA}^{hnd}, x_1^{hnd})$
13. $type_6 \leftarrow \text{get\_type}(pke_v^{hnd})$
14. if $(type_6 \neq \text{pke})$ then
15.     Abort
16. end if
17. $y_1^{hnd} \leftarrow \text{msg\_of\_sig}(x_2^{hnd})$    $\{y_1 \approx t_C, n_2\}$
18. $b \leftarrow \text{verify}(x_2^{hnd}, pke_v^{hnd}, y_1^{hnd})$    $\{x_2 \approx [t_C, n_2]_{sk_C}\}$
19. if $b = false$ then
20.     Abort
21. end if
22. $y_{1i}^{hnd} \leftarrow \text{list\_proj}(y_1^{hnd}, i)$ for $i = 1, 2$    $\{y_{11} \approx t_C, y_{12} \approx n_2\}$
23. $type_{12} \leftarrow \text{get\_type}(y_{12}^{hnd})$
24. if $(type_{12} \neq \text{nonce}) \vee ((y_{12}^{hnd}, .) \in Nonce3_K)$ then
25.     Abort
26. end if
27. $Nonce3_K := Nonce3_K \cup \{(y_{12}^{hnd}, v)\}$
28. $k^{hnd} \leftarrow \text{gen\_symenc\_key}()$
29. $AK^{hnd} \leftarrow \text{gen\_symenc\_key}()$
30. $auth^{hnd} \leftarrow \text{auth}(k^{hnd}, m^{hnd})$    $\{auth \approx ck\}$
31. $z_1^{hnd} \leftarrow \text{list}(k^{hnd}, auth^{hnd})$    $\{z_1 \approx k, ck\}$
32. $s_2^{hnd} \leftarrow \text{sign}(ske_K^{hnd}, z_1^{hnd})$    $\{s_2 \approx [k, ck]_{sk_K}\}$
33. $z_2^{hnd} \leftarrow \text{list}(cert_K^{hnd}, s_2^{hnd})$    $\{z_2 \approx Cert_K, [k, ck]_{sk_K}\}$
34. $m_{21} \leftarrow \text{encrypt}(pke_K^{hnd}, z_2^{hnd})$    $\{m_{21} \approx \{\{Cert_K, [k, ck]_{sk_K}\}\}_{pk_C}\}$
35. $z_3^{hnd} \leftarrow \text{list}(AK^{hnd}, x_3^{hnd}, t_K^{hnd})$    $\{z_3 \approx AK, C, t_K, T\}$
36. $TGT^{hnd} \leftarrow \text{sym\_encrypt}(skse_{K,x_4}^{hnd}, z_3^{hnd})$    $\{TGT \approx \{AK, C, t_K\}_{k_T}\}$
37. $z_4^{hnd} \leftarrow \text{list}(AK^{hnd}, x_5^{hnd}, t_K^{hnd}, x_4^{hnd})$    $\{z_4 \approx AK, n_1, t_K, T\}$
38. $m_{24} \leftarrow \text{sym\_encrypt}(k^{hnd}, z_4^{hnd})$    $m_{24} \approx \{Ak, n_1, t_K, T\}_k\}$
39. $m_2^{hnd} \leftarrow \text{list}(m_{21}^{hnd}, x_3^{hnd}, TGT^{hnd}, m_{24}^{hnd})$
        $\{m_2 \approx \{\{Cert_K, [k, ck]_{sk_K}\}\}_{pk_C}, C, TGT, \{Ak, n_1, t_K, T\}_k\}$
40. $\text{send\_i}(v, m_2^{hnd})$
```

Fig. 7. Algorithm 2 of Public-key Kerberos: Behavior of the KAS

4 Formal Results

4.1 Security in the Symbolic Setting

In order to use the BPW model to prove the computational security of Kerberos, we first formalize the respective security properties and verify them in the BPW model. We first prove that Kerberos keeps the symmetric key, which the TGS T generated for use between user u and server S, symbolically secret from the adversary. In order to prove this, we show that Kerberos also keeps the keys generated by KAS K for the use between u and the TGS T secret. Furthermore, we prove entity authentication of the user u to a server S (and subsequently entity authentication of S to u). This form of authentication is weaker than the authentication Kerberos offers, since we do not consider the purpose of timestamps in Kerberos. (Recall that timestamps are currently not included in the BPW model.)

Secrecy and Authentication Requirements. We now define the notion of key secrecy, which was informally captured already in Property 1 of Sect. 2, as the following formal requirement in the language of the BPW model.

Definition 1 (Key secrecy requirement). *For Kerb* $\in\{PK, K5\}$ *the secrecy requirement* $Req^{\mathsf{Sec}}_{Kerb}$ *is:*
For all $u \in \mathcal{H} \cap \{1, \ldots, n\}$, *and* $S \in \mathcal{H} \cap \{S_1, \ldots, S_l\}$, *and* $t_1, t_2, t_3 \in \mathbb{N}$*:*

$$\begin{aligned} &(t_1 : \mathrm{KA\_out}_S!\,(\mathrm{ok}, \mathrm{Kerb}, u, SK^{hnd}) \\ \vee\ & t_2 : \mathrm{KA\_out}_u!\,(\mathrm{ok}, \mathrm{Kerb}, S, SK^{hnd}) \\ \Rightarrow\ & t_3 : D[hnd_u = SK^{hnd}].hnd_a = \downarrow \end{aligned}$$

where $t : D$ denotes the contents of database D at time t. Similarly $t : p?m$ and $t : p!m$ denotes that message m occurs at input (respectively output) port p at time t. As above PK refers to Public-key Kerberos and K5 to Kerberos 5. In the next section Thm. 1 will show that the symbolic Kerberos systems specified in Sect. 3.2 satisfy this notion of secrecy, and therefore Kerberos enjoys Property 1.

Next we define the notion of authentication in Property 2 in the language of the BPW model.

Definition 2 (Authentication requirements). *For* Kerb $\in \{PK, K5\}$*:*

i. The authentication requirement $Req^{\mathsf{Auth1}}_{Kerb}$ *is: For all* $v \in \mathcal{H} \cap \{1, \ldots, n\}$, *for all* $S \in \mathcal{H} \cap \{S_1, \ldots, S_l\}$, *and* K, T*:*

$$\begin{aligned} &\exists\, t_3 \in \mathbb{N}.\ t_3 : \mathrm{KA\_out}_S!\,(\mathrm{ok}, \mathrm{Kerb}, v, SK^{hnd}) \\ \Rightarrow\ &\exists\, t_1, t_2 \in \mathbb{N} \textit{ with } t_1 < t_2 < t_3.\ t_2 : \mathrm{KA\_in}_v!\,(\mathrm{continue\_prot}, \mathrm{Kerb}, \mathrm{T}, S, \cdot) \\ &\wedge\ t_1 : \mathrm{KA\_in}_v!\,(\mathrm{new\_prot}, \mathrm{Kerb}, K, T) \end{aligned}$$

ii. The authentication requirement $Req^{\mathsf{Auth2}}_{Kerb}$ *is: For all* $u \in \mathcal{H} \cap \{1, \ldots, n\}$, *for all* $S \in \mathcal{H} \cap \{S_1, \ldots, S_l\}$, *and* K, T*:*

$$\begin{aligned} &\exists\, t_2 \in \mathbb{N}.\ \ t_2 : \mathrm{KA\_out}_u!\,(\mathrm{ok}, \mathrm{Kerb}, S, SK^{hnd}) \\ \Rightarrow\ &\exists\, t_1 \in \mathbb{N} \textit{ with } t_1 < t_2.\ t_1 : \mathrm{KA\_in}_S!\,(\mathrm{ok}, \mathrm{Kerb}, u, SK^{hnd}) \end{aligned}$$

iii. The overall authentication $Req^{\mathsf{Auth}}_{Kerb}$ *for protocol Kerb is:*

$$Req^{\mathsf{Auth}}_{Kerb} := Req^{\mathsf{Auth1}}_{Kerb} \wedge Req^{\mathsf{Auth2}}_{Kerb}$$

Theorem 1 will show that this notion of authentication is satisfied by the symbolic Kerberos system. Therefore Kerberos has Property 2.

When proving that Kerberos has these properties, we will use the notion of a system *Sys perfectly fulfilling a requirement Req*, $Sys \models^{\mathrm{perf}} Req$. This means the property *Req* holds with probability one over the probability space of runs for a fixed security parameter (as defined in Sect. 3.1). Later we will also need the notion of a system *Sys computationally fulfilling a requirement Req*, $Sys \models^{\mathrm{poly}} Req$, i.e., the property holds with negligible error probability for all polynomially bounded users and adversaries (again, over the probability space of all runs for a fixed security parameter). In particular, perfect fulfillment implies computational fulfillment.

In order to prove Thm. 1, we first need to prove a number of auxiliary properties (previously called *invariants* in, e.g., [4,11]). Although these properties are nearly identical for Kerberos 5 and Public-key Kerberos, their proofs had to be carried out separately. We consider it interesting future work to augment the BPW model with proof techniques that allow for conveniently analyzing security protocols in a more modular manner. In fact, a higher degree of modularity would simplify the proofs for each individual protocol as it could exploit the highly modular structure of Kerberos; moreover, it would also simplify the treatment of the numerous optional behaviors of this protocol.

Some of the key properties needed in the proof of Thm. 1, which formalizes Properties 1 and 2, make authentication and confidentiality statements for the first two rounds of Kerberos. These properties are described in English below; they are formalized and proved in [5].

i) **Authentication of KAS to client and Secrecy of AK:** If a user u receives a valid AS_REP message then this message was indeed generated by K for u and an adversary cannot learn the symmetric keys contained in this message.
ii) **TGS Authentication of the TGT:** If a TGS T receives a TGT and an authenticator $\{v, t_v\}_{AK}$ where the key AK and the username v are contained in the TGT, then the TGT was generated by K and the authenticator was created by v.
iii) **Authentication of TGS to client and Secrecy of SK:** If a user u receives a valid TGS_REP then it was generated by T for u and S and no adversary can learn the session key SK contained in this message.
iv) **Server Authentication of the ST:** If a server S receives an ST and an authenticator $\{v, t_v\}_{SK}$ where the key SK and the name v are contained in the ST, then the ST was generated by T and the authenticator was created by v.

We can now capture the security of Kerberos in the BPW model in the following theorem, which says that Properties 1 and 2 hold symbolically for Kerberos.

We show a proof excerpt in the case of Public-key Kerberos (the outline is analogous for Kerberos 5).

Theorem 1. *(Security of the Kerberos Protocol based on the BPW Model)*

- *Let* $Sys^{\mathrm{K5,\,symb}}$ *be the symbolic Kerberos 5 system defined in Sect. 3.2, and let* Req_{K5}^{Sec} *and* Req_{K5}^{Auth} *be the secrecy and authentication requirements defined above. Then* $Sys^{\mathrm{K5,\,symb}} \models^{\mathrm{perf}} Req_{K5}^{\mathsf{Sec}} \wedge Req_{K5}^{\mathsf{Auth}}$.
- *Let* $Sys^{\mathrm{PK,\,symb}}$ *be the symbolic Public-key Kerberos system, and let* Req_{PK}^{Sec} *and* Req_{PK}^{Auth} *be the secrecy and authentication requirements defined above. Then* $Sys^{\mathrm{PK,\,symb}} \models^{\mathrm{perf}} Req_{PK}^{\mathsf{Sec}} \wedge Req_{PK}^{\mathsf{Auth}}$.

Proof (sketch). We assume that all parties are honest. If user u successfully terminates a session run with a server S, i.e., there was an output (ok, PK, S, k^{hnd}) at KA_out$_u$!, then the key k was stored in the set $Session\_KeysS_u$. This implies that the key was generated by T and sent to u in a valid TGS_REP. By auxiliary property iv), an adversary cannot learn k. The case that S successfully terminates a session run is analogous. This shows the key secrecy property Req_{PK}^{Sec}. As for the authentication property $Req_{PK}^{\mathsf{Auth1}}$, if server S successfully terminates a session with u, i.e., there was an output (ok, PK, u, k^{hnd}) at KA_out$_S$!, then S must have received a ticket generated by T (for S and u) and also a matching authenticator generated by user u (by auxiliary property iv)). But the ticket will only be generated if u sends the appropriate request to T, i.e., there was an input (continue_prot,PK, T, S, AK^{hnd}) at KA_in_u?. The request, on the other hand, contains a TGT that was generated by K for u (by auxiliary property ii)), therefore u must have sent an request to K. In particular, there had been an input (new_prot, PK, K, T) at KA_in_u?. As for the authentication property $Req_{PK}^{\mathsf{Auth2}}$, if the user u successfully terminates a session with server S, i.e., there was an output (ok, PK, S, k^{hnd}) at KA_out$_u$!, then it must have received a message encrypted under k that does not contain u's name. The key k was contained in a valid TGS_REP and was therefore generated by T, by auxiliary property iii). Only T, u, or S could know the key k, but only S uses this key to encrypt and send a message that u received. On the other hand, S follows sending such a message immediately by an output (ok, PK, u, k^{hnd}) at KA_out$_S$!. □

This proof shares similarities with the Dolev-Yao style proofs of analogous properties for Kerberos 5 and PKINIT using the MSR framework [18,20]. The two approaches are similar in the sense that both reconstruct a necessary trace backward from an end state, and in that they rely on some form of induction (based on rank/co-rank functions in MSR). In future work, we plan to draw a formal comparison between these two Dolev-Yao encodings of a protocol, and the proof techniques they support.

4.2 Security in the Cryptographic Setting

The results of [14] allow us to take the authentication results in Thm. 1 and derive a corresponding authentication results for a cryptographic implementation of Kerberos. Just as Property 2 holds symbolically for Kerberos, this shows that it holds in a cryptographic implementation as well. In particular, entity authentication between a user and a server in Kerberos holds with overwhelming probability (over the probability space of runs). However, symbolic results on key secrecy can only be carried over to cryptographic implementations if the protocol satisfies certain additional conditions. Kerberos unfortunately does not fulfill these definitions, and it can easily be shown that cryptographic implementations of Kerberos do not fulfill the standard notion of cryptographic key secrecy, see below. This yields the following theorem.

Theorem 2. *(Computational security of the Kerberos protocol)*

– *Let* $Sys^{\mathrm{K5,\ comp}}$ *denote the computational Kerberos 5 system implemented with provable secure cryptographic primitives. Then* $Sys^{\mathrm{K5,\ comp}} \models^{\mathrm{poly}} Req^{\mathsf{Auth}}_{K5}$.
– *Let* $Sys^{\mathrm{PK,\ comp}}$ *denote the computational Public-key Kerberos system implemented with provable secure cryptographic primitives. Then* $Sys^{\mathrm{PK,\ comp}} \models^{\mathrm{poly}} Req^{\mathsf{Auth}}_{PK}$.

Proof (Sketch for public-key Kerberos). By Thm. 1, we know that $Sys^{PK,\ id} \models^{perf} Req^{\mathsf{Auth}}_{PK}$. And, as we mentioned earlier, the cryptographic implementation of the BPW model (using provably secure cryptographic primitives) is at least as secure as the BPW model, $Sys^{\mathrm{cry,\ comp}} \geq^{\mathrm{poly}}_{\mathrm{sec}} Sys^{\mathrm{cry,\ id}}$. After checking that the "Commitment Problem" does not occur in the protocol, we can use the Preservation of Integrity Properties Theorem from [6] to automatically obtain Thm. 2.

The Commitment Problem occurs when keys that have been used for cryptographic work are revealed later in the protocol. If the simulator in [14] (with which one can simulate a computational adversary attack on the symbolic system) learns in some abstract way that e.g. a ciphertext was sent, the simulator generates a distinguishable ciphertext without knowing the symmetric key nor the plaintext. If the symmetric key is revealed later in the protocol then the trouble for the simulator will be to generate a suitable symmetric key that decrypts the ciphertext into the correct plaintext. This is typically an impossible task. In order for the simulation with the BPW model to work, one thus needs to check that the Commitment Problem does not occur in the protocol. □

As far as key secrecy is concerned, it can be proven that the adversary attacking the cryptographic implementation does not learn the secret key string as a whole. However, it does not necessarily rule out that an adversary will be able to distinguish the key from other fresh random keys, as required by the definition of *cryptographic key secrecy.* This definition of secrecy says that an adversary cannot learn any partial information about such a key and is hence considerably stronger than requiring that an adversary cannot obtain the whole key. For Kerberos we can show that the key SK does not satisfy cryptographic key secrecy after the last round of Kerberos, i.e., SK is distinguishable from other fresh random keys. It should also be noted that this key SK is still indistinguishable from random after the second round but before the start of the third round of Kerberos. We have the following proposition.

Proposition 1. *a) Kerberos does not offer cryptographic key secrecy for the key SK generated by the TGS T for the use between client C and server S after the start of the last round of Kerberos.*

b) After the TGS exchange and before the start of the CS exchange the key SK generated by the TGS T is still cryptographically secret.

Proof. a) To see that Kerberos does not offer cryptographic key secrecy for SK after the start of the third round, note that the key SK is used in the protocol for symmetric encryption. As symmetric encryption always provides partial information to an adversary if the adversary also knows the message that was encrypted. An adversary can exploit this to distinguish the key SK as follows: the adversary first completes a regular Kerberos execution between C and S learning the message $\{C, t'\}_{SK}$ encrypted under the unknown key SK. The adversary will also learn a bounded time period TP (of a few seconds) in which the timestamp t' was generated. Next a bit b is flipped and the adversary receives a key k, where $k = SK$ for $b = 0$ and k is a fresh random key for $b = 1$. The adversary now attempts to decrypt $\{C, t'\}_{SK}$ with k yielding a message m. If $m \neq C, t$ for a timestamp t then the adversary guesses $b = 1$. If $m = C, t$ for a timestamp t then the adversary checks whether $t \in TP$ or not. If $t \notin TP$ then the adversary guesses $b = 1$ otherwise the adversary guesses $b = 0$. The probability of the adversary guessing correctly is then $1 - \epsilon$, where ϵ is the probability that for random keys k, SK the ciphertext $\{C, t'\}_{SK}$ decrypted with k is C, t with $t \in TP$. Clearly, ϵ is negligible (since the length of the time period TP does not depend on the security parameter). Hence, SK is distinguishable and cryptographic key secrecy does not hold.

b) However, before the third round has been started the key SK is not only unknown to the adversary but, in particular, SK has not been used for symmetric encryption yet. We can therefore invoke the key secrecy preservation theorem of [9], which states that a key that is symbolically secret and symbolically unused is also cryptographically secret. This allows us to conclude that SK is cryptographically secret from the adversary.

For similar reasons, we also have the following proposition

Proposition 2. *a) Kerberos does not offer cryptographic key secrecy for the key AK generated by the KAS K for the use between client C and TGS T after the start of the second round of Kerberos.*

b) After the AS exchange and before the start of the TGS exchange the key AK generated by the KAS K is still cryptographically secret.

Finally, we note the following

Remark 1. Kerberos allows the client or the server to generate a sub-session key [39]. This optional key, which can then be used for the encryption of further communication between the two parties, is cryptographically secret as it can be proven symbolically secret and symbolically unused. This proof can easily be conducted symbolically similar to Thm. 1, and then the key secrecy preservation theorem of [10] can be used to automatically obtain a proof of cryptographic key secrecy for the optional sub-key. Note that this preservation theorem could not be used for proving cryptographic key secrecy for the main key as this key is already used within the key exchange protocol.

5 Conclusions and Future Work

In this paper, we have exploited the Dolev-Yao style model of Backes, Pfitzmann, and Waidner [8,12,13] to obtain the first computational proof of authentication for the core exchanges of the Kerberos protocol and its extension to public keys (PKINIT). Although the proofs sketched here are conducted symbolically, grounding the analysis on the BPW model automatically lifts the results to the computational level, assuming that all cryptography is implemented using provably secure primitives. Cryptographic key secrecy in the sense of indistinguishability of the exchanged key from a random key could only be established for the optional sub-key exchanged in Kerberos while for the actually exchanged key, cryptographic key secrecy could be proven not to hold.

Potentially promising future work includes the augmentation of the BPW model with specialized proof techniques that allow for conveniently performing modular proofs. Such techniques would provide a simple and elegant way to integrate the numerous optional behaviors supported by Kerberos and nearly all commercial protocols; for example, this would facilitate the analysis of DH mode in PKINIT which is part of our ongoing work. We intend to tackle the invention of such proof techniques that are specifically tailored towards the BPW model in the near future, e.g., by exploiting recent ideas from [24]. Another potential improvement we plan to pursue in the near future is to augment the BPW model with timestamps; this would in particular allow us to establish authentication properties that go beyond entity authentication [18,20,21]. A further item on our research agenda is to fully understand the relation between the symbolic correctness proof for Kerberos 5 presented here and the corresponding results achieved in the MSR framework [18,20].

References

1. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In *Proc. of Computer-aided Verification (CAV)*. Springer, 2005. URL: `www.avispa-project.org`.
2. M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. TACS*, pages 82–94, 2001.
3. M. Abadi and P. Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP International Conference on Theoretical Computer Science*, pages 3–22. Springer LNCS 1872, 2000.
4. M. Backes. A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol. In *Proc. ESORICS*, pages 89–108. Springer LNCS 3193, 2004.

5. M. Backes, I. Cervesato, A. D. Jaggard, A. Scedrov, and J.-K. Tsay. Cryptographically sound security proofs for basic and public-key Kerberos. IACR Cryptology ePrint Archive, Report 2006/219, `http://eprint.iacr.org/`, June 2006.
6. M. Backes and C. Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *Proc. 20th STACS*, pages 675–686. Springer LNCS 2607, 2003.
7. M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. *Journal on Selected Areas in Communications*, 22(10):2075–2086, 2004.
8. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. CSFW'04*, pages 204–218, June 2004.
9. M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Trans. Dependable Secure Comp.*, 2(2):109–123, April–June 2005.
10. M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. In *Proc. 26th IEEE Symposium on Security & Privacy*, pages 171–182, 2005. Extended version in IACR Cryptology ePrint Archive 2004/300.
11. M. Backes and B. Pfitzmann. On the cryptographic key secrecy of the stregthened Yahalom protocol. In *Proceedings of 21st IFIP SEC'06*, To appear.
12. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. CCS'03*, pages 220–230, 2003.
13. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. ESORICS'03*, pages 271–290. Springer LNCS 2808, 2003.
14. M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. IACR Cryptology ePrint Archive, Report 2003/015, `http://eprint.iacr.org/`, January 2003.
15. G. Bella and L. C. Paulson. Kerberos Version IV: Inductive Analysis of the Secrecy Goals. In *Proc. ESORICS'98*, pages 361–375. Springer LNCS 1485, 1998.
16. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Proc. CRYPTO '93*, pages 232–249. Springer LNCS 773, 1994.
17. B. Blanchet. A computationally sound mechanized prover for security protocols. In *Proc. 27th IEEE Symposium on Security & Privacy*, 2006.
18. F. Butler, I. Cervesato, A. D. Jaggard, and A. Scedrov. An Analysis of Some Properties of Kerberos 5 Using MSR. In *Proc. CSFW'02*, 2002.
19. R. Canetti and J. Herzog. Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange). In *Proc. 3rd Theory of Cryptography Conference (TCC)*, 2006.
20. I. Cervesato, A. D. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad. Breaking and fixing public-key Kerberos. In *Proc. WITS'06*, 2006.
21. I. Cervesato, A. D. Jaggard, A. Scedrov, and C. Walstad. Specifying Kerberos 5 Cross-Realm Authentication. In *Proc. WITS'05*, pages 12–26, 2005.
22. V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. ESOP-14*, pages 157–171, 2005.
23. A. Datta, A. Derek, J. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proc. ICALP*, pages 16–29. Springer LNCS 3580, 2005.
24. A. Datta, A. Derek, J. Mitchell, and B. Warinschi. Key exchange protocols: Security definition, proof method, and applications. In *19th IEEE Computer Security Foundations Workshop (CSFW 19)*, Venice, Italy, 2006. IEEE Press.
25. D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Trans. Info. Theory*, 2(29):198–208, 1983.

26. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. STOC*, pages 218–229, 1987.
27. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
28. J. D. Guttman, F. J. Thayer Fabrega, and L. Zuck. The faithfulness of abstract protocol analysis: Message authentication. In *Proc. CCS-8*, pages 186–195, 2001.
29. C. He and J. C. Mitchell. Security Analysis and Improvements for IEEE 802.11i. In *Proc. NDSS'05*, 2005.
30. J. Herzog, M. Liskov, and S. Micali. Plaintext awareness via key registration. In *Proc. CRYPTO*, pages 548–564. Springer LNCS 2729, 2003.
31. IETF. Public Key Cryptography for Initial Authentication in Kerberos, 1996–2006. Sequence of Internet drafts available from `http://tools.ietf.org/wg/krb-wg/draft-ietf-cat-kerberos-pk-init/`.
32. R. Impagliazzo and B. M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. FOCS*, pages 372–381, 2003.
33. P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. Symp. Security and Privacy*, pages 71–85, 2004.
34. C. Meadows. Analysis of the internet key exchange protocol using the NRL Protocol Analyzer. In *Proc. IEEE Symp. Security and Privacy*, pages 216–231, 1999.
35. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. TCC*, pages 133–151. Springer LNCS 2951, 2004.
36. Microsoft. Security Bulletin MS05-042. `http://www.microsoft.com/technet/security/bulletin/MS05-042.mspx`, Aug. 2005.
37. J. Mitchell, M. Mitchell, A. Scedrov, and V. Teague. A probabilistic polynominal-time process calculus for analysis of cryptographic protocols (preliminary report). *Electronic Notes in Theoretical Computer Science*, 47:1–31, 2001.
38. C. Neuman and T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications*, 32(9):33–38, Sept. 1994.
39. C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5), July 2005. `http://www.ietf.org/rfc/rfc4120.txt`.
40. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. S&P*, pages 184–200, 2001.

Deriving Secrecy in Key Establishment Protocols

Dusko Pavlovic[1] and Catherine Meadows[2]

[1] Kestrel Institute, Palo Alto, CA 94304
dusko@kestrel.edu
[2] Naval Research Laboratory, Washington, DC 20375
meadows@itd.nrl.navy.mil

Abstract. Secrecy and authenticity properties of protocols are mutually dependent: every authentication is based on some secrets, and every secret must be authenticated. This interdependency is a significant source of complexity in reasoning about security. We describe a method to simplify it, by encapsulating the authenticity assumptions needed in the proofs of secrecy. This complements the method for encapsulating the secrecy assumptions in proofs of authenticity, presented in [1]. While logically straightforward, this idea of encapsulation in general, and the present treatment of secrecy in particular, allow formulating scalable and reusable reasoning patterns about the families of protocols of practical interest. The approach evolved as a design strategy in the Protocol Derivation Assistant (Pda), a semantically based environment and toolkit for derivational approach to security [2,3].

1 Introduction

All secure communication on public networks begins with key establishment. Many diverse key distribution and key agreement schemes have evolved, with subtle, complex, and sometimes unclear security properties Since they are critical for the functioning of the networks, it is desirable to establish their provable, rather than just empirical security.

1.1 Derivational Approach to Security

Practical methods for proving security evolve slowly. While most branches of engineering largely consist of methodologies for building complex systems from simple components, formulating the incremental and compositional methods for security engineering has been a challenging task: in general, security properties are not preserved under composition.

Although not straightforward, the progress towards methods for the incremental design and analysis of security protocols has been steady. The present work is a part of a continued effort towards capturing and formalizing a sound part of the incremental practices of security engineering, and supporting it in a semantically based integrated development environment for secure systems [2,3]. The logic of [4,5] is a protocol analysis logic which incorporates incremental practices of security engineering such as refinement and composition. In [6] we developed a streamlined and simplified version of the authentication fragment of the logic in [4,5], that was still sufficient to uncover a flaw an IETF standardized protocol, that had already undergone extensive formal analysis. This led us to the idea, presented in [1], that many authentication proofs can be simplified, by encapsulating the needed secrecy requirements, and leaving them as open assumptions, to be discharged in separate proof modules. In the present work we present a secrecy logic, intended to support these separate proofs of secrecy properties. This time, the authentication assumptions, needed for secrets, are encapsulated, and left as open assumptions.

1.2 Encapsulation Trick

An important source of complexity in reasoning about security is the mutual dependancy of secrecy and authenticity: every secret must be authenticated, and every authentication must be based on a secret. This feedback loop of positive and negative knowledge statements, logical weaving and interleaving of authentication and secrecy, often generates confusion. The method of *encapsulation* allows us to untie some such knots, and to make some general patterns of formal reasoning actually and unexpectedly *simpler* then the informal counterparts, which were their source.

In previous work [1], we simplified authenticity proofs by encapsulating the secrecy assumptions on which they depended: e.g., an assumption that a key, used for authentication, was uncompromised, was simply left open, to be discharged in a separate proof module. A more common approach would have been to unfold the proof that the key remains secret within the given protocol. This proof would require an assumption that the key was authenticated — and this assumption would then usually be left open, as an assumption about the infrastructure. The authentication performed in the analyzed protocol would thus be reduced to another authentication, performed in the preceding key establishment protocol. While this approach is reasonable, unfolding a secrecy proof within each authenticity proof, and vice versa, does seem to complicate things, and does not seem necessary. In some cases, reducing authentications to the relevant secrecy assumptions, that encapsulate their own authenticity assumptions, often allows considerably simpler, and more insightful proofs.

Outline of the Paper. We present a method to derive secrecy properties of key establishment protocols, while assuming, and encapsulating for later implementation, the needed authentication properties of their components. Section 2 opens the exposition with an informal overview of the counterpart of this approach, where the authentication properties are derived, while the secrecy properties are assumed and encapsulated. An outline of the general framework used in protocol derivations is in the Appendix. Section 3 introduces the relations needed for modeling secrecy, and the derivation rules needed for proving it. Section 4 provides a method for proving a family of inductive statements to which the crucial rule reduces the secrecy statements. Section 5 presents the basic derivation templates for two key establishment patterns: key distribution and key agreement. Section 6 concludes the paper with a summary of its contributions.

2 Overview of Authentication with Encapsulated Secrecy

The goal of an authentication protocol $\mathcal{Q}$ for a group G is to realize the authentication predicate

$$\mathsf{Auth}^{\mathcal{Q}}(G) = \forall XY \in G.\ \overline{L}_X^{\mathcal{Q}} \approx \overline{L}_Y^{\mathcal{Q}}$$

where $\overline{L}_X^{\mathcal{Q}}$ is the *complete view* of the principal X at the final state of the protocol $\mathcal{Q}$. The relation $\approx$ requires the views to be equal, except for the last send-receive pair, which the sender cannot ascertain. The predicate $\mathsf{Auth}^{\mathcal{Q}}(G)$ thus formalizes entity authentication through "matching histories", introduced in [7] and formalized in [8]. Message authentication can be captured in a similar way, by requiring that principals' records match when restricted to the actions containing the relevant term:

$$\mathsf{Auth}^{\mathcal{Q}}(t; G) = \forall XY \in G.\ \left(\overline{L}_X^{\mathcal{Q}} \restriction t\right) \approx \left(\overline{L}_Y^{\mathcal{Q}} \restriction t\right)$$

To represent, say, a freshly generated value, we take t to be a variable x bound by an action (νx) in $\mathcal{Q}$. Its propagation is tracked through the send, receive and assign actions in the various runs, and its authenticity means that the various principals' views of this coincide at the final state of each run. For a set of terms Θ, we abbreviate $\mathsf{Auth}(\Theta; G) = \bigwedge_{t \in \Theta} \mathsf{Auth}(t; G)$.

In general, the complete view $\overline{L}_X^{\sigma}$ of the principal X at the state σ is obtained by applying the available *authentication axioms* and logical rules to the (incomplete) view L_X^{σ}, which consists of the actions observed by X up to σ. The views grow as the runs progress. Each principal directly observes only her own actions (and the actions of her subprincipals); but the authentication axioms allow her to draw conclusions about the actions of others, roughly in the form: "If I am receiving this message, someone must have sent it."

The authentication axioms are subsumed under the authentication template

$$A:\ \forall x.\ \Phi(x) \wedge ((f^{AB}x))_A \Longrightarrow \Psi(x) \wedge \langle\langle f^{AB}x\rangle\rangle_{B<} < ((f^{AB}x))_A \qquad (\mathsf{au})$$

instantiated to particular formulas Φ and Ψ. Here $((t))_A$ means *"A receives a message containing t"*, and $\langle\langle t\rangle\rangle_{A(<)}$ means *"A sends (originates) a message containing t"* [6]. The prefix "A :" means that the schema is used in A's local reasoning. The most important instance of this schema is the challenge-response axiom

$$A:\ (\nu x)_A\Big(\langle\langle c^{AB}x\rangle\rangle_A < ((r^{AB}x))_A$$
$$\Longrightarrow \langle\langle c^{AB}x\rangle\rangle_A < ((c^{AB}x))_B < \langle\langle r^{AB}x\rangle\rangle_{B<} < ((r^{AB}x))_A\Big) \qquad (\mathsf{cr})$$

obtained by setting

$$\begin{aligned} f^{AB}x &= r^{AB}x \\ \Phi(x) &= \mathsf{New}(x) \wedge \langle\langle c^{AB}x\rangle\rangle_A < ((r^{AB}x))_A \\ \Psi(x) &= ((c^{AB}x))_B < \langle\langle r^{AB}x\rangle\rangle_B \end{aligned}$$

in (au), and abbreviating "$\forall x.\ \mathsf{New}(x) \Rightarrow$" to "$(\nu x)$". This axiom allows a principal A, who can only observe her own actions, to draw conclusions about B's actions, using the assumption that no one but B could have transformed $c^{AB}x$ to $r^{AB}x$. Axiom (cr) should thus be construed as a specification of the property of the challenge-response functions c and r, required for the authentication. This property is simply asserted (postulated) for the abstract challenge-response protocol, but the task is to refine this protocol, and implement c and r as more concrete cryptographic functions, which come with their own axioms, from which (cr) can then be derived as a theorem [6,1]. This is where the encapsulated *secrecy assumptions* enter scene. For instance, when the challenge-response functions are implemented as

$$\begin{aligned} c^{AB}x &= x \\ r^{AB}x &= S^B(A, x) \end{aligned}$$

where S^B is B's signature, then axiom (cr) is implied by the statement that this signature is B's secret, i.e. that only he can generate it:

$$A : \langle\langle S^B x\rangle\rangle_{X<} \Longrightarrow X = B$$

And this, furthermore, can only be true if the session where B's signature is established has been authenticated, so that no other principal can come in possession of his signature. And that authentication depended on some previous secrets, and so on.

In general, given a protocol refinement $\mathcal{Q}(f) \longrightarrow \mathcal{Q}(F/f)$, where an abstract function f is implemented as a more concrete function F, refining the authentication proof often requires an additional assumption that the function F, used to authenticate group G, is G's secret

$$\mathsf{Auth}^{\mathcal{Q}(f)}(G) \wedge \mathsf{Secr}^{\mathcal{Q}(F/f)}(F; G) \Longrightarrow \mathsf{Auth}^{\mathcal{Q}(F/f)}(G)$$

For instance, when the abstract response function r^{AB} is refined to signature S^B, axiom (cr) will be satisfied because S^B is B's secret, and no one else could generate his signed response.

Similarly, in order to derive a secrecy property of a refinement F of an abstract operation f in a protocol $\mathcal{Q}$, we shall often need the authenticity assumption, in the form

$$\mathsf{Secr}^{\mathcal{Q}(f)}(f; G) \wedge \mathsf{Auth}^{\mathcal{Q}(F/f)}(F; G) \Longrightarrow \mathsf{Secr}^{\mathcal{Q}(F/f)}(F; G)$$

The goal of the rest of the paper is to make this precise.

3 Modeling Secrecy

In this section we introduce the conceptual and notational infrastructure needed for the formal definition of secrecy and the rules for deriving it. There are several layers of structure, and the tempo is brisk: the details must be left for the subsequent sections.

3.1 Order and Security

The simplest process model suitable for capturing the various aspects of security seems to be the based on partial orders, or more precisely partially ordered multisets (pomsets) [9]. It extends the trace based models, such as strand spaces [10], and simplifies the process-calculus based models.

While authenticity is achieved through partial ordering of actions, secrecy is concerned with the *computability* relation between terms, as they propagate through communication. The abstract pomset model, used for deriving authenticity, is now extended by an abstract computability relation. This will suffice for secrecy derivations. The model (outlined in the Appendix) consisted of:

- partial order of *terms* and subterms $(\mathcal{T}, \sqsubseteq)$,
- partial order of *principals* and subprincipals $(\mathcal{W}, \Subset)$,
- set of *actions* $\mathcal{A}$ generated over the terms,
- *processes* as partially ordered multisets of actions [9], i.e. maps $\mathbb{L} \xrightarrow{L} \mathcal{A} \times \mathcal{W}$, where $\mathbb{L}$ is a partial order, and
- *runs*, which extend processes by assigning to each receive action a unique send action

Now we add:

- partial order $\Gamma \vdash \Theta$ between finite sets $\Gamma, \Theta \subseteq \mathcal{T}$, meaning that each term $t \in \Theta$ can be computed from a tuple[1] of terms $\boldsymbol{g} \in \Gamma$.

In the present paper, we study the useful symbolic interpretations of computability relation.

3.2 Symbolic Computability

In general, the computability relation among the terms used in a protocol can be given by rewrite rules, e.g.

$$x, y \vdash \langle x, y \rangle \qquad \langle x, y \rangle \vdash x, y \qquad k, x \vdash Ekx \qquad k, Ekx \vdash x$$

where $\langle x, y \rangle$ represents a pairing operation, and Ekx the encryption of x by k. Abadi and Rogaway [11] use such computability relation. Paulson's **analz** operator [12] corresponds to the closure $\Gamma^{\vdash} = \{t \in \mathcal{T} \mid \Gamma \vdash t\}$ for the special case of the theory of encryption and tupling. More generally, given an algebraic theory

[1] We often abbreviate $g_1, g_2, \ldots, g_n$ to $\boldsymbol{g}$.

T with a signature Σ_T, and the set of derived operations $\overline{\Sigma}_T$, the computability relation can be defined by

$$\Gamma \vdash \Theta \iff \forall t \in \Theta \exists \boldsymbol{g} \in \Gamma \exists \varphi \in \overline{\Sigma}_T.\ T \models (t = \varphi \boldsymbol{g})$$

In words, for every element $t \in \Theta$ we can find a tuple $\boldsymbol{g} \in \Gamma$ and an algebraic operation φ, such that the equation $t = \varphi\boldsymbol{g}$ can be proven in algebra T. The rewrite rules given above are obtained for the algebraic theory T over the signature $\Sigma_T = \big\{\langle -, - \rangle, \pi_1, \pi_2, E, D\big\}$ and the equations

$$\pi_1\langle x, y\rangle = x \qquad \pi_2\langle x, y\rangle = y \qquad Dk(Ekx) = x$$

where we make the restriction that π_i is applied only to the result of concatenation and Dk is applied only to the result of applying Ek. [2] Note that the encryption E and decryption D are presented as *curried* binary operations on keys and messages. This will simplify notation in the sequel: e.g. the encryption and decryption by a key k become unary operations

Notation. It is convenient to extend the computability relation from the elements of $\mathcal{T}$ to functions $f : \mathcal{T} \longrightarrow \mathcal{T}$, as follows:

$$\begin{aligned}
\Gamma \vdash f &= \forall x.\ \Gamma, x \vdash fx \\
\Gamma \vdash f^{-1} &= \forall x.\ \Gamma, fx \vdash x \\
\Gamma, f \vdash t &= \forall \Xi.\ \Gamma, \Xi \vdash f \Rightarrow \Gamma, \Xi \vdash t \\
\Gamma, f^{-1} \vdash t &= \forall \Xi.\ \Gamma, \Xi \vdash f^{-1} \Rightarrow \Gamma, \Xi \vdash t
\end{aligned}$$

This extends to sets of functions in the obvious way. E.g., if $E_G = \{E_X : \mathcal{T} \to \mathcal{T} | X \in G\}$ is a family of public key encryptions for the principals from some group $G \subseteq \mathcal{W}$, then $\Gamma \vdash E_G^{-1}$ means that the keys to decrypt the messages encrypted by any of $E_X \in E_G$ can be computed from Γ.

For an algebraic theory T with signature Σ_T, the above definition of computability implies that $x \vdash fx$, i.e. $\vdash f$, holds for every $f \in \Sigma_T$.

Finally, each order relation induces a closure operator on sets of terms:

$$\Gamma^{\vdash} = \{t \in \mathcal{T} \mid \Gamma \vdash t\} \qquad \Gamma^{\sqsupseteq} = \{t \in \mathcal{T} \mid \exists u \in \Gamma.\ \ u \sqsupseteq t\}$$

3.3 Guard Relation

A set of sets of terms $\mathcal{G} \in \wp\wp\mathcal{T}$ is said to *guard* a term t with respect to a set of terms $\Upsilon \subseteq \mathcal{T}$ if every computation of t from Υ must traverse some $\Gamma \in \mathcal{G}$, i.e.

$$\mathcal{G} \ \mathsf{guards}_{\Upsilon}\ \Theta \quad = \quad \forall t \in \Theta \exists \Gamma \in \mathcal{G} \forall \Xi \subseteq \Upsilon.\ \Xi \vdash t \Rightarrow \Xi \vdash \Gamma$$

We say that $\mathcal{G}$ guards Θ in a process L if it guards it with respect to the set of terms $\Upsilon = \mathcal{T}_L$ which become computable to all observers in any run of L.

[2] We note that without such restrictions, not only are the theories not equivalent, but as pointed out in [13], it is possible to have protocols that are secure in the rewrite theory that are not secure in the algebraic theory.

This notion is implicit in many symbolic analyses of secrecy. Since the environment Υ depends on the possible runs of L (of which there can be many!), proving that a term is guarded can be complex. Our approach is to prove that guardedness is satisfied if certain syntactic conditions on the runs that are easy to verify using the authentication logic are also satisfied. This allows us to encapsulate the complex secrecy proofs as well. A framework for a large class of such proofs is presented in section 4.

3.4 Secrecy Predicates and Rules

The information available to a principal $A \in \mathcal{W}$ at a state[3] σ in a run ℓ of a process $\mathbb{L} \xrightarrow{L} \mathcal{A} \times \mathcal{W}$ is conveniently subdivided into

- a *view* $L_A^\sigma : \mathbb{L}_A^\sigma \longrightarrow \mathcal{A} \times \mathcal{W}_{\Subset A}$, which is the restriction of the run $L = \langle L_\mathcal{A}, L_\mathcal{W}\rangle : \mathbb{L}^\ell \longrightarrow \mathcal{A} \times \mathcal{W}$ to the actions executed before the state σ by A's subprincipals, i.e. to the subposet $\mathbb{L}_A^\sigma = \{\xi \in \mathbb{L}^\ell \mid \xi \leq \blacktriangledown_\sigma \wedge L_\mathcal{W}\xi \Subset A\}$, and
- an *environment* Γ_A^σ, which consists of the fresh variables from σ_A^2 and the terms which occur in σ_A^3, which together capture all terms that A has generated, received or computed up to the state σ.

Secrecy of a set of terms Θ for a group $G \subseteq \mathcal{W}$ at a state σ is then defined[4]

$$\begin{aligned}
\mathsf{Have}^\sigma(\Theta; G) &= \forall X \in G.\ \Gamma_X^\sigma \vdash \Theta \\
\mathsf{Only}^\sigma(\Theta; G) &= \forall X \in \mathcal{W} \forall t \in \Theta.\ \Gamma_X^\sigma \vdash t \Longrightarrow X \in G \\
\mathsf{Secr}^\sigma(\Theta; G) &= \mathsf{Have}^\sigma(\Theta; G) \wedge \mathsf{Only}^\sigma(\Theta; G) \\
&= \forall X \in \mathcal{W} \forall t \in \Theta.\ \Gamma_X^\sigma \vdash t \iff X \in G
\end{aligned}$$

Lemma 1. *(a) For $\Phi \in \{\mathsf{Have}, \mathsf{Only}, \mathsf{Secr}\}$ holds*

$$\Phi(\Theta_1; G) \wedge \Phi(\Theta_2; G) \iff \Phi(\Theta_1 \cup \Theta_2; G)$$

and therefore

$$\Theta_1 \supseteq \Theta_2 \Longrightarrow \Phi(\Theta_1; G) \Rightarrow \Phi(\Theta_2; G)$$

(b) Furthermore

$$\begin{aligned}
\mathsf{Have}(\Theta; G_1) \wedge \mathsf{Have}(\Theta; G_2) &\iff \mathsf{Have}(\Theta; G_1 \cup G_2) \\
\mathsf{Only}(\Theta; G_1) \wedge \mathsf{Only}(\Theta; G_2) &\iff \mathsf{Only}(\Theta; G_1 \cap G_2)
\end{aligned}$$

and therefore

$$G_1 \supseteq G_2 \Longrightarrow \mathsf{Have}(\Theta; G_1) \Rightarrow \mathsf{Have}(\Theta; G_2) \wedge \mathsf{Only}(\Theta; G_2) \Rightarrow \mathsf{Only}(\Theta; G_1)$$

[3] The definitions are in the Appendix.

[4] Note that $\mathsf{Only}(\Theta; G)$ is logically equivalent to $\forall X \in \mathcal{W}(\exists t \in \Theta.\ \Gamma_X^\sigma \vdash t) \Longrightarrow X \in G$.

Notation. It will be convenient to introduce the relation of *relative* computability

$$\Xi \vdash^{\sigma}_{A} \Theta = \Xi, \Gamma^{\sigma}_{A} \vdash \Theta$$

We write $\Xi \vdash^{Q}_{A} \Theta$ and $\Xi \vdash_{A} \Theta$ for computability in all runs of a protocol Q. We also elide the empty set, and write $\vdash^{\sigma}_{A} \Theta$ instead of $\emptyset \vdash^{\sigma}_{A} \Theta$. For a group $G \subseteq \mathcal{W}$, we write

$$\Xi \vdash_{G} \Theta = \forall X \in G.\ \Xi \vdash_{X} \Theta$$

Rules. The basic steps in the derivations of secrecy will be

$$\frac{\mathsf{Have}^{\sigma}(\Xi; G) \qquad \Xi \vdash^{\sigma}_{G} \Theta}{\mathsf{Have}^{\sigma}(\Theta; G)}\,(\mathsf{have}) \qquad \frac{\mathsf{Only}^{\sigma}(\Xi_i; G_i)\ \Big|_{i=1}^{n} \qquad \{\Xi_i\}_{i=1}^{n}\ \mathsf{guards}\ \Theta}{\mathsf{Only}^{\sigma}(\Theta; \bigcup_{i=1}^{n} G_i)}\,(\mathsf{only})$$

$$\frac{\mathsf{Secr}^{\sigma}(\Xi_i; G_i)\ \Big|_{i=1}^{n} \qquad \Xi_i \vdash^{\sigma}_{G_i} \Theta\ \Big|_{i=1}^{n} \qquad \{\Xi_i\}_{i=1}^{n}\ \mathsf{guards}\ \Theta}{\mathsf{Secr}^{\sigma}(\Theta; \bigcup_{i=1}^{n} G_i)}\,(\mathsf{secr})$$

4 Proving Guards

In this section, we explore the ways in which the guarding assumptions of (only) and (secr) can often be proved.

In practice, guards are often realized using functions which are hard to invert, or functions which are easy to invert with a key, but hard to invert without it. In modern cryptography, such functions are developed as *one-way* functions, and as *trapdoor* functions respectively [14, 2.4]. Very roughly, the idea is that the output of a one-way function does not contribute to computations that may yield its input

$$\Gamma, Fk \vdash k \Longrightarrow \Gamma \vdash k$$

whereas a trapdoor function allows computing a part m of its input only if another part k, called *trapdoor*, is also computable

$$\Gamma, Ekm \vdash m \Longrightarrow \Gamma \vdash k\ \vee\ \Gamma \vdash m$$

However, these formulations abstract away the fact that a function may not be invertible on its own, but may become invertible in a context, via equations like $Dk(Ekx) = x$. The definitions that we propose below capture this fact, but remain an algebraic approximation, inevitably crude, of concepts that essentially involve probabilistic computation. However, this initial simplification seems necessary for incremental proofs of secrecy, and open for refinements to more precise models.

4.1 Guarding by One-Way Functions

Fix a term algebra $\mathcal{T}$, given with a subterm relation $\sqsubset$, a set of function symbols Σ, and a computability relation $\vdash$.

If $f \in \Sigma$ is a function symbol of arity n and $i \leq n$, we call a pair $\langle f, i\rangle$ a *position*, and denote by $f(\sigma)_i$ a term in the form $fs_1s_2\ldots s_{i-1}\sigma s_{i+1}\ldots s_n$, i.e. where σ occurs as i-th argument of f.

Definition 1. *Given sets $\Upsilon, \Theta \subseteq \mathcal{T}$ of terms and a set $\Pi \subseteq \Sigma \times \mathbb{N}$ of positions, we extract the Υ-terms which occur in Θ just in the Π-positions by the operator*

$$\mathsf{par}_\Upsilon(\Theta, \Pi) = \left\{\sigma \in \Upsilon \setminus \Theta \;\mid\; \forall t \in \Theta \forall s \sqsubseteq t \forall f \in \Sigma \forall i \in \mathbb{N}.\ s = f(\sigma)_i \;\Rightarrow\; \langle f, i\rangle \in \Pi\right\}$$

The context Υ is often all of the term algebra $\mathcal{T}$, or the set $\mathcal{T}_L^{\sqsupseteq}$ of the subterms from the set $\mathcal{T}_L$ of the terms that may be sent in runs of the process L. We write $\mathsf{par}_L(\Theta, \Pi) = \mathsf{par}_{\mathcal{T}_L^{\sqsupseteq}}(\Theta, \Pi)$, and $\mathsf{par}(\Theta, \Pi) = \mathsf{par}_{\mathcal{T}}(\Theta, \Pi)$.

Definition 2. *A term σ is purged from a position $\langle f, i\rangle$ in the terms of Θ by replacing in every $t \in \Theta$, all occurrences of σ as i-th argument of f by a fresh variable x, not occurring in Θ. Formally,*

$$\mathsf{pur}_\sigma(\Theta, \langle f, i\rangle) = \left\{t[f(x)_i] \mid t[f(\sigma)_i] \in \Theta\right\}$$

where $t[f(\sigma)_i]$ displays all the occurrences of $f(\sigma)_i$ in t. Then $\mathsf{pur}_\Upsilon(\Theta, \Pi)$ is obtained by sequentially purging all $\sigma \in \Upsilon$ the $\sqsubset$-maximal ones first.

Note that, if σ never occurs in a position from Π, then $\mathsf{pur}_\Upsilon(\Theta, \Pi) = \emptyset$.

Lemma 2. *The operation $\mathsf{pur}_\Upsilon(\Theta, \Pi)$ is well defined: the order of purges of the individual terms $\sigma \in \Theta$ does not matter, as long as the maximal terms are purged first.*

Proof. This follows from the fact that two terms are either disjoint, or one is entirely contained in another.

Definition 3. *A position set Π is said to be one-way if it satisfies*

$$\mathsf{Onwy}(\Pi) = \forall \Theta\ \forall \sigma \in \mathsf{par}(\Theta, \Pi).\ \ \Theta \vdash \sigma \quad\Longrightarrow\quad \mathsf{pur}_\sigma(\Theta, \Pi) \vdash \sigma$$

We write $\mathsf{Onwy}(f, i)$ instead of $\mathsf{Onwy}(\{\langle f, i\rangle\})$, and we write $\mathsf{Onwy}(f)$ instead of $\mathsf{Onwy}(\{\langle f, 1\rangle, \ldots, \langle f, n\rangle\})$, where f is an n-ary function symbol.

Lemma 3. *For the function E, with the computability relation as defined in sec. 3.2, holds $\mathsf{Onwy}(E, 1) \wedge \neg\mathsf{Onwy}(E, 2)$.*

Proof. The second conjunct is a consequence of the rewrite $k, Ekm \vdash m$. For the proof of the first, let $\mathcal{D} = \Gamma_1 \vdash \ldots \vdash \Gamma_n \vdash \sigma \in \mathsf{par}(\Theta, \langle E, 1\rangle)$ where $\Gamma_1 \subseteq \Theta$ and each $\vdash$-step consists of a single application of a rewrite rule. We want to show that if $\mathcal{D}$ is a derivation, then so is the result of applying the purge function to each term in $\mathcal{D}$. The proof will be by induction on the length of $\mathcal{D}$. The result clearly holds when the length of $\mathcal{D}$ is one. Suppose the result holds for length less than n. Suppose $\mathsf{pur}_\sigma(\Theta, \Pi) \not\vdash \sigma$. Then one of the steps Γ_i in $\mathcal{D}$ must be of the form $S \cup \{\sigma, E\sigma y\} \vdash S \cup \{\sigma, E\sigma, y\}$ or $S \cup \{\sigma, y\} \vdash S \cup \{\sigma, y, E\sigma y\}$. Then $\Gamma_1 \vdash \ldots \vdash \Gamma_i$ is a derivation of σ, and we are done.

Definition 4. *We say that t is only sent under one-way functions in the runs of a process L if there is a one-way position set Π such that $t \in \mathsf{par}(\mathcal{T}_L, \Pi)$, where $\mathcal{T}_L$ is the set of terms of L.*

A term t is protected in L if it is only sent under one-way functions, or not at all. The set of the terms protected in L is thus

$$\mathsf{prot}_L(\Theta, \Pi) = \mathsf{par}_L(\mathcal{T}_L, \Pi) \cup (\mathcal{T} \setminus \mathcal{T}_L^{\exists})$$

Proposition 1. *If a term t is only sent under one-way functions in the runs of a process L, then t cannot be derived by the observers of these runs. Formally,*

$$\mathsf{Onwy}(\Pi) \wedge t \in \mathsf{par}_L(\Gamma_X, \Pi) \Longrightarrow \Gamma_X \nvdash t$$

Proof. Suppose that t is derivable from the terms Θ in a run. By the definition of Onwy, $\mathsf{pur}_t(\Theta, \Pi) \vdash t$. But by assumption, t does not appear in $\mathsf{pur}_t(\Theta, \Pi)$. Since t is an atomic term, that means that t cannot be derived from $\mathsf{pur}_t(\Theta, \Pi)$, and so it cannot be derived from Θ.

Corollary 1. *If a term t is protected in a process L, and in the initial environments of all principals, then no run of L will make it computable for any principal for which it was not computable initially, before any runs of L, i.e.*

$$\mathsf{Onwy}(\Pi) \;\wedge\; t \in \mathsf{prot}_L(\mathcal{T}_L, \Pi) \;\wedge\; \forall X \in \mathcal{W}.\ t \in \mathsf{prot}_L(\Gamma_X^{\iota}, \Pi) \;\Longrightarrow\; \mathsf{Only}(t; G_t)$$

where $G_t = \{X \in \mathcal{W} \mid \Gamma_X^{\iota} \vdash t\}$.

Proposition 2. *Let L be a process, and let k be an atomic term which only appears as the first argument of E in any term in the runs of L. Then, k cannot be derived by the observers of these runs.*

Proof. Since k is atomic, and the $\vdash$ relation introduces no new variables on the right-hand side, the fact that k does not appear in any element of $\mathsf{pur}_k(\mathcal{T}_L, \langle E, 1\rangle)$ implies $\mathsf{pur}_k(\mathcal{T}_L, \langle E, 1\rangle) \nvdash k$. Since $\langle E, 1\rangle$ is one-way (by lemma 3), definition 3 yields $\Theta \nvdash k$.

Remark. If $\Theta = \{E(Fkm)n\}$, and $\mathsf{Onwy}(E, 1)$, then $\Theta \nvdash Fkm$. Note, however that this does not imply $\Theta \nvdash k$, since the algebraic theory may include, e.g. the equation $E(Fxy)z = x$. However, if $\mathsf{Onwy}\{\langle E, 1\rangle, \langle F, 1\rangle\}$, then $\Theta \nvdash k$ can indeed be proven.

4.2 Guarding by Trapdoor Functions

Definition 5. *Given sets $\Upsilon, \Theta \subseteq \mathcal{T}$ and $\Psi \subseteq \Sigma \times \mathbb{N} \times \mathbb{N}$, with the projections $\Psi_0 = \{\langle f, i\rangle \mid \exists k.\ \langle f, i, k\rangle \in \Psi\}$ and $\Psi_1 = \{\langle f, k\rangle \mid \exists i.\ \langle f, i, k\rangle \in \Psi\}$, then the operator*

$$\begin{aligned}\mathsf{pad}_{\Upsilon}(\Theta, \Psi) \quad = \quad \big\{\langle \sigma, \kappa\rangle \in \Upsilon^2 \setminus \Theta^2 \;\mid\; & \forall t \in \Theta \forall s \sqsubseteq t \forall f \in \Sigma \forall i \in \mathbb{N}.\ \big(s = f(\sigma)_i \\ & \Rightarrow \exists k \in \mathbb{N}.\ \langle f, i, k\rangle \in \Psi \;\wedge\; s = f(\kappa)_k\big)\big\}\end{aligned}$$

extracts just those pairs of terms $\langle \sigma, \kappa\rangle \in \Upsilon^2$ where σ only occurs in a Ψ_0-position, if κ occurs in a Ψ_1-position within the same subterm.

Like before, $\mathsf{pad}(\Theta, \Psi)$ stands for $\mathsf{pad}_{\mathcal{T}}(\Theta, \Psi)$. We further define

$$\mathsf{msg}_{\Upsilon}(\Theta, \Psi) = \mathsf{par}_{\Upsilon}(\Theta, \Psi_0) \qquad \mathsf{key}_{\Upsilon}(\Theta, \Psi)\sigma = \{\kappa \mid \langle \sigma, \kappa\rangle \in \mathsf{pad}_{\Upsilon}(\Theta, \Psi)\}$$

Definition 6. *We say that $\Psi \subseteq \Sigma \times \mathbb{N} \times \mathbb{N}$ is a trapdoor set if for every $\langle f, i, j\rangle \in \Psi$ holds that whenever an $\langle f, i\rangle$-subterm σ can be extracted from $f(\sigma)_i$, then some $\langle f, j\rangle$-subterm κ must also be computable. Formally,*

$$\mathsf{Trap}(\Psi) = \forall \Theta\ \forall \sigma \in \mathsf{msg}(\Theta, \Psi).\ \Theta \vdash \sigma \implies \exists \kappa \in \mathsf{key}(\Theta, \Psi)\sigma.\ \Theta \vdash \kappa$$

We abbreviate $\mathsf{Trap}(\{\langle f, i, k\rangle\})$ to $\mathsf{Trap}(f, i, k)$, and for n-ary f write $\mathsf{Trap}(f, k)$ instead of $\mathsf{Trap}(\{\langle f, 1, k\rangle, \ldots, \langle f, n, k\rangle\})$.

Proposition 3. *The function E from sec. 3.2 satisfies* $\mathsf{Trap}(E, 2, 1)$.

Proof. Let Θ be a set of terms, $\Pi = \langle E, 1\rangle$, and $\sigma \in \mathsf{msg}(\Theta, \Psi)$ such that $\Theta \vdash \sigma$ but $\Theta \not\vdash g$ for any $g \in \mathsf{key}(\Theta, \Psi)$. Let $\mathcal{T}$ be a sequence $T_1 \vdash \ldots \vdash T_n \vdash \sigma$ where $T_1 \subseteq \Theta$ and each $\vdash$ step consists of a single application of a rewrite rule. Then some T_i must be of the form $S \cup \{k, Eky\} \vdash S \cup \{k, Eky, y\}$ where $\sigma \sqsubseteq y$. Hence $\Theta \vdash k$ and $k \in \mathsf{key}(\Theta, \Psi)$, contradicting our assumption.

The following three results follow directly from the definitions, so we omit their proofs.

Lemma 4. *For every Θ and a trapdoor set Ψ holds*

$$\big\{\mathsf{key}(\Theta, \Psi)\sigma \mid \sigma \in \mathsf{msg}(\Theta, \Psi)\big\}\ \ \mathsf{guards}_{\Theta}\ \ \mathsf{msg}(\Theta, \Psi)$$

Proposition 4. *If a term σ is only sent under trapdoor functions in the runs of a process L, then every computation leading to σ must pass through a trapdoor κ. Formally,*

$$\mathsf{Trap}(\Psi)\ \wedge\ \sigma \in \mathsf{msg}(\mathcal{T}, \Psi) \implies \mathsf{key}(\mathcal{T}, \Psi)\sigma \neq \emptyset.$$

Corollary 2. *If a term σ is only sent under trapdoor functions in the runs of a process L, and if all of its trapdoors are contained in Γ, then σ is guarded by Γ, i.e.*

$$\mathsf{Trap}(\Psi)\ \wedge\ \sigma \in \mathsf{msg}(\mathcal{T}_L, \Psi)\ \wedge\ \mathsf{key}(\mathcal{T}_L, \Psi)\sigma \subseteq \Gamma \implies \{\Gamma\}\ \mathsf{guards}_L\ \sigma$$

Remark. One-way functions implement a simple form of guarding, where s is guarded by the function symbol f in the term fs. Trapdoor functions implement guarding where s can be extracted from a term fst only if t is known. Equations between terms lead to more complicated forms of guarding. E.g., if $g^\wedge(-)$ is a one-way function satisfying $(g^\wedge y)^\wedge z = (g^\wedge z)^\wedge y$, then z cannot be extracted from $(g^\wedge y)^\wedge z$, but this term can be computed without knowing z, from $\{g^\wedge z, y\}$.

5 Secrecy Derivations

5.1 Key Distribution

Consider the abstract key distribution protocol $\mathsf{KD}[A, B, S]$

$$\begin{tikzcd}
A & S & B \\
\circ \arrow[r, "{A,B}"] & \circ \arrow[d, "\nu x"] & \\
\circ & \circ \arrow[l, "g_{SA}x"'] \arrow[r, "g_{SB}x"] & \circ
\end{tikzcd}$$

The principal A here requests a key to communicate with B and the server S generates a fresh key and distributes it, guarded by the functions g_{SA} and g_{SB}. The desired secrecy property is that at the final state of KD, only A,B and S have x, provided that they are honest.

To assure this, we impose the following axioms

$$\forall Y \in \mathcal{W}.\ \mathsf{Secr}^{\iota}(g_{SY}, g_{SY}^{-1}; S, Y) \tag{sg}$$

$$\mathsf{Auth}^{\mathsf{KD}}(g_{SA}, g_{SA}^{-1}, g_{SB}, g_{SB}^{-1}; A, B, S) \tag{ag}$$

$$\mathsf{Auth}^{\mathsf{KD}}(x; A, B, S) \tag{ax}$$

which say (sg) that at the *initial* state ι, only S and Y can construct and remove g_{SY}, and (ag,ax) that A, B and S can each ascertain that there are no undesired flows involving g's and x. The desired secrecy property is now obtained using the secrecy rule

$$\dfrac{\dfrac{(\mathsf{sg}) \wedge (\mathsf{ag})}{\mathsf{Secr}^{\mathsf{KD}}(g_{SY}^{-1}; S, Y)\,\big|_{Y\in\{A,B\}}} \qquad \dfrac{g_{SY}x \in \Gamma^{\mathsf{KD}}_{S,Y}\,\big|_{Y\in\{A,B\}}}{g_{SX}^{-1} \vdash_{S,Y} x\,\big|_{Y\in\{A,B\}}} \qquad \dfrac{(\mathsf{ax}) \wedge \mathsf{H}(A,B,S)}{\big\{\{g_{SA}^{-1}\},\{g_{SB}^{-1}\}\big\}\ \mathsf{guards}\ x}}{\mathsf{Secr}^{\mathsf{KD}}(x; A, B, S)}$$

where $\mathsf{H}(A, B, S)$ is the usual honesty assumption, that principals act according to the protocol. To see how $(\mathsf{ax}) \wedge \mathsf{H}(A, B, S)$ implies that $\big\{\{g_{SA}^{-1}\},\{g_{SB}^{-1}\}\big\}$ guards x, recall that $\mathsf{Auth}(x; A, B, S)$ means that $\big(\overline{L}_A \restriction x\big) \approx \big(\overline{L}_S \restriction x\big) \approx \big(\overline{L}_B \restriction x\big)$, at the final state of KD. In other words, all three principals A, B and S can in each run of KD establish the same complete view of all of their actions with the terms containing x. Of course, each of them observes only her own actions, but the assumption $\mathsf{Auth}(x; A, B, S)$ asserts that this suffices to allow each of them to also prove the exact order of actions of the other two. In particular, A and B can both prove that S has only sent $g_{SA}x$ and $g_{SB}x$, and nothing else.

In this reasoning, the authentication assumptions are encapsulated in axioms (ag) and (ax), which *postulate* that there are no runs viewed differently by different principals. This means that neither of them can be proven, nor needs to be proven, for the abstract protocol. They just specify requirements which the implementations of the abstract protocol need to satisfy. This is analogous to the derivational approach to authentication, where the axiom (cr) specifies the required property of the challenge-response functions c and r, as an implementation task. The task here is to implement the guard function g, and discharge the open authenticity assumption. The abstract secrecy property of KD, proven above, must be preserved and realized through such implementations.

Refining KD. Suppose that we are given an encryption algorithm E, with the decryption algorithm D, and a symmetric key k_{SY} shared by the server S with every $Y \in \mathcal{W}$. Assume that these data satisfy the following axioms[5]

$$\mathsf{Onwy}\langle E, 1\rangle \tag{oE}$$
$$\mathsf{Trap}\langle E, 2, 1\rangle \tag{tE}$$
$$\forall Y \in \mathcal{W}.\mathsf{Secr}^{\iota}(k^{SY}; S, Y) \tag{sk}$$

First Try. Setting $g_{SY}x = Ek_{SY}x$, one can easily derive (sg) from (sk), and the secrecy of k_{SY} follows from (sk), (oE), and $\mathsf{Auth}^{\mathsf{KD}}(k_{SY}; Y, S)$. However, (ag)[6] and (ax) are not satisfied, because $g_{SY} = Ek_{SY}$ allows runs which may leave each principal with a different view, e.g.

$$\begin{tikzcd}
A & S & C \\
\circ \arrow[rr, "{A,B}"] & & \circ \arrow[d] \\
& \circ \arrow[d, "\nu x"] & \circ \arrow[l, "{A,C}"'] \\
\circ & \circ \arrow[l, "Ek_{SA}x"'] \arrow[r, "Ek_{SC}x"] & \circ
\end{tikzcd}$$

Here S does not know that A wanted to speak to B, A does not know that C participates the run, and B does not know that anything happened at all.

Second Try. The principal A needs to prove (ax) that S has only sent x in the two tokens, intended for A and B, guarded by g_{SA} or g_{SB}; and (ag) that the guards have not been compromised. Since A does not have g_{SB}^{-1}, she cannot check (ax) directly. So she must check it indirectly, relying for the authentications (ag) and (ax) on the information received from S. This idea is realized by adding peer's identity in the term $g_{SA}x$, constructed by S for A. Ditto for $g_{SB}x$.

Proposition 5. *The protocol* KD, *with axioms* (sg,ag,ax), *can be refined by setting*

$$g_{SA}x = Ek_{SA}Bx \qquad g_{SB}x = Ek_{SB}Ax$$

[5] For simplicity, the term algebra is here untyped, and any term can be used as a key; typing can be introduced as needed.

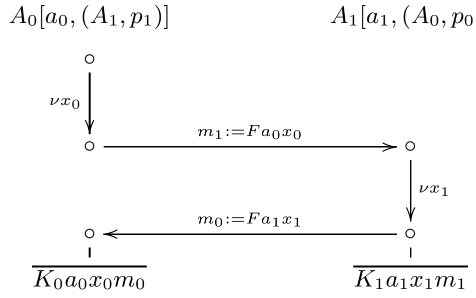
[6] Note that $\mathsf{Trap}\langle E, 2, 1\rangle$ implies $\mathsf{Auth}(Ek, Ek^{-1}) \iff \mathsf{Auth}(k)$.

provided that (oE) $\mathsf{Onwy}\langle E, 1\rangle$*,* (tE) $\mathsf{Trap}\langle E, 2, 1\rangle$*, and* (sk) $\mathsf{Secr}^{\iota}(k^{SY}; S, Y), Y \in \{A, B\}$ *are satisfied. Under these assumptions, the proof of* KD*'s secrecy property specializes to its refinement.*

The validity of (sg) follows as in the first try. The validity of (ag) and (ax) follows from the honesty assumption: if A receives $Ek_{SA}Bx$, then the presence of B in that message means the only other token that the honest server S would send is $Ek_{SB}Ax$.

5.2 Key Agreement

The generic key agreement protocol KA is based on abstract public key infrastructure: each principal $X \in \mathcal{W}$ is given a long term private key a_X, corresponding to a public key p_X, as well as an "address book" $\{(Y, p_Y)\}_{Y \in \mathcal{W}}$ of public keys of all principals. This means that the view Γ_X^{σ} for every σ and X contains $\{a_X, (Y, p_Y) \mid Y \in \mathcal{W}\}$. But the participants of the abstract key agreement protocol $\mathsf{KA}[A_0, A_1]$ only need to know each other's public key, so it becomes



Besides the displayed secret data, the operations F and K_i may also depend on the public data (but this clutters notation and plays no role in the reasoning). The following axioms are imposed:

$$\mathsf{Onwy}(F) \tag{oF}$$

$$\forall X \in \mathcal{W}.\ \mathsf{Secr}^{\iota}(a_X; X) \tag{sa}$$

$$\mathsf{Auth}^{\mathsf{KA}}(a_0, x_0, a_1, x_1; A_0, A_1) \tag{aax}$$

$$K_0a_0x_0(Fa_1x_1) \;=\; K_1a_1x_1(Fa_0x_0) \tag{agr}$$

$$\big\{\{a_0, x_0\}, \{a_1, x_1\}\big\} \text{ guards } \{K_0a_0x_0m_0, K_1a_1x_1m_1\} \tag{gua}$$

The secrecy rule now gives

$$\dfrac{\dfrac{(\mathsf{oF}) \wedge (\mathsf{sa}) \wedge (\mathsf{aax}) \wedge \mathsf{H}(A_0, A_1)}{\mathsf{Secr}^{\mathsf{KA}}(a_i, x_i; A_i)\ \big|_{i \in \{0,1\}}} \qquad \dfrac{m_i \in \Gamma_{A_i}^{\mathsf{KA}}\ \big|_{i \in \{0,1\}}}{a_i, x_i \vdash_{A_i} K_ia_ix_im_i\ \big|_{i \in \{0,1\}}} \qquad (\mathsf{gua})}{\mathsf{Secr}^{\mathsf{KA}}(K_0a_0x_0m_0, K_1a_1x_1m_1; A_0, A_1)}$$

It further follows from (aax) that the messages Fa_0x_0 and Fa_1x_1 are exchanged as desired and assigned to m_1 and m_0 respectively. Axiom (agr) then implies that at the final state of KA, the secret keys $K_0a_0x_0m_0$ and $K_1a_1x_1m_1$ agree, and yield the key $k = K_0a_0x_0m_0 = K_0a_1x_1m_1$. This is the desired outcome of the protocol.

Refining KA. The abstract KA-scheme subsumes the large family of protocols arising from Diffie and Hellman's paradigm [15]. It includes the MTI family and their descendants, UM, KEA, MQV and others. In combination with other security components, they are used in several widely used protocol suites [16,17].

The minimal algebraic structure needed for the DH-style key agreement is a multiplicative group $\mathbb{G}$ with a binary operation $(-^\wedge-) : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}$, satisfying

$$x^\wedge 1 = x \tag{eq1}$$

$$(x^\wedge y)^\wedge z = (x^\wedge z)^\wedge y \tag{eq2}$$

$$\mathsf{Onwy}(x^\wedge) \tag{dl}$$

$$\{\{x^\wedge y, z\}, \{x^\wedge z, y\}\} \ \mathsf{guards}_{\mathsf{KA}} \ \{(x^\wedge y)^\wedge z, (x^\wedge z)^\wedge y\} \tag{cdh}$$

Axiom (dl) is the abstract version of the Discrete Logarithm (DL) assumption, and (cdh) is related to the Computational Diffie-Hellman (CDH) assumption.

Besides the group $\mathbb{G}$, the assumed infrastructure includes a chosen element $g \in \mathbb{G}$, intended to generate a large subgroup of $\mathbb{G}$. For each principal $X \in \mathcal{W}$, the public key p_X, corresponding to the long-term private key a_X is now specified in the form $p_X = g^\wedge a_X$. So for all X and σ, the view Γ_X^σ now contains a_X, g and $\{(Y, g^\wedge a_Y)\}_{Y \in \mathcal{W}}$.

The DH-style key agreement protocols now fall into two subfamilies, depending on whether the key derivation functions K_0 and K_1 are the same or not.

Asymmetric Key Derivation. Given functions H_0 and H_1, that satisfy

$$H_0 a_0 x_0 (F a_1 x_1) \ = \ H_1 a_1 x_1 (F a_0 x_0) \tag{agrH}$$

$$\{\{a_0, x_0\}, \{a_1, x_1\}\} \ \mathsf{guards} \ \{H_0 a_0 x_0 (F a_1 x_1), H_1 a_1 x_1 (F a_0 x_0)\} \tag{guaH}$$

setting

$$K_0 axm = J(H_0 axm)(H_1 axm) \qquad K_1 axm = J(H_1 axm)(H_0 axm)$$

validates (agr) and (gua). This subsumes the general scheme of the first two MTI protocols [18]

protocol	Fax	Jxy	$H_0 axm$	$H_1 axm$
MTI/A(f)	$g^\wedge(fax)$	$x \cdot y$	$p^\wedge(fax)$	$m^\wedge a$
MTI/B(f)	$p^\wedge(fax)$	$x \cdot y$	$g^\wedge(fax)$	$m^\wedge(1/a)$

The variable p still denotes peer's public key. When proving (agr), substitute each p_i by $g^\wedge a_i$. The original MTI protocols are obtained by setting $fxy = x^i \cdot y$. For $i = 1$, replacing the function $Jxy = xy$ in MTI/A by $Jxy = x+y$ and extending the group axioms with ring axioms yields the core of the KEA protocol; taking Jxy to append and then hash x and y yields the core of the "Unified Model" protocol (UM). The security properties gained by these variations, discussed in [19,17], are beyond the scope of our current axioms, but can be captured in refinements.

Symmetric Key Derivation. Given a function K, satisfying

$$Ka_0x_0(Fa_1x_1) \;=\; Ka_1x_1(Fa_0x_0) \qquad \text{(agrK)}$$

$$\{\{a_0, x_0\}, \{a_1, x_1\}\} \text{ guards } \{Ka_0x_0(Fa_1x_1), Ka_1x_1(Fa_0x_0)\} \qquad \text{(guaK)}$$

can, of course, implement both K_0 and K_1 in the protocol KA. This subsumes the third MTI protocol, and the scheme which we denote MQV/D.

protocol	Fax	$Kaxm$
MTI/C(f)	$p^\wedge(fax)$	$(m^\wedge(1/a))^\wedge(fax)$
MQV/D(f)	$g^\wedge(fax)$	$(Rpm)^\wedge(ra(fax))$

where R and r are required to satisfy

$$x^\wedge(ryz) = R(x^\wedge y)(x^\wedge z) \qquad \text{(agrR)}$$

$$\{a, x\} \text{ guards } ra(fax) \qquad \text{(guar)}$$

Instantiating to $Rxy = (x^\wedge y) \cdot y$ and $rxy = x \cdot (g^\wedge y) + y$ validates these axioms and gives the MQV protocol [20,21], but there are other interesting choices. Instantiating $Rxy = rxy = fxy = y$ yields the DH protocol, which, of course, does not validate (guar).

Proposition 6. *The protocol* KA, *with axioms* (oF,sa,aax,agr,gua), *can be refined as above, using the asymmetric key derivation functions* H_0, H_1, *or the symmetric key derivation scheme* K, *provided that axioms* (oF,sa,aax,agrX,guaX) *hold for* $\mathsf{X} \in \{H, K\}$.

Using the algebraic structure satisfying (eq1,eq2,cdh,dl), *the protocols* MTI/A, MTI/B,UM *and* KEA *can be obtained as further refinements of the asymmetric scheme, whereas the protocols* MTI/C *and* MQV/D *are further refinements of the symmetric scheme, assuming, in the latter case, also* (agrR,guar).

The generic secrecy property, proven for the protocol KD, *is inherited by all of its refinements.*

6 Conclusions

The main contribution of this work is an extension of the framework for composing and refining security protocols, that allows proofs of the secrecy properties realized by the protocols. The secrecy concepts are defined in terms of the abstract computability relation, and the secrecy derivations are built from the rules derivable from the minimal assumptions about this relation. The secrecy properties are derived from the axioms attached to the basic protocol components. The axioms specify the requirements/assumptions that need to be discharged through refinement and implementation of abstract operations. Some axioms encapsulate the authenticity assumptions. Since they embody a different, logically dual concern from secrecy, these assumptions are cumbersome to realize concurrently with it. Encapsulating them in some cases significantly simplifies the secrecy proofs. The genericity of the approach allows reusable treatment of broad families of related protocols. All derivations are stored, and were originally constructed, in the prototype version of a software tool, which is freely available for download [3]. The structure of the presented modeling methodology is very much influenced by its role of the semantical underpinning of this tool.

There are also interesting further directions to be explored. Although our initial explorations in this area are based on a symbolic model, there is no need to limit ourselves in this direction. Indeed, we could develop different secrecy logics with different semantics based on, for example, information-theoretic or computational aspects of cryptography, depending on the types of cryptosystems being used. The ability to derive and employ different secrecy logics would allow us to develop a pluggable semantics for cryptographic protocol analysis that would allow us to reason over multiple domains.

Acknowledgements. We thank Carolyn Talcott for careful reading and many useful comments.

References

1. Cervesato, I., Meadows, C., Pavlovic, D.: An encapsulated authentication logic for reasoning about key distribution protocols. In Guttman, J., ed.: Proceedings of CSFW 2005, IEEE (2005) 48–61
2. Anlauff, M., Pavlovic, D., Waldinger, R., Westfold, S.: Proving authentication properties in the Protocol Derivation Assistant. In: Proceedings of ARSPA 2006. Lecture Notes in Computer Science, IEEE (2006) to appear.
3. Anlauff, M., Pavlovic, D.: Pda download web site. `http://www.kestrel.edu/software/pda` (2003–6)
4. Durgin, N., Mitchell, J., Pavlovic, D.: A compositional logic for proving security properties of protocols. J. of Comp. Security **11**(4) (2004) 677–721
5. Datta, A., Derek, A., Mitchell, J., Pavlovic, D.: A derivation system and compositional logic for security protocols. J. of Comp. Security **13** (2005) 423–482
6. Meadows, C., Pavlovic, D.: Deriving, attacking and defending the GDOI protocol. In Ryan, P., Samarati, P., Gollmann, D., Molva, R., eds.: Proc. ESORICS 2004. Volume 3193 of Lecture Notes in Computer Science., Springer Verlag (2004) 53–72
7. Diffie, W., van Oorschot, P.C., Wiener, M.J.: Authentication and Authenticated Key Exchanges. Designs, Codes, and Cryptography **2** (1992) 107–125
8. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Proc. CRYPTO '93, Springer-Verlag (1994) 232–249
9. Pratt, V.: Modelling concurrency with partial orders. Internat. J. Parallel Programming **15** (1987) 33–71
10. Fabrega, F.J.T., Herzog, J., Guttman, J.: Strand spaces: What makes a security protocol correct? Journal of Computer Security **7** (1999) 191–230
11. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). J. of Cryptology **15**(2) (2002) 103–127
12. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security **6** (1998) 85–128
13. Millen, J.: On the freedom of decryption. Information Processing Letters **86**(6) (2003) 329–333

14. Goldreich, O.: Foundations of Cryptography. Volume I: Basic Tools. Cambridge University Press (2000)
15. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **IT-22**(6) (1976) 644–654
16. Kaufman, C., Perlman, R., Speciner, M.: Network Security. Private Communication in a Public World. 2 edn. Computer Networking and Distributed System. Prentice Hall PTR (2002)
17. Boyd, C., Mathuria, A.: Protocols for Authentication and Key Establishment. Information Security and Cryptography. Springer-Verlag (2003)
18. Matsumoto, T., Takashima, Y., Imai, H.: On seeking smart public-key distribution systems. Transactions of the IECE (Japan) **69** (1986) 99–106
19. Blake-Wilson, S., Menezes, A.: Authenticated Diffie-Hellman key agreement protocols. In: SAC '98: Proceedings of the Selected Areas in Cryptography, London, UK, Springer-Verlag (1999) 339–361
20. Menezes, A., Qu, M., Vanstone, S.: Some new key agreement protocols providing mutual implicit authentication. In: SAC '95: Proceedings of the Selected Areas in Cryptography, London, UK, Springer-Verlag (1995) 22–32
21. Law, L., Menezes, A., Qu, M., Solinas, J., Vanstone, S.: An efficient protocol for authenticated key agreement. Technical Report CORR 98-05, University of Waterloo (1998) also in [22].
22. P1363 Working Group: The IEEE P1363 home page. standard specifications for public-key cryptography. `http://grouper.ieee.org/groups/1363/` (2005)
23. Abadi, M., Burrows, M., Lampson, B., Plotkin, G.: A calculus for access control in distributed systems. ACM Transactions on Programming Languages and Systems **21**(4) (1993) 706–734
24. Lampson, B., Abadi, M., Burrows, M., Wobber, E.: Authentication in distributed systems: theory and practice. ACM Trans. on Comput. Syst. **10**(4) (1992) 265–310
25. Myers, A.C., Liskov, B.: Protecting privacy using the decentralized label model. ACM Transactions on Software Engineering and Methodology **9**(4) (2000) 410–442
26. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7) (1978) 558–565

Appendix: Protocol Derivation System

Syntactic Categories: Conceptually, authenticity and secrecy are dual, in the sense that authenticity says that some principals know something, while secrecy says that that they do not know something else. Formally, though, both secrecy and authenticity can be analyzed in terms of partial ordering: principals authenticate each other by establishing the same order of their joint actions, whereas the failures of secrecy can be viewed as connecting some data that are known, and some that should not be known through a relation of computability, which is transitive and reflexive. So we reason over several partially ordered syntactic categories:

– *data* are represented as terms from an algebra $\mathcal{T}$, given with an abstract subterm relation $\sqsubseteq$;
  - to support information-theoretic security analyses, $\mathcal{T}$ must be given with a frequency distribution $\mathsf{Prob} : \mathcal{T} \longrightarrow [0, 1]$;

- to support computational security analyses, besides the frequency distribution, $\mathcal{T}$ must be given with a representation $\mathcal{T} \longrightarrow \Sigma^*$ of terms as strings (usually from the alphabet $\Sigma = \{0, 1\}$), available for processing e.g. by turing machines;

– *principals* are collected in the set $\mathcal{W}$ ordered by the relation $\Subset$, which may be interpreted as "speaks for" [23,24] or "acts for" [25], etc.

– *actions* are generated from terms and principals by the constructors including

action	constructor	form	meaning
send	$\mathcal{T} \times \mathcal{W}^2 \overset{\langle\rangle}{\hookrightarrow} \mathcal{A}$	$\langle t : A \to B\rangle$	the term t is sent, purportedly from A to B
receive	$\mathsf{Var}_{\mathcal{T}} \times \mathsf{Var}^2_{\mathcal{W}} \overset{()}{\hookrightarrow} \mathcal{A}$	$(x : Y \to Z)$	a term, source and destination are received into the variables x, Y, and Z
match	$\mathcal{T} \times \Sigma_{\mathcal{T}} \times \mathsf{Var}_{\mathcal{W}} \overset{(/)}{\hookrightarrow} \mathcal{A}$	$(t/p(x))$	the term t is matched with the pattern $p(x)$
new	$\mathsf{Var}_{\mathcal{T}} \overset{(\nu)}{\hookrightarrow} \mathcal{A}$	(νx)	a fresh value is created and stored in the variable x

We often use partial descriptions, and elide e.g. the source and the destination of a message, as in $\langle t\rangle$, or (y). For simplicity, we also omit the explicit type structure. However, all term variables are assumed to be local to a principal, and thus come with a map $\mathsf{Var}_{\mathcal{T}} \longrightarrow \mathcal{W}$; each principal is assumed to have an infinite supply of local variables.

Execution Model: Processes are represented as partially ordered multisets (pomsets [9]) of actions attributed to principals. More precisely, a **process** is an assignment $\mathbb{L} \xrightarrow{L} \mathcal{A} \times \mathcal{W}$ such that (a) $(\mathbb{L}, <)$ is a well-founded partial order, and (b) $p < q$ implies $L_{\mathcal{W}}(p) \Subset L_{\mathcal{W}}(q)$ or $L_{\mathcal{W}}(p) \Supset L_{\mathcal{W}}(q)$. We abuse notation and write an action $p \in \mathbb{L}$ such that $L_{\mathcal{A}}p = a$ and $L_{\mathcal{W}}p = A$ as $a_A \in \mathbb{L}$, or even a, although, of course, several elements of $\mathbb{L}$ may correspond to the same action by the same principal.

A **run** ℓ extends a process L by a choice of *communication links*, which assign to each receive action a unique send action. Formally, a run is thus a pair

$$\ell \;=\; \langle L,\;\; \surd :\; \mathsf{recvs}(L) \longrightarrow \mathsf{sends}(L)\rangle \qquad (x : Y \to Z)_A \mapsto \langle t : S \to R\rangle_B$$

such that $\surd(x) \not> (x)$. A run ℓ thus induces the partially ordered set $\mathbb{L}^\ell$, obtained by extending the ordering of $\mathbb{L}$ by $\surd(x) < (x)$. One can thus think of a run as the pomset extension $\mathbb{L} \hookrightarrow \mathbb{L}^\ell$, where each receive action (x) from $\mathbb{L}$ has in $\mathbb{L}^\ell$ a chosen predecessor $\langle t\rangle = \surd(x)$. This is, of course, just another formalization of Lamport's ordering of actions [26]. The resulting model is also similar, and has been influenced by strand spaces and bundles [10].

The actions in a run $\ell = \langle \mathbb{L}, \surd\rangle$ are then executed in order: each $a \in \mathbb{L}$ can be executed only after all $b < a$ have been executed. Executing an action changes state. A **state** of a run ℓ is a triple $\sigma = \langle \sigma^1, \sigma^2, \sigma^3\rangle$, where

– $\sigma^1 \supset \mathbb{L}^\ell$ is a poset, extending each maximal chain of $\mathbb{L}^\ell$ by exactly one element, denoted $\blacktriangledown$, that marks the execution point;

– $\sigma^2 = \{x_1, \dots, x_m\}$ is a finite set of the variables bound to *freshly generated* nonces or keys,
– $\sigma^3 = \{y_1 := t_1, \dots y_n := t_n\}$ is a finite set of the assignments, that result from the executed actions; formally, it can be viewed as a partial map from the variables y_i to the terms t_i, such that its domain $\{y_1 \dots y_n\}$ is disjoint from $\sigma^2 = \{x_1 \dots x_m\}$.

As always, the variables are taken up to renaming (i.e. modulo α-conversion).

At the *initial state* ι of every run ℓ, all the markers in ι_1 are set below the minimal elements of $\mathbb{L}^\ell$, and $\iota_2 = \iota_3 = \emptyset$.If the execution of the run ℓ has reached a state σ, the transition $\sigma \longrightarrow \tau$ proceeds as follows[7]:

action	if in σ^1	and if	then set τ^3 to	set τ^2 to	and in τ^1
send	$^\blacktriangledown\langle t\rangle_C$	$FV(t) \subseteq \sigma^2$	σ^3	σ^2	$\langle t\rangle_C^\blacktriangledown$
receive	$^\blacktriangledown(x)_D$	$\surd(x)_D = \langle t\rangle_C \wedge\ x \notin \sigma^2$	$\sigma^3 \cup \{x := t\}$	σ^2	$(x)_D^\blacktriangledown$
match	$^\blacktriangledown(t/p(x))_D$	$t = p(u) \wedge\ x \notin \sigma^2$	$\sigma^3 \cup \{x := t\}$	σ^2	$(t/p(x))_D^\blacktriangledown$
new	$^\blacktriangledown(\nu x)_D$	$x \notin \sigma^2$	σ^3	$\sigma^2 \cup \{x\}$	$(\nu x)_D^\blacktriangledown$

The local state σ_A is the part of the state σ that can be observed by the principal A. Its components are thus

– the poset $\sigma_A^1 \subseteq \sigma^1$, spanned[8] by the actions of A;
– the set σ_A^2 obtained by restricting σ^2 to A's local variables, and
– the partial map σ_A^3, obtained by restricting σ^3 to A's local variables.

A **protocol** is a specification of a process and a nonempty set of desired runs. The participants of a protocol are usually called *roles*, and are denoted by the variables for principals. Since the pomset representing a run extends the pomset representing a process, and the restriction of the run to the process is usually obvious, a protocol is usually specified just by its desired runs. Such a specification should thus be construed as a proof task: show that the principals can prove that the run which they participated is as desired.

[7] For compactness of the table, we omit the source and the destination fields, which are just passed from the received message to the receiving variables.
[8] Just like σ^1 is a $\blacktriangledown$-marking of $\mathbb{L}^\ell$, σ_A^1 is a $\blacktriangledown$-marking of $\mathbb{L}_A^\ell = \{\xi \in \mathbb{L}^\ell \mid L_{\mathcal{W}}\xi = A\}$.

Limits of the BRSIM/UC Soundness of Dolev-Yao Models with Hashes

Michael Backes[1], Birgit Pfitzmann[2], and Michael Waidner[2]

[1] Saarland University, Saarbrücken, Germany
backes@cs.uni-sb.de
[2] IBM Zurich Research Lab, Switzerland
{bpf, wmi}@zurich.ibm.com

Abstract. Automated tools such as model checkers and theorem provers for the analysis of security protocols typically abstract from cryptography by Dolev-Yao models, i.e., abstract term algebras replace the real cryptographic operations. Recently it was shown that in essence this approach is cryptographically sound for certain operations like signing and encryption. The strongest results show this in the sense of blackbox reactive simulatability (BRSIM)/UC with only small changes to both Dolev-Yao models and natural implementations. This notion essentially means the preservation of arbitrary security properties under active attacks in arbitrary protocol environments.

We show that it is impossible to extend the strong BRSIM/UC results to usual Dolev-Yao models of hash functions in the general case. These models treat hash functions as free operators of the term algebra. This result does not depend on any restriction of the real hash function; even probabilistic hashing is covered. In contrast, we show that these models are sound in the same strict sense in the random oracle model of cryptography. For the standard model of cryptography, we also discuss several conceivable restrictions and extensions to the Dolev-Yao models and classify them into possible and impossible cases in the strong BRSIM/UC sense.

1 Introduction

Tools for proving security protocols typically abstract from cryptography by deterministic operations on abstract terms and simple cancellation rules. An example term is $\mathsf{E}_{pke_w}(\mathsf{hash}(\mathsf{sign}_{sks_u}(m, N_1), N_2))$, where m denotes a payload message and N_1, N_2 two nonces, i.e., representations of fresh random numbers. We wrote the keys as indices only for readability; formally they are normal operands in the term. A typical cancellation rule is $\mathsf{D}_{ske}(\mathsf{E}_{pke}(m)) = m$ for corresponding keys. The proof tools handle these terms symbolically, i.e., they never evaluate them to bitstrings. In other words, the tools perform abstract algebraic manipulations on trees consisting of operators and base messages, using only the cancellation rules, the message-construction rules of a particular protocol, and abstract models of networks and adversaries. Such abstractions, although different in details, are collectively called Dolev-Yao models after their first authors [23].

It is not obvious that a proof in a Dolev-Yao model implies security with respect to real cryptographic definitions. Recently, this long-standing gap was essentially closed by proving that an almost normal Dolev-Yao model of several important cryptographic system types can be implemented with real cryptographic systems secure according to standard cryptographic definitions in a way that offers blackbox reactive simulatability [8]. We abbreviate blackbox reactive simulatability by BRSIM in the following. This security (in other words soundness) notion essentially means that one system, here the cryptographic realization, can be plugged into arbitrary protocols instead of another system, here the Dolev-Yao model, without any noticeable difference [35,36]. Essentially the same notion is also called UC for its universal composition properties [16].[1] In other words, this result shows that the Dolev-Yao model as such can serve as an ideal functionality that is correctly implemented by a real functionality given by actual cryptographic systems. Extensions of this BRSIM/UC result to more cryptographic primitives were presented in [9,6], uses in protocol proofs in [5,3,4], stronger links to conventional Dolev-Yao-style type systems in [29], and an integration into the Isabelle theorem prover in [37]. Earlier results on relating Dolev-Yao models and real cryptography considered passive attacks only [2,1,27]. Later papers [31,28,18] consider to what extent restrictions to weaker security properties, such as integrity only, and/or less general protocol classes, e.g., a specific class of key exchange protocols, allow simplifications compared with [8]. [2]

No prior paper relating Dolev-Yao models and cryptography considers hashing or one-way functions although they are important operators in automated proof tools based on Dolev-Yao models, e.g., [30,34,13,11]. (A very recent report does, but only under passive attacks [24].) The standard model is that hash is a free operator in the term algebra, i.e., there is no inverse operator, nor any other cancellation rule with operators like E and D. Only a party who knows or guesses a potentially hashed term t can test whether hashing t equals a given hash term h. The goal of our paper is to close this gap, and to study how the soundness results in the sense of BRSIM/UC can be extended when hash or one-way functions are added to a Dolev-Yao model and its cryptographic implementation. In the following, we only speak of hash functions since the standard Dolev-Yao model for the two classes is the same.

1.1 Our Contributions

It turns out that proving BRSIM/UC soundness for Dolev-Yao models with hash functions is impossible in a general way. Note that the question is not whether a hash function is a good and generally usable cryptographic primitive by itself, but only whether its idealization as a free operator in a term algebra, or a similar plausible idealization, is sound in this strong and pluggable sense. Prior work showed that certain (classes

[1] Recent revisions of the long version of [16] also contain an explicit blackbox version of UC, which is proven to be equivalent to UC. A similar equivalence was first shown in the long version of [35] for universal and blackbox synchronous reactive simulatability.

[2] There is also work on formulating syntactic calculi for dealing with probabilism and polynomial-time considerations and encoding them into proof tools, in particular [32,33,26,22,14]. This is orthogonal to the work of justifying Dolev-Yao models, which offer a higher level of abstraction and thus much simpler proofs where applicable, so that proofs of larger systems can be automated.

of) ideal functionalities are not BRSIM/UC-securely realizable, e.g., for bit commitments [17], coin tossing, zero knowledge, and oblivious transfer [16], classes of secure multi-party computation [19] and certain game-based definitions [21]. However, none of these works investigated a Dolev-Yao model. Impossibility of BRSIM/UC soundness of a Dolev-Yao model with XOR was shown in [7]. For our case of hashes, the reasons for impossibility and thus the proofs are quite different. Furthermore, the proofs in [7] are reduction proofs, essentially saying that if an idealization of XOR and other cryptographic operations is soundly implementable in the sense of BRSIM/UC, it can be used to compute cryptographic algorithms and is therefore not intuitively Dolev-Yao. In contrast, we obtain absolute impossibility results. We achieve this by making stronger definitions on what makes an ideal functionality of hashing and other cryptographic operations a Dolev-Yao model.

It is important to note that there is so far no rigorous definition of "any Dolev-Yao model" in the literature that is independent of specific underlying system models such as CSP, π-calculus, IO automata, or strand spaces. For positive results, this is not a problem. However, an impossibility result that only holds for one such model would not be very convincing. (In particular, the closest model to build on would be that from [8], and due to syntax idiosyncrasies many people find it hard to transfer basic ideas from that model to others.) Hence, instead of proving impossibility for one specific Dolev-Yao model, we will only make certain assumptions on the Dolev-Yao model; we believe they are fulfilled by all such models existing so far. Essentially we only assume that the hash functions are abstracted as free operators as informally explained above, that they are applicable to arbitrary terms, and that the model contains some other typical operators and base types.

One reason (but not the only one) for the impossibility in the general case is that hash functions, at least those with a one-way property, are by nature committing, i.e., if one first gets $h = \mathsf{hash}(m)$ and later m one can validate whether indeed $h = \mathsf{hash}(m)$. It is well known that such a commitment property often causes problems in proofs of BRSIM/UC: If the simulator has to simulate a bitstring for h before knowing m, then whatever it picks will most likely not match m. Thus the simulation fails if m is later revealed. In some cases, the commitment problem can be circumvented by using non-standard models of cryptography, e.g., the random oracle model [12] or the common random string model, cf. [17]. Indeed we can show BRSIM/UC for the standard Dolev-Yao model of hashes if the cryptographic realization of the hash function is treated as a random oracle.

We can also consider probabilistic hashing [15,20]. First, it is not an alternative for justifying the Dolev-Yao abstraction of hashing by a free operator in the sense of BRSIM/UC: Instead one needs hash and verify operators that cancel, similar to authentication. (Even for the purely passive case this extension is made in [24].) Moreover, hashing the same message must produce a fresh term each time, similar to standard models of nonces, or the Dolev-Yao style model of probabilistic encryption in [8]. This freshness is needed because it is distinguishable in the real system whether a message was hashed twice, or whether an existing hash value was forwarded. Hence the implementation would be incorrect in the sense of BRSIM/UC if this were not possible in the ideal, Dolev-Yao style system. Even then, however, our impossibility results hold; we sketch this extension below. Roughly, this is so because probabilistic hashing mainly improves the secrecy of the hashed message; however, the imperfect secrecy offered by deterministic hash functions is not an argument in our counterexamples.

For the standard model of cryptography, the next question is whether certain restrictions on the use of hash functions enable a BRSIM/UC soundness result. One option is to restrict the types of terms that can be hashed, in particular to forbid the hashing of payloads. By "payload" we mean an application message, e.g., an email text or the amount and currency of a payment in a payment protocol that uses the cryptographic functionality. Technically, payloads play a special role (as m in the example of the commitment problem shows) because they are known outside the cryptographic system and thus can typically not be modified by the simulator, in contrast to nonces, keys, etc. As to practical usage, this restriction is serious but not unreasonable; e.g., key exchange protocols typically do not use general payloads, and indeed [18] relies on their absence. However, we still obtain an impossibility result if excluding payloads is the only restriction on hashable terms. The basic idea in that case is that by constructing large enough terms, the users of the cryptographic system can simulate payloads. Another conceivable restriction is therefore on the size of hashable terms. Again this restriction is serious (e.g., general hash chains and trees are now excluded) but not unreasonable because many protocols only use rather small terms. In fact, for the restriction to hashing single nonces, we obtain a positive result, but so far not for any other type of restriction to small terms. This result may seem surprising: A typical counterargument is "how can this be true if we allow real hash functions that leak parts of the message?" The point is that nonces are system-internal, and thus leaking bits about them is not harmful by itself as long as the application-level properties of the Dolev-Yao model remain fulfilled, i.e., essentially that the adversary cannot guess nonces of which it has only seen hash values. Clearly this argument does not hold for payloads, and it also does not hold for keys, because leaking key bits would harm later key usage. An example of protocols that use hashing only for single nonces are protocols that use hashing only for one-time signatures.

Another restriction is to give up the ideal secrecy property of the hash functions, i.e., to give at least the ideal adversary an operator that inverts hash. The technical motivation is that this clearly prevents the commitment problem. On the application side, this restriction excludes all protocols where one-wayness is the core property for which a hash function is used. However, there are protocols where shortening of messages with collision resistance is the core property desired; here such a model could be used. Note that in a Dolev-Yao model we can have collision freeness, i.e., no equality between hashes of different terms, without secrecy. The realization of such an operator by a real cryptographic function would of course still have to be collision-resistant and thus one-way if it is sufficiently shortening. Anyway, we still obtain an impossibility result in this case: No shortening hash function exists that enables a secure realization of a Dolev-Yao model with hashes even without secrecy. If we combine giving up secrecy with not hashing payloads and only terms of constant size, then we obtain a positive result.

Of course the restrictions that we considered are not the only conceivable ones; in particular it may be interesting to find other positive cases in the standard model of cryptography than the two that we prove. Furthermore, we do not exclude that even the standard Dolev-Yao model of hashes may be sound with respect to weaker soundness definitions.

2 Summary of Reactive Simulatability/UC, Also with Random Oracle

As our results are for the security definitions of BRSIM/UC, we first briefly review this notion. BRSIM/UC is used for comparing an ideal and a real system with respect to security [35,36,16]. We believe that our following results are independent of the small differences between the definition styles and therefore write "BRSIM/UC" and similar term pairs like "ideal system/functionality". For the actual results, we have to use a specific formalism, and we use that from [36]. Here one speaks of ideal and real systems (the functionalities and protocols of UC). The ideal system is often called TH for "trusted host", see Figure 1, and the protocol machines of the real system are often called M_u, where u is a user index. The ideal or real system interacts with arbitrary so-called honest users, often collectively denoted by a machine H; this corresponds to potential protocols or human users to whom the functionality is offered. Furthermore, the ideal or real system interacts with an adversary, often denoted by A, who is often given more power than the honest users; in particular in real systems A typically controls the network. A and H can interact; this corresponds to known- and chosen-message attacks etc.

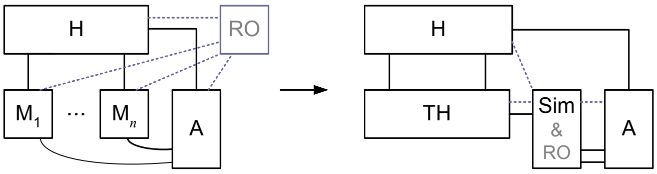


Fig. 1. Overview of blackbox reactive simulatability with a real system on the left and an ideal system on the right, and a potential random oracle. The views of H must be indistinguishable.

Reactive simulatability between the real and ideal system essentially means that for every attack on the real system there exists an equivalent attack on the ideal system. More specifically, blackbox reactive simulatability (BRSIM) states that there exists a simulator Sim that can use an arbitrary real adversary as a blackbox, such that arbitrary honest users cannot distinguish whether they interact with the real system and the real adversary, or with the ideal system and the simulator with its blackbox. Indistinguishability, here applied to the two families of views of the honest users, is the well-known notion from [38]. We always assume that all parties are polynomial-time. The original formulation of BRSIM as sketched above would allow the simulator to also modify the interaction between A and H; however, all known positive BRSIM proofs work as in Figure 1. The reason is similar to why all known positive RSIM proofs are BRSIM proofs: It is hard to make concrete use of the communication of two unknown machines A and H just as of the program text of the unknown machine A. We therefore show impossibility for this usual, stronger notion of BRSIM. Moreover, the blackbox version of UC also does not allow the simulator access to the corresponding interaction (called adversary and environment there), i.e., it corresponds to Figure 1 immediately.

A formal representation of random oracles in the UC notation was given in [25]. This can be used one-to-one in the BRSIM terminology. In a real system, each machine has distinguished connections for querying the random oracle RO, which is the usual stateful machine from [12] that generates a random string as "hash value" for each message m when it is first queried about m. In the ideal system with a simulator, these distinguished connections connect to the simulator, i.e., the simulator learns every query to the random oracle and can give arbitrary answers. This is also shown in Figure 1.[3]

3 Informal Overview of the Impossibility Proofs

In this section, we present our results as proof sketches with a minimum amount of notation. Later we define more notation and precise assumptions, and then extend the proof sketches to full proofs. A reader with a specific Dolev-Yao model in mind and not interested in the generalization to "arbitrary Dolev-Yao-like models" should be able to see already in this section that the results could be instantiated for that model.

In the following, messages may occur in several representations, which we distinguish by superscripts. We write terms in the Dolev-Yao sense without superscript, e.g., $h := \mathsf{hash}(m)$ for a hash term. The real cryptographic versions get a superscript $^\mathsf{r}$, e.g., $h^\mathsf{r} := \mathsf{hash}^\mathsf{r}(m^\mathsf{r})$ for the corresponding real bitstring, computed by applying a real hash function hash^r to the real representation m^r of m. The users and adversaries may concretely address the terms/bitstrings in yet another way when interacting with the real or ideal functionality (hopefully indistinguishably, hence we need only one notation); we write these representations with superscripts ^{u} for honest user u and $^\mathsf{a}$ for the adversary. (Using the actual terms as these representations is a special case.) In the figures we write H_u for the actual user u, which is a part of the global H in Figure 1.

3.1 Scenarios with Payloads

Our first scenario in Figure 2 demonstrates that the real hash function hash^r must at least be collision-resistant in order to offer a sound implementation of a Dolev-Yao model with hashes and payloads. This is not surprising, but we need the collision resistance in the next proofs. The proof idea is that otherwise the adversary can find two colliding payloads m^r and $m^{*\mathsf{r}}$ and send their hash h^r to an honest party u. It also exchanges m^r and $m^{*\mathsf{r}}$ secretly with the user u outside the system. Recall that such outside exchanges are allowed for chosen-message attacks etc. and that the notion of BRSIM/UC considered here does not allow the simulator to learn or change them, i.e., it must produce an overall indistinguishable view for H and A; we denote these outside exchanges by dashed arrows in the interaction figures.

[3] Alternatively, one could use a correct random oracle also in the ideal system and only allow the simulator to eavesdrop the queries of some or all parties. However, this weakens the power of the simulator considerably, and most of our impossibility proofs for the standard model of cryptography would hold for this model with only minor changes. Thus the strength of the simulator means a certain weakness in our positive results, but the need for this is another indicator that the impossibility problems are strong.

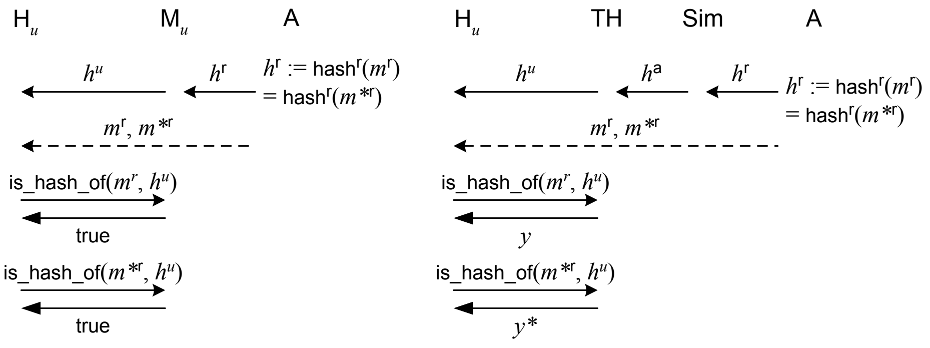

H_u
M_u
A
h^u
h^r
$h^r := \mathsf{hash}^r(m^r) = \mathsf{hash}^r(m^{*r})$
m^r, m^{*r}
is_hash_of(m^r, h^u)
true
is_hash_of(m^{*r}, h^u)
true
H_u
TH
Sim
A
h^u
h^a
h^r
$h^r := \mathsf{hash}^r(m^r) = \mathsf{hash}^r(m^{*r})$
m^r, m^{*r}
is_hash_of(m^r, h^u)
y
is_hash_of(m^{*r}, h^u)
y^*

Fig. 2. Counterexample with payload hashing for not collision-resistant hash function

Then user u uses the ideal or real functionality to check whether the message received through the system is the hash of both payloads. In the real system (on the left in all our interaction figures), the answer is true both times by the choice of h^r. However, the ideal collision freeness of the ideal system (on the right in all our interaction figures) does not allow this; hence the ideal and the real system are distinguishable.

The major challenge in formalizing this proof sketch is in the treatment of payloads, because most Dolev-Yao models do not put the actual payloads into the terms. Below we make precise assumptions on this treatment and define the ideal collision freeness, and then turn this sketch into a proof.

Our second scenario in Figure 3 demonstrates that even with a collision-resistant function hash^r, a sound implementation of a Dolev-Yao model with hashes and payloads is impossible if the ideal Dolev-Yao functionality offers ideal secrecy. By ideal secrecy we mean that an adversary who obtains the hash of an otherwise unknown term cannot do better than comparing this hash with self-made hashes of guessed terms. The scenario is that an honest party u selects a random payload m^r of length $2k$ (where k is the security parameter), sends it to the adversary outside the system, and sends the hash of this payload to the adversary through the ideal or real system. From the real system, the adversary gets the real hash $h^r := \mathsf{hash}^r(m^r)$, tests whether this is indeed the correct hash value of the payload, and tells the result to u outside the system. By the ideal secrecy, the simulator Sim for the ideal system cannot find out more about m^r than excluding polynomially many guesses. Using the collision resistance of the real hash function, we show that Sim can consequently only guess h^r with negligible probability, and thus cannot simulate this scenario correctly.

The technical difficulties with the full proof for this scenario lie in an appropriate formulation of the ideal secrecy, independent of one specific Dolev-Yao model.

Our third scenario in Figure 4 demonstrates that omitting the ideal secrecy requirement does not help as long as the real hash function is shortening as well as collision-resistant, and thus one-way. Here the real adversary agrees on a random payload m^r with an honest user u outside the system, and sends its hash h^r to u through the system. The user tests, using the ideal or real functionality, whether the obtained message is indeed a hash of m^r. In the real system, the output is clearly true. The best way for the

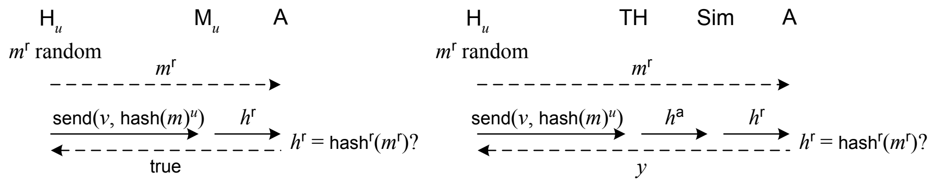


Fig. 3. Counterexample with payload hashing and ideal secrecy for collision-resistant hash function

simulator to cause the same output from the ideal functionality would be to send the term $h = \mathsf{hash}(m)$ via TH in the first step. However, this would require guessing m^{r} and thus breaking the one-way property of $\mathsf{hash}^{\mathsf{r}}$.

The difficulty with the full proof for this scenario, besides the question of payload representations as in the first scenario, is that $\mathsf{hash}(m)$ is not the only term that causes the output true. For instance, $\mathsf{D}(\mathsf{E}(\mathsf{hash}(m)))$ or $\mathsf{hash}(\mathsf{D}(\mathsf{E}(m)))$ for en- and decryption operators E and D are other such terms. We therefore have to be careful in how we can argue that every successful strategy for the simulator really leads to a successful algorithm that extracts m^{r} and thus breaks the one-way property.

3.2 Scenarios Without Payloads

After showing that payloads and hashes in Dolev-Yao models lead to comprehensive impossibility results for secure realizations in the sense of BRSIM/UC, we consider restricted Dolev-Yao models without payloads. However, as long as this is the only restriction, we still prove impossibility. The basic proof idea is to let the users and the adversary simulate payloads by encoding them into the structure of long terms. Concretely, we use a list of $2k$ nonces and encode a payload as a bit vector $\boldsymbol{b}$ that selects a sublist of these nonces. Instead of nonces, any other type can be used of which one can generate $2k$ instances that are ideally different, e.g., keys.

With a scenario similar to Figure 2, only adding the random choice of the nonces for the encoding, we show that a hash function must be collision-resistant on these bit vectors in order to offer a BRSIM/UC-sound implementation of a Dolev-Yao model with hashes and lists of nonces. Then with a scenario similar to Figure 3, we show that if ideal secrecy is offered no sound implementation exists at all. With a scenario similar to Figure 4, we show that even without ideal secrecy, no sufficiently shortening hash function, in particular one whose output length depends only on the security parameter, yields a sound implementation.

3.3 Probabilistic Hash Functions

Finally, we sketch why the scenarios show impossibility also for probabilistic hashing: The real hash function $\mathsf{hash}^{\mathsf{r}}$ can simply be probabilistic now. On the user side, checking whether a received term or message is the hash of a known message is already written is_hash_of; this is suitable also to express the new verification procedure. On the adversary side in Figure 3, the test "$h^{\mathsf{r}} = \mathsf{hash}^{\mathsf{r}}(m^{\mathsf{r}})$?" must be replaced by a call

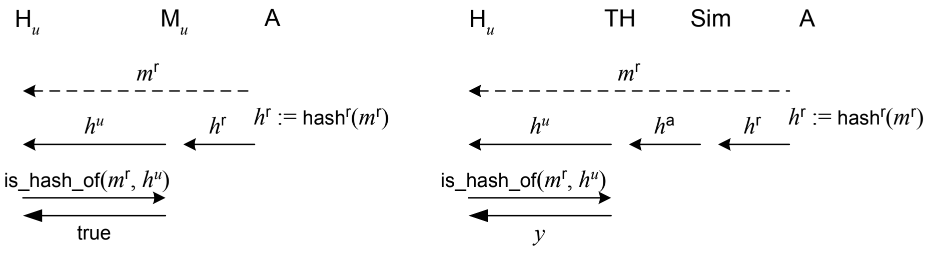


Fig. 4. Counterexample with payload hashing for shortening hash function

to the real verification algorithm. Everything else remains the same. For readability, we do not make these extensions in the following formal part, but stick to the standard free hash operator.

4 Assumptions on Dolev-Yao Models for Our Impossibility Results

As explained in the introduction, we would like to work out the impossibility proofs sketched in Section 3 not only for one specific Dolev-Yao model (then we could simply use an arbitrary definition from the literature and would not need our own definitions), but for all of them. However, "all Dolev-Yao models" is not a notion that anyone tried to formalize before. Hence we now characterize Dolev-Yao models and their realizations by weak rigorous requirements. In other words, we define common properties that are fulfilled by all standard Dolev-Yao models, and hopefully also by all other conceivable variants that might count as Dolev-Yao models. This makes our impossibility results as strong as possible.

4.1 Minimum Assumptions on a Dolev-Yao Model with Hashes

In this section we describe the functionality that we assume every Dolev-Yao system with hashes offers. We start with the basic notions of terms, including a hash operator. Recall that hash is essentially a free operator in the term algebra of typical Dolev-Yao models. However, we do not define this strong freeness, but only a weaker property, ideal collision freeness (both because this makes our results stronger, and because not all Dolev-Yao models are actually defined as initial models of an equational specification).

Definition 1 (Terms of a Dolev-Yao Model with Hashes). *We require that we can derive definitions of the following concepts from a Dolev-Yao model with hashes:*

a. A set $Terms$ denoting the overall set of valid terms. We speak of atoms *and* operators *denoting the potential leaves and inner nodes, respectively, of the terms considered as trees. The terms, atoms and operators may be typed. There is an equivalence relation "$\equiv$" on $Terms$. We call $(Terms, \equiv)$ the term algebra.*[4]

[4] Clearly syntactic term equality "=" implies equivalence. Typically "$\equiv$" is constructed from cancellation rules.

b. *A unary operator* hash*, which fulfils* ideal collision freeness*, i.e.,* $\mathsf{hash}(t) \equiv \mathsf{hash}(t') \Rightarrow t \equiv t'$ *for all* $t, t' \in \mathit{Terms}$*.*
c. *A set* $\mathit{Hashable\_Terms} \subseteq \mathit{Terms}$ *of the terms that are valid operands of the operator* hash*. We speak of a model with* unrestricted hashing *if* $\mathit{Hashable\_Terms} = \mathit{Terms}$*.*
d. *A list operator (possibly implemented by repeated pairing in the original syntax). Two lists are equivalent iff all their corresponding elements are.* $\diamond$

Next we define some minimum actions that the users and the adversary can carry out on the terms, and the results of these actions. In our context, this is the basis for showing that our impossibility scenarios are at least executable in every Dolev-Yao model (which probably nobody doubted).

While our notion of term representations t^u for individual users is certainly more general than notions that may be familiar to some readers, and thus can only strengthen our impossibility results, let us briefly motivate how it relates to such notions: An important concept in Dolev-Yao models is that of terms t constructible for some user u or the adversary (by applying operators and cancellation rules to previously known messages); however, the syntax for this concept varies considerably. Some high-level representations simply use t itself in the protocol representations (e.g., "$\mathsf{hash}(m)$" even when someone who does not know m forwards this term). More detailed representations, e.g., in CSP or π-calculus, typically use the concepts of variables inherent to these calculi, usually by matching received messages with a pattern describing the expected message format, and then using the pattern variables in subsequent message constructions. The syntax explicitly made for BRSIM/UC of the Dolev-Yao-style model in [8] uses local variables called handles and explicit parsing of received messages. The syntax from all these models can easily be mapped to that in our following definition.

We do not need a full definition of how a user acquires term representations. However, we define that terms can be sent and that the ideal adversary controls the network as usual in Dolev-Yao models. Furthermore we define that users can hash terms and compare whether one term is the hash of another term. In some, but not all, Dolev-Yao models this comparison can be made by using a general equality operator corresponding to the term equivalence $\equiv$.

Definition 2 (Actions on a Dolev-Yao Model with Hashes). *Users and the ideal adversary can make at least the following inputs into the ideal functionality of a Dolev-Yao model with hashes, with the described results.*

a. *If an honest user* u *inputs* $\mathsf{send}(v, t^u)$ *for a term representation* t^u*, this leads to an output* $\mathsf{receive}(u, v, t^{\mathsf{a}})$ *for the adversary.*
b. *If the adversary inputs* $\mathsf{send}(u, v, t^{\mathsf{a}})$ *for a term representation* t^{a}*, this leads to an output* $\mathsf{receive}(u, t^v)$ *for user* v *(i.e., the adversary impersonates* u*).*
c. *If a user* u *(honest or the adversary represented by* $u = \mathsf{a}$*) has a term representation* t^u*, then it also has a representation for the term* $\mathsf{hash}(t)$*. (Typically this is something like the string "*$\mathsf{hash}(t^u)$*".)*
d. *An input* $\mathsf{is\_hash\_of}(t^u, h^u)$ *by a user* u *(honest or* a*) leads to a Boolean output* y *for user* u *with* $y = \mathsf{true}$ *iff* $h \equiv \mathsf{hash}(t)$*.* $\diamond$

4.2 Payload Assumptions

All Dolev-Yao models in real proof tools have at least payload messages, nonces, and keys as atoms. However, as payloads are particularly problematic in simulations and some protocol classes do not need general payloads, we define Dolev-Yao models with and without them. A payload m models an application message, i.e., its cryptographic realization m^{r} can be an arbitrary bitstring; examples are emails, payment messages, and digital pictures. In this sense, our scenarios in Section 3 are perfectly natural: The users and the adversary select payloads as arbitrary bitstrings. However, the internal representation of payloads in the terms in Dolev-Yao proof tools is usually a constant supply of payload names or a nonce-like construction of fresh names. We therefore assume that the full ideal functionality maintains a translation table between the real payloads that occur in a system execution and their internal representations.[5]

Definition 3 (Payloads in Dolev-Yao Models in the BRSIM/UC Setting). *A* Dolev-Yao model with payloads *allows us to derive a type (subset)* $\mathsf{payload}$ *in the set* Terms. *In every execution, every occurring payload term has a fixed realization* m^{r}*, and* $m^{\mathsf{r}} = m'^{\mathsf{r}}$ *implies* $m \equiv m'$*. The range of payload realizations* m^{r} *is at least* $\{0,1\}^{2k}$*. A real payload* m^{r} *can always be used as an input representation* m^u *by every user.* $\diamond$

We now consider how secret a hashed term is in a Dolev-Yao model when the adversary learns its hash. We only need this in our second scenario, where we want to show that an adversary receiving a (representation of) a term $t = \mathsf{hash}(m)$ containing a payload m cannot get significant information about the real payload m^{r} and thus its real hash. In normal Dolev-Yao models, the hash operator is free, and thus there is no inverse operator that the adversary can use to extract m, nor a sequence of such operators. In addition, in many Dolev-Yao models one would represent the initial situation where the adversary does not know m by not giving the adversary any representation of m, thus excluding any possibility that the adversary guesses m. With such a strong assumption the impossibility proof would be easy. However, we allow the more realistic case that the adversary might guess payloads (as, e.g., in [8]). Furthermore, we only make the minimum assumption that payloads are secret in hash terms except for this guessing.

Definition 4 (Ideal Secrecy of New Payloads). *A Dolev-Yao model with hashes offers* ideal secrecy of new payloads *iff the following holds: If user* u *inputs* $\mathsf{send}(v, h^u)$ *where* $h = \mathsf{hash}(m)$ *for a newly chosen payload* m^{r} *(i.e., one that was not input to the Dolev-Yao model before), then the ideal adversary, from its output* $\mathsf{receive}(u, v, h^{\mathsf{a}})$ *and without further involvement of the user* H*, cannot obtain more information about* m^{r} *than by learning for* x *bitstrings* m'^{r} *whether* $m'^{\mathsf{r}} = m^{\mathsf{r}}$ *(in addition to its a-priori*

[5] As an additional motivation for this assumption, recall that we want to compare the Dolev-Yao model and its cryptographic realization in the sense of BRSIM/UC. Thus they must offer the same syntactic user interfaces, i.e., in- and output formats. This holds for all definition variants of BRSIM/UC. In particular, in Figure 1 this is the interface between TH or $\mathsf{M}_1, \ldots, \mathsf{M}_n$, respectively, and the entirety of honest users H. In [16], it is the input and output formats of the ideal and real functionality. Syntactically different user interfaces would either simply prevent the same users from using alternatively the real or the ideal system, or lead to trivial distinguishability.

information), if it interacts at most x times with the ideal system and thus in particular if it runs in time x. ◇

Finally, we define the weak freeness property of Dolev-Yao hashes that we need for the third scenario. Essentially this is that without knowing a payload m or its real representation m^{r} one cannot construct a term equivalent to $\mathsf{hash}(m)$. Like other definitions of "knowledge" in cryptography, this is done by defining that the capability to construct such a term implies the capability to find out m^{r}. This reduction is done constructively by an extractor algorithm.

Definition 5 (Minimum Non-Constructibility of Unknown Payload Hashes). *A Dolev-Yao model with hashes offers* minimum non-constructibility of unknown payload hashes *if there exists a polynomial-time algorithm* Ext*, called extractor, such that the following holds: If the ideal adversary (for simplicity at the system start) makes a sequence of inputs and then sends a term t to an honest user such that $t \equiv \mathsf{hash}(m)$ for a payload m, then the extractor, given the transcript of the ideal adversary's in- and outputs, outputs m^{r}.* ◇

For Dolev-Yao models with well-defined and constructible normalizations of terms, the extractor is essentially this normalization: It constructs t and the relation of payload terms and their representations from the transcript (typically the transcript is simply of the form "$\mathsf{send}(v, t^{\mathsf{a}})$" where payloads in t^{a} are in their real representation) and normalizes t; the result is $\mathsf{hash}(m)$, from which m^{r} can be looked up. This clearly holds for typical Dolev-Yao models that only have constructors and destructors like encryption and decryption. It gets more complex in Dolev-Yao models with algebraic operations like XOR; however, specifically XOR is known not to be realizable in BRSIM/UC [7].

4.3 Nonce List Assumptions

For the case without payloads, our scenarios use lists of nonces. We therefore define what we assume about nonces (lists are already in Definition 1). The first assumption is extremely simple and normal, except that some basic Dolev-Yao models only allow a fixed number of nonces, while we need at least $2k$ (as does every Dolev-Yao model suitable for arguing about an unbounded number of sessions).

Definition 6 (Nonces in Dolev-Yao Models). *A* Dolev-Yao model with nonce lists *allows us to derive a type (subset)* nonce *in the set Terms. Every participant can use, or explicitly generate, at least $2k$ new nonces (we do not need a fixed syntax for this generation); such nonces are pairwise not equivalent.* ◇

The next definition extends the ideal secrecy of hashed terms, which we earlier defined only for new payloads, to new lists of nonces. More precisely, we define that an ideal hash term does not divulge which of the many potential sublists of a list of nonces was hashed. (We make these weak special assumptions to strengthen the impossibility results, and to avoid complex considerations about prior knowledge in the general case.)

Definition 7 (Ideal Secrecy of New Nonce Lists). *A Dolev-Yao model with hashes offers* ideal secrecy of new nonce lists *iff the following holds: Let user u generate $2k$*

new nonces $\boldsymbol{n} = (n_1, \ldots, n_{2k})$ *and potentially send them to the adversary, select a random bit vector* $\boldsymbol{b} = (b_1, \ldots, b_{2k}) \xleftarrow{\mathcal{R}} \{0,1\}^{2k}$, *and input* $\mathsf{send}(v, (\mathsf{hash}(\boldsymbol{b} \odot \boldsymbol{n}))^u)$ *where* $\boldsymbol{b} \odot \boldsymbol{n}$ *denotes the sublist consisting of the nonces* n_i *with* $b_i = 1$. *Then the ideal adversary, from its output* $\mathsf{receive}(u, v, h^{\mathsf{a}})$ *and without further interaction with the user* u, *cannot obtain more information about* $\mathbf{b}$ *than by learning for* x *bit vectors* $\boldsymbol{b'}$ *whether* $\boldsymbol{b'} = \boldsymbol{b}$, *if it interacts at most* x *times with the ideal system and thus in particular if it runs in time* x. ◇

Finally, we define that the ideal adversary cannot construct a hash over a sublist of nonces without knowing which sublist of the nonces it is using.

Definition 8 (Minimum Non-Constructibility of Unknown Nonce-list Hashes). *A Dolev-Yao model with hashes offers* minimum non-constructibility of unknown nonce-list hashes *if there exists a polynomial-time algorithm* Ext, *called extractor, such that the following holds: If the ideal adversary (for simplicity at the system start) makes a sequence of inputs and then sends a list* $\boldsymbol{n}$ *of* $2k$ *pairwise different nonces and a term* h *to an honest user such that* $h \equiv \mathsf{hash}(\boldsymbol{b} \odot \boldsymbol{n})$, *then the extractor, given the transcript of the ideal adversary's in- and outputs, outputs* $\boldsymbol{b}$. ◇

Again, the existence of such an extractor is clear for Dolev-Yao models with a normalization algorithm because the term $\mathsf{hash}(\boldsymbol{b} \odot \boldsymbol{n})$ cannot be further reduced and is thus the normal form of every equivalent term. Given the overall list of nonces $\boldsymbol{n}$, which the ideal adversary sent separately, the selection of nonces in this term and thus $\boldsymbol{b}$ can be read off.

4.4 Minimum Assumptions on a Cryptographic Realization

A general characteristics of real systems is that they are distributed. This means that each participant u has its own machine, here called M_u, and the machines are only connected by channels that offer well-defined possibilities for observations and manipulations by a real adversary. Specifically for the realization of Dolev-Yao models with hashes, we make the following (natural) minimum assumptions in the standard model of cryptography: Real channels are insecure; the input to send a term t leads to actual sending of a bitstring t^{r}; and hash terms are realized by applying a fixed (hash) function to the realization of the contained terms.

Definition 9 (Realization of a Dolev-Yao Model with Hashes). *In a* realization of a Dolev-Yao model with hashes in the standard model of cryptography*, an input* $\mathsf{send}(v, t^u)$ *to a machine* M_u *releases a bitstring* t^{r} *to the real adversary, such that within one execution of the system* $t \equiv t' \Rightarrow t^{\mathsf{r}} = t'^{\mathsf{r}}$ *for all terms* t, t'. *There must be a deterministic, polynomial-time function* $\mathsf{hash}^{\mathsf{r}}$ *such that* $(\mathsf{hash}(t))^{\mathsf{r}} = \mathsf{hash}^{\mathsf{r}}(t^{\mathsf{r}})$ *for all* $t \in \mathit{Hashable\_Terms}$. *An input* $\mathsf{is\_hash\_of}(t^u, h^u)$ *to a machine* M_u *leads to the output* true *iff* $\mathsf{hash}^{\mathsf{r}}(t^{\mathsf{r}}) = h^{\mathsf{r}}$.

For nonces, there must be a probabilistic polynomial-time algorithm $\mathsf{G_n}$ *that is used to generate* n^r *when it is needed for a new nonce* n, *and* $2k$ *executions of* $\mathsf{G_n}$ *must yield pairwise different results* $n_1^{\mathsf{r}}, \ldots, n_{2k}^{\mathsf{r}}$ *with overwhelming probability.* ◇

In realizations with type tagging we can consider an original cryptographic hash function together with the type tag as $\mathsf{hash}^{\mathsf{r}}$. Note that we made no assumptions on the cryptographic properties of $\mathsf{hash}^{\mathsf{r}}$ and only a weak one on $\mathsf{G_n}$; we will show that neither "good" nor "bad" realizations lead to soundness in the sense of BRSIM/UC. [6]

5 Details of the Impossibility Proofs

We now present the missing details for the impossibility proof sketches in Section 3, using the definitions from Section 4.

5.1 Unsoundness of Dolev-Yao Models with Payloads

The first scenario from Section 3.1 becomes the following lemma. Its proof is contained in the long version [10].

Lemma 1. *(Collision Resistance of the Real Hash Function) If a Dolev-Yao model with hashes and payloads (Definitions 1 to 3) has a realization in the standard model of cryptography (Definition 9) that is secure in the sense of BRSIM/UC, then the hash function* $\mathsf{hash}^{\mathsf{r}}$ *in this realization is collision-resistant. For simplicity we define here that a collision for security parameter k consists of two messages of length $2k$.* □

The second scenario from Section 3.1 together with this lemma gives us the following theorem.

Theorem 1. *(Unsoundness of Dolev-Yao Models with Hashes and Ideal Secrecy of New Payloads) No Dolev-Yao model with hashes and ideal secrecy of new payloads (Definitions 1 to 4) has a realization in the standard model of cryptography (Definition 9) that is secure in the sense of BRSIM/UC.* □

Proof. Assume that a Dolev-Yao model and a realization as specified in the theorem exist. By Lemma 1, the hash function $\mathsf{hash}^{\mathsf{r}}$ in the realization must be collision-resistant. Then $\delta(k) := \max_{h^{\mathsf{r}} \in \{0,1\}^*}(\mathsf{Pr}[\mathsf{hash}^{\mathsf{r}}(m^{\mathsf{r}}) = h^{\mathsf{r}} :: m^{\mathsf{r}} \overset{\mathcal{R}}{\leftarrow} \{0,1\}^{2k}])$ is negligible (as a function of k), because otherwise two random messages of length $2k$ are a collision with not negligible probability. We elaborate our second scenario in Figure 3: The user H_u chooses the payload as $m^{\mathsf{r}} \overset{\mathcal{R}}{\leftarrow} \{0,1\}^{2k}$. By Definitions 2 and 3, the adversary and the user can indeed act as described in Section 3.1, and by Definition 9, the output for A in the real system is $\mathsf{hash}^{\mathsf{r}}(m^{\mathsf{r}})$. In the ideal system, the simulator Sim gets an output $\mathsf{receive}(u, v, h^{\mathsf{a}})$ from the Dolev-Yao model TH and has to produce a string h^{r} for the adversary. For indistinguishability, this string must fulfill $h^{\mathsf{r}} = \mathsf{hash}^{\mathsf{r}}(m^{\mathsf{r}})$ with overwhelming probability.

Definition 4 is applicable and implies that Sim (which acts as the ideal adversary here), with x calls to TH, cannot obtain more information about m^{r} than by learning

[6] In computational considerations about $\mathsf{hash}^{\mathsf{r}}$ we allow $\mathsf{hash}^{\mathsf{r}}$ to depend on the security parameter k, which is fixed in each system execution. To allow collision resistance in the sense of the typical cryptographic definition, it should even depend on a key pk chosen at the beginning of each system execution; our proofs could easily be adapted to this case.

for x bitstrings m'^{r} whether $m'^{\mathsf{r}} = m^{\mathsf{r}}$. As m^{r} is uniformly random, the probability that Sim hits $m'^{\mathsf{r}} = m^{\mathsf{r}}$ in this process is $x/2^{2k}$, where x is polynomial because Sim is polynomial-time. Thus this probability is negligible. In the other case, the optimal choice of h^{r} for Sim is the most likely hash value over the remaining $2^{2k} - x$ possible payloads. The probability that this value is correct is at most $\delta(k)2^{2k}/(2^{2k} - x)$. This is negligible because x is polynomial. ■

Next we consider the case without secrecy of hashed terms, but with the additional assumption that the output length of the real hash function depends only on the security parameter, not on the input length. (Weaker definitions of significantly shortening hash functions would also suffice.) The third scenario from Section 3.1 together with Lemma 1 gives us the following theorem.

Theorem 2. *(Unsoundness of Dolev-Yao Models with Hashes and Payloads without Secrecy) No Dolev-Yao model with hashes and payloads, even without ideal secrecy, but with minimum non-constructibility of unknown payload hashes, (Definitions 1 to 3 and 5) has a realization in the standard model of cryptography (Definition 9) that is secure in the sense of BRSIM/UC and where the real hash function is shortening. For simplicity, we require that the range of a shortening hash function is* $\{0,1\}^k$. □

Proof. Assume that a Dolev-Yao model and a realization as specified in the theorem exist. By Lemma 1, the hash function $\mathsf{hash}^{\mathsf{r}}$ in the realization must be collision-resistant. As $\mathsf{hash}^{\mathsf{r}}$ is also shortening, it is one-way. (Otherwise the following algorithm finds a collision with not negligible probability: Select a random payload $m^{\mathsf{r}} \xleftarrow{\mathcal{R}} \{0,1\}^{2k}$, use the assumed inversion algorithm $\mathsf{A}_{\mathsf{owf}}$ to find a preimage m'^{r} of $\mathsf{hash}^{\mathsf{r}}(m^{\mathsf{r}})$, and output m^{r} and m'^{r} if they are unequal. This holds because all payloads, except less than 2^k and thus negligibly many, collide with another one. If $\mathsf{A}_{\mathsf{owf}}$ succeeds for such a payload m^{r}, then with probability at least $1/2$ we have $m'^{\mathsf{r}} \neq m^{\mathsf{r}}$.)

We now elaborate our third scenario in Figure 4. By Definitions 2 and 3, the adversary and the user can indeed act as described in Section 3.1, and by Definition 9 the output for H_u in the real system is indeed true. By the assumption of the proof, the simulator can achieve the same result in the ideal system with overwhelming probability. Hence it makes an input $\mathsf{send}(v, u, h^{\mathsf{a}})$ where, by Definition 2, $h \equiv \mathsf{hash}(m)$ for the term m that is realized as m^{r}. Definition 5 is applicable to our scenario and essentially states that Sim, which acts as the ideal adversary, must know m^{r} for this. More precisely, we use the postulated extractor Ext to extract m^{r} from the transcript of Sim whenever Sim is successful. This gives us an inversion algorithm $\mathsf{A}_{\mathsf{owf}}$ for the function $\mathsf{hash}^{\mathsf{r}}$ that succeeds with not negligible probability, in contradiction to the one-wayness of $\mathsf{hash}^{\mathsf{r}}$. Concretely, $\mathsf{A}_{\mathsf{owf}}$ is the combination of TH, Sim and Ext. This is indeed a non-interactive algorithm as required in the definition of one-wayness: Sim initially gets one input h^{r} and TH has no input so far. Then Sim and TH interact with each other, but interaction with H or A would be distinguishable from the real system. ■

5.2 Unsoundness Without Payloads

Now we work out the scenarios for restricted Dolev-Yao models without payloads. As sketched in Section 3.2, we proceed similar to the scenarios with payloads, letting the users and the adversary replace payloads by bit vectors that select sublists of nonces. For this, we first define collision resistance and one-wayness with respect to the bit vectors.

Definition 10 (Bit-vector Collision Resistance and One-Wayness). *Let a Dolev-Yao model with hashes and a realization in the standard model of cryptography with the hash function* hash^r *be given (Definitions 1, 2, and 9). We say that* hash^r *is* bit-vector collision-resistant *if every polynomial-time adversary can only find a list* $\boldsymbol{n}^\mathsf{r} = (n^\mathsf{r}_1, \ldots, n^\mathsf{r}_{2k})$ *of* $2k$ *pairwise different real nonces and bit vectors* $\boldsymbol{b} \neq \boldsymbol{b}^* \in \{0,1\}^{2k}$ *with* $\mathsf{hash}^\mathsf{r}(\boldsymbol{b} \odot \boldsymbol{n}^\mathsf{r}) = \mathsf{hash}^\mathsf{r}(\boldsymbol{b}^* \odot \boldsymbol{n}^\mathsf{r})$ *with negligible probability, where* $\boldsymbol{b} \odot \boldsymbol{n}^\mathsf{r}$ *for a bit vector* $\boldsymbol{b} = b_1, \ldots, b_{2k}$ *denotes the sublist consisting of the nonces* n^r_i *with* $b_i = 1$.

We say that hash^r *is* bit-vector one-way *if every polynomial-time algorithm* A_owf*, on input* $h^\mathsf{r} := \mathsf{hash}^\mathsf{r}(\boldsymbol{b} \odot \boldsymbol{n}^\mathsf{r})$ *for random* $\boldsymbol{b} \xleftarrow{\mathcal{R}} \{0,1\}^{2k}$ *and real nonces generated with* G_n*, can only output a bit vector* $\boldsymbol{b}^* \in \{0,1\}^{2k}$ *with* $h^\mathsf{r} = \mathsf{hash}^\mathsf{r}(\boldsymbol{b}^* \odot \boldsymbol{n}^\mathsf{r})$ *with negligible probability.*

Lemma 2. *(Bit-vector Collision Resistance of the Real Hash Function) If a Dolev-Yao model with hashes and nonces (Definitions 1, 2, and 6) where* $Hashable\_Terms$ *contains at least all lists of up to* $2k$ *nonces has a realization in the standard model of cryptography (Definition 9) that is secure in the sense of BRSIM/UC, then the hash function* hash^r *in this realization is bit-vector collision-resistant.* □

The proofs of this lemma and the two following theorems are postponed to the long version [10].

Theorem 3. *(Unsoundness of Dolev-Yao Models with Hashes and Ideal Secrecy of New Nonce Lists) No Dolev-Yao model with hashes and ideal secrecy of new nonce lists (Definitions 1, 2, 6, and 7) has a realization in the standard model of cryptography (Definition 9) that is secure in the sense of BRSIM/UC.* □

Theorem 4. *(Unsoundness of Dolev-Yao Models with Hashes and Nonce Lists without Secrecy) Let a Dolev-Yao model with hashes be given whose set* $Hashable\_Terms$ *contains at least all lists of up to* $2k$ *nonces, and where minimum non-constructibility of unknown nonce-list hashes holds (Definitions 1, 2, 6, and 8). Then no cryptographic implementation in the standard model of cryptography (Definition 9) with a shortening real hash function is sound in the sense of BRSIM/UC.* □

6 Soundness Results

In this section we show that Dolev-Yao-style hashes can be proven sound in the random oracle model, and under specific restrictions on the usage of hash functions or their properties in the ideal system even in the standard model of cryptography.

6.1 Soundness of Dolev-Yao Models with Hashes in the Random Oracle Setting

The first soundness result states that normal Dolev-Yao models without specific restrictions can be proven sound in the random oracle model. As an overall result for an operator-rich Dolev-Yao model with hashes, this requires an underlying Dolev-Yao model with the other usual cryptographic operators and a realization secure in the sense of BRSIM/UC. Hence we have to use that of [8]. However, what happens specifically with the hashes can be explained well without specific notation from [8]. We sketch this in this section, leaving more details to the long version of this paper [10].

The Dolev-Yao functionality is that of a free hash operator with unrestricted hashing and with ideal secrecy in the typical sense that the adversary, upon learning a hash value, has no deconstruction operator or other ways to obtain information about the contained term except by the input is_hash_of for comparing a message and a potential hash. The only additional power that the ideal adversary gets compared with honest users is to make hashes with unknown preimages, i.e., terms that could be written $\mathsf{hash}(?)$. The preimages will remain unknown forever; that this works in the realization is a consequence of the random oracle model.

In the cryptographic realization, the operator hash is essentially realized by the random oracle. The only addition is that the bitstrings are typed, i.e., the realization $\mathsf{hash}(t)^{\mathsf{r}}$ of a hash term is the pair $(\text{`}\mathsf{hash}\text{'}, \mathsf{RO}(t^{\mathsf{r}}))$ where 'hash' is a fixed string and RO abbreviates the result of a (stateful) random oracle call.

Our security claim is that this realization is as secure as this ideal Dolev-Yao-style system in the sense of BRSIM/UC in the random oracle model; see Section 2. The proof of the theorem can be found in the long version [10].

Theorem 5. *(Soundness of a Dolev-Yao Model with Hashes in the Random Oracle Model) A Dolev-Yao model with unrestricted hashing and secrecy of hashed terms can be securely implemented by a canonical cryptographic realization in the sense of BRSIM/UC in the random oracle model. Both the Dolev-Yao model and cryptographic realization are defined in detail in the long version [10].* □

6.2 Soundness Results in the Standard Model

Finally, we briefly present two restricted but still practically useful types of Dolev-Yao models with hashes that have secure realizations even in the standard model of cryptography. Both models allow the ideal adversary to construct hash terms with unknown preimages, i.e., terms $\mathsf{hash}(?)$. In contrast to the model in Section 6.1, the adversary can later provide a preimage for such a term. Both realizations require a collision-resistant hash function; in the first case the hash function must also be one-way.

The first type of Dolev-Yao model gives up the ideal secrecy, and can then work with a significant class of hashable terms. By the size of a term t we mean the number of nodes in the tree representation of t.

Theorem 6. *(Soundness Without Payloads or Secrecy for Constant-Sized Terms) A Dolev-Yao model without secrecy of hashed terms where terms in $Hashable\_Terms$ do not contain payloads and are at most of a constant size l can be securely realized in the sense of BRSIM/UC with arbitrary collision-resistant, one-way hash functions in the standard model of cryptography.* □

We believe that this theorem can be extended to terms that contain payloads, but only together with fresh nonces that remain secret, but the overhead of such a condition that must be defined over an overall system execution does not seem justified because the case is of limited usefulness: Such a nonce cannot be sent over an insecure channel, and thus unless a secret channel is available no other party can do anything with such a hash term, such as test if for correctness.

The second theorem offers the ideal secrecy of typical Dolev-Yao models of hashing, but only individual nonces can be hashed, as for instance in one-time signatures. Recall from the introduction that while this may seem surprising given that real hash functions (even probabilistic ones) do not offer perfect secrecy, it is correct because nonces are system-internal objects whose Dolev-Yao abstraction essentially only requires freshness and unguessability of the nonces as a whole.

Theorem 7. *(Soundness With Secrecy for Nonce Hashing) A Dolev-Yao model with secrecy of hashed terms and where the set $Hashable\_Terms$ contains only individual nonces can be securely realized in the sense of BRSIM/UC with arbitrary collision-resistant hash functions in the standard model of cryptography.* □

Similar to the random oracle case, for the precise model we rely on the existing Dolev-Yao model of [8] and extend it with hashes. The detailed models and sketches of both proofs are given in the long version [10].

7 Conclusion

We have investigated whether Dolev-Yao models with hashes or one-way functions can be realized in the sense of BRSIM/UC, i.e., such that the Dolev-Yao model is regarded as an ideal functionality that is securely implemented by its realization. We have shown that this is not possible for the standard type of such Dolev-Yao models where hashing is a free operator. This impossibility result holds for all polynomial-time computable functions in the role of the real hash function.

We then considered restrictions and extensions of the Dolev-Yao model or its ideal properties that have a potential to simplify simulations. For these, we obtained additional BRSIM/UC impossibility results: First, modeling probabilistic hashing makes no difference. Secondly, it does not help if no payloads can be hashed, but only cryptographic terms (in fact, lists of nonces are sufficient). Thirdly, even if we give up the ideal secrecy property of hashes (retaining the ideal collision freeness so that the model is still reasonable), we obtain BRSIM/UC impossibility for all realizations of the hash operator by polynomial-time computable functions whose output length is independent of the input length, and thus for all typical real hash functions. This is the first impossibility proof for a Dolev-Yao model that does not assume any ideal secrecy property.

On the positive side, we showed that a BRSIM/UC-sound realization of standard Dolev-Yao hashes is possible in the random oracle model. we also obtain BRSIM/UC soundness in the standard model of cryptography for two cases: One includes ideal secrecy, but only allows hashing of single nonces, e.g., for the use in one-time signatures. The other gives up ideal secrecy, but allows hashing of arbitrary cryptographic terms, i.e., terms without payloads, up to an arbitrary but constant size.

Acknowledgments. We thank Martín Abadi, Anupam Datta, Ante Derek, Cathy Meadows, John Mitchell, and Andre Scedrov for interesting discussions.

References

1. M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th TACS*, pages 82–94, 2001.
2. M. Abadi and P. Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP TCS*, volume 1872 of *LNCS*, pages 3–22. Springer, 2000.
3. M. Backes. A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol. In *Proc. 9th ESORICS*, volume 3193 of *LNCS*, pages 89–108. Springer, 2004.
4. M. Backes and M. Dürmuth. A cryptographically sound Dolev-Yao style security proof of an electronic payment system. In *Proc. 18th IEEE CSFW*, pages 78–93, 2005.
5. M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. *Journal on Selected Areas in Communications*, 22(10):2075–2086, 2004.
6. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE CSFW*, pages 204–218, 2004.
7. M. Backes and B. Pfitzmann. Limits of the cryptographic realization of Dolev-Yao-style XOR. In *Proc. 10th ESORICS*, volume 3679 of *LNCS*, pages 178–196. Springer, 2005.
8. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proc. 10th ACM CCS*, pages 220–230, 2003.
9. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. 8th ESORICS*, volume 2808 of *LNCS*, pages 271–290. Springer, 2003.
10. M. Backes, B. Pfitzmann, and M. Waidner. Limits of the Reactive Simulatability/UC of Dolev-Yao models with hashes. IACR Cryptology ePrint Archive 2006/068, Feb. 2006.
11. D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *Intern. Journal of Information Security*, 2004.
12. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proc. 1st ACM CCS*, pages 62–73, 1993.
13. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE CSFW*, pages 82–96, 2001.
14. B. Blanchet. A computationally sound mechanized prover for security protocols. In *Proc. 27th IEEE Symp. on Security & Privacy*, pages 140–154, 2006.
15. R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Proc. CRYPTO 97*, pages 455–469. Springer, 1997.
16. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE FOCS*, pages 136–145, 2001.
17. R. Canetti and M. Fischlin. Universally composable commitments. In *Proc. CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, 2001.
18. R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *Proc. 3rd Theory of Cryptography Conf. (TCC)*, pages 380–403. Springer, 2006.
19. R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In *Proc. EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 337–351. Springer, 2002.
20. R. Canetti, D. Micciancio, and O. Reingold. Perfectly one-way probabilistic hash functions. In *Proc. 30th ACM STOC*, pages 131–140, 1998.
21. A. Datta, A. Derek, J. Mitchell, A. Ramanathan, and A. Scedrov. Games and the impossibility of realizable ideal functionality. In *Proc. 3rd Theory of Cryptography Conf. (TCC)*. Springer, 2006. To appear.

22. A. Datta, A. Derek, J. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proc. 32nd Intern. Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *LNCS*, pages 16–29. Springer, 2005.
23. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
24. F. D. Garcia and P. van Rossum. Sound computational interpretation of formal hashes. IACR Cryptology ePrint Archive 2006/014, Jan. 2006.
25. D. Hofheinz and J. Müller-Quade. Universally composable commitments using random oracles. In *Proc. 1st Theory of Cryptography Conf. (TCC)*, volume 2951 of *LNCS*, pages 58–76. Springer, 2004.
26. R. Impagliazzo and B. M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. 44th IEEE FOCS*, pages 372–381, 2003.
27. P. Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symp. on Programming (ESOP)*, pages 77–91, 2001.
28. P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symp. on Security & Privacy*, pages 71–85, 2004.
29. P. Laud. Secrecy types for a simulatable cryptographic library. In *Proc. 12th ACM CCS*, pages 26–35, 2005.
30. M. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, 1983.
31. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conf. (TCC)*, volume 2951 of *LNCS*, pages 133–151. Springer, 2004.
32. J. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39th FOCS*, pages 725–733, 1998.
33. J. Mitchell, M. Mitchell, A. Scedrov, and V. Teague. A probabilistic polynominal-time process calculus for analysis of cryptographic protocols. *ENTCS*, 47:1–31, 2001.
34. L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.
35. B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM CCS*, pages 245–254, 2000.
36. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symp. on S & P*, pages 184–200, 2001.
37. C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, and M. Waidner. Cryptographically sound theorem proving. In *Proc. 19th IEEE CSFW*, 2006. To appear.
38. A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd FOCS*, pages 80–91, 1982.

Conditional Reactive Simulatability

Michael Backes[1], Markus Dürmuth[1], Dennis Hofheinz[2], and Ralf Küsters[3]

[1] Saarland University
{backes, duermuth}@cs.uni-sb.de
[2] CWI, Cryptology and Information Security Group, Prof. Dr. R. Cramer
Dennis.Hofheinz@cwi.nl
[3] Christian-Albrechts-Universität zu Kiel
kuesters@ti.informatik.uni-kiel.de

Abstract. Simulatability has established itself as a salient notion for defining and proving the security of cryptographic protocols since it entails strong security and compositionality guarantees, which are achieved by universally quantifying over all environmental behaviors of the analyzed protocol. As a consequence, however, protocols that are secure except for certain environmental behaviors are not simulatable, even if these behaviors are efficiently identifiable and thus can be prevented by the surrounding protocol.

We propose a relaxation of simulatability by conditioning the permitted environmental behaviors, i.e., simulation is only required for environmental behaviors that fulfill explicitly stated constraints. This yields a more fine-grained security definition that is achievable for several protocols for which unconditional simulatability is too strict a notion, or at lower cost for the underlying cryptographic primitives. Although imposing restrictions on the environment destroys unconditional composability in general, we show that the composition of a large class of conditionally simulatable protocols yields protocols that are again simulatable under suitable conditions. This even holds for the case of cyclic assume-guarantee conditions where protocols only guarantee suitable behavior if they themselves are offered certain guarantees. Furthermore, composing several commonly investigated protocol classes with conditionally simulatable subprotocols yields protocols that are again simulatable in the standard, unconditional sense.

1 Introduction

As a tool to define and prove the security of cryptographic protocols, the concept of simulatability has a long history, e.g., [34,23,22,11,30]. In recent years, in particular the general simulatability frameworks of reactive simulatability [8,6] and universal composability [14,15] proved useful for analyzing security properties of cryptographic protocols in distributed systems. In such a simulatability framework, a protocol is compared to an ideal specification of the respective protocol task, usually given by a single machine called trusted host that is immune to any adversarial attacks by construction. A protocol is said to be secure, if for every adversary interacting with the real protocol there exists another adversary interacting with the ideal specification such that no protocol environment can distinguish the ideal specification (with the ideal adversary) from the real implementation (with the real adversary). This essentially means that every attack that an adversary may successfully mount against the real implementation can be mounted as well against the ideal specification. In that sense, the real protocol is at least as secure as the ideal specification.

This definition is very appealing due to its simplicity, and at the same time it provides very strong security guarantees. Specifically, both mentioned frameworks allow for very general composition theorems (see, e.g., [32,14,7]). In a nutshell, these theorems guarantee that a secure protocol can be composed with arbitrary other protocols and still retains its security. These strong results are essentially entailed by the universal quantification over all protocol environments. However, such strong compositionality properties are bought at the price that several protocol tasks are not securely realizable at all in the sense of simulatability. This includes important cryptographic tasks such as bit commitment, zero-knowledge, oblivious transfer [16,14], and (authenticated) Byzantine agreement [29], classes of secure multi-party computation [17], classes of functionalities that fulfill certain game-based definitions [18], and Dolev-Yao style abstractions of symmetric encryption, XOR, and hash functions [3,4,9].

This nuisance led to several attempts to weaken the simulatability definition, either by strengthening the ideal adversary or by limiting the attack capabilities of the real adversary, which, however, results in restricted adversary models and thus in less realistic scenarios. A more detailed review of related work is given below.

Our Contribution. In this paper, we also endeavor to circumvent a specific class of the aforementioned impossibility results, namely those that arise due to certain environmental behaviors that cannot be properly simulated. The prime example contained in this class is Dolev-Yao style symmetric encryption, i.e., symbolic abstractions of symmetric encryption as constructors of a term algebra with a small set of algebraic properties. This kind of encryption can only be correctly simulated if the protocol using the encryption scheme does not cause a so-called commitment problem. Our approach for circumventing impossibility in these cases does however not follow the prevalent idea of augmenting or constraining the capabilities of the adversary. Instead, we limit the number of protocol environments in which a protocol is required to be secure. This idea applies particularly nicely to protocols that can be securely realized except for certain distinguished environmental behaviors, especially if these behaviors are efficiently identifiable and thus can be prevented by the surrounding protocol; among others, Dolev-Yao style symmetric encryption is of this kind. The resulting security notion is named *conditional reactive simulatability*. In addition to circumvent known impossibility results for unconditional simulatability, the notion of conditional reactive simulatability may also allow for securely realizing ideal functionalities at lower cost on the underlying cryptographic primitives. For instance, if Dolev-Yao style symmetric encryption permits the construction of key cycles, e.g., encrypting a key with itself, it is only securely realizable by encryption schemes that fulfill certain strong, non-standard assumptions such as dynamic KDM security [5]. When, however, conditioning the functionality to those cases that exclude key cycles, successful simulation based on weaker, more standard security notions such as IND-CCA2 security is possible.

Despite imposing restrictions on the surrounding protocol and thus giving up the universal quantification of environments that allows for general compositionality, we show that the notion of conditional reactive simulatability still entails strong compositionality guarantees. More specifically, we prove that if one composes protocols each of which is conditionally simulatable provided that their surrounding protocol fulfills an arbitrary trace property, and if these properties do not give rise to cyclic dependencies, then the composition of these protocols is conditionally simulatable under natural conditions on the (overall) surrounding protocol. Technically, the theorem establishes a cryptographic statement on the acyclic composition of assume-guarantee specifications, i.e., specifications that guarantee suitable behaviors only if they themselves are offered suitable guarantees. Assume-guarantee specifications have been well investigated in the past, mostly for non-security-specific contexts [31,26,1,20] but also specifically for security aspects [24] (but without investigations of simulatability and composition). The postulation of acyclicity applies to most cases in practice, e.g., to protocols that provide specific security guarantees to their subprotocols without making these guarantees dependent on the outputs they obtain from these subprotocols.

Interestingly, we can even prove compositionality for cyclic dependencies of such specifications, i.e., compositions of protocols that mutually promise to adhere to a certain behavior only if they mutually receive guarantees from each other. This case is technically more demanding since an inductive proof by proceeding through the acyclic dependency graph as done in the proof of the acyclic case is no longer possible. In fact, it is easy to show that for cyclic dependencies, subprotocols that are conditionally simulatable under *arbitrary* trace properties might not be securely composable. However, we prove that the theorem for the acyclic case can be carried over to the cyclic case if the constraints imposed on protocols for conditional simulatability are safety properties. Safety properties arguably constitute the most important class of properties for which conditional simulatability is used, especially since liveness properties usually cannot be achieved unless one additionally constraints the adversary to fair scheduling.

Finally, we note that composing protocol classes with conditionally simulatable subprotocols can yield protocols that are simulatable in the standard, unconditional sense.

Our results are formalized in the Reactive Simulatability framework. However, we do not use any specific characteristics of this framework, so our results can naturally be carried over to the Universal Composability framework.

Related Work. There have been several attempts to relax simulatability to avoid impossibility results. The work closest to ours is the work on proving Dolev-Yao style symmetric encryption sound in the sense of simulatability [3]. There it was shown that Dolev-Yao style symmetric encryption can be securely realized if the environmental protocol does not cause the commitment problem and in addition key cycles are excluded. This definition thus constitutes a special case of conditional reactive simulatability yet without investigating more general conditions or corresponding compositionality aspects. Nevertheless, our work is inspired by their idea of augmenting simulatability with conditions on environments.

The impossibility of simulating a specific bit commitment was shown in [16]. The remedy proposed there was to augment the real protocol with certain "helping trusted hosts" which are, by definition, immune to any attack on the real protocol; thus, effectively this weakens the real adversary. More specifically, [16] presented simulatably secure protocols for bit commitment and zero-knowledge. However, these protocols rely on a so-called Common Reference String (CRS), which is a form of a trusted setup assumption on the protocol participants. In a similar vein, [17] shows that basically every trusted host can be realized using a CRS as a helper functionality. One point of criticism against the CRS approach is that the proposed protocols lose security in a formal and also very intuitive sense as soon as the CRS setup assumption is invalidated. The related approach [25] uses a Random Oracle (RO) instead of a CRS to help real protocols achieve simulatable security. The benefit of their construction is that the proposed protocols retain at least classical (i.e., non-simulatable) security properties when the RO assumption is invalidated. However, also there, simulatability in the original sense is lost as soon as this happens.

In [33], the real and ideal adversaries are equipped with a so-called imaginary angel. This is an oracle that (selectively) solves a certain class of hard computational problems for the adversary. Under a very strong computational assumption, this notion could be shown to avoid known impossibility results for simulatability. Yet, as the imaginary angels behave in a very specific way tailored towards precisely circumventing these impossibility results, e.g., these angels make their response dependent on the set of corrupted parties, the model might be considered unintuitive.

In [10], it is shown how to realize any trusted host in a simulatable manner, if the ideal adversary is freed from some of its computational restrictions. However, it is substantial that in their security notion, the ideal adversary is not restricted to polynomial-time, but the real adversary is. So in particular, the security notion they consider is not transitive and it is generally not easy in their framework to construct larger protocols modularly.

Outline. We first review the underlying Reactive Simulatability framework in Section 2 and subsequently define the more fine-grained version of conditional reactive simulatability in Section 3. The bulk of the paper is dedicated to the investigation of the compositionality aspects of this new security notion for both acyclic and cyclic assume-guarantee conditions, which is done in Section 4. The usefulness of conditional reactive simulatability is further exemplified in Section 5 by showing how this notion can be exploited to cryptographically justify common idealizations of cryptography. Section 6 concludes. More details and proofs can be found in our technical report [2].

2 Review of the Reactive Simulatability Framework

Our work builds upon the Reactive Simulatability framework. We will briefly review relevant definitions and refer the reader to [8] for details.

2.1 Overall Framework

A protocol is modeled as a *structure* (M, S) consisting of a set of protocol *machines* and a set of *service ports*, to which the *protocol user* connects[1]. Machines are probabilistic, polynomial-time I/O automata, and are connected by *ports*. The model differentiates in-ports and out-ports, where each out-port is connected to exactly one in-port by naming convention. Moreover, in- and out-ports may be service or non-service ports. In what

[1] Actually, a structure represents a protocol in a specific corruption situation. To handle different corruption situations, *systems* (i.e., sets of structures) are used. However, in the style of [8,19], we concentrate on a given specific corruption situation for ease of presentation.

follows, by S^{in} we denote the service in-ports of S and by $S^{C,out}$ the complement of M's service out-ports, i.e., the set of service in-ports of machines M connects to.

Two structures (M_1, S_1) and (M_2, S_2) are *composable* iff they connect through their respective service ports only. Their *composition* is given by $(M_1 \cup M_2, S)$ where S includes all ports from S_1 and S_2 that are not connected to another machine in $M_1 \cup M_2$.

A set of machines M is *closed* iff all ports are connected to corresponding ports of machines that are in the same set. A structure can be complemented to a closed set by a so-called *honest user* H and an *adversary* A, where H connects to service ports only, and A connects to all remaining open ports, and both machines may interact. The tuple $(M, S, \mathsf{H}, \mathsf{A})$ is then called a *configuration* of (M, S) where one of the machines H or A plays the role of the *master scheduler*, i.e., if no machine was activated by receiving a message, the master schedule is activated. A closed set C is a *runnable system*. The transcript of a single run is called a *trace* (often denoted by $\mathbf{t}$ and decorations thereof) and is defined to be a sequence of transitions performed by the machines. A *transition* of a machine M is of the form $(\overline{p}, s, s', \overline{p}')$ where $\overline{p}$ describes the in-ports of M along with the current message written on these ports, s is the current configuration of M, s' is a successor configuration (computed depending on $\overline{p}$ and s), and $\overline{p}'$ are the out-ports along with the output produced. We denote by $run_{C,k}$ the distribution of traces induced by runs of C with security parameter k. The restriction $\mathbf{t}\lceil_S$ of a trace $\mathbf{t}$ to a set of in-ports S is defined in the obvious way. (Note that $\mathbf{t}\lceil_S$ only depends on the first component ($\overline{p}$) of the transitions of $\mathbf{t}$). Now, $run_{C,k}\lceil_S$ denotes the distribution of the traces induced by runs of C with security parameter k when restricted to S. The *restriction of a trace* $\mathbf{t}$ *to a machine* M is obtained from $\mathbf{t}$ by removing all transitions not done by M. Now, the distribution of such traces given k is denoted by $view_{C,k}(\mathsf{M})$. We refer to the k-indexed family $\{view_{C,k}(\mathsf{M})\}_k$ of these views by $view_C(\mathsf{M})$.

2.2 Simulatability

Simulatability is used in different areas of cryptography. Informally speaking, for reactive systems it says that whatever might happen to a protocol (M, S) can also happen to another protocol (M', S). Here both protocols need to have the same set of service ports S to allow for a meaningful comparison. Typically, (M', S) is an idealization, or specification, of the protocol task that (M, S) is to implement. We therefore call (M, S) the *real* and (M', S) the *ideal protocol*. (Typically, the ideal protocol consists only of a single machine TH, a trusted host, that guarantees an ideal behaviour to a user of the protocol.) For simulatability one requires that for every configuration $(M, S, \mathsf{H}, \mathsf{A})$, with honest user H and real adversary A, there is a configuration $(M', S, \mathsf{H}, \mathsf{A}')$ of (M', S), with the same honest user H and a (possibly different) ideal adversary A', such that H cannot distinguish both scenarios. This is illustrated in Figure 1.

The notion that H cannot distinguish both scenarios is captured by the notion of computational indistinguishability: Two families $(\mathsf{var}_k)_{k\in\mathbb{N}}$, $(\mathsf{var}'_k)_{k\in\mathbb{N}}$ of random variables on common domains D_k are *computationally indistinguishable* ("$\approx$") if no polynomial-time algorithm can distinguish both distributions with non-negligible probability, i.e., if for all polynomial-time algorithms Dis the following holds:

$$\left|\Pr\left[\mathsf{Dis}(1^k, \mathsf{var}_k) = 1\right] - \Pr\left[\mathsf{Dis}(1^k, \mathsf{var}_k) = 1\right]\right| \text{ is negligible in } k,$$

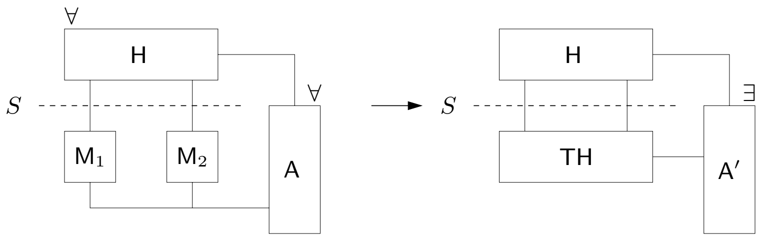


Fig. 1. Simulatability: The two views of H must be indistinguishable

where a function $g : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ is said to be *negligible* iff for all positive polynomials $Q, \exists k_0 \forall k \geq k_0 : g(k) \leq 1/Q(k)$.

Definition 1 (Reactive Simulatability). *Let structures (M, S) and (M', S) with identical sets of service ports be given. We write $(M, S) \geq_{\mathsf{sec}}^{\mathsf{poly}} (M', S)$, where $\geq_{\mathsf{sec}}^{\mathsf{poly}}$ is read as* computationally at least as secure as *or* securely realizes*, if for every configuration $conf = (M, S, \mathsf{H}, \mathsf{A})$, there exists a configuration $conf' = (M', S, \mathsf{H}, \mathsf{A}')$ (with the same* H*) such that*

$$view_{conf}(\mathsf{H}) \approx view_{conf'}(\mathsf{H}).$$

One also defines *universal simulatability*, where A' in $conf'$ does not depend on H, i.e., the order of quantifiers is reversed, and *blackbox simulatability*, where A' is the composition of a fixed part Sim (the *simulator*) and A. In the sequel, we omit the superscript poly.

3 Conditional Reactive Simulatability

Reactive simulatability (Definition 1) permits configurations with arbitrary honest users H (satisfying some syntactic requirements on ports). In other words, reactive simulatability requires a faithful simulation of the combination of the real adversary and real protocol by the ideal adversary and ideal protocol for *every* honest user. This universal quantification over all honest users allows for a general composition theorem [32,7], which says that if protocol (M, S) is as secure as protocol (M', S), then (M, S) can be substituted for (M', S) in *any* larger protocol without invalidating simulatability. For this type of compositional property, simulatability can even be shown to be necessary [28].

However, reactive simulatability may be too strict in certain practical scenarios: The simulation might fail for certain honest users, but in the application under consideration such users may not occur since the protocol in question may always be used in a certain (secure) way. For example, consider Dolev-Yao style symmetric encryption. It was shown in [3] that this kind of encryption is not securely realizable in the sense of reactive simulatability, due to the so-called commitment problem: If an encrypted message is sent to the adversary, where the adversary neither knows the message nor the key, the best the simulator can do is to create a new key and encrypt a random message with this key. If later the message becomes known, indistinguishability guarantees that the simulation is still correct. However, if later the key becomes known, the simulator has to come up with a suitable key that decrypts the chosen ciphertext to the correct message. This is not possible in general. However, in the application under consideration the way Dolev-Yao style symmetric encryption is used, e.g., by a larger protocol (representing the honest user), may guarantee that the encryption key is never exposed. It turns out that in this situation faithful simulation is still possible.

Following this idea, we propose a relaxation of reactive simulatability, called conditional reactive simulatability, where instead of quantifying over all honest users, we quantify only over those honest users which satisfy a certain condition. In this way troublesome honest users which would not occur in the application anyway can be ruled out.

The conditions on honest users are expressed in terms of what we call predicates. A predicate, which is defined with respect to a set S of ports (typically service in-ports), is a set of sequences of bit strings for every port of S. Using predicates, we can restrict the kind and the order of messages on ports of S in a run of a system. To formally define these predicates, we need the following notation: For sets A and B, we denote by B^A the set of mappings from A to B. If A is a finite set, then the elements of B^A can be considered to be tuples where every component is an element of B and corresponds to an element of A. For $i \geq 0$ and a set A, we denote by A^i the set of all words over A of length i. Now, predicates are defined as follows:

Definition 2 (Predicates). *Let S be a set of ports. We call a set π with*

$$\pi \subseteq \bigcup_{i \geq 0} ((\{0,1\}^*)^S)^i.$$

a predicate π over S *if the following conditions are satisfied:*

1. *If $\pi = s_1 \cdots s_i$, $s_j \in (\{0,1\}^*)^S$, then we have that for every $j \in \{1,\ldots,i\}$ there exists $p \in S$ such that $s_j(p) \neq \varepsilon$, i.e., for every s_j at least one port contains a non-empty message.*
2. *π is decidable in polynomial-time, i.e., there is a probabilistic polynomial-time algorithm that, on input t, outputs whether or not $t \in \pi$.*

We call $t \in \pi$ an S-trace.

Instead of a single predicate, one could also consider a family of predicates indexed by the security parameter. However, for the application presented in this paper, simple predicates suffice. Also, all results presented in this paper easily carry over to the case of families of predicates.

We will use the following notation. We write $\pi = \texttt{true}$ for a predicate π over S with $\pi = \bigcup_{i \geq 0}((\{0,1\}^*)^S)^i$. Furthermore, for two predicates π_1 and π_2 over two disjoint port sets S_1 and S_2, we write $\pi_1 \wedge \pi_2$ for the predicate containing all $(S_1 \cup S_2)$-traces such that for every trace in $\pi_1 \wedge \pi_2$ its restriction to S_1 and S_2 belongs to π_1 and π_2, respectively.[2] Intuitively, $\pi_1 \wedge \pi_2$ represents the conjunction of π_1 and π_2.

An S-trace t' is a *prefix* of an S-trace t if there exist t'' such that $t = t' \cdot t''$ where '$\cdot$' denotes concatenation. A predicate π over S is *prefix-closed* iff for every S-trace $t \in \pi$

[2] In run restricted to some port set S, all entries with inputs only in non-S ports are deleted.

every prefix of t belongs to π as well. We also call such a predicate a *safety property* since once it is violates it stays violated.

Now, we say that a set of machines M fulfills a predicate π over a set of ports S, if in runs of M with any other set of machines the sequences of messages written on ports in S belong to π. More precisely, it suffices if this is true with overwhelming probability:

Definition 3 (Predicate Fulfillment). *Let M be a set of machines with service ports S and let π be a predicate over a subset S' of the ports $S^{C,out}$ of machines to which machines in M connect. Then, M* fulfills *π if for any set of machines $\overline{M}$ such that $C := \{M, \overline{M}\}$ is closed,*

$$\mathsf{Pr}_{t \leftarrow run_{C,k}} [(t\lceil_{S'}) \in \pi] \text{ is overwhelming as a function in } k.$$

We are now ready to present the definition of conditional reactive simulatability.

Definition 4 (Conditional Reactive Simulatability). *Let structures (M, S) and (M', S) with identical set S of service ports be given, and let π be a predicate over a subset of the service in-ports of S. We say that (M, S)* is at least as secure as (or realizes) *(M', S)* under condition *π (written $(M, S) \geq^{\pi}_{\mathsf{sec}} (M', S)$) if for every configuration $conf = (M, S, \mathsf{H}, \mathsf{A})$ such that H fulfills π, there exists a configuration $conf' = (M', S, \mathsf{H}, \mathsf{A}')$ (with the same H) such that*

$$view_{conf}(\mathsf{H}) \approx view_{conf'}(\mathsf{H}).$$

Conditional universal simulatability and *conditional blackbox simulatability* are defined with the obvious modifications.

4 Composition Under Conditional Reactive Simulatability

In this section, we present composition theorems for conditional reactive simulatability. As mentioned in the introduction, when composing protocols which assume certain conditions (predicates) to hold on their service in-ports and in turn guarantee certain conditions (predicates) to hold on service in-ports of other protocols, cyclic dependencies may occur. In what follows, we first introduce the general setting (Section 4.2) and then present general composition theorems both for the acyclic and cyclic case (Section 4.2 and 4.3). While for the acyclic case no restrictions on predicates are put, for the cyclic case we require predicates to be safety properties.

4.1 The General Setting

One would expect that a protocol M_0 (for brevity we omit the service ports) that is simulatable under condition π can be securely composed with a protocol M_1 that fulfills π. In some applications, the larger protocol M_1 may fulfill π only if M_1 itself is used in a "sane" way, i.e., a predicate, say τ, is fulfilled on the service in-ports of M_1. Then, one would expect that M_0 securely composes with M_1 as long as τ is fulfilled. More generally, we consider the composition of several protocols with assume-guarantee conditions among them. In what follows, this is formalized.

Let π and τ be predicates over S_π and S_τ, respectively, and let $\mathbf{t}$ be a trace. We say that ***t*** *satisfies* $\tau \rightarrow \pi$ if $\mathbf{t}\lceil_{S_\tau} \in \tau$ implies $\mathbf{t}\lceil_{S_\pi} \in \pi$.

Definition 5 (Conditional Predicate Fulfillment). *Let M be a set of machines with service ports S, τ be a predicate over a subset S_τ of S^{in}, and π be a predicate over a subset S_π of $S^{C,out}$.*

Then, M fulfills π under condition τ *if $\tau \to \pi$ is satisfied with overwhelming probability no matter with which machines M interacts, i.e., for all sets $\overline{M}$ of machines such that $C := \{M, \overline{M}\}$ is closed, we have that*

$$\mathsf{Pr}_{t \leftarrow run_{C,k}} [t \text{ satisfies } \tau \to \pi] \text{ is overwhelming as a function in } k.$$

In what follows, for every $i = 1, \ldots, n$, let $P_i := (M_i, S_i)$ and $P_i' := (M_i', S_i)$ be real and ideal protocols, respectively. We consider the following predicates for these protocols.

Let τ_i^j be a predicate over $S_j^{C,out} \cap S_i^{in}$ (service in-ports of P_i to which P_j connects) and τ_i^{H} be a predicate over $S_i^{in} \setminus \bigcup_{j=1}^n S_j^{C,out}$ (service in-ports of P_i to which no other protocol connects). Intuitively, τ_i^j denotes the guarantees the ith protocol expects from the jth one. Analogously, τ_i^{H} specifies the guarantees the ith protocol expects from H. (Note that H may connect to all service in-ports of P_i the other protocols do not connect to.) We denote by

$$\tau_i = \tau_i^{\mathsf{H}} \wedge \bigwedge_{j \neq i} \tau_i^j \tag{1}$$

the guarantees the ith protocol expects from other protocols. Note that τ_i is a predicate over S_i^{in}.

Similarly, we now define the guarantees the ith protocol provides to other protocols. Let π_i^j be a predicate over $S_i^{C,out} \cap S_j^{in}$ (service in-ports of P_j to which P_i connects). Intuitively, π_i^j denotes the guarantees the ith protocol gives to the jth one. Note that we do not consider a predicate π_i^{H}. This simplifies our presentation and is without loss of generality since we are only interested in the compositionality properties of the composed protocol. We denote by

$$\pi_i = \bigwedge_{j \neq i} \pi_i^j. \tag{2}$$

the guarantees the ith protocol provides to other protocols. Note that π_i is a predicate over $\bigcup_{j \neq i} (S_i^{C,out} \cap S_j^{in})$.

In order for the composition theorems to hold, we clearly need that

$$\pi_j^i \subseteq \tau_i^j, \tag{3}$$

i.e., the guarantees τ_i^j the ith protocol expects from the jth one are actually met by the guarantees π_j^i the jth protocol offers to the ith protocol.

Obviously, in the setting above the guarantees among the protocols may be cyclic: the ith protocol provides guarantee π_i^j (and hence, τ_j^i) to the jth protocol only if the jth protocol guarantees τ_i^j, and vice versa, i.e., the jth protocol provides guarantee π_j^i (and hence, τ_i^j) to the ith protocol only if the ith protocol guarantees τ_j^i. Hence, in case $\tau_j^i \neq \texttt{true}$ *and* $\tau_i^j \neq \texttt{true}$ the dependencies between the ith and jth protocol are cyclic. The following is a concrete example.

Example 1. Say that an encryption system P_1 guarantees that the secret key is not output in plain as long as this secret key is not submitted as part of a plaintext for encryption. However, a larger protocol P_2 that uses that encryption system might want to encrypt plaintexts multiple times, possibly tagged with some syntactic type information. In particular, as long as no ciphertext itself contains the secret key in plain, this secret key will not be submitted for encryption. In other words, there is a mutual dependency between P_1 and P_2. (Obviously, in this particular case secure composition *is* possible.)

More generally, cyclic dependencies are defined as follows: Let the (directed) dependency graph $G = (V, E)$ be given by

$$V = \{V_1, \ldots, V_n\}, \quad E = \{(V_i, V_j) \ : \ \tau_i^j \neq \mathtt{true}\}. \tag{4}$$

If G is acyclic, we say that the dependencies between the protocols are *acyclic* or *non-mutual*, and otherwise, we say that they are *cyclic* or *mutual*.

In the following two subsections, we prove theorems for securely composing protocols, both in the case of acyclic and cyclic dependencies between the protocols. In these theorems we need to argue that the condition τ_i the ith protocol expects to be satisfied are in fact fulfilled when composing all protocols. In case of acyclic dependencies between the protocols, this is possible because the fulfillment of τ_i can be traced back to the conditions satisfied by other protocols or the honest users. In case of cyclic dependencies this is in general not possible because one runs into cycles. However, as we will see, if the predicates involved are safety properties, cyclic dependencies can be resolved. We note that the predicates informally stated in Example 1 are in fact safety predicates.

4.2 Composition in the Acyclic Case

In this section, we prove the following general composition theorem for the case of acyclic dependencies between the protocols.

Theorem 1. *For every $i = 1, \ldots, n$, let $P_i = (M_i, S_i)$ and $P_i' = (M_i', S_i)$ be protocols as introduced above with $P_i \geq_{\mathsf{sec}}^{\tau_i} P_i'$, and assume that M_i' fulfills π_i under condition τ_i where π_i and τ_i are defined as above and condition (3) is satisfied. If the dependencies between the protocols are acyclic, we have, for every i, that*

$$P_1 || \ldots || P_n \quad \geq_{\mathsf{sec}}^{\tau} \quad P_1 || \ldots || P_{i-1} || P_i' || P_{i+1} || \ldots || P_n, \tag{5}$$

where $\tau := \bigwedge_{j=1}^{n} \tau_j^{\mathsf{H}}$. Moreover,

$$P_1 || \ldots || P_n \quad \geq_{\mathsf{sec}}^{\tau} \quad P_1' || \ldots || P_n'. \tag{6}$$

□

Before we prove this theorem, we present useful corollaries of this theorem. The first corollary considers the case of two protocols and it easily follows from Theorem 1 using that $P_2 \geq_{\mathsf{sec}} P_2$.

Corollary 1 (Conditional Subroutine Composition). *Assume that $P_1 \geq^{\pi}_{\mathsf{sec}} P'_1$. Let $P_2 = (M_2, S_2)$ be a protocol such that M_2 i) connects to all ports over which π is defined and ii) fulfills π under condition τ where τ is a predicate over the service in-ports of P_2 to which P_1 does not connect. Then,*

$$P_1||P_2 \geq^{\tau}_{\mathsf{sec}} P'_1||P_2.$$

If $\tau = \mathsf{true}$, i.e., M_2 fulfills π unconditionally, we obtain

$$P_1||P_2 \geq_{\mathsf{sec}} P'_1||P_2.$$

□

Theorem 1 also allows to combine two protocols that are not connected via service ports:

Corollary 2 (Parallel Composition). *Assume that $P_1 \geq^{\pi_1}_{\mathsf{sec}} P'_1$ and $P_2 \geq^{\pi_2}_{\mathsf{sec}} P_2$ such that P_1 and P_2 are not connected via service ports. Then,*

$$P_1||P_2 \quad \geq^{\pi_1 \wedge \pi_2}_{\mathsf{sec}} \quad P'_1||P'_2.$$

□

Proof of Theorem 1. The proof relies on the following definition:

Definition 6. *Let M, τ, π be as in Definition 5. Then, M* fulfills π under enforced condition τ *if the predicate π is true with overwhelming probability when M interacts with machines that fulfill τ, i.e., for all sets $\overline{M}$ of machines that fulfill τ and such that $C := \{M, \overline{M}\}$ is closed, it holds that*

$$\mathsf{Pr}_{t \leftarrow run_{C,k}} [\textit{t satisfies } \pi] \text{ is overwhelming as a function in } k.$$

Obviously, if M fulfills π under condition τ, then M fulfills π under enforced condition τ.

As a preparation for our proof, note that for $i = 1, \ldots, n$, both M'_i and M_i fulfill π_i under enforced condition τ_i. For M'_i, this is clear by assumption, and for M_i it follows from $M_i \geq^{\tau}_{\mathsf{sec}} M'_i$. (Assuming that it is not true for M_i, one obtains an honest user which cannot be simulated, contradicting the assumption that $M_i \geq^{\tau}_{\mathsf{sec}} M'_i$.) Now fix $i \in \{1, \ldots, n\}$ and set

$$\tilde{P}_i := P_1|| \ldots ||P_n \text{ and } \tilde{P}'_i := P_1|| \ldots ||P_{i-1}||P'_i||P_{i+1}|| \ldots ||P_n.$$

Theorem statement (5): We need to show that for every configuration $conf = (\tilde{P}_i, \mathsf{H}, \mathsf{A})$ of $\tilde{P}_i$, where H fulfills τ, there is a valid configuration $conf' = (\tilde{P}'_i, \mathsf{H}, \mathsf{A}')$ of $\tilde{P}'_i$ with the same H such that

$$view_{conf}(\mathsf{H}) \approx view_{conf'}(\mathsf{H}). \tag{7}$$

Step 1: We construct a new user H_i as a combination of H with all protocol machines M_j except for M_i. Note that H_i is polynomial-time, so in any case, $conf_i := (P_i, \mathsf{H}_i, \mathsf{A})$ is a configuration of P_i.

H_i *fulfills* τ_i*:* Note that this statement makes sense because H_i connects to all of M_i's service ports. Due to space limitations, the somewhat technical proof is only presented in the technical report [2]. In this proof we use that M_i fulfills π_i under enforced condition τ_i.

Step 2: Now, since H_i fulfills τ_i, the conditional simulatability of M_i guarantees the existence of a configuration $conf'_i := (P'_i, \mathsf{H}_i, \mathsf{A}')$ with

$$view_{conf_i}(\mathsf{H}_i) \approx view_{conf'_i}(\mathsf{H}_i).$$

In particular, this yields

$$view_{conf_i}(\mathsf{H}) \approx view_{conf'_i}(\mathsf{H}) \tag{8}$$

for the submachine H of H_i.

Step 3: Decomposing H_i into H and the machines M_j $(j \neq i)$ yields a valid configuration $(\tilde{P}'_i, \mathsf{H}, \mathsf{A}')$ of protocol $\tilde{P}'_i$ such that (7) follows from (8) as desired.

Theorem statement (6): We show

$$P'_1 || \dots || P'_{i-1} || P_i \dots || P_n \quad \geq^{\tau}_{\mathsf{sec}} \quad P'_1 || \dots || P'_i || P_{i+1} \dots || P_n \tag{9}$$

for $i = 1, \dots, n$ by repeatedly applying (5). The case $i = 1$ is directly implied by (5), and for $i > 1$, all P_j with $j < i$ can be set to P'_j. Then by transitivity, (9) implies (6), which completes the proof. ■

4.3 Dealing with Mutual Dependencies – Composition in the Cyclic Case

In this section, we show that protocols can securely be composed even in case of cyclic dependencies given that the predicates considered are safety properties.

Theorem 2. *For every $i = 1, \dots, n$, let $P_i = (M_i, S_i)$ and $P'_i = (M'_i, S_i)$ be protocols as introduced in Section 4.1 with $P_i \geq^{\tau_i}_{\mathsf{sec}} P'_i$, and assume that M'_i and M_i fulfills π_i under condition τ_i where π_i and τ_i are defined as in Section 4.1 and condition (3) is satisfied. Also, assume that all predicates τ_i^j, τ_i^{H}, and π_i^j are safety properties. Then, for all i, we have:*

$$P_1 || \dots || P_n \quad \geq^{\tau}_{\mathsf{sec}} \quad P_1 || \dots || P_{i-1} || P'_i || P_{i+1} || \dots || P_n, \tag{10}$$

where $\tau := \bigwedge_{j=1}^{n} \tau_j^{\mathsf{H}}$. Moreover,

$$P_1 || \dots || P_n \quad \geq^{\tau}_{\mathsf{sec}} \quad P'_1 || \dots || P'_n. \tag{11}$$

□

We note that in Theorem 2 the requirement that M_i fulfills π_i under condition τ_i can be dispensed with if service out-ports are scheduled locally (which in most scenarios is the case): The reason is that, as in the proof of Theorem 1, it easily follows that if M'_i fulfills π_i under condition τ_i, then M_i fulfills π_i under enforced condition τ_i. Now, it is not hard to see that if service out-ports are scheduled locally, then the notion of Definition 6 implies the one of Definition 5. Hence, M_i fulfills π_i under condition τ_i.

Proof of Theorem 2. For the proof of Theorem 2, we need some terminology. For a trace **t** and predicates τ and π such that τ and π are safety properties, we say that ***t** satisfies* $\tau \rightarrow \pi$ *at any time* if $\mathbf{t}'$ satisfies $\tau \rightarrow \pi$ for every prefix $\mathbf{t}'$ of **t**.

Definition 7. *Let M, π, τ be as in Definition 5 such that π and τ are safety properties. Then, M* fulfills π under condition τ at any time *if the predicate $\tau \rightarrow \pi$ is satisfied at any time with overwhelming probability, no matter with which machines M interacts, i.e., for all sets $\overline{M}$ such that $C := \{M, \overline{M}\}$ is closed, it holds that*

$$\mathsf{Pr}_{\boldsymbol{t} \leftarrow run_{C,k}}\,[\boldsymbol{t} \textit{ satisfies } \tau \rightarrow \pi \textit{ at any time}] \textit{ is overwhelming as a function in } k. \quad (12)$$

We can show that the above notion is equivalent to the one defined in Definition 5.

Lemma 1. *Let M, π, and τ be as in Definition 7, and such that M contains no master scheduler. Then we have that M fulfills π under condition τ at any time iff M fulfills π under condition τ.* □

Proof. The implication from left to right is obvious. To see the converse direction, let $\overline{M}$ be a set of machines such that $C = \{M, \overline{M}\}$ is closed and let the polynomial $p(k)$ bound the runtime of $\overline{M}$. (Note that $\overline{M}$ necessarily contains a master scheduler.) First, by definition, if a trace **t** of C does not satisfy $\tau \rightarrow \pi$ at any time, then there exists a prefix $\mathbf{t}'$ of **t** which does not satisfy $\tau \rightarrow \pi$, i.e., $\mathbf{t}'\lceil_{S_\tau} \in \tau$ but $\mathbf{t}'\lceil_{S_\pi} \notin \pi$. Let $\mathbf{t}'$ be of minimal length with this property. It is easy to see that the last transition of $\mathbf{t}'$ must be a transition of $\overline{M}$. Now, assume that (12) is not satisfied, i.e., $\mathsf{Pr}_{\mathbf{t} \leftarrow run_{C,k}}\,[\mathbf{t}$ does not satisfy $\tau \rightarrow \pi$ at any time$]$ is a non-negligible function in k. Consider the machine $\overline{M}^*$ which simulates $\overline{M}$ but at the beginning randomly chooses a position $i \in \{1, \ldots, p(k) + 1\}$ and when activated for the ith time it stops (simulating $\overline{M}$). Let $C^* = \{M, \overline{M}^*\}$. Intuitively, $\overline{M}^*$ has a high probability to stop a run of C^* exactly when the trace produced so far does not satisfy $\tau \rightarrow \pi$. In fact, using that (12) is not satisfied it is easy to verify that $\mathsf{Pr}_{\mathbf{t} \leftarrow run_{C^*,k}}\,[\mathbf{t}$ does not satisfy $\tau \rightarrow \pi]$ is a non-negligible function in k. This implies that M does not fulfill π under condition τ. ■

We can now prove Theorem 2. For an overview of the proof, see Figure 2. We first prove (10), from which then (11) follows as in the proof of Theorem 1. Fix $i \in \{1, \ldots, n\}$ and set

$$\tilde{P}_i := P_1 || \ldots || P_n \text{ and } \tilde{P}'_i := P_1 || \ldots || P_{i-1} || P'_i || P_{i+1} || \ldots || P_n.$$

We need to show that for every configuration $conf = (\tilde{P}_i, \mathsf{H}, \mathsf{A})$ of $\tilde{P}_i$, where H fulfills τ, there is a valid configuration $conf' = (\tilde{P}'_i, \mathsf{H}, \mathsf{A}')$ of $\tilde{P}'_i$ with the same H, such that

$$view_{conf}(\mathsf{H}) \approx view_{conf'}(\mathsf{H}). \quad (13)$$

Step 1: We construct a new user H_i as a combination of H with all protocol machines M_j except for M_i. Note that H_i is polynomial-time, so in any case, $conf_i := (P_i, \mathsf{H}_i, \mathsf{A})$ is a configuration of P_i.

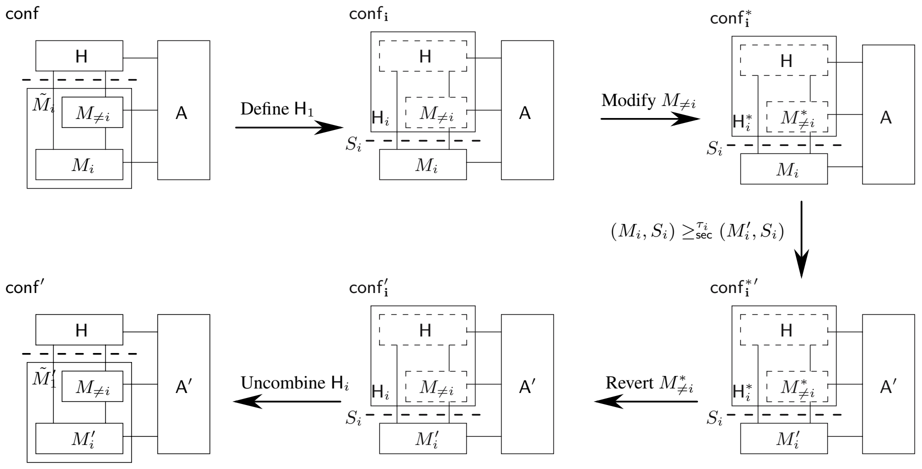


Fig. 2. Overview of the proof of Theorem 2

Step 2: We modify H_i into a new user H_i^* such that H_i^* fulfills τ_i. This is done by substituting all sets of submachines M_j $(j \neq i)$ of H_i by sets of machines M_j^* that fulfill their respective predicates π_j *without any preconditions*. More specifically, M_j^* simulates M_j and in addition checks whether τ_j is fulfilled, i.e., whether the observed sequence of inputs on in-ports of M_j lies in τ_j. By assumption, this can be done efficiently. If τ_j is not fulfilled, then M_j^* halts immediately.

First claim regarding H_i^**:* We claim that the view of the submachine H of H_i is not changed (non-negligibly) by this modification, i.e., we claim

$$view_{conf_i}(\mathsf{H}) \approx view_{conf_i^*}(\mathsf{H}) \tag{14}$$

where $conf_i^* = (P_i, \mathsf{H}_i^*, \mathsf{A})$.

Assume for contradiction that (14) does not hold. Then the probability that some τ_j $(j \neq i)$ is not fulfilled in a run of $conf_i$ is non-negligible (since otherwise, $conf_i$ and $conf_i^*$ behave identical). Let j be such that τ_j is with non-negligible probability the *first* of all predicates τ_ℓ $(1 \leq \ell \leq n)$ to become false in a run of $conf_i$. By "first", we mean that there is a prefix of the considered run that does not lie in τ_j, but all shorter prefixes lie in *all* τ_ℓ. (Note that by the prefix-closeness of all τ_ℓ such a prefix must exist for some j.)

Because of (1), there is thus a τ_j^r (with $r \in \{1, \ldots, n, \mathsf{H}\} \setminus \{j\}$) such that with non-negligible probability, τ_j^r becomes false before any other predicate τ_ℓ, $\ell \neq j$, and $\tau_j^{r'}$, $r' \neq r$, does. As $r = \mathsf{H}$ directly contradicts the assumption on H, we may assume $r \neq \mathsf{H}$.

Now by assumption, M_r fulfills π_r, and thus, by (3) and (1), also τ_j^r under condition τ_r (in the sense of Definition 5). By Lemma 1 and the just derived statement about τ_j^r, this implies that with non-negligible probability, τ_r is false *before* τ_j is. This is a contradiction to the choice of j.

Second claim regarding H_i^**:* We claim that H_i^* fulfills τ_i (without any precondition). By (1) and the assumption on H, it suffices to prove that for any $j \neq i$, M_j^* fulfills τ_i^j without any precondition. Now since M_j fulfills π_j under condition τ_j, it also does so at any time (Lemma 1). That is, it holds with overwhelming probability that at any point during a run of M_j, π_j is true unless τ_j becomes false.

By construction, M_j^* and M_j behave identically unless τ_j becomes false. That is, also M_j^* fulfills π_j under condition τ_j at any time. In particular, by definition of M_j^*, with overwhelming probability π_j is true when M_j^* halts. It is also easy to see that π_j cannot become false after M_j^* has halted. Hence, M_j^* fulfills π_j, and thus, τ_i^j unconditionally.

Step 3: As H_i^* fulfills τ_i, the conditional simulatability of M_i guarantees the existence of a configuration $conf_i^{*\prime} := (P_i', \mathsf{H}_i^*, \mathsf{A}')$ with

$$view_{conf_i^*}(\mathsf{H}_i^*) \approx view_{conf_i^{*\prime}}(\mathsf{H}_i^*).$$

In particular, this yields

$$view_{conf_i^*}(\mathsf{H}) \approx view_{conf_i^{*\prime}}(\mathsf{H}) \tag{15}$$

for the submachine H of H_i^*.

Step 4: We substitute H_i^* again by H_i. Since, by assumption, M_i' fulfills π_i under condition τ_i, analogously to Step 2 we can show that

$$view_{conf_i^{*\prime}}(\mathsf{H}) \approx view_{conf_i'}(\mathsf{H}) \tag{16}$$

where $conf_i' = (P_i', \mathsf{H}_i, \mathsf{A}')$.

Step 5: Decomposing H_i into H and the machines M_j $(j \neq i)$ yields a valid configuration $(\tilde{P}_i', \mathsf{H}, \mathsf{A}')$ of protocol $\tilde{P}_i'$ such that (13) and thus (10) follows from (14), (15) and (16) as desired. ■

5 Applications and Examples

In this section, we provide examples substantiating the claim that conditional reactive simulatability constitutes a suitable security notion for circumventing known impossibility results of simulating interesting abstractions of cryptography. In addition, we illustrate that imposing suitable constraints on the environment may allow for a simulation proof based on much weaker assumptions on the underlying cryptography. Generally speaking, conditional reactive simulatability allows for exploiting knowledge of which protocol class will use the protocol under investigation, resulting in more fine-grained reasoning about cryptographic protocols.

More specifically, we prove that Dolev-Yao style abstractions of symmetric encryption can be correctly simulated by conditioning environments to those cases that do not cause a so-called commitment problem. For unconditional simulatability, Dolev-Yao style symmetric encryption is known not to be simulatable at all [3]. If one further constraints the environment not to create key cycles, e.g., encrypting a key with itself, we can even establish conditional simulatability based on considerably weaker assumptions on the underlying cryptographic encryption scheme. Finally, we show that conditional simulatability may naturally entail unconditional simulatability for composed protocols again.

5.1 Conditional Simulatability of Dolev-Yao Style Symmetric Encryption

For Dolev-Yao style symmetric encryption, the following so-called commitment problem inherently prevents the successful application of unconditional reactive simulatability. The ideal encryption system must somehow allow that secret keys are sent from one participant to another. This is used for example in key-exchange protocols. If the ideal system simply allows keys to be sent at any time (and typical Dolev-Yao models do allow all valid terms to be sent at any time), the following problem can occur: An honest participant first sends a ciphertext such that the adversary can see it, and later sends both the contained plaintext and the key. This behavior may even be reasonably designed into protocols, e.g., the ciphertext might be an encrypted bet that is later opened. The simulator will first learn in some abstract way that a ciphertext was sent and has to simulate it by some bitstring, which the adversary sees. Later the simulator sees abstractly that a key becomes known and that the ciphertext contains a specific application message. It cannot change the application message, thus it must simulate a key that decrypts the old ciphertext bitstring (produced without knowledge of the application message) to this specific message.

We omit a rigorous definition of the absence of the commitment problem for Dolev-Yao style symmetric encryption as given in [3,5] but only give an informal definition for the sake of readability:

Definition 8 (No Commitment Property of Dolev-Yao Style Symmetric Encryption, informally). *The No Commitment property* NoComm *of Dolev-Yao style symmetric encryption consists of those traces of Dolev-Yao style symmetric encryption that satisfy the following trace predicate: If a term is encrypted at time t_1 in this trace by an honest user u with secret key sk, and at this time sk is not known to the adversary, then the adversary does not learn the key sk at any future time t_2 in this trace.*

Technically, the requirement that an adversary does not learn certain keys relies on the state of the Dolev-Yao model which keeps track of who knows which term; thus Definition 8 is syntactically not a predicate in the sense of Definition 2. However, those parts of the state that capture if an adversary already knows keys generated by honest users are uniquely determined by the preceding inputs at the service in-ports. Thus NoComm can naturally be recast as a property that is only defined at the service in-ports of the Dolev-Yao model and thus as a predicate in the sense of Definition 2 (however with a much more tedious notation).

The main result of [5] provides a simulation for those cases in which NoComm is fulfilled provided that the cryptographic encryption scheme fulfills the notion of dynamic KDM security [5]. We can now rephrase their result in our formalism to benefit from the compositionality guarantees entailed by our composition theorems. In the following, let $(\{\mathsf{TH}_{\mathcal{H}}^{\mathsf{cry\_sym,id}}\}, S_{\mathcal{H}})$ and $(\{\mathsf{M}_{\mathcal{E},u}^{\mathsf{cry\_sym,real}} \mid u \in \mathcal{H}\}, S_{\mathcal{H}})$ denote the Dolev-Yao model of symmetric encryption and its cryptographic realization from [3,5], respectively, for a set $\mathcal{H} \subseteq \{1, \ldots, n\}$ of honest users, and an encryption scheme $\mathcal{E}$.

Theorem 3 (Conditional Reactive Simulatability of Dolev-Yao Style Symmetric Encryption). *For all symmetric encryption schemes $\mathcal{E}$ that satisfy dynamic KDM security, and for all sets $\mathcal{H} \subseteq \{1,\dots,n\}$ of honest users, the realization of the Dolev-Yao model is at least as secure as the Dolev-Yao model under condition* NoComm, *i.e.,* $(\{\mathsf{M}^{\mathsf{cry\_sym,real}}_{\mathcal{E},u} \mid u \in \mathcal{H}\}, S_{\mathcal{H}}) \geq^{\mathsf{NoComm}}_{\mathsf{sec}} (\{\mathsf{TH}^{\mathsf{cry\_sym,id}}_{\mathcal{H}}\}, S_{\mathcal{H}})$. □

5.2 Securely Realizing Dolev-Yao Style Symmetric Encryption with Weaker Cryptography

While Theorem 3 shows that Dolev-Yao style symmetric encryption can be conditionally simulated by excluding the commitment property, it still relies on the strong assumption that the underlying encryption scheme satisfies dynamic KDM security – a very strong, non-standard notion for which no realization in the standard model of cryptography is known. However, it turns out that this strong notion is only necessary to deal with the quite exotic case that symmetric keys are encrypted in a cyclic manner, e.g., a key with itself. Most protocols however avoid such constructions by definition, and indeed further constraining simulatability to traces that do not contain key cycles yields a simulatability result based on considerably weaker assumptions on the underlying encryption scheme. More precisely, it suffices that the encryption scheme satisfies indistinguishability under adaptive chosen-ciphertext attacks as well as integrity of ciphertexts. This is the standard security definition of authenticated symmetric encryption [13,12], and efficient symmetric encryptions schemes provably secure in this sense exist under reasonable assumptions [21,27].

Definition 9 (No Key Cycles for Dolev-Yao Style Symmetric Encryption, informally). *The No Key Cycles property* $\mathsf{NoKeyCycles}$ *of Dolev-Yao style symmetric encryption consists of those traces of Dolev-Yao style symmetric encryption in which honest users do not create encryptions $E(sk_i, m_i)$ such that sk_{i+1} is a subterm of m_i for $i = 0,\dots,j-1$ for some j, and sk_0 is a subterm of m_j.*

Theorem 4 (Conditional Reactive Simulatability of Dolev-Yao Style Symmetric Encryption w/o Key Cycles). *For all authenticated symmetric encryption schemes $\mathcal{E}$ and all sets $\mathcal{H} \subseteq \{1,\dots,n\}$ of honest users, the realization of the Dolev-Yao model is at least as secure as the Dolev-Yao model under condition* $\mathsf{NoComm} \wedge \mathsf{NoKeyCycles}$, *i.e.,* $(\{\mathsf{M}^{\mathsf{cry\_sym,real}}_{\mathcal{E},u} \mid u \in \mathcal{H}\}, S_{\mathcal{H}}) \geq^{\mathsf{NoComm}\wedge\mathsf{NoKeyCycles}}_{\mathsf{sec}} (\{\mathsf{TH}^{\mathsf{cry\_sym,id}}_{\mathcal{H}}\}, S_{\mathcal{H}})$. □

5.3 Simulatable Protocols from Conditionally Simulatable Subprotocols

We finally illustrate, exploiting Corollary 1, that conditional simulatability can often be turned into unconditional simulatability again (and in fact, it seems hard to come up with a non-artificial example for which Corollary 1 does not apply). Consider a secure channel between two parties that uses Dolev-Yao style symmetric encryption as a subprimitive, which itself is only conditionally simulatable. The secure channel consists of two machines M_1 and M_2. M_1 expects a message m as input at a service port $\mathsf{in?}$, and encrypts this message with a symmetric key k shared with M_2. The encryption is computed using Dolev-Yao style symmetric encryption as a subprimitive, i.e., m

is output at a service port $\mathsf{enc\_out}_1!$ and the resulting encryption e is obtained at a service port $\mathsf{enc\_in}_1?$. M_2 outputs the message at a service port $\mathsf{out}!$. We do not give a rigorous definition of this behavior here since this would presuppose introducing a significant amount of notion from [3] but it should be clear already that this secure channel neither causes a commitment problem nor any key cycles by construction. Let $(M^{\mathsf{sc}}, S^{\mathsf{sc}}) := (\{\mathsf{M}_1, \mathsf{M}_2\}, \{\mathsf{in}?, \mathsf{out}!, \mathsf{enc\_out}_1!, \mathsf{enc\_in}_1?\})$ denote the secure channel.

Theorem 5. *For all authenticated symmetric encryption schemes $\mathcal{E}$, and for $\mathcal{H} = \{1,2\}$, the secure channel based on the realization is unconditionally at least as secure as the secure channel based on the Dolev-Yao model, i.e., $(M^{\mathsf{sc}}, S^{\mathsf{sc}})||(\{\mathsf{M}^{\mathsf{cry\_sym,real}}_{\mathcal{E},u} \mid u \in \mathcal{H}\}, S_{\mathcal{H}}) \geq_{\mathsf{sec}} (M^{\mathsf{sc}}, S^{\mathsf{sc}})||(\{\mathsf{TH}^{\mathsf{cry\_sym,id}}_{\mathcal{H}}\}, S_{\mathcal{H}})$.* □

6 Conclusion

We presented a relaxation of simulatability, one of the central concepts of modern cryptography for defining and analyzing the security of multi-party protocols, by permitting to constrain environments to adhere to certain behaviors. The resulting notion is called conditional reactive simulatability. It constitutes a more fine-grained security notion that is achievable i) for protocols for which traditional simulatability is too strong a notion, and ii) based on weaker requirements on the underlying cryptography. In addition, conditional reactive simulatability maintains the interesting property that for various protocol classes, composition of conditionally simulatable protocols yield protocols that are simulatable in the traditional sense.

We furthermore showed that despite imposing restrictions on the surrounding protocol and thus giving up the universal quantification of environments that naturally allowed for compositionality proofs in earlier works, the notion of conditional reactive simulatability still entails strong compositionality guarantees. In particular, this holds for the common case of composing so-called assume-guarantee specifications, i.e., specifications that are known to behave properly if offered suitable inputs, provided that these assumptions and guarantees constitute arbitrary trace properties that do not give rise to cyclic dependencies. We further investigated the theoretically more demanding (but arguably practically less interesting) case of cyclic dependencies among such specifications and proved a similar composition theorem under the additional assumption that conditions are expressible as safety properties.

Acknowledgments. We thank *Martín Abadi* for interesting discussions.

References

1. Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactional on Programmming Languages and Systems.*, 17(3):507–534, 1995.
2. M. Backes, M. Dürmuth, D. Hofheinz, and R. Küsters. Conditional Reactive Simulatability. Technical Report 132, Cryptology ePrint Archive, 2006. Online available at `http://eprint.iacr.org/2006/132.ps`.

3. Michael Backes and Birgit Pfitzmann. Symmetric encryption in a simulatable dolev-yao style cryptographic library. In *17th IEEE Computer Security Foundations Workshop, Proceedings of CSFW 2004*, pages 204–218. IEEE Computer Society, 2004.
4. Michael Backes and Birgit Pfitzmann. Limits of the cryptographic realization of Dolev-Yao-style XOR. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *Computer Security, Proceedings of ESORICS 2005*, number 3679 in Lecture Notes in Computer Science, pages 178–196. Springer-Verlag, 2005.
5. Michael Backes, Birgit Pfitzmann, and Andre Scedrov. Key-dependent message security under active attacks. ePrint Archive, 2005/421, 2006.
6. Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations. In *10th ACM Conference on Computer and Communications Security, Proceedings of CCS 2003*, pages 220–230. ACM Press, 2003. Extended abstract.
7. Michael Backes, Birgit Pfitzmann, and Michael Waidner. A general composition theorem for secure reactive systems. In Moni Naor, editor, *Theory of Cryptography, Proceedings of TCC 2004*, number 2951 in Lecture Notes in Computer Science, pages 336–354. Springer-Verlag, 2004.
8. Michael Backes, Birgit Pfitzmann, and Michael Waidner. Secure asynchronous reactive systems. IACR ePrint Archive, March 2004.
9. Michael Backes, Birgit Pfitzmann, and Michael Waidner. Limits of the Reactive Simulatability/UC of Dolev-Yao models with hashes. Cryptology ePrint Archive 2006/068, 2006.
10. Boaz Barak and Amit Sahai. How to play almost any mental game over the net — concurrent composition via super-polynomial simulation. In *46th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2005*, pages 543–552. IEEE Computer Society, 2005.
11. Donald Beaver. Foundations of secure interactive computing. In Joan Feigenbaum, editor, *Advances in Cryptology, Proceedings of CRYPTO '91*, number 576 in Lecture Notes in Computer Science, pages 377–391. Springer-Verlag, 1992.
12. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology - ASIACRYPT 2000*, pages 531–545, 2000.
13. Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient constructions. In *Advances in Cryptology - ASIACRYPT 2000*, pages 317–330, 2000.
14. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2001*, pages 136–145. IEEE Computer Society, 2001.
15. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. IACR ePrint Archive, January 2005. Full and revised version of [14].
16. Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology, Proceedings of CRYPTO 2001*, number 2139 in Lecture Notes in Computer Science, pages 19–40. Springer-Verlag, 2001.
17. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2002*, pages 494–503. ACM Press, 2002. Extended abstract.
18. Anupam Datta, Ante Derek, John C. Mitchell, Ajith Ramanathan, and Andre Scedrov. Games and the impossibility of realizable ideal functionality. In *To appear in Proceedings of Theory of Cryptography (TCC 2006)*, 2006.
19. Anupam Datta, Ralf Küsters, John C. Mitchell, and Ajith Ramanathan. On the relationships between notions of simulation-based security. In *In Theory of Cryptography, Proceedings of TCC 2005*, pages 476–494, 2005.

20. Dimitra Giannakopoulou, Corina S. Pasareanu, and Jamieson M. Cobleigh. Assume-guarantee verification of source code with design-level assumptions. In *Proceedings 26th International Conference on Software Engineering*, pages 211–220, 2004.
21. Virgil D. Gligor and Pompiliu Donescu. Fast encryption and authentication: Xcbc encryption and xecb authentication modes. In *Proceedings 8th Fast Software Encryption"*, pages 82–108, 2001.
22. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game—a completeness theorem for protocols with honest majority. In *Nineteenth Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1987*, pages 218–229. ACM Press, 1987. Extended abstract.
23. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
24. Heather Hinton. Composing partially-specified systems. In *IEEE Symposium on Security and Privacy, Proceedings of SSP 1998*, pages 27–39. IEEE Computer Society, 1998.
25. Dennis Hofheinz and Jörn Müller-Quade. Universally composable commitments using random oracles. In Moni Naor, editor, *Theory of Cryptography, Proceedings of TCC 2004*, number 2951 in Lecture Notes in Computer Science, pages 58–76. Springer-Verlag, 2004.
26. C. Jones. Specification and design of (parallel) programs. In *Information Processing 83: Proceedings 9th IFIP World Congress*, pages 321–322, 1983.
27. Charanjit Jutla. Encryption modes with almost free message integrity. In *Advances in Crptology - EUROCRYPT 2001*, pages 529–544, 2001.
28. Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *44th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2003*, pages 394–403. IEEE Computer Society, 2003.
29. Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated byzantine agreement. In *34th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2002*, pages 514–523. ACM Press, 2002.
30. Silvio Micali and Phillip Rogaway. Secure computation. In Joan Feigenbaum, editor, *Advances in Cryptology, Proceedings of CRYPTO '91*, number 576 in Lecture Notes in Computer Science, pages 392–404. Springer-Verlag, 1992. Abstract.
31. J. Misra and M. Chandy. Proofs of networks of processes. *IEEE Transactions of Software Engineering*, 7(4):417–426, 1981.
32. Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy, Proceedings of SSP 2001*, pages 184–200. IEEE Computer Society, 2001.
33. Manoj Prabhakaran and Amit Sahai. New notions of security: Achieving universal composability without trusted setup. In *36th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2004*, pages 242–251. ACM Press, 2004.
34. Andrew Chi-Chih Yao. Theory and applications of trapdoor functions. In *23th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 1982*, pages 80–91. IEEE Computer Society, 1982.

SessionSafe: Implementing XSS Immune Session Handling*

Martin Johns

Security in Distributed Systems (SVS)
University of Hamburg, Dept of Informatics
Vogt-Koelln-Str. 30, D-22527 Hamburg
johns@informatik.uni-hamburg.de

Abstract. With the growing trend towards the use of web applications the danger posed by cross site scripting vulnerabilities gains severity. The most serious threats resulting from cross site scripting vulnerabilities are session hijacking attacks: Exploits that steal or fraudulently use the victim's identity. In this paper we classify currently known attack methods to enable the development of countermeasures against this threat. By close examination of the resulting attack classes, we identify the web application's characteristics which are responsible for enabling the single attack methods: The availability of session tokens via JavaScript, the pre-knowledge of the application's URLs and the implicit trust relationship between webpages of same origin. Building on this work we introduce three novel server side techniques to prevent session hijacking attacks. Each proposed countermeasure removes one of the identified prerequisites of the attack classes. SessionSafe, a combination of the proposed methods, protects the web application by removing the fundamental requirements of session hijacking attacks, thus disabling the attacks reliably.

1 Introduction

Web applications as frontends for online services enjoy an ever growing popularity. In addition, a general direction towards web applications replacing traditional client side executables can be observed during the last years. Email, banking, project management or business services move from specialized programs to the web browser.

In close connection to this trend, web application vulnerabilities move from being mere annoyances towards posing severe threats. With companies like Google and Yahoo starting to integrate various web applications under one authentication process the impact of single vulnerabilities even increases, as one weakness could now endanger a whole range of different applications. One of the most common threats is session hijacking, an attack method that targets the victim's identity. Session hijacking is often feasible because web applications frequently

* This work was supported by the German Ministry of Economics and Technology (BMWi) as part of the project "secologic" , www.secologic.org.

suffer from cross site scripting (XSS) vulnerabilities. In most cases one single XSS vulnerability suffices to compromise the entire application.

Even though XSS is known at least since the year 2000 [4], this class of weaknesses is still a serious issue. In 2005 alone 167 XSS vulnerabilities have been reported to the BugTraq mailing list. Also, the year 2005 brought a shift to more sophisticated XSS attacks: In July 2005 Anton Rager presented the XSS proxy, a tool that allowed for the first time a systematic, semi automatic exploitation of XSS vulnerabilities [18]. Wade Alcorn described in September 2005 an XSS virus that is able to self propagate from server to server, provided all these servers run the same vulnerable web application [2]. Finally, in October 2005 the self replicating "mySpace XSS worm" infected more than one million user profiles [19]. While the cause of XSS vulnerabilities is almost always insufficient output sanitization in connection with handling of user provided input strings, ensuring the absence of this flaw is of growing difficulty. Today's web applications are complex. They often consist of numerous different server side technologies, legacy code and third party components. Thus, enforcing consistent input handling a non-trivial task. In this paper we describe a novel approach to protect web applications against XSS session hijacking attacks. Instead concentrating on user input, we disable session hijacking by removing the attacks' basic requirements.

The remainder of the paper is organized as follows. Section 2 discusses web application security topics that are relevant for the proposed methods. Section 3 describes and classifies currently known XSS session hijacking attacks. In Section 4 follows a description of our countermeasures; those countermeasures then will be discussed in Section 5. Finally, after looking at related work in Section 6, we conclude in Section 7.

2 Technical Background

2.1 Session Management

Because of http's stateless nature [7] web applications that require authentication need to implement additional measures to keep their users authenticated. To achieve this, session identifiers (SID) are used: After a successful authentication the web application generates the SID and transmits it to the client. Every following http request that contains this SID is regarded as belonging to this particular user. Thus the SID is a credential that both identifies and authenticates the user. The protection of this information is therefore essential for the security of the application. There are three methods of implementing SIDs: inclusion of the identifier in the URL, communication of the identifier via POST parameters or storing the identifier in browser cookies:

URL query strings: The SID is included in every URL that points to a resource of the web application: `<a href="some_page?SID=g2k42a">...</a>`

POST parameters: Instead of using hyperlinks for the navigation through the application HTML forms are used. In this case, the SID is stored in a hidden form field. Whenever a navigation is initiated, the according HTML form is submitted, thus sending the SID as part of the request's body.

Cookies: Using cookies for SID storage is broadly used in today's web applications. Web browser cookies technology [14] provides persistent data storage on the client side. A cookie is a data set consisting at least of the cookie's name, value and domain. It is sent by the web server as part of an http response message using the `Set-Cookie` header field. The cookie's domain property is controlled by the URL of the http response that is responsible for setting the cookie. The property's value must be a valid domain suffix of the response's full domain and contain at least the top level domain (tld) and the original domain. The cookie's domain property is used to determine in which http requests the cookie is included. Whenever the web browser accesses a webpage that lies in the domain of the cookie (the domain value of the cookie is a valid domain suffix of the page's URL), the cookie is automatically included in the http request using the `Cookie` field.

2.2 Cross Site Scripting

Cross Site Scripting (XSS) denotes a class of attacks. These attacks are possible, if a web application allows the inclusion of insufficiently filtered data into web pages. If a web application fails to remove HTML tags and to encode special characters like `"`, `'`, `<` or `>` from user provided strings, an attacker will be capable of inserting malicious script code into webpages. Consequently, this script code is executed by the victim's web browser and runs therefore in the victim's security context. Even though XSS is not necessarily limited to JavaScript, attacks may also use other embedded scripting languages, this paper focuses its description of attacks and countermeasures on JavaScript. While completely removing scripting code through output filtering is feasible to disarm XSS threats posed by more obscure client side scripting languages like VBScript, this procedure is not an option in the case of JavaScript. JavaScript is ubiquitous, deeply integrated in common DHTML techniques and used on the vast majority of websites. A rogue JavaScript has almost unlimited power over the webpage it is contained in. Malicious scripts can, for example, change the appearance of the page, steal cookies, or redirect form actions to steal sensitive information (see Section 3 and [9] for further details).

2.3 JavaScript Security Aspects

JavaScript contains semantics of object oriented programming as well as aspects that are usually found in functional languages. In this paper we describe JavaScript from an object orientated point of view. JavaScript in webpages is executed "sandboxed": It has no access to the browser's host system and only limited access to the web browser's properties. JavaScript's capabilities to manipulate the appearance and semantics of a webpage are provided through the global object `document` which is a reference to the root element of the page's DOM tree [11]. A script can create, delete or alter most of the tree's elements. JavaScript 1.5 has been standardized by ECMA as "ECMAScript" [6] in 1999.

The same-origin policy: The "same-origin policy" defines what is accessible by a JavaScript. A JavaScript is only allowed read and/or write access to properties of windows or documents that have the same origin as the script itself. The origin of an element is defined by specific elements of the URL it has been loaded from: The host, the port and the protocol [8]. While port and protocol are fixed characteristics, JavaScript can influence the host property to mitigate this policy. This is possible because a webpage's host value is reflected in its DOM tree as the `domain` attribute of the `document` object. JavaScript is allowed to set this property to a valid domain suffix of the original host. For example, a JavaScript could change `document.domain` from `www.example.org` to the suffix `example.org`. JavaScript is not allowed to change it into containing only the top level domain (i.e. `.org`) or some arbitrary domain value. The same-origin policy defines also which cookies are accessible by JavaScript.

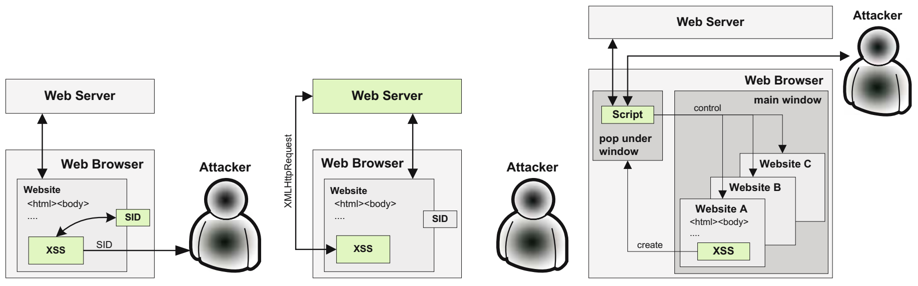
Public, privileged and private members in JavaScript objects: A little known fact is, that JavaScript supports information hiding via encapsulation. The reason for this obscurity is that JavaScript does not provide access specifiers like "private" to implement encapsulation. Encapsulation in JavaScript is implemented via the scope of a variable. Depending on the context in which a variable or a method is created, its visibility and its access rights are defined [6]. From an object oriented point of view this translates to three access levels: "public", "privileged" and "private" [5]: "Public" members of objects are accessible from the outside. They are either defined by prototype functions [6] or created as anonymous functions and added to the object after object creation. Either way: They are created within the global scope of the object's surroundings. Public methods cannot access private members. "Private" members are only accessible by private or privileged methods in the same object. They are defined on object creation and only exist in the local scope of the object. Private methods cannot be redefined from the outside after object creation. "Privileged" methods are accessible from the outside. They can read and write private variables and call private methods. Privileged methods have to be defined on object creation and exist therefore in the local scope of the object. The keyword `this` is used to export the methods to the global scope, so that they can be accessed from outside the object. If a privileged method is redefined from the outside after object creation, it will become part of the global scope and its state will change therefore to "public".

3 A Classification of XSS Session Hijacking Attacks

All currently known XSS session hijacking attack methods can be assigned to one of the following different classes: "Session ID theft", "Browser Hijacking" and "Background XSS Propagation".

3.1 Session ID Theft

As described in Section 2.1, web applications usually employ a SID to track the authenticated state of a user. Every request that contains this SID is regarded as belonging to the authenticated user. If an attacker can exploit an XSS vulnerability of the web application, he might use a malicious JavaScript to steal



a. SID Theft b. Browser Hijacking c. XSS Propagation

Fig. 1. The three classes of XSS session hijacking attacks [24]

the user's SID. It does not matter which of the methods described in Section 2.1 of SID storage is used by the application - in all these cases the attacking script is able to obtain the SID. The attacking script is now able to communicate the SID over the internet to the attacker. As long as the SID is valid, the attacker is now able to impersonate the attacked client [13].

3.2 Browser Hijacking

This method of session hijacking does not require the communication of the SID over the internet. The whole attack takes place in the victim's browser. Modern web browsers provide the XMLHttpRequest object, which can be used to place GET and POST requests to URLs, that satisfy the same-origin policy. Instead of transferring the SID or other authentication credentials to the attacker, the "Browser Hijacking" attack uses this ability to place a series of http requests to the web application. The application's server cannot differentiate between regular, user initiated requests and the requests that are placed by the script. The malicious script is therefore capable of acting under the identity of the user and commit arbitrary actions on the web application. In 2005 the so called "mySpace Worm" employed this technique to create a self replicating JavaScript worm that infected approximately one million profiles on the website myspace.com [19].

3.3 Background XSS Propagation

Usually not all pages of a web application are vulnerable to cross site scripting. For the attacks described above, it is sufficient that the user visits only one vulnerable page in which a malicious script has been inserted. However, other attack scenarios require the existence of a JavaScript on a certain webpage to work. For example, even when credit card information has been submitted it is seldom displayed in the web browser. In order to steal this information a malicious script would have to access the HTML form that is used to enter it. Let us assume the following scenario: Webpage A of the application is vulnerable against XSS whereas webpage B is not. Furthermore, webpage B is the page containing the credit card entry form. To steal the credit card information, the attacker would have to propagate the XSS attack from page A to page B. There are two techniques that allow this attack:

Propagation via iframe inclusion: In this case, the XSS replaces the displayed page with an iframe that takes over the whole browser window. Furthermore, the attacking script causes the iframe to display the attacked webpage, thus creating the impression that noting has happened. From now on every user navigation is done inside the iframe. While the user keeps on using the application, the attacking script is still active in the document that contains the iframe. As long as the user does not leave the application's domain, the malicious script is able to monitor the user's surfing and to include further scripts in the webpages that are displayed inside the iframe. A related attack is described in [18].

Propagation via pop under windows: A second way of XSS propagation can be implemented using "pop under" windows. The term "pop under" window denotes the method of opening a second browser window that immediately sends itself to the background. On sufficiently fast computers users often fail to notice the opening of such an unwanted window. The attacking script opens such a window and inserts script code in the new window's body. The new window has a link to the DOM tree of the original document (the father window) via the `window.opener` property. This link stays valid as long as the `domain` property of the father window does not change, even after the user resumes navigating through the web application. The script that was included in the new window is therefore able to monitor the user's behavior and include arbitrary scripts in web pages of the application that are visited during the user's session.

4 Countermeasures Against Session Hijacking

In the next Sections we propose countermeasures against the described session hijacking attacks. Each of these countermeasures is designed to disarm at least one of the specified threats.

4.1 Session ID Protection Through Deferred Loading

The main idea of the proposed technique is twofold: For one, we store the SID in such a way that malicious JavaScript code bound by the "same-origin policy" is not able to access it any longer. Secondly, we introduce a deferred process of loading the webpage, so that security sensitive actions can be done, while the page is still in a trustworthy state. This deferred loading process also guarantees the avoidance of timing problems.

To successfully protect the SID, it has to be kept out of reach for any JavaScript that is embedded into the webpage. For this reason, we store the SID in a cookie that does not belong to the webpage's domain. Instead, the cookie is stored for a different (sub-)domain that is also under the control of the web application. In the following paragraphs the main web application will reside on `www.example.org`, while the cookies will be set for `secure.example.org`. The domain `secure.example.org` is hosted on the same server as the main web application. Server scripts of the main web application have to be able to share data and/or communicate with the server scripts on `secure.example.org` for this technique to work. On the `secure` domain only two simple server scripts exist: `getCookie.ext` and `setCookie.ext`. Both are only used to transport the cookie data. The data that they respond is irrelevant - in the following description they return a 1 by 1 pixel image.

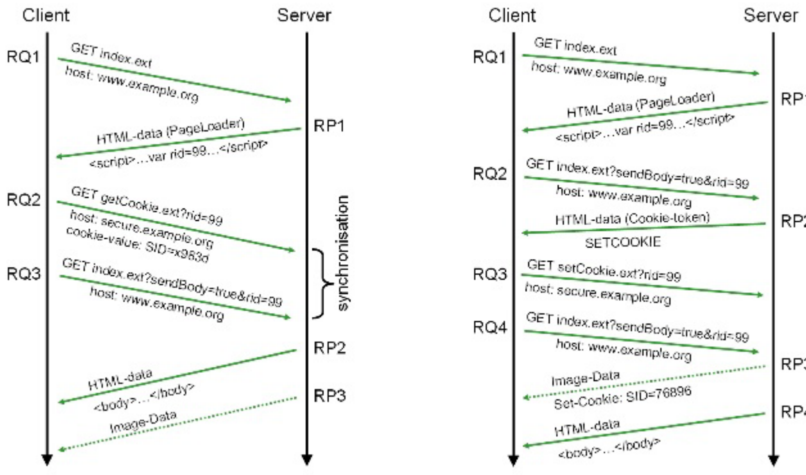
To carry out the deferred loading process we introduce the “PageLoader”. The PageLoader is a JavaScript that has the purpose to manage the cookie transport and to load the webpage’s content. To transport the cookie data from the client to the server it includes an image with the `getCookie.ext` script as URL. For setting a cookie it does the same with the `setCookie.ext` script. To display the webpage’s body the PageLoader requests the body data using the XMLHttpRequest object. In the following specifications the abbreviations “RQ” and “RP” denote respectively “http request” and “http response”.

Getting the cookie data. The process of transferring an existing cookie from the client to the server is straight forward. In the following scenario the client web browser already possesses a cookie for the domain `secure.example.org`. The loading of a webpage for which a cookie has been set consists of the following steps (see figure 1.a):

1. The client’s web browser sends an http request for `www.example.org/index.ext` (RQ1).
2. The web server replies with a small HTML page that only contains the PageLoader (RP1).
3. The PageLoader includes the `getCookie.ext` image in the DOM tree of the webpage. This causes the client’s web browser to request the image from the server (RQ2). The cookie containing the SID that is stored for `secure.example.org` is included in this request automatically.
4. The PageLoader also requests the webpage’s body using the XMLHttpRequest object (RQ3). This http request happens parallel to the http request for the `getCookie.ext` image.
5. The web server waits with the answer to RQ3 until it has received and processed the request for the `getCookie.ext` image. According to the cookie data that this request contained, the web server is able to compute and send the webpage’s body (RP2).
6. The PageLoader receives the body of the webpage and uses the `document.write` method to display the data.

The web server has to be able to identify that the last two http requests (RQ2 and RQ3) where initiated by the same PageLoader and therefore came from the same client. For this reason the PageLoader uses a request ID (RID) that is included in the URLs of the request RQ2 and RQ3. The RID is used by the web server to synchronize the request data between the domains `www` and `secure`.

Setting a cookie: The usually preceding process of transferring existing cookie data from the client to the server, as described above, is left out for brevity. With this simplification the setting of a cookie consists of the following steps (see figure 1.b):



a. Getting a cookie b. Setting a cookie

Fig. 2. schematic view of the processes

1. The client's web browser sends an http request for `www.example.org/index.ext` (RQ1).
2. The web server replies with the PageLoader (RP1) and the PageLoader subsequently requests the body data (RQ2).
3. The web server computes the request RQ2. Because of the outcome of the computation the server decides to place a cookie. The server replies with "`SETCOOKIE`" to the PageLoader's request for the body data (RP2).
4. The PageLoader receives the "`SETCOOKIE`" token and includes the `setCookie.ext` image in the DOM tree of the webpage. This causes the client's web browser to request the image from the server (RQ3).
5. The PageLoader also requests the webpage's body once more (RQ4). This http request happens parallel to the http request for the `setCookie.ext` image.
6. The web server receives the request for the image and includes the cookie data in the response (RP3). The web server marks the RID as "used" (see below).
7. The web server waits with the answer to RQ4 until it successfully delivered the `setCookie.ext` image to the client. After the image request has been processed the body data gets sent (RP4).

There is an important timing aspect to take into consideration: The PageLoader should not display the HTML body data before the cookie setting process is finished, and the web server should never reply more than once to a `setCookie.ext` request containing the same RID value. Otherwise, the security advantage of the proposed method would be lost, because after the HTML body data is displayed in the client's browser a malicious JavaScript might be executed. This script then could read the DOM tree to obtain the full URL of the `setCookie.ext` image and communicate this information via the internet to the attacker. If at this point of time the web server still treats this image URL (more precise: the RID value) as valid, the attacker would be able to successfully request the image including the cookie data from the web server. If no invalidation of the RID happens, the described technique will only shift the attack target from losing the cookie value to losing the RID value. For the same reason the RID value must be random and of sufficient length in order to prevent guessing attacks. Because of the restrictions posed by the same-origin policy, the cookies stored for `secure.example.org` are not accessible by JavaScript embedded into a page from `www.example.org`. Furthermore, JavaScript is not allowed to change `document.domain` to `secure.example.org` because this value is not a valid domain suffix of the original host value `www.example.org`. The `secure` subdomain only contains the two specified server scripts for cookie transportation. The reply data of these server scripts does not contain any dynamic data. Thus, an XSS attack on `secure.example.org` is not feasible. Therefore, the proposed technique successfully prevents cookie stealing attacks without limiting cookie usage.

4.2 One-Time URLs

To defend against browser hijacking (see 3.2) we have to remove the fundamental basis of this attack class. Every browser hijacking attack has one characteristic in common: The attacking script submits one or more http requests to the server and potentially parses the server's response. The basis for this attack is therefore the attacker's knowledge of the web application's URLs. The main idea of the proposed countermeasure is to enhance the application's URLs with a secret component which cannot be known, obtained, or guessed by the attacking JavaScript. As long as the server responds only to requests for URLs with a valid secret component, the attacker is unable to execute a browser hijacking attack.

To determine the requirements for successful URL hiding we have to examine the abilities of rogue JavaScript. The secret URL component has to satisfy the following limitations:

– It has to be unguessable.
– It must not be stored in an HTML element, e.g. a hidden form field. JavaScript can access the DOM tree and therefore is able to obtain any information that is included in the HTML code.
– It must not be stored in public JavaScript variables. All JavaScript code in one webpage exists in the same namespace. Therefore, a malicious script is able to execute any existing JavaScript function and read any available public variable.
– It must not be hard coded in JavaScript. Every JavaScript element (i.e. object, function or variable) natively supports the function `toString()` which per default returns the source code of the element. Malicious script could use this function to parse code for embedded information.

– It has to be valid only once. Otherwise, the attacker's script could use the value of `document.location` to emulate the loading process of the displayed page.

Thus, the only place to keep data protected from malicious JavaScript is a private variable of a JavaScript object. In the following paragraphs we show how this approach can be implemented. We only describe this implementation in respect of randomizing hyperlink URLs. The randomization of HTML forms is left out for brevity - the applicable technique is equivalent.

The URLRandomizer Object: Our approach uses a URL GET parameter called "rnonce" to implement the URL randomization. Only URLs containing a valid rnonce are treated as authorized by the web server. To conduct the actual randomization of the URLs we introduce the URLRandomizer, a JavaScript object included in every webpage. As introduced above, the URLRandomizer object contains a private variable that holds all valid randomization data. During object creation the URLRandomizer requests from the web server a list of valid nonces for the webpage's URLs. This request has to be done as a separate http request on runtime. Otherwise, the list of valid nonce would be part of the source code of the HTML page and therefore unprotected against XSS attacks. The URLRandomizer object also possesses a privileged method called "go()" that has the purpose to direct the browser to new URLs. This method is called by hyperlinks that point to URLs that require randomization:

```
<a href="#" onclick="URLRandomizer.go('placeOrder.ext');">Order</a>
```

The "go()" method uses the function parameter and the object's private randomization data to generate a URL that includes a valid rnonce. This URL is immediately assigned to the global attribute `document.location` causing the client's web browser to navigate to that URL. Listing 1 shows a sketch of the URL-Randomizer's go() function. In this code "validNonces" is a private hashtable containing the valid randomization data.

```
this.go = function(path){
    nonce = validNonces[path];
    document.location =
         "http://www.example.org/"+path+"?rnonce="+nonce;
}
```

Listing 1.1. sketch of the URLRandomizers go() function

Timing aspects: As mentioned above, the URLRandomizer obtains the valid randomization data from the server by requesting it via http. This leads to the following requirement: The URL that is used to get this data also has to be randomized and limited to one time use. It is furthermore important, that the URLRandomizer object is created early during the HTML parsing process and that the randomization data is requested on object creation. Otherwise, malicious JavaScript could examine the source code of the URLRandomizer to obtain the URL for the randomization data and request it before the legitimate object does. As long as the definition and creation of the URLRandomizer object is the first JavaScript code that is encountered in the parsing process, this kind of timing attack cannot happen.

Entering the randomized domain: It has to be ensured that the first webpage, which contains the URLRandomizer object, was not requested by a potential malicious JavaScript, but by a proper user of the web application. Therefore, an interactive process that cannot be imitated by a program is required for the transition. The natural solution for this problem is combining the changeover to one-time URLs with the web application's authentication process. In situations where no authentication takes place CAPTCHA (Completely Automated Public Turing-Test to Tell Computers and Humans Apart) technology [23] can be employed for the transition. If no interactive boundary exists between the realms of static and one-time URLs, a malicious JavaScript would be able to request the URL of the entry point to the web application and parse its HTML source code. This way the script is able to acquire the URL that is used by the URLRandomizer to get the randomization data.

Disadvantages of this approach: The proposed method poses some restrictions that break common web browser functionality: Because it is forbidden to use a random nonce more than once, the web server regards every http request that includes a invalidated nonce as a potential security breach. Depending on the security policy such a request may result in the termination of the authenticated session. Therefore, every usage of the web browser's "Back" or "Reload" buttons pose a problem because these buttons cause the web browser to reload pages with invalid nonces in their URLs. A web application using one-time URLs should be verbose about these restrictions and provide appropriate custom "Back" and "Reload" buttons as part of the application's GUI. It is also impossible to set bookmarks for URLs that lie in the randomized area of the web application, as the URL of such a bookmark would contain an invalid random nonce. Other issues, e.g. the opening of new browser windows, can be solved using DHTML techniques. Because of the described restrictions, a limitation on the usage of one-time URLs for only security sensitive parts of the web application may be recommendable.

Alternative solutions: Some of the limitations mentioned above exist because the proposed URLRandomizer object is implemented in JavaScript. As described above the separation of two different JavaScript objects running in the same security context is a complex and limited task. Especially the constraint that a random nonce can be used only once is due to the described problems. An alternative approach would be using a technology that can be separated cleanly from potential malicious JavaScript. There are two technologies that might be suitable candidates: Java applets [22] and Adobe Flash [1]. Both technologies have characteristics that suggest that they might be suitable for implementing the URL randomizing functionality: They provide a runtime in the web browser for client side code which is separated from the JavaScript runtime, they possess interfaces to the web browser's controls, they are able to export functionality to JavaScript routines and they are widely deployed in today's web browsers. Before implementing such an solution, the security properties of the two technologies have to be examined closely, especially in respect of the attacker's capability to include a malicious Java or Flash object in the attacked web page.

4.3 Subdomain Switching

The underlying fact which is exploited by the attacks described in Section 3.3 is, that webpages with the same origin implicitly trust each other. Because of this circumstance rogue iframes or background windows are capable of inserting malicious scripts in pages that would not be vulnerable otherwise. As years of security research have taught us, implicit trust is seldom a good idea - instead explicit trust should be the default policy. To remove this implicit trust between individual webpages that belong to the same web application, we have to ensure that no trust relationship between these pages induced by the same-origin policy exists: As long as the `document.domain` property for every page differs, background XSS propagation attacks are impossible.

To achieve this trust removal, we introduce additional subdomains to the web application. These subdomains are all mapped to the same server scripts. Every link included into a webpage directs to a URL with a subdomain that differs from the domain of the containing webpage. For example a webpage loaded from `http://s1.www.example.org` only contains links to `http://s2.www.example.org`. Links from `s2.www.example.org` would go to `s3.www`... and so on. As a result every single page possesses a different `document.domain` value. In cases where a page A explicitly wants to create a trust relationship to a second page B, pages A and B can change their `document.domain` setting to exclude the additional subdomain.

Tracking subdomain usage: As mentioned above, all added subdomains map to the same server scripts. Therefore, the URL `http://s01.www.example.org/order.ext` points to the same resource as for example the URL `http://s99.www.example.org/order.ext`. The subdomains have no semantic function; they are only used to undermine the implicit trust relationship. If a malicious script rewrites all URLs in a page to match the script's `document.domain` value, the web application will still function correctly and a background propagation attack will again be possible. For this reason, the web server has to keep track which mapping between URLs and subdomains have been assigned to a user's session.

Implementation aspects: The implementation of the subdomains is highly dependent on the application server used. For our implementation we used the Apache web server [16] which allows the usage of wildcards in the definition of subdomain names. Consequently, we had unlimited supply of applicable subdomain names. This allows the choice between random subdomain names or incrementing the subdomain identifier (`s0001.www` links to `s0002.www` which links to `s0003.www` and so on). On application servers that do not offer such an option and where therefore the number of available subdomain names is limited, the web application has to be examined closely. It has to be determined how many subdomains are required and how the mapping between URLs and subdomains should be implemented. These decisions are specific for each respective web application.

5 Discussion

5.1 Combination of the Methods

Before implementing the countermeasures described in Section 4, the web application's security requirements and environment limitations have to be examined. A combination of all three proposed methods provides complete protection against all known session hijacking attacks: The Deferred Loading Process prevents the unauthorized transmission of SID information. Subdomain Switching limits the impact of XSS vulnerabilities to only the vulnerable pages. Furthermore Browser Hijacking attacks that rely on the attacker's capability to access the content of the attack's http responses are also prevented as the XMLHttpRequest object is bound by the same origin policy: With Subdomain Switching in effect the attacking script would have to employ iframe or image inclusion to create the attack's http request. One-Time URLs prevent all Browser Hijacking attacks. Furthermore Session Riding [20] attacks would also be impossible as this attack class also relies on the attacker's prior knowledge of the application's URLs. It is strongly advisable to implement all three methods if possible. Otherwise, the targeted security advantage might be lost in most scenarios.

5.2 Limitations

As shown above, a combination of the countermeasures protect against the session hijacking attacks described in Section 3. However, on the actual vulnerable page in which the XSS code is included, the script still has some capabilities, e.g altering the page's appearance or redirecting form actions. Thus, especially webpages that include HTML forms should be inspected thoroughly for potential weaknesses even if the described techniques were implemented.

The described techniques are not meant to replace input checking and output sanitation completely. They rather provide an additional layer of protection to mitigate the consequences of occurring XSS vulnerabilities.

5.3 Transparent Implementation

An implementation of the proposed methods that is transparent to existing web applications is desirable. Such an implementation would allow to protect legacy applications without code changes.

Deferred Loading: There are no dependencies between the deferred loading process and the content of the application's webpages. Therefore, a transparent implementation of this method is feasible. It can be realized using an http proxy positioned before the server scripts: The proxy intercepts all incoming and outgoing http messages. Prior to transferring the request to the actual server scripts,

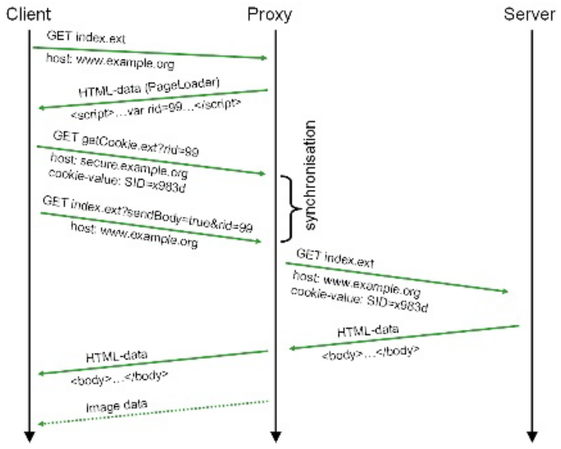


Fig. 3. Transparent implementation of the "get cookie" process

the "get cookie" process is executed (see figure 3). Before sending the http response to the client, all included cookies are stripped from the response and send to the client via the "set cookie" process.

One-Time URLs and Subdomain Switching: We only describe the problems in respect to One-Time URLs. Most of these issues also concern Subdomain Switching, while Subdomain Switching does not pose additional difficulties. A transparent implementation of these methods also would employ proxy like functionality. All incoming requests are examined whether their URLs are valid, i.e. contain a valid random nonce. All outgoing HTML data is modified to use the specified URLs. Implementing such a proxy is difficult because all application local URLs have to be rewritten for using the randomizer object. While standard HTML forms and hyperlinks pose no special challenge, prior existing JavaScript may be harder to deal with. All JavaScript functions that assign values to `document.location` or open new windows have to be located and modified. Also all existing `onclick` and `onsubmit` events have to be rewritten. Furthermore, HTML code might include external referenced JavaScript libraries, which have to be processed as well. Because of these problems, a web application that is protected by such a solution has to be examined and tested thoroughly. Therefore, an implementation of the proposed methods as a library for hyperlink and form creation is preferable.

5.4 Future Work

It still has to be specified how the proposed methods can be integrated into established frameworks and application servers. Such an integration is the prerequisite for examinations concerning performance issues and backwards compatibility. Recently we finished developing a transparent solution as described in Section 5.3 for the J2EE framework [24]. This implementation will be the basis for further investigations .

6 Related Work

The first line of defense against XSS attacks is input filtering. As long as JavaScript code is properly stripped from all user provided strings and special characters are correctly encoded, XSS attacks are impossible. However, implementing correct input filtering is a non-trivial task. For example, in 2005 an XSS input filter was added to the PHPNuke content management system that still was vulnerable against numerous known XSS attack vectors [3].

Scott and Sharp [21] describe an application level firewall which is positioned in front of the web server. The firewall's ruleset is defined in a specialized security policy description language. According to this ruleset incoming user data (via POST, GET and cookie values) are sanitized. Only requests to URLs for which policies have been defined are passed to the web server. The Sanctum AppShield Firewall is another server side proxy solution [13]. AppShield executes default filter operations on all user provided data in order to remove potential XSS attacks. Opposed to Scott and Sharp's approach, AppShield requires no application specific configuration which makes it easy to install but less powerful.

Kirda et al. proposed "Noxes", a client side personal firewall [12]. Noxes prevents XSS induced information leakage, e.g. stealing of cookie data, by selectively disallowing http requests to resources that do not belong to the web application's domain. The firewall's ruleset is a combination of automatically constructed and manually configured rules. Noxes does not offer protection against bowser hijacking attacks.

"Taint analysis" is a method for data flow tracking in web applications. All user controlled data is marked as "tainted". Only if the data passes sanitizing functions its status will change to "untainted". If a web application tries to include tainted data into a webpage a warning will be generated. Taint analysis was first introduced by Perl's taint mode [15]. In 2005 Huang et al. presented with WEBSSARI a tool that provides static taint analysis for PHP [10].

Microsoft introduced an "http only" option for cookies with their web browser Internet Explorer 6 SP1 [17]. Cookies that are set with this option are not accessible by JavaScript and therefore safe against XSS attacks. The http only option is not standardized and until now there are no plans to do so. It is therefore uncertain if and when other web browsers will implement support for this option.

7 Conclusion

In this paper we presented SessionSafe, a combination of three methods that successfully prohibits all currently known XSS session hijacking attacks.

To achieve this, we classified currently known methods for session hijacking. Through a systematic examination of the resulting attack classes, we identified the basic requirements for each of these attack methodologies: SID accessibility in the case of Session ID Theft, prior knowledge of URLs in the case of Browser Hijacking and implicit trust between webpages in the case of Background XSS Propagation.

There are only two instruments provided by the web browser architecture that can be used to enforce access restrictions in connection with JavaScript: The same-origin policy and private members in JavaScript objects. Using the knowledge gained by the classification, we were able to apply these security mechanisms to remove the attack classes' foundations: To undermine the SID accessibility, the SID is kept in a cookie which belongs to a different subdomain than the main web application. To achieve this, we developed a deferred loading process which allows to execute the cookie transport while the web page is still in a trustworthy state. To undermine the pre-knowledge of the application's URLs, valid One-Time URLs are hidden inside private members of the URL-Randomizer JavaScript object. Finally, additional subdomains are introduced by Subdomain Switching, in order to create a separate security domain for every single webpage. This measure employs the same-origin policy to limit the impact of XSS attacks to the vulnerable pages only. Consequently, each proposed countermeasure removes the fundamental necessities of one of the attack classes, hence disabling it reliably. By preventing session hijacking, a large slice of the attack surface of XSS can be removed.

The proposed countermeasures do not pose limitations on the development of web applications and only moderate restrictions on web GUI functionality. They can be implemented as an integral component of the application server and thus easily be integrated in the development or deployment process.

References

1. Adobe flash. Website <http://www.adobe.com/products/flash/flashpro/>.
2. Wade Alcorn. The cross-site scripting virus. Whitepaper, <http://www.bindshell.net/papers/xssv/xssv.html>, September 2005.
3. Maksymilian Arciemowicz. Bypass xss filter in phpnuke 7.9. mailing list BugTraq, <http://www.securityfocus.com/archive/1/419496/30/0/threaded>, December 2005.
4. CERT/CC. Cert® advisory ca-2000-02 malicious html tags embedded in client web requests. [online], <http://www.cert.org/advisories/CA-2000-02.html> (01/30/06), February 2000.
5. Douglas Crockford. Private members in javascript. website, <http://www.crockford.com/javascript/private.html>, last visit 01/11/06, 2001.
6. ECMA. Ecmascript language specification, 3rd edition. Standard ECMA-262, <http://www.ecma-international.org/publications/standards/Ecma-262.htm>, December 1999.
7. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, <http://www.w3.org/Protocols/rfc2616/rfc2616.html>, June 1999.
8. David Flanagan. *JavaScript: The Definitive Guide*. O'Reilly, 4th edition, November 2001.
9. Jeremiah Grossman. Phishing with super bait. Presentation at the Black Hat Asia 2005 Conference, <http://www.blackhat.com/presentations/bh-jp-05/bh-jp-05-grossman.pdf>, October 2005.

10. Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, Der-Tsai Lee, and Sy-Yen Kuo. Securing web application code by static analysis and runtime protection. In *Proceedings of the 13th conference on World Wide Web*, pages 40–52. ACM Press, 2004.
11. Philippe Le Hégaret, Ray Whitmer, and Lauren Wood. Document object model (dom). W3C recommendation, <http://www.w3.org/DOM/>, January 2005.
12. Engin Kirda, Christopher Kruegel, Giovanni Vigna, and Nenad Jovanovic. Noxes: A client-side solution for mitigating cross site scripting attacks, security. In *Security Track of the 21st ACM Symposium on Applied Computing (SAC 2006)*, April 2006.
13. Amit Klein. Cross site scripting explained. White Paper, Sanctum Security Group, <http://crypto.stanford.edu/cs155/CSS.pdf>, June 2002.
14. D. Kristol and L. Montulli. Http state management mechanism. RFC 2965, <http://www.ietf.org/rfc/rfc2965.txt>, October 2000.
15. LarryWall, Tom Christiansen, and Jon Orwant. *Programming Perl.* O'Reilly, 3rd edition, July 2000.
16. Ben Laurie and Peter Laurie. *Apache: The Definitive Guide.* O'Reilly, 3rd edition, December 2002.
17. MSDN. Mitigating cross-site scripting with http-only cookies. website, <http://msdn.microsoft.com/workshop/author/dhtml/httponly_cookies.asp>, last visit 01/23/06.
18. Anton Rager. Xss-proxy. website, <http://xss-proxy.sourceforge.net>, last visit 01/30/06, July 2005.
19. Samy. Technical explanation of the myspace worm. website, <http://namb.la/popular/tech.html>, last visit 01/10/06, October 2005.
20. Thomas Schreiber. Session riding - a widespread vulnerability in today's web applications. Whitepaper, SecureNet GmbH, <http://www.securenet.de/papers/Session_Riding.pdf>, December 2004.
21. D. Scott and R. Sharp. Abstracting application-level web security. In *Proceedings of 11th ACM International World Wide Web Conference*, pages 396 – 407. ACM Press New York, NY, USA, 2002.
22. Sun. Java. Website <http://java.sun.com/>.
23. L. von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *Proceedings of Eurocrypt*, pages 294–311, 2003.
24. Christian Weitendorf. Implementierung von maßnahmen zur sicherung des web-session-managements im j2ee-framework. Master's thesis, University of Hamburg, 2006.

Policy-Driven Memory Protection for Reconfigurable Hardware

Ted Huffmire, Shreyas Prasad, Tim Sherwood, and Ryan Kastner

University of California, Santa Barbara
Santa Barbara CA 93106, USA
{huffmire,sherwood}@cs.ucsb.edu
{shreyas,kastner}@ece.ucsb.edu
http://www.cs.ucsb.edu/~arch

Abstract. While processor based systems often enforce memory protection to prevent the unintended sharing of data between processes, current systems built around reconfigurable hardware typically offer no such protection. Several reconfigurable cores are often integrated onto a single chip where they share external resources such as memory. While this enables small form factor and low cost designs, it opens up the opportunity for modules to intercept or even interfere with the operation of one another. We investigate the design and synthesis of a memory protection mechanism capable of enforcing policies expressed as a formal language. Our approach includes a specialized compiler that translates a policy of legal sharing to reconfigurable logic blocks which can be directly transferred to an FPGA. The efficiency of our access language design flow is evaluated in terms of area and cycle time across a variety of security scenarios.

Keywords: Computer Security, Embedded Systems, Reference Monitors, Separation Kernels, Security Policies, Policy Languages.

1 Introduction

Reconfigurable hardware is at the heart of many high performance embedded systems. Satellites, set-top boxes, electrical power grids, and the Mars Rover all rely on Field Programmable Gate Arrays (FPGAs) to perform their respective functions. The bit-level reconfigurability of these devices can be used to implement highly optimized circuits for everything from encryption to FFT, or even entire customized processors. Because one device is used for so many different functions, special-purpose circuits can be developed and deployed at a fraction of the cost associated with custom fabrication. Furthermore, if the design needs to be updated, the logic on an FPGA board can even be changed in the field. These advantages of reconfigurable devices have resulted in their proliferation into critical systems, yet many of the security primitives which software designers take for granted are simply nonexistent.

Due to Moore's law, digital systems today have enough transistors on a single chip to implement over 200 separate RISC processors. Increased levels of integration are inevitable, and reconfigurable systems are no different. Current reconfigurable systems-on-chip include diverse elements such as specialized multiplier units, integrated memory tiles, multiple fully programmable processor cores, and a sea of reconfigurable gates capable of implementing significant ASIC or custom data-path functionality. The complexity of these systems and the lack of separation between different hardware modules has increased the possibility that security vulnerabilities may surface in one or more components, which could threaten the entire device. New methods that can provide separation and security in highly integrated reconfigurable devices are needed.

One of the most critical aspects of separation that needs to be addressed is in the management of external resources such as off-chip DRAM. While a processor will typically use virtual memory and TLBs to enforce some form of memory protection, reconfigurable devices usually operate in the physical addresses space with no operating system support. Lacking these mechanisms, any hardware module can read or write to the memory of any other module at any time, whether purposefully, accidentally, or maliciously. This situation calls for a *memory access policy* that all modules on chip must obey. In this paper we present a method that utilizes the reconfigurable nature of field programmable devices to provide a mechanism to enforce such a policy.

In the context of this paper, a *memory access policy* is a formal description that establishes what accesses to memory are legal and which are not. Our method rests on the ability to formally describe the access policy using a specialized language. We present a set of tools through which the policy description can be automatically transformed and *directly synthesized to a circuit.* This circuit, represented as a bit-stream, can then be loaded into a reconfigurable hardware module and used as an execution monitor to analyze memory accesses and enforce the policy.

The techniques presented in this paper are steps towards a cohesive methodology for those seeking to build reconfigurable systems with modules acting at different security clearance levels on a single chip. In order for such a methodology to be adopted by the embedded design community it is critical that the resulting hardware is both high performance and low overhead. Furthermore, it is important that our methods are both formally grounded and yet understandable to those outside the security discipline. Throughout this paper we strive to strike a balance between engineering and formal evaluation. Specifically, this paper makes the following contributions:

– We specify a memory access policy language, based on formal regular languages, which expresses the set of legal accesses and allowed policy transitions.
– We demonstrate how our language can express classical security scenarios, such as compartmentalization, secure hand-offs, Chinese walls, access control lists and an example of redaction.
– We present a policy compiler that translates an access policy described in this language into a synthesizable hardware module.

– We evaluate the effectiveness and efficiency of this novel enforcement mechanism by synthesizing several policies down to a modern FPGA and analyzing the area and performance.

2 Reconfigurable Systems

Increasingly we are seeing reconfigurable devices emerge as the flexible and high-performance workhorses inside a variety of high performance embedded computing systems [4,6,8,15,20,29]. The power of reconfigurable systems lies in the immense amount of flexibility that is provided. Designs can be customized down to the level of individual bits and logic gates. They combine the post-fabrication programmability of software running on a general purpose processor with the spatial computational style most commonly employed in hardware designs [8]. Reconfigurable systems use programmability and regularity to create a flexible computing fabric that can lower design costs, reduce system complexity, and decrease time to market, while achieving 100x performance gain per unit silicon as compared to a similar microprocessor [5,7,33]. The growing popularity of reconfigurable logic has forced practitioners to start to consider the security implications, yet the resource constrained nature of embedded systems is a challenge to providing a high level of security [16]. To provide a security technique that can be used in practice, it must be both robust and efficient.

Protecting Memory on an FPGA. A successful run-time management system must protect different logical modules from interfering, intercepting, or corrupting any use of a shared resource. On an embedded system, the primary resource of concern is memory. Whether it is on-chip block RAM, off-chip DRAM, or backing-store such as Flash, a serious issue in the design of any high performance secure system is the allocation and reallocation of memory in a way that is efficient, flexible, and protected. On a high performance processor, security domains may be enforced through the use of a page table. Superpages, which are very large memory pages, can also be used to provide memory protection, and their large size makes it possible for the TLB to have a lower miss rate [22]. Segmented Memory [27] and Mondrian Memory Protection [35], a finer-grained scheme, address the inefficiency of providing memory protection at the granularity of a page (or a superpage) by allowing different protection domains to have different permissions on the same memory region.

While a TLB may be used to speed up page table accesses, this requires additional associative memory (not available on FPGAs) and greatly decreases the performance of the system in the worst case. Therefore, few embedded processors and even fewer reconfigurable devices support even this most basic method of protection. Instead, reconfigurable architectures on the market today support a simple linear addressing scheme that exactly mirrors the physical memory. **Hence, on a modern FPGA the memory is essentially flat and unprotected.**

Preventing unauthorized accesses to memory is fundamental to both effective debugging and computer security. Even if the system is not under attack, many of the most insidious bugs are a result of errant memory accesses which affect multiple sub-systems. Ensuring protection and separation of memory when multiple concurrent logic modules are active requires a new mechanism to ensure that the security properties of the system are enforced.

To provide separation in memory between multiple different interacting modules, we adapt some of the key concepts from separation kernels. Rushby originally proposed that a separation kernel [12] [18] [24] [25] creates within a single shared machine an environment which supports the various components of the system, and it provides the communication channels between them in such a way that individual components of the system cannot distinguish this shared environment from a physically distributed one. A separation kernel divides all resources under its control into blocks such that the actions of a subject in one block are isolated from (viz., cannot be detected by or communicated to) a subject in another block, unless an explicit means for that communication has been established. For a multilevel secure system, each block typically represents a different classification level. Unfortunately, separation kernels have high overhead and complexity due to the need to implement software virtualization, and the design complexity of modern out-of-order CPUs makes it difficult to implement separation kernels with a verifiable level of trust. A solution is needed that is located somewhere along a continuum between the two extremes of physical separation and software separation in order to have the best of both worlds.

We propose that the reconfigurable nature of FPGAs offers a new method by which the fine grain control of access to off-chip memory is possible. By building a specialized circuit that recognizes a *language of legal accesses*, and then by realizing that circuit directly onto the reconfigurable device as a specialized state machine, every memory access can be checked with only a small additional latency. Although incorporating the enforcement module into a separate hardware module would lessen the impact of covert channel attacks, this would introduce additional latency. We describe techniques we are working on to isolate the enforcement module in Section 5.2.

3 Policy Description and Synthesis

While reconfigurable systems typically do not have traditional memory protection enforcement mechanisms, the programmable nature of the devices means that we can build whatever mechanisms we need as long as they can be implemented efficiently. In fact, we exploit the fine grain re-programmability of FPGAs to provide word-level stateful memory protection by implementing a compiler that can translate a memory access policy directly into a circuit. The enforcement mechanisms generated by our compiler will help prevent a corrupted module or processor from compromising other modules on the FPGA with which it shares memory.

We begin with an explanation of our memory access policies, and we describe how a policy can be expressed and then compiled down to a synthesizable module. In this section we explain both the high level policy description and the automated sequence of steps, or *design flow*, for converting a memory access policy into a hardware enforcement module.

3.1 Memory Access Policy

Once a high level policy is developed based on the requirements of the system and the organizational security policy [32], it must be expressed in a concrete form to allow engineers to build enforcement mechanisms. In the context of this paper we concentrate on policies as they relate to memory accesses. In particular, the enforcement mechanisms we consider in this paper belong to the Execution Monitoring (EM) class [30], which monitor the execution of a *target*, which in our case is one or more modules on the FPGA. An execution monitor must be able to monitor all memory accesses and able to halt or block the execution of the target if it attempts to violate the security policy. Allowing a misbehaving module to continue executing might let it use the state of the enforcement mechanism as a covert channel. In addition, all modules must be isolated from the enforcement mechanism so that they cannot interfere with the DFA transitions. We discuss techniques for module isolation in Section 5.2. The enforcement mechanism is also a Reference Validation Mechanism (RVM) [3]. Although Erlingsson et al. have proposed the idea of merging the reference monitor in-line with the target system [9], in a system with multiple interacting cores, this approach has the drawback that the reference monitors are distributed, which is problematic for stateful policies. Although there exist security policies that execution monitors are incapable of enforcing, such as *information flow* policies [26], we argue that in the future our execution monitors could be combined with static analysis techniques to enforce a more broad range of policies if required. We therefore begin by describing a well defined method for describing memory access policies.

The goal of our memory access policy description is to precisely describe the set of legal memory access patterns, specifically those that can be recognized by an execution monitor capable of tracking address ranges of arbitrary size. Furthermore, it should be possible to describe complex behaviors such as sharing, exclusivity, and atomicity, in an understandable fashion. An engineer can then write a policy description in our input form (as a series of productions) and have it transformed automatically to an extended type of regular expression. By extending regular languages to fit our needs we can have a human-readable input format, and we can build off of theoretical contributions which have created a path to state machines and hardware [1].

There are three pieces of information that we will incorporate into our execution monitor. The *Accessing Modules* (M) are the unique identifiers for a specific principal on the chip, such as a specific intellectual property core or one of the on-chip processors. Throughout this paper we simply refer to these units of separation of the FPGA as Modules. The *Access Methods* (A) are typically Read and Write, but may include special memory operators such as zeroing or incrementing if required. The set P is a partitioning of physical memory into ranges. The *Memory Range Specifier* (R in P) describes a physical address or set of physical addresses to which a specific permission can be assigned. Our language describes an access policy through a sequence of *productions*, which specify the relationship between principals (M: modules), access rights (A: read, write, etc.), and objects (R: memory ranges[1]).

The terminals of the language are *memory accesses descriptors* which ascribe a specific right to a specific module for a specific object for the duration of the next memory access. Formally, the terminals of the productions are tuples of the form (M, A, R), and the universe of tuples forms an alphabet $\Sigma = M \times A \times R$. The memory access policy description precisely defines a formal language $L \subseteq \Sigma*$ which is almost always infinite (unless the device only supports a fixed number of accesses). L needs to satisfy the property that $\forall xt \mid t \in \Sigma, xt \in L : x \in L$. This has the effect that any legal access pattern will be incrementally recognized as legal along the way.

One thing to note is that memory accesses refer to a specific memory address, while memory access descriptors are defined over the set of all memory ranges R. A memory access (M, A, k), where k is a particular address, is *contained* in a memory access descriptor (M', A', R) iff $M = M', A = A'$, and $R_{low} \leq k \leq R_{high}$. A sequence of memory accesses $a = a_0, a_1, ..., a_n$ is said to be legal iff $\exists s = s_0, s_1, ..., s_n \in L$ s.t. $\forall_{0 \leq i \leq n}\ s_i$ contains a_i. In order to turn this into an enforceable method we need two things.

1. A method by which L can be precisely defined
2. An automatically created circuit which recognizes memory access sequences that are legal under L

We begin with a description of the first item through the use of a simple example. Consider a very straightforward compartmentalization policy. $Module_1$ is only allowed to access memory in the range of [0x8e7b008,0x8e7b00f], and $Module_2$ is only allowed to access memory in the range of [0x8e7b018,0x8e7b01b]. Figure 2 shows this memory access policy expressed as a set of productions.

Each of these productions is a re-writing rule as in a standard grammar. The non-terminal *Policy* is the start symbol of the grammar and defines the overall access policy. Note that *Policy* is essentially a regular expression that describes L. Through the use of a grammar we allow the hierarchical composition of more complex policies. In this case $Access_1$ and $Access_2$ are simple access descriptors, but in general they could be more complex expressions that recognize a set of legal memory access.

Since we eventually want to compile the access policy to hardware, we limit our language to constructs with computational power no greater than a regular expression [19] with the added ability to detect ranges. Although a regular language must have a type-3 grammar in the Chomsky hierarchy, it is inconvenient for security administrators to express policies in right-linear or left-linear form. Since a language can be recognized by *many* grammars, any grammar that

[1] An interval of the address space including high (R_{high}) and low (R_{low}) bounds.

can be transformed into type-3 form is acceptable. This transformation can be accomplished by extracting first terminals from non-terminals.

Note that the atomic unit of enforcement is an address range, and that the ranges are of arbitrary size. The smallest granularity that we enforce currently is at the word boundary, and we can support any sized range from a single word to the entire address space. There is no reason that ranges have to be of the same size or even close, unlike pages. We will later show how this ability can be used to set up special *control words* that help in securely coordinating between modules.

Although we are restricted to policies that are equivalent to a finite automata with range checking, we have constructed many example policies including compartmentalization and Chinese wall in order to demonstrate the versatility and efficiency of our approach. In Section 4.4 we describe a "redaction policy," in which modules with multiple security clearance levels are interacting within a single embedded system. However, now that we have introduced our memory access policy specification language, we describe how it can be transformed automatically to an efficient circuit for implementation on an FPGA.

3.2 Hardware Synthesis

We have developed a policy compiler that converts an access policy, as described above, into a circuit that can be loaded onto an FPGA to serve as the enforcement module. At a high level the technique partitions the module into two parts, range discovery and language recognition.

3.3 Design Flow Details

Access Policy – To describe the process of transforming a policy to a circuit, we consider a simple compartmentalization policy with two modules, which can only access their own single range:

$Access \rightarrow \{Module_1, rw, Range_1\} \mid \{Module_2, rw, Range_2\};$
$Policy \rightarrow (Access)*;$

Building and Transforming a Parse Tree – Next, we use Lex [17] and Yacc [14] to build a parse tree from our security policy. Internal nodes represent operators such as concatenation, alternation, and repetition. Figure 1 shows the parse tree for our example policy.

We must then transform the parse tree into a large single production with no non-terminals on the right hand side, from which we can generate a regular expression. This process of macro expansion requires an iterative replacement of all the non-terminals in the policy. We apply the productions to the parse tree by substituting the left hand side of each production with its right hand side. Figure 1 shows the transformed parse tree for our policy.

Building the Regular Expression – Next, we find the subtree corresponding to *Policy* and traverse this subtree to obtain the regular expression. By this stage

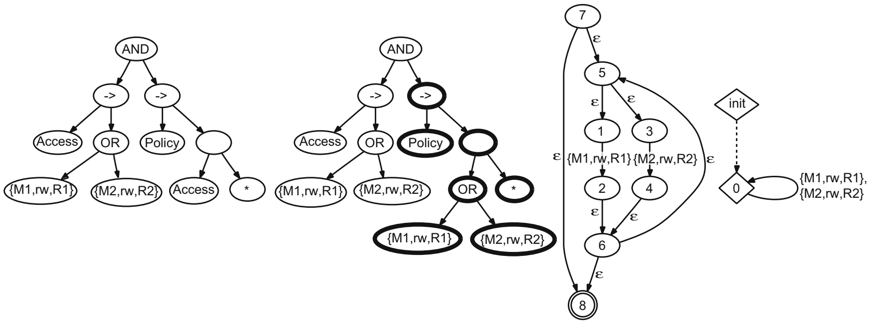


Fig. 1. Our policy compiler translates the policy to a regular expression by building, transforming, and traversing a parse tree. From the regular expression, an NFA is constructed, which is then converted into a minimized DFA.

we have completely eliminated all of the non-terminals, and we are left with a single regular expression which can then be converted to an NFA. The regular expression for our access policy is: $((\{Module_1, rw, Range_1\}) \,|\, (\{Module_2, rw, Range_2\}))*$.

Constructing the NFA – Once the regular expression has been formed, an NFA can be constructed from this regular expression using Thompson's Algorithm [1]. Figure 1 shows the NFA for our policy.

Converting the NFA to a DFA – From this NFA we can construct a DFA through subset construction [1]. Following the creation of the DFA, we apply Hopcroft's Partitioning Algorithm [1] as implemented by Grail [23] to minimize the DFA. Figure 1 shows the minimized DFA for our policy on the right.

Processing the Ranges – Before we can convert the DFA into Verilog, we must perform some processing on the ranges so that the circuit can efficiently determine which range contains a given address. Our system converts the ranges to an internal format using don't care bits. For example, 10XX can be 1000, 1001, 1010, or 1011, which is the range [8,11]. Hardware can be easily synthesized to check if an address is within a particular range by performing a bit-wise XOR on just the significant bits.[2] Using this optimization, any aligned power of two range can be efficiently described, and any non-power of two range can be converted into a covering set of $O(\log_2 |range|)$ power of two ranges. For example the range [7,12] (0111, 1000, 1001, 1010, 1011, 1100) is not an aligned power of two range but can be converted to a set of aligned power of two ranges: {[7,7],[8,11],[12,12]} (or equivalently {0111|10XX|1100}).

Converting the DFA to Verilog – Because state machines are a very common hardware primitive, there are well-established methods of translating a descrip-

[2] This is equivalent to performing a bit-wise XOR, masking the lower bits, and testing for non-zero except that in hardware the masking is unnecessary.

tion of state transitions into a hardware description language such as Verilog. Figure 3 shows the hardware module we wish to build. There are three inputs: the module ID, the op {read, write, etc.}, and the address. The output is a single bit: 1 for grant and 0 for deny. The DFA transitions are the concatenation of the module ID, op, and a range ID bit vector. The range ID bit vector contains one bit for each range ID in the policy. **The hardware will check all the ranges in parallel and set to 1 the bit corresponding to the range ID that contains the input address.** If there is no transition for an input character, the machine always transitions to the rejecting state, which is a "dummy" sink state. This is important for security because an attacker might try to insert illegal characters into the input.

State Machine Synthesis. The final step in the design flow is the actual conversion of Verilog code to a bit-stream that can be loaded onto an FPGA. Using the Quartus tools from Altera, which does synthesis, optimization, and place-and-route, we turn each machine into an actual implementation. After testing the circuit to verify that it accepts a sample of valid accesses and rejects invalid accesses, we are ready to measure the area and cycle time of our design.

4 Example Applications

To further demonstrate the usefulness of our language, we use it to express several different policies. We have already demonstrated how to compartmentalize access to different modules, and it is trivial to extend the above policy to include overlapping ranges, shared regions, and most any static policy. The true

Compartmentalization:

$rw \longrightarrow r \mid w;$
$Range_1 \longrightarrow [0x8e7b008, 0x8e7b00f];$
$Range_2 \longrightarrow [0x8e7b018, 0x8e7b01b];$
$Access_1 \longrightarrow \{Module_1, rw, Range_1\};$
$Access_2 \longrightarrow \{Module_2, rw, Range_2\};$
$Policy \longrightarrow (Access_1 \mid Access_2)^*;$

Access Control:

$Class_1 \longrightarrow Module_1 \mid Module_2;$
$Class_2 \longrightarrow Module_3 \mid Module_4;$
$List_1 \longrightarrow Class_1 \mid Class_2;$
$List_2 \longrightarrow Class_2;$
$Access_1 \longrightarrow \{List_1, rw, Range_1\};$
$Access_2 \longrightarrow \{List_2, rw, Range_2\};$
$Policy \longrightarrow (Access_1 \mid Access_2)^*;$

Secure Hand-Off:

$Module_{1|2} \longrightarrow Module_1 \mid Module_2;$
$Access_1 \longrightarrow \{Module_1, rw, Range_1\} \mid \{Module_{1|2}, rw, Range_2\};$
$Access_2 \longrightarrow \{Module_2, rw, (Range_1 \mid Range_2)\};$
$Trigger \longrightarrow \{Module_1, rw, Range_2\};$
$Policy \longrightarrow (Access_1^*)\ (\in \mid Trigger\ (Access_2)^*);$

Chinese Wall:

$Access_1 \longrightarrow \{Module_1, rw, (Range_1 \mid Range_3)\}^*;$
$Access_2 \longrightarrow \{Module_1, rw, (Range_1 \mid Range_4)\}^*;$
$Access_3 \longrightarrow \{Module_1, rw, (Range_2 \mid Range_3)\}^*;$
$Access_4 \longrightarrow \{Module_1, rw, (Range_2 \mid Range_4)\}^*;$
$Policy \longrightarrow Access_1 \mid Access_2 \mid Access_3 \mid Access_4;$

Redaction:

$rw \longrightarrow r \mid w;$
$Access_2 \longrightarrow \{Module_1, rw, Range_1\} \mid \{Module_1, r, Range_3\} \mid \{Module_2, rw, Range_2\} \mid \{Module_2, w, Range_4\} \mid \{Module_3, rw, Range_3\};$
$Access_1 \longrightarrow \{Module_2, r, Range_3\} \mid Access_2;$
$Trigger \longrightarrow \{Module_1, w, Range_4\};$
$Clear \longrightarrow \{Module_3, z, Range_3\};$
$SteadyState \longrightarrow (Access_2 \mid Clear\ Access_1^*\ Trigger)^*;$
$Policy \longrightarrow \in \mid Access_1^* \mid Access_1^*\ Trigger\ SteadyState \mid Access_1^*\ Trigger\ SteadyState\ Clear\ Access_1^*;$

Fig. 2. Several example policies expressed in our language

power of our system comes from the description of *stateful* policies that involve revocation or conditional access. In particular we demonstrate how data may be securely handed off between modules, and we also show the Chinese wall policy. Before we do that let us first discuss another more traditional example: access control lists.

4.1 Access Control List

A secure system that employs access control lists will associate every object in the system with a list of principals along with the rights of each principal to access the object. For example, suppose our system has two objects, $Range_1$ and $Range_2$. $Class_1$ is a class of principals ($Module_1$ and $Module_2$), and $Class_2$ is another class of principals ($Module_3$ and $Module_4$). Either $Class_1$ or $Class_2$ may access $Range_1$, but only $Class_2$ may access $Range_2$. Figure 2 shows this policy.

In general, since access control list policies are stateless, the resulting DFA will have one state, and the number of transitions will be the sum of the number of principals that may access each object. In this example, $Module_1$, $Module_2$, $Module_3$, and $Module_4$ may access $Range_1$, and $Module_3$ and $Module_4$ may access $Range_2$. The total number of transitions in this example is 4+2=6.

4.2 Secure Hand-Off

Many protocols require the ability to securely hand-off information from one party to another. Embedded systems often implement these protocols, and our language makes these transfers possible. Rather than requiring large communication buffers or multiple copies of the data, we can simply transfer the control of a specified range of data from one module to the next. For example, suppose $Module_1$ wants to securely hand-off some data to $Module_2$. $Module_1$ writes some data to memory, to which it must have exclusive access, and then $Module_2$ reads the data from memory. Rather than communicating the data, an access policy can be compiled that will allow the critical transition of permissions in synchronization with the hand-off. Using formal languages to express security policies makes such a temporal hand-off possible.

After a certain trigger event occurs, it is possible to revoke the permissions of a module so that it may no longer access one or more ranges. Consider the example policy in Figure 2. At first, $Module_1$ can access $Range_1$ or $Range_2$, and $Module_2$ can access $Range_2$. However, the first time $Module_1$ accesses $Range_2$ (indicating that $Module_1$ is ready to hand off), $Access_1$ is deactivated by this trigger event, revoking the permissions for $Module_1$ from both Ranges. As a result of the trigger, $Module_2$ has exclusive access to $Range_1$ and $Range_2$.

4.3 Chinese Wall

Another security scenario that can be efficiently expressed using a policy language is the Chinese wall. Consider an example of this scenario, in which a lawyer who looks at the set of documents of $Company_1$ should not view the set of files of $Company_2$ if $Company_1$ and $Company_2$ are in the same conflict-of-interest class. This lawyer may also view the files of $Company_3$ provided that $Company_3$ belongs to a different conflict-of-interest class than $Company_1$ and $Company_2$. Figure 2 shows a Chinese wall security policy expressed in our language. There are two conflict-of-interest classes. One contains $Range_1$ and $Range_2$, and the other contains $Range_3$ and $Range_4$. For simplicity, we have restricted this policy to one module.

In general, for Chinese wall security policies, the number of states scales exponentially in the number of conflict-of-interest classes. This occurs because the number of possible legal accesses is the product of the number of sets in each conflict-of-interest class. The number of transitions also scales exponentially in the number of classes for the same reason. Fortunately, the number of states scales linearly in both the number of sets and the number of modules. Even better, the number of states is not affected by the number of ranges. The number of transitions scales linearly in the number of sets, ranges, and modules.

4.4 Redaction

Our security language can also be used to enforce instances of redaction [28], even at very high throughputs (such as for video). Military hardware such as avionics [34] may contain components with different clearance levels, and a component with a top secret clearance must not leak sensitive information to a component with a lower clearance [31]. Figure 5 shows the architecture of a redaction scenario that is based on separation. A multilevel database contains both Top Secret and Unclassified data. $Module_1$ has a top secret (TS) clearance, but $Module_2$ has an unclassified (U) clearance. $Module_1$ and $Module_2$ are initially compartmentalized, since they have different clearance levels. Therefore, $Range_1$ belongs to $Module_1$, and $Range_2$ belongs to $Module_2$. $Module_3$ acts as a trusted server of information contained in the database, and this server must have a security label range from U to TS. $Range_3$ is temporary storage used for holding information that has just been retrieved from the database by the trusted server. $Range_4$ (the control word) is used for performing database queries: a module writes to $Range_4$ to request that $Module_3$ retrieve some information from the database and then write the query result to temporary storage. Any database query performed by $Module_2$ must have all TS data redacted by the trusted server. If a request is made by $Module_1$ for top secret information, it is necessary to revoke $Module_2$'s read access to the temporary storage, and this access must not be reinstated until the trusted server zeroes out the sensitive information contained in temporary storage. Figure 2 shows our redaction policy, and Figure 4 shows the DFA that recognizes this policy. State 1 corresponds to a less restrictive mode ($Access_1$), and State 0 corresponds to a more restrictive mode ($Access_2$). The Trigger event causes the state machine to transition from State 1 to State 0, and the Clear event causes the machine to transition from State 0 to State 1. In general, the DFA for a redaction policy will have one state for each access mode. For example, if we have three different modules that each have a different clearance level, there will be three access modes and three states.

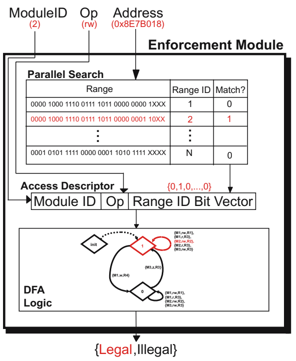

ModuleID
(2)
Op
(rw)
Address
(0x8E7B018)
Enforcement Module
Parallel Search
Range
Range ID
Match?
0000 1000 1110 0111 1011 0000 0000 1XXX
1
0
0000 1000 1110 0111 1011 0000 0001 10XX
2
1
0001 0101 1111 0000 0001 1010 1111 XXXX
N
0
Access Descriptor
{0,1,0,...,0}
Module ID
Op
Range ID Bit Vector
DFA Logic
init
1
0
{M1,rw,R1}, {M1,r,R3}, {M2,rw,R2}, {M2,r,R3}, {M3,rw,R3}
{M3,z,R3}
{M1,w,R4}
{M1,rw,R1}, {M1,r,R3}, {M2,rw,R2}, {M3,rw,R3}
{Legal,Illegal}

Fig. 3. A hardware reference monitor

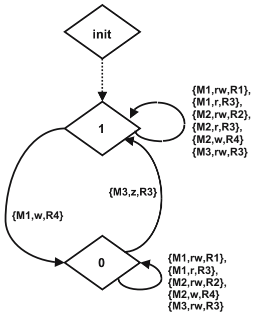

init
1
{M1,rw,R1}, {M1,r,R3}, {M2,rw,R2}, {M2,r,R3}, {M2,w,R4} {M3,rw,R3}
{M3,z,R3}
{M1,w,R4}
0
{M1,rw,R1}, {M1,r,R3}, {M2,rw,R2}, {M2,w,R4} {M3,rw,R3}

Fig. 4. DFA recognizing legal behavior for a redaction policy

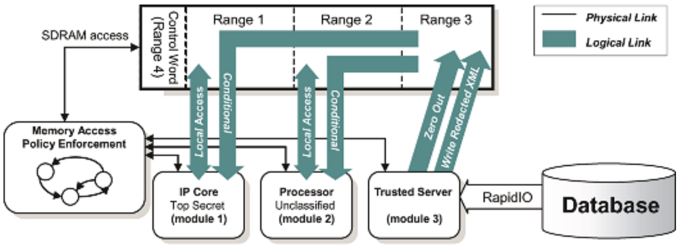


Fig. 5. A redaction architecture. IP stands for Intellectual Property.

5 Integration and Evaluation

Now that we have described several different memory access policies that could be enforced using a stateful monitor, we need to demonstrate that such systems could be efficiently realized on reconfigurable hardware.

5.1 Enforcement Architecture

The placement of the enforcement mechanism can have a significant impact on the performance of the memory system. Figure 6 shows two architectures for the enforcement mechanism which assumes that modules on the FPGA can only access shared memory via the bus. In the figure on the left, the enforcement mechanism (E) sits between the memory and the bus, which means that every access must pass through the enforcement mechanism before going to memory. In the case of a read, the request cannot proceed to memory until the enforcement mechanism approves the access. This results in a large delay which is the sum of the time to determine the legality of the access and the memory latency. We can mitigate this problem by having the enforcement mechanism snoop on the bus or through the use of various caching mechanisms for keeping track of accesses that have already been approved. This scenario is shown in the figure on the right. In the case of a read, the request is sent to memory, and the memory access occurs in parallel with the task of determining the legality of the read. A buffer (B) holds the data until the enforcement mechanism grants approval, at which time the data is sent across the bus. In the case of a write, the data to be written is stored in the buffer until the enforcement mechanism grants approval, at which time the write request is sent to memory. Thus, both architectures provide to the enforcement mechanism the isolation and omnipotence required of a reference or execution monitor.

Since a module may be sending sensitive data over the bus, it is necessary to prevent other modules from accessing the bus at the same time. We address this problem by placing an arbiter between each module and the bus. In a system with two modules, the arbiters will allow one module to access the bus on even clock cycles and the other module to access the bus on odd clock cycles.

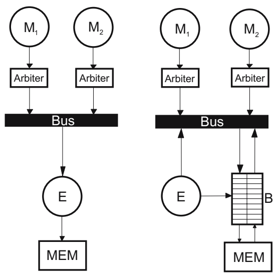


Fig. 6. Two alternative architectures for the enforcement mechanism

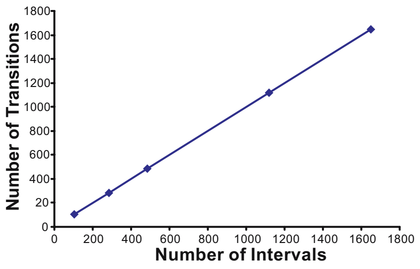


Fig. 7. DFA Transitions versus number of ranges for compartmentalization

5.2 Isolation of the Reference Monitor

It is critical that the reference module be isolated from other modules on the FPGA. Ensuring the physical separation of the modules entails distributing the computation spatially. We are working on methods to ensure that modules are placed in separate spatial areas and that there are no extraneous connections between the modules. Although we are working on addressing this problem by developing techniques that work at the gate level, this work is beyond the scope of this paper. In our attack model, there may be malicious modules or remote attacks that originate from the network, but we assume that the attacker does not have physical access to the device.

5.3 Evaluation

Of the different policies we discussed in Section 4, we focus primarily on characterizing compartmentalization as this isolates the effect of range detection on system efficiency. Rather than tying our results to the particular reconfigurable system prototype we are developing, we quantify the results of our design flow on a randomly generated set of ranges over which we enforce compartmentalization. The range matching constitutes the majority of the hardware complexity (assuming there are a large number of ranges), and there has already been a great deal of work in the CAD community on efficient state machine synthesis [21].

To obtain data detailing the timing and resource usage of our range matching state machines, we ran the memory access policy description through our front-end and synthesized[3] the results with Quartus II 4.2 [2]. Compilations are optimized for the target FPGA device (Altera Stratix EPS1S10F484C5), which has 10,570 available logic cells, and Quartus will utilize as many of these cells as possible.

5.4 Synthesis Results

In general, a DFA for a compartmentalization policy always has exactly one state, and there is one transition for each $\{ModuleID, op, RangeID\}$ tuple. Figure 7 shows that there is a linear relationship between the number of transitions and the number of ranges.

Figure 8 shows that the area of the resulting circuit scales linearly with the number of ranges for the compartmentalization policy. The slope is approximately four logic cells for every range. Figure 9 shows the cycle time (T_{clock}) for machines of various sizes, and Figure 10 shows the setup time (T_{su}), which is primarily the time to determine the range to which the input address belongs. T_{clock} is primarily the time for one DFA transition, and it is very close to the maximum frequency of this particular Altera Stratix device. Although T_{clock} is relatively stable, T_{su} increases linearly with the number of ranges. Fortunately, T_{su} can be reduced by pipelining the circuitry that determines what range contains the input address.

Figure 11 shows the area of the circuits resulting from the example policies presented in this paper. These circuits are much smaller in area than the series of compartmentalization circuits above because the example policies have very few ranges. The complexity of the circuit is a combination of the number of ranges and the number of DFA states and transitions. Since the circuit for the Chinese wall policy has the most states, transitions, and ranges, it has the greatest area, followed by redaction, secure hand-off, access control list, and compartmentalization. Figure 12 shows that the cycle time is greatest for redaction, followed by compartmentalization, Chinese wall, secure hand-off, and access control list. Figure 13 shows that the setup time is greatest for redaction, followed by Chinese wall, compartmentalization, access control list, and secure hand-off.

5.5 Impact of the Reference Monitor on System Performance

FPGAs do not operate at high frequency. Since they operate at a lower frequency, they achieve their performance from spatial parallelism. FPGA applications such as DSPs, signal processing, and intrusion detection systems are

[3] The back-end handles netlist creation, placement, routing, and optimization for both timing and area.

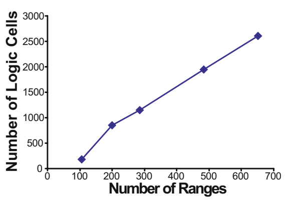


Fig. 8. Circuit area versus number of ranges. There is a nearly linear relationship between the circuit area and the number of ranges.

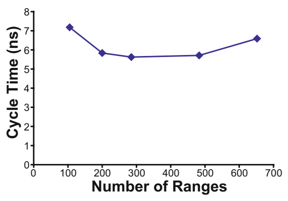


Fig. 9. Cycle time versus number of ranges. There is a nearly constant relationship between the cycle time and the number of ranges.

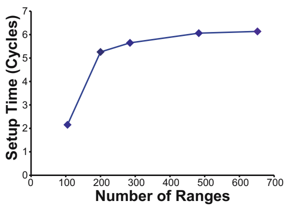


Fig. 10. Setup time versus number of ranges. There is a nearly linear relationship between the setup time and the number of ranges.

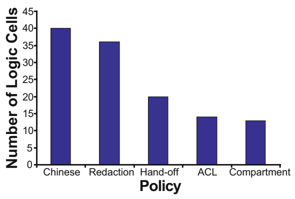


Fig. 11. Circuit area versus access policy. The area is related to the number of states, transitions, and ranges. The circuit area is greatest for the Chinese wall policy.

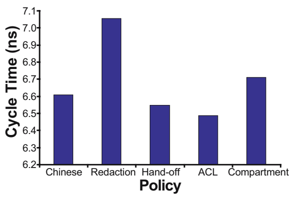


Fig. 12. Cycle time for each access policy. Cycle time is greatest for redaction, followed by compartmentalization, Chinese wall, secure hand-off, and access control list.

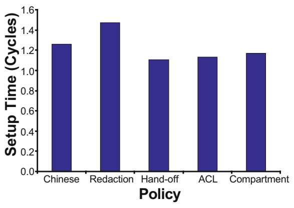


Fig. 13. Setup time for each access policy. Setup time is greatest for redaction, followed by Chinese wall, compartmentalization, access control list, and secure hand-off.

throughput-driven and therefore are latency-insensitive. These applications are designed using careful scheduling and pipelining techniques. For these reasons, we argue that our technique does not impact the performance significantly. For example, since an FPGA operating at 200MHz will have a cycle time of 5ns, our reference monitor only adds at most a two cycle delay in this case.

6 Conclusions

Reconfigurable systems are blurring the line between hardware and software, and they represent a large and growing market. Due to the increased use of reconfigurable logic in mission-critical applications, a new set of synthesizable security techniques is needed to prevent improper memory sharing and to contain memory bugs in these physically addressed embedded systems. We have demonstrated a method and language for specifying access policies that can be used as both a description of legal access patterns and as an input specification for direct synthesis to a reconfigurable logic module. Our architecture ensures that the policy module is invoked for every memory access, and we are currently developing gate-level techniques to ensure the physical isolation of the policy module.

The formal access policy language provides a convenient and precise way to describe the fine-grained memory separation of modules on an FPGA. The flexibility of our language allows modules to communicate with each other securely by precisely transferring the privilege to access a buffer from one module to another. We have used our policy compiler to translate a variety of security policies to hardware enforcement modules, and we have analyzed the area and performance of these circuits. Our synthesis data show that the enforcement module is both efficient and scalable in the number of ranges that must be recognized. In addition to the reconfigurable domain, our methods can be applied to systems-on-a-chip as part of a more general scheme.

Since usability is fundamental to system security [13] [11], we plan to provide an incremental method of constructing mathematically precise policies by building on the policy engineering work of Fong et al. [10]. In a correctly formed policy, there should be no intersection between legal and illegal behavior. Our tools will allow a policy engineer to check whether there is any conflict between a policy under construction that specifies legal behavior and a specific instance of behavior that is known to be illegal. If a conflict exists, the tool will inform the policy engineer of the exact problem that needs to be fixed.

References

1. A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison Wesley, Reading, MA, 1988.
2. Altera Inc. Quartus II Manual, 2004.
3. J.P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, ESD/AFSC, Hanscorn AFB, Bedford, MA, 1972.
4. K. Bondalapati and V.K. Prasanna. Reconfigurable computing systems. In *Proceedings of the IEEE*, volume 90(7), pages 1201–17, 2002.
5. D.A. Buell and K.L. Pocek. Custom computing machines: an introduction. In *Journal of Supercomputing*, volume 9(3), pages 219–29, 1995.
6. K. Compton and S. Hauck. Reconfigurable computing: a survey of systems and software. In *ACM Computing Surveys*, volume 34(2), pages 171–210, USA, 2002. ACM.
7. A. DeHon. Comparing computing machines. In *SPIE-Int. Soc. Opt. Eng. Proceedings of SPIE - the International Society for Optical Engineering*, volume 3526, pages 124–33, 1998.
8. A. DeHon and J. Wawrzynek. Reconfigurable computing: what, why, and implications for design automation. In *Proceedings of the Design Automation Conference*, pages 610–15, West Point, NY, 1999.
9. Ulfar Erlingsson and Fred B. Schneider. Sasi enforcement of security policies: A retrospective. In *Proceedings of the 1999 Workshop on New Security Paradigms*, 1999.
10. Philip W. L. Fong. Access control by tracking shallow execution history. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, 2004.

11. Peter Gutmann and Ian Grigg. Security usability. *IEEE Security and Privacy Magazine*, July/August 2005.
12. C. Irvine, T. Levin, T. Nguyen, and G. Dinolt. The trusted computing exemplar project. In *Proceedings of the 5th IEEE Systems, Man and Cybernetics Information Assurance Workshop*, pages 109–115, West Point, NY, June 2004.
13. Cynthia E. Irvine, Timothy E. Levin, Thuy D. Nguyen, David Shifflett, Jean Khosalim, Paul C. Clark, Albert Wong, Francis Afinidad, David Bibighaus, and Joseph Sears. Overview of a high assurance architecture for distributed multilevel security. In *Proceedings of the 2002 IEEE Workshop on Information Assurance and Security*, West Point, NY, June 2002.
14. S. Johnson. Yacc: Yet another compiler-compiler. Technical Report CSTR-32, Bell Laboratories, Murray Hill, NJ, 1975.
15. Ryan Kastner, Adam Kaplan, and Majid Sarrafzadeh. *Synthesis Techniques and Optimizations for Reconfigurable Systems*. Kluwer Academic, Boston, MA, 2004.
16. P. Kocher, R. Lee, G. McGraw, A. Raghunathan, and S. Ravi. Security as a new dimension in embedded system design. In *Proceedings of the 41st Design Automation Conference (DAC '04)*, San Diego, CA, June 2004.
17. M. Lesk and E. Schmidt. Lex: A lexical analyzer generator. Technical Report 39, Bell Laboratories, Murray Hill, NJ, October 1975.
18. Timothy E. Levin, Cynthia E Irvine, and Thuy D. Nguyen. A least privilege model for static separation kernels. Technical Report NPS-CS-05-003, Naval Postgraduate School, 2004.
19. Peter Linz. *An Introduction to Formal Languages and Automata*. Jones and Bartlett, Sudbury, MA, 2001.
20. W.H. Mangione-Smith, B. Hutchings, D. Andrews, A. DeHon, C. Ebeling, R. Hartenstein, O. Mencer, J. Morris, K. Palem, V.K. Prasanna, and H.A.E. Spaanenburg. Seeking solutions in configurable computing. In *Computer*, volume 30(12), pages 38–43, 1997.
21. Giovanii De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, New York, 1994.
22. J. Navarro, S. Iyer, P. Druschel, and A. Cox. Practical, transparent operating system support for superpages. In *Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, MA, December 2002.
23. D. Raymond and D. Wood. Grail: A C++ library for automata and expressions. *Journal of Symbolic Computation*, 11:341–350, 1995.
24. John Rushby. Design and verification of secure systems. *ACM Operating Systems Review*, 15(5):12–21, December 1981.
25. John Rushby. A trusted computing base for embedded systems. In *Proceedings 7th DoD/NBS Computer Security Conference*, pages 294–311, September 1984.
26. Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1), January 2003.
27. J. Saltzer. Protection and the control of information sharing in multics. *Communications of the ACM*, 17(7):388–402, July 1974.
28. O. Sami Saydjari. Multilevel security: Reprise. *IEEE Security and Privacy Magazine*, September/October 2004.
29. P. Schaumont, I. Verbauwhede, K. Keutzer, and M. Sarrafzadeh. A quick safari through the reconfiguration jungle. In *Proceedings of the Design Automation Conference*, pages 172–7, 2001.
30. Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1), February 2000.

31. Richard E. Smith. Cost profile of a highly assured, secure operating system. In *ACM Transactions on Information and System Security*, 2001.
32. D.F. Stern. On the buzzword "security policy". In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pages 219–230, Oakland, CA, 1991.
33. J.E. Vuillemin, P. Bertin, D. Roncin, M. Shand, H.H. Touati, and P. Boucard. Programmable active memories: Reconfigurable systems come of age. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 4(1), pages 56–69, 1996.
34. Clark Weissman. MLS-PCA: A high assurance security architecture for future avionics. In *Proceedings of the Annual Computer Security Applications Conference*, pages 2–12, Los Alamitos, CA, December 2003. IEEE Computer Society.
35. E. Witchel, J. Cates, and K. Asanovic. Mondrian memory protection. In *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, San Jose, CA, October 2002.

Privacy-Preserving Queries on Encrypted Data*

Zhiqiang Yang[1], Sheng Zhong[2], and Rebecca N. Wright[1]

[1] Computer Science Department, Stevens Institute of Technology, Hoboken, NJ 07030 USA
[2] Computer Science and Engineering Department, SUNY Buffalo, Amherst, NY 14260 USA

Abstract. Data confidentiality is a major concern in database systems. Encryption is a useful tool for protecting the confidentiality of sensitive data. However, when data is encrypted, performing queries becomes more challenging. In this paper, we study efficient and provably secure methods for queries on encrypted data stored in an outsourced database that may be susceptible to compromise. Specifically, we show that, in our system, even if an intruder breaks into the database and observes some interactions between the database and its users, he only learns very little about the data stored in the database and the queries performed on the data.

Our work consists of several components. First, we consider databases in which each attribute has a finite domain and give a basic solution for certain kinds of queries on such databases. Then, we present two enhanced solutions, one with a stronger security guarantee and the other with accelerated queries. In addition to providing proofs of our security guarantees, we provide empirical performance evaluations. Our experiments demonstrate that our solutions are fast on large-sized real data.

1 Introduction

As anyone who reads newspapers is aware, there have been a staggering number of data breaches reported in the last two years. Some of the largest of these revealed sensitive information of millions of individuals. For example, in June 2005, names and credit card numbers of more than 40 million MasterCard cardholders were exposed [12]. In May 2006, disks containing the names, social security numbers, and dates of birth of more than 26 million United States veterans were stolen from the home of an employee of the Department of Veterans Affairs [29]. In the wrong hands, this kind of sensitive information can be to carry out identity theft and other fraudulent activities that harm the individuals involved and have a large cost to society.

Techniques such as access control, intrusion detection, and policies about how data is to be used attempt to prevent such thefts and intrusion. However, existing techniques cannot ensure that a database is fully immune to intrusion and unauthorized access.

Encryption is a well-studied technique to protect sensitive data [13] so that even if a database is compromised by an intruder, data remains protected even in the event that a database is successfully attacked or stolen. Provided that the encryption is done properly and the decryption keys are not also accessible to the attacker, encryption can provide protection to the individuals whose sensitive data is stored in the databases,

* This work was supported by the National Science Foundation under Grant No. CCR-0331584.

reduce the legal liability of the data owners, and reduce the cost to society of fraud and identity theft.

While encrypting the data provides important protection, encrypted data is much less convenient to use encrypted data than to use cleartext data. Specifically, in a database that stores encrypted data, how can queries be processed? If a database user fully trusts the database server, she can simply send the encryption key to the database server together with her query. However, because the database may be compromised at some point during such an interaction, revealing the key to the database server is at the risk of leaking all sensitive data to an intruder. Theoretically, if efficiency was not a concern, a user could retrieve all encrypted tables from the database, decrypt the tables, and then perform queries on the cleartext tables. However, this is clearly impractical when we take efficiency into consideration.

In this paper, we investigate efficient methods for processing queries on encrypted data in such a way that the data remains secure even if the database server may be compromised at some point by an intruder. Such methods are very useful in strengthening the protection of sensitive data in databases.

1.1 Related Work

Various methods have been proposed recently for securing databases in various settings [7,27,30,22,25,21]. In particular, encryption is an important technique to protect sensitive data [13]. An analysis of how to encrypt and securely store data in relational database management systems has been given in [24]. Recognizing the importance of encryption techniques, some database vendors have included encryption functionality in their products [1,2]. By considering different privacy policies for different data records, Hippocratic databases, which combine privacy policies with sensitive data, are very useful in preventing unauthorized users from accessing sensitive data [3].

With data stored in an encrypted form, a crucial question is how to perform queries. Hacigumus et al. [19] studied querying encrypted data in the database-as-service (DAS) model where sensitive data is outsourced to an untrusted server [20]. Their solution divides attribute domains into partitions and maps partitions ids to random numbers to achieve privacy. This idea is simple, practical, and elegant. However, it relies on an implicit tradeoff between privacy and efficiency. Specifically, if the partitions are larger, then less information is leaked, but the database server needs to send more false positives (i.e., data that should not have been in the results of queries) to the user. If the partitions are smaller, then the database server needs to send fewer false positives, but more information is leaked. (This issue is further explored in [23].) Furthermore, no precise quantifications are given of either of the information leak relative to the size of partitions or of the amount of communication overhead. In comparison, the solutions we provide in this paper do not have such a tradeoff; our solutions enjoy strong privacy without wasting communication resources on false positives; our security guarantee is precisely quantified using a cryptographic measure. Another issue is that, although the partition ids in [19] can be used for indexing to speed up queries, such an index can incur inference and linking attacks as is pointed out in [11]. In comparison, our solution in Section 5 speeds up queries using metadata without introducing any additional information leakage.

Agrawal et al. [4] propose a solution for range queries on numerical data that allows convenient indexing. Their solution is built on an encoding that preserves the order of the numerical data in each column. Consequently, if a database intruder observes the encrypted data, he learns the order of all cells in every column, which is a significant amount of information. They give no rigorous analysis quantifying the information leak of their solution. In comparison, in this paper we show that our solutions reveal only a small amount (as quantified in later sections) of information to a potential intruder.

In the scenario considered in [4,24], the adversary is modeled to have access to the data storage and has no access to the transmitted messages between the users and the database. In the setting of DAS model [19], since the database server is untrusted, the server itself is a potential adversary who tries to breach the data privacy. The server has access to all encrypted data and all the transmitted messages between the users and the server. In this sense, the server has the strongest power to breach data privacy. In comparison, we model that the adversary can have access to all encrypted data in the server, and he also can monitor some transmitted messages (up to t queries) between the server and the users. We give the details of the attack (or adversary) model in Section 2.1, and we also prove the security properties of our solutions under the adversary model.

The study of "search on encrypted data" is closely related to our work. Specifically, Song, Wagner, and Perrig [28] propose practical techniques for finding keywords in encrypted files, which allow a user, when given a trapdoor for a keyword, to check the existence of the key word in a file. But their solution needs to scan the entire file sequentially and no provably secure index technique is provided. A follow-up by Chang and Mitzenmacher [9] has interesting analysis but their solution is restricted to searching for a keyword chosen from a pre-determined set. Boneh et al. present a searchable public key scheme [6]; the scenario they considered is analogous to that of [28] but uses public-key encryption rather than symmetric-key encryption. In the same scenario, Goh demonstrates a method for secure indexes using Bloom filters [15]. These solutions are possibly useful in searching for keywords in a file; however, it is unclear how to apply them to the problem of efficiently querying encrypted relational databases. Yet another piece of related work is by Feigenbaum et al. [14], in which an encryption scheme was proposed to efficiently retrieve tuples from a look-up dictionary by using hash functions. The basic idea is that a tuple can only be retrieved if a valid key is provided.

In contrast to the goals of our work, private information retrieval [10,8,26] is designed to hide entirely from the database which queries a user is making. As we discuss later, we take a more pragmatic view that allows more efficient solutions.

1.2 Our Contributions

In this paper, we address the problem of performing queries on an encrypted database. We consider a pragmatic notion of privacy that trades off a small amount of privacy for a gain in efficiency. Specifically, in our solutions all data is stored and processed in its encrypted form. We note that given any solution that returns a response to a query to the user consisting of precisely the encryptions in the database of the items that match the query, this solution leaks the location of the returned cells to an attacker with access to the database. Similarly, even if a solution were to return different encryptions of the matching items, if the database is able to access only those cells, then the location of those cells is revealed. In order to admit the most efficient solutions, we therefore consider this information to be the "minimum information revelation," as described in more detail in Section 2. We allow solutions that leak this minimum information revelation, while we seek to prevent leakage of any additional information.

The contributions of this paper can be summarized as follows.

- We present a basic solution for simple queries (Section 3). We give a rigorous security analysis to show that, beyond the minimum information revelation, our solution only reveals very little information, namely which attributes are tested in the "where" condition. Our security guarantee is quantitative and cryptographically strong.
- We present a solution with enhanced security (Section 4). We show that, for a broad class of tables, this solution reveals nothing beyond the minimum information revelation.
- We present a solution that adds metadata to further speed up queries (Section 5).
- Compared with previous solutions, an advantage of our schemes is that a database user does not need to maintain a large amount of confidential information (like the partitioning ids in [19] or the large keys in [4]). In our schemes, a user only needs to store several secret keys that amount to at most tens of bytes. Thus the storage overhead on the user side is negligible.

2 Technical Preliminaries

We consider a system as illustrated in Figure 1. In this system, data is encrypted and stored in tables. In the front end, when the user has a query, the query is translated to one or more messages that are sent to the database. Upon receiving the message(s), the database finds the appropriate encrypted cells and returns them to the front end. Finally, the front end decrypts the received cells. For ease of presentation, we do not distinguish the human user from the front end program; when we say "user," we mean the human user plus the front end program.

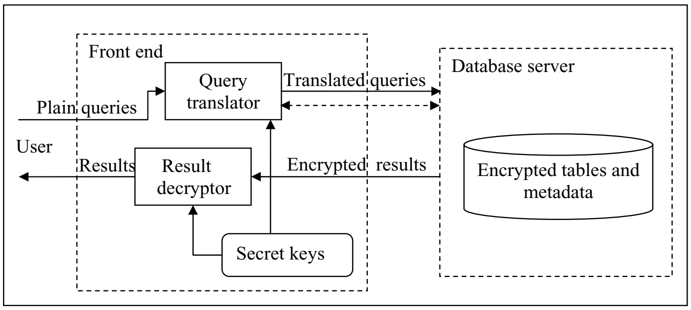


Fig. 1. Overall architecture

2.1 Trust and Attack Model

In this paper, we focus on the possibility that an intruder might successfully attack the database server. The goal of our work is that an intruder who has complete access to the database server for some time should learn very little about the data stored in the database and the queries performed on the data. Our trust and attack model is as follows:

1. We do not fully trust the database server because it may be vulnerable to intrusion. Furthermore, we assume that, once a database intruder breaks into the database, he can observe not only the encrypted data in the database, but can also control the whole database system for a time interval. During the time interval, a number of query messages sent by the user, as well as the database's processing of these queries, can be observed by the intruder. We note that assumption that an intruder can only control the whole database system for only a bounded time period is reasonable, for example, in the setting that a database administrator can physically reset the database server from time to time or when intrusions are detected.
2. We assume the communication channel between the user and the database is secure, as there exist standard protocols to secure it—e.g., SSL and IPsec. We also trust the user's front-end program; protecting the front-end program against intrusion is outside of the scope of this paper.
3. We require all data and metadata, including user logs and scheme metadata, to be stored encrypted. (Otherwise, these may open the door for intruders.)

2.2 Table and Queries

We consider a database represented by a table T and we discuss queries performed on T. Suppose that T has n rows (i.e., n records) and m columns (i.e., m attributes). We denote by $T_{i,j}$ the cell at the intersection of the ith row and the jth column; we also refer to (i, j) as the *coordinates* of the cell. We denote the ith row by T_i. Each attribute of the table has a finite domain. For the jth attribute A_j, we denote the domain by D_j.

As we have mentioned, we store our tables in an encrypted form. More precisely, for a table T, we store an encrypted table T' in the database, where each $T'_{i,j}$ is an encryption of $T_{i,j}$. Without loss of generality, we assume that each cell $T_{i,j}$ of the plaintext table is a bitstring of exactly k_1 bits—that is, $\forall j \in [1, m]$, $D_j \subseteq \{0, 1\}^{k_1}$. (We can always encode any value of an attribute as a sufficiently long bitstring.) When we encrypt a cell, the encryption algorithm appends a random string of k_2 bits to the plaintext.[1] Hence, the input to the encryption algorithm is a k_0-bit string, where $k_0 = k_1 + k_2$. For simplicity (and following the practice of most symmetric encryption schemes), we assume the output of the encryption algorithm and the encryption key are k_0-bit strings as well. We therefore note that k_0 should be chosen to be long enough to resist brute-force key search attacks.

Suppose that a user intends to perform a query Q on the table T. As discussed earlier, in this paper, in order to allow solutions that are as efficient as possible, we consider query protocols that return to the user precisely the set of encrypted cells stored in the database that satisfy the condition of the query, with the same encryptions as in T'. We call such query protocols *precise query protocols*. Denote by $R(Q)$ the set of coordinates of the cells satisfying the condition of query Q— i.e., the cells satisfying the condition of query Q are $\{T'_{i,j} : (i, j) \in R(Q)\}$. Clearly, in *any* precise query protocol,

[1] As explained in more detail in Section 3, the purpose of using a random string is that multiple occurrences of a plaintext should lead to different ciphertexts.

if there is a database intrusion of the type discussed in Section 2.1, then $R(Q)$ is always revealed to the intruder. This is because the intruder can simply check T' to see which encrypted cells are in the returned result. Therefore, we say $R(Q)$ is the *minimum information revelation* of query Q. We allow solutions that reveal this minimum information revelation; we seek solutions that do not yield any additional information.

2.3 Privacy-Preserving Queries

We give a cryptographic definition of *privacy-preserving query protocols*. In particular, we consider that an intruder may observe up to t queries $Q_1, \ldots, Q_t$, where t is a polynomially bounded function of k_0. We quantify the information leaked by the protocol using a random variable α. Specifically, we say the protocol only reveals α beyond the minimum information revelation if, after these queries are processed, what the database intruder has observed can be simulated by a probabilistic polynomial-time algorithm using only α, $R(Q)$, and the encrypted table. For simplicity, we only provide here a definition of a privacy-preserving *one-round* query protocol. It is straightforward to extend this definition to multi-round query protocols.

Definition 1. *(Privacy-Preserving Query) A one-round query protocol* reveals only α beyond the minimum information revelation *if for any polynomial* $\mathsf{poly}()$ *and all sufficiently large* k_0*, there exists a probabilistic polynomial-time algorithm* $\mathcal{S}$ *(called a* simulator*) such that for any* $t < \mathsf{poly}(k_0)$*, any polynomial-size circuit family* $\{\mathcal{A}_{k_0}\}$*, any polynomial* $p()$*, and any* $Q_1, \ldots, Q_t$*,*

$$\begin{aligned}&|\Pr[\mathcal{A}_{k_0}(Q_1, \ldots, Q_t, q_1, \ldots, q_t, T') = 1] - \\ &\qquad \Pr[\mathcal{A}_{k_0}(Q_1, \ldots, Q_t, \mathcal{S}(\alpha, R(Q_1), \ldots, R(Q_t), T')) = 1]| < 1/p(k_0).\end{aligned}$$

A query protocol is ideally private *if it reveals nothing beyond the minimum information revelation.*

The above definition can be viewed as an adaptation of the definition of secure protocol in the semi-honest model (i.e., assuming the intruder does not modify the database software but attempts to violate data privacy by analyzing what he observes) [17]. However, note that a secure one-round query protocol as defined here *remains secure even in the case the intruder is fully malicious* (i.e., even when the intruder modifies the database software such that the database deviates from the protocol). The reason is that the the database's behavior does not affect the user's behavior in this case.

3 Basic Solution

In this section, we give a basic solution for queries of the format "*select*...*from* T *where* $A_j = v$," where $v \in D_j$ is a constant. We provide rigorous cryptographic specifications and proofs.

3.1 Solution Overview

Our basic idea is to encode each cell in a special redundant form. Specifically, for each cell $T_{i,j}$, the encrypted cell $T'_{i,j} = (T'_{i,j}\langle 1\rangle, T'_{i,j}\langle 2\rangle)$ has two parts. The first part $T'_{i,j}\langle 1\rangle$, is a simple encryption of $T_{i,j}$ using a block cipher $E()$; the second part, $T'_{i,j}\langle 2\rangle$, is a "checksum" that, together with the first part, enables the database to check whether this cell satisfies the condition of the query or not. $T'_{i,j}\langle 1\rangle$ and $T'_{i,j}\langle 2\rangle$ satisfy a secret equation determined by the value of $T_{i,j}$. When the database is given the equation corresponding to value v, it can easily check whether a cell satisfies the condition or not by substituting the two parts of the encrypted cell into the equation.

The remaining question is what equation to use as the secret equation. We use the following simple equation:

$$E_{f(T_{i,j})}(T'_{i,j}\langle 1\rangle) = T'_{i,j}\langle 2\rangle,$$

where f is a function. When the user has a query with condition $A_j = v$, she only needs to send $f(v)$ to the database so that the database can check, for each i, whether

$$E_{f(v)}(T'_{i,j}\langle 1\rangle) = T'_{i,j}\langle 2\rangle$$

holds. It should be infeasible to derive v from $f(v)$ because otherwise an intruder learns v when observing $f(v)$. To achieve this goal, we define $f(\cdot)$ to be an encryption of v using the block cipher $E(\cdot)$. Additional care needs to be taken when we use the block cipher E. As previously mentioned, we append a random string to $T_{i,j}$ before applying E to obtain $T'_{i,j}\langle 1\rangle$; this is done in order to prevent the database from being able to determine whether two cells have the same contents. Additionally, in order to avoid having the same $f(v)$ for different attributes, we append j to $f(v)$ before applying E.

3.2 Solution Details

Data Format. Let $E(\cdot)$ be a symmetric encryption algorithm whose key space, plaintext space, and ciphertext space are all $\{0,1\}^{k_0}$. We often use the notation $E_S(M_1, M_2)$ to denote a message (M_1, M_2) encrypted using secret key S, where M_1 (resp., M_2) is either a k_1-bit (resp., k_2-bit) string. We denote the corresponding decryption algorithm by D, and we assume that the key generation algorithm simply picks a uniformly random key from the key space $\{0,1\}^{k_0}$.

To create the table T in the database, the user first picks two secret keys s_1, s_2 from $\{0,1\}^{k_0}$ independently and uniformly. The user keeps s_1, s_2 secret. For each cell $T_{i,j}$, the user picks $r_{i,j}$ from $\{0,1\}^{k_2}$ uniformly at random and stores

$$\begin{aligned} T'(i,j) &\triangleq (T'(i,j)\langle 1\rangle, T'(i,j)\langle 2\rangle) \\ &= (E_{s_1}(T_{i,j}, r_{i,j}), E_{E_{s_2}(T_{i,j},j)}(E_{s_1}(T_{i,j}, r_{i,j}))) \end{aligned}$$

Query Protocol. Denote by A_j the jth attribute of T. Suppose there is a query *select* A_{j_1},..., A_{j_ℓ} *from* T *where* $A_{j_0} = v$. To carry out this query, the user computes $q = E_{s_2}(v, j_0)$ and sends j_0, q, and $(j_1, \ldots, j_\ell)$ to the database.

For $i = 1, \ldots, n$, the database tests whether $T'_{i,j_0}\langle 2\rangle = E_q(T'_{i,j_0}\langle 1\rangle)$ holds. For any i such that the above equation holds, the database returns $T'_{i,j_1}\langle 1\rangle, \ldots, T'_{i,j_\ell}\langle 1\rangle$ to the user. The user decrypts each received cell using secret key s_1 and discards the k_2-bit tail of the cleartext.

In our scheme, note that each encrypted cell with the same plaintext value has a different encryption. Thus if an intruder breaks into the database and sees the encrypted table, he cannot tell whether two cells have the same plaintext value or not.

3.3 Security Analysis

We can prove the security of our scheme by using standard cryptographic techniques. Recall that for security we need to consider t queries. Suppose the uth query ($1 \leq u \leq t$) is of the format "select $A_{j_{u,1}}$,..., $A_{j_{u,\ell}}$ from T where $A_{j_{u,0}} = v_u$." We show that our basic solution only reveals $j_{1,0}, \ldots, j_{t,0}$ beyond the minimum information revelation. That is, the only extra information leakage by the basic solution is which attributes are tested in the "where" conditions.

The security of our scheme derives from the security of the block cipher we use. In cryptography, secure block ciphers are modeled as *pseudorandom permutations* [18]. Here, encryption key of the block cipher is the random seed for the pseudorandom permutation. For each value of the key, the mapping from the cleartext blocks to the ciphertext blocks is the permutation indexed by the value of the seed. In the following theorem, we assume the block cipher we use satisfies this security requirement.

Theorem 1. *If the block cipher E is a pseudorandom permutation (with the encryption key as the random seed), the basic protocol reveals only $j_{1,0}, \ldots, j_{t,0}$ beyond the minimum information revelation.*

Proof. We construct a simulator $\mathcal{S}$ as follows. First, let $R_1(Q_u) = \{i : (i,j) \in R(Q_u)\}$ and $R_2(Q_u) = \{j : (i,j) \in R(Q_u)\}$. Then, for any $u \in \{1, \cdots, t\}$, if there exists $u' < u$ such that $j_{u,0} = j_{u',0}$ and that $R_1(Q_u) = R_1(Q_{u'})$, $\mathcal{S}$ sets $\overline{q}_u = \overline{q}_{u'}$. otherwise, $\mathcal{S}$ chooses $\overline{q}_u$ from $\{0,1\}^{k_0} - \{\overline{q}_{u'} : u' < u \wedge j_{u,0} = j_{u',0}\}$ uniformly at random. Next, for $i = 1$ through n and $j = 1$ through m, $\mathcal{S}$ chooses $\overline{T'}_{i,j}\langle 1\rangle$ uniformly and independently from $\{0,1\}^{k_0}$. For $u = 1$ through t, for each $i \in R_1(Q_u)$, $\mathcal{S}$ computes

$$\overline{T'}_{i,j_{u,0}}\langle 2\rangle = E_{\overline{q}_u}(\overline{T'}_{i,j_{u,0}}\langle 1\rangle).$$

For any pair (i,j) for which $\overline{T'}_{i,j}\langle 2\rangle$ has not been defined, $\mathcal{S}$ chooses $\overline{T'}_{i,j}\langle 2\rangle$ from $\{0,1\}^{k_0}$ uniformly and independently. Finally, $\mathcal{S}$ outputs $\overline{q}_1, \ldots, \overline{q}_t$, $\overline{T'}$. The indistinguishability by polynomial-size algorithms follows from the pseudorandomness of E.

In this setting, even if the intruder has access to the whole database, the intruder can learn nothing about the encrypted data. By combining $j_{1,0}, \cdots, j_{t,o}$ with the minimum information revelation, an intruder can derive some statistical information about the underlying data or the queries (Theorem 1 does catch this case). In Section 4, we present a solution that leaks less information to make such attacks more difficult.

3.4 Performance Evaluations

To evaluate the efficiency of our basic solution in practice, we implemented the basic solution. Our experiments use the Nursery dataset from the UCI machine learning repository [5]. The Nursery dataset is a table with eight categorical attributes and one class attribute. There are 12,960 records in total. The total number of data cells is about $100,000$. The only change we made to the Nursery dataset is that we added an ID attribute to the Nursery dataset so that the table would have a primary key.

Because the time spent on communication is highly dependent on the network bandwidth, we focus on the computational overhead and ignore the communication overhead. The experimental environment is the NetBSD operating system running on an

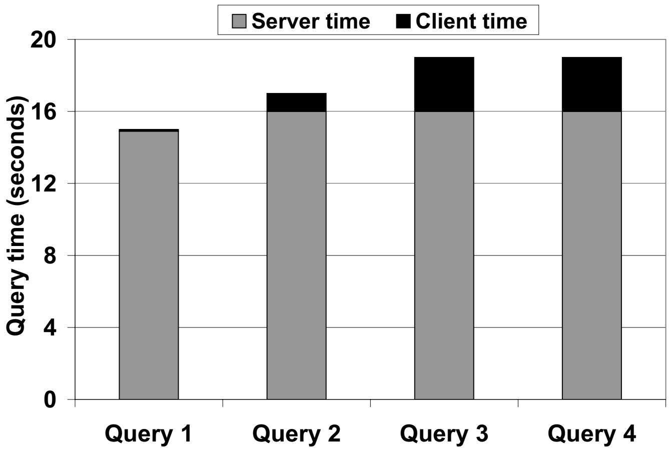


Fig. 2. Query time in the basic solution

AMD Athlon 2GHz processors with 512M memory. For the block cipher, we use the Blowfish symmetric encryption algorithm with a 64-bit block size (i.e., $k_0 = 64$).

Clearly, the overhead to encrypt a database is linear in the size of the database. Specifically, in our experiments it took only 25 seconds to encrypt the entire Nursery dataset. On average, encrypting a single record requires only 0.25 milliseconds. Figure 2 shows the time consumed by four SELECT queries. Those queries are SELECT $*$ FROM Nursery WHERE Parent=usual, WHERE Class=recommend, WHERE Class=very_recom and WHERE Class=priority.

For each query, the database server needs almost the same amount of time for computation (about 16 seconds). The user's computational time depends on the number of returned records from the database. In the first query, only 2 records are returned, and so the computational time by the user is extremely small. In contrast, the last two queries return 4320 and 3266 records, respectively. Therefore, the computational time of the user in each of these two queries is about 4 seconds.

4 Solution with Enhanced Security

In this section, we enhance the security of the basic solution so that the query protocol reveals less information. For a broad class of tables, we can show our solution with enhanced security is ideally private.

4.1 Solution Overview

Recall that the basic solution reveals which attributes are tested in the "where" conditions. Our goal is to hide this information (or at least part of this information). A straightforward way for doing this is to randomly permute the attributes in the encrypted table, in order to make it difficult for a database intruder to determine which attributes are tested.

There remains the question of which distribution to use for the random permutation. If the distribution has a large probability mass on some specific permutations, then the intruder can guess the permutation with a good probability. So the ideal distribution is the uniform distribution. However, if the permutation is chosen uniformly from all permutations of the attributes, the user needs to "memorize" where each attribute is after the permutation. When the number of attributes is large, this is a heavy burden for the user. To eliminate this problem, we use a pseudorandom permutation, which is by definition indistinguishable from a uniformly random permutation [16]. The advantage of this approach is that it requires the user to memorize the random seed.

In fact, we note that we do not need to permute all the attributes in the encrypted table. For each (i, j), we can keep $T'_{i,j}\langle 1\rangle$ as defined in the basic solution; we only need to permute the equations satisfied by $T'_{i,j}\langle 1\rangle$ and $T'_{i,j}\langle 2\rangle$ because only these equations are tested when there is a query. Specifically, the equation satisfied by $T'_{i,j}\langle 1\rangle$ and $T'_{i,j}\langle 2\rangle$ is no longer decided by the value $T_{i,j}$; instead, it is decided by $T_{i,\pi_S(j)}$, where $\pi_S()$ is a pseudorandom permutation. Consequently, when there is a query whose condition involves attribute A_j, the database actually tests an equation on attribute $A_{\pi_S^{-1}(j)}$.

4.2 Solution Details

Data Format. Let $\pi_S()$ be a pseudorandom permutation on $\{1, \cdots, m\}$ for a uniformly random seed $S \in \{0,1\}^{k_0}$. To store the table T in the database, the user first picks secret keys s_1, s_2, s'_2 from $\{0,1\}^{k_0}$ independently and uniformly. The user keeps s_1, s_2, and s'_2 secret. For each cell $T_{i,j}$, the user picks $r_{i,j}$ from $\{0,1\}^{k_2}$ uniformly at random, computes $\hat{\jmath} = \pi_{s'_2}(j)$ and stores

$$\begin{aligned} T'(i,j) &\triangleq (T'(i,j)\langle 1\rangle, T'(i,j)\langle 2\rangle) \\ &= (E_{s_1}(T_{i,j}, r_{i,j}), E_{E_{s_2}(T_{i,\hat{\jmath}},\hat{\jmath})}(E_{s_1}(T_{i,j}, r_{i,j}))), \end{aligned}$$

Query Protocol. Suppose there is a query *select* A_{j_1},..., A_{j_ℓ} *from* T *where* $A_{j_0} = v$. To carry out this query, the user computes $j'_0 = \pi_{s'_2}^{-1}(j_0)$ and $q = E_{s_2}(v, j'_0)$, then sends $j'_0, q, (j_1, \ldots, j_\ell)$ to the database.

For $i = 1, \ldots, n$, the database tests whether $T'_{i,j'_0}\langle 2\rangle = E_q(T'_{i,j'_0}\langle 1\rangle)$ holds. For any i such that the above equation holds, the database returns $T'_{i,j_1}\langle 1\rangle, \ldots, T'_{i,j_\ell}\langle 1\rangle$ to the user. The user decrypts each received cell using secret key s_1 and discards the k_2-bit tail of the cleartext.

4.3 Security Analysis

Again, recall that for security we need to consider t queries, where the uth query $(1 \le u \le t)$ is of the format "select $A_{j_{u,1}}$,..., $A_{j_{u,\ell}}$ from T where $A_{j_{u,0}} = v_u$." We introduce a new variable that represents whether two queries involve testing the same attribute: for $u, u' \in [1, t]$, we define

$$\epsilon_{u,u'} = \begin{cases} 1 & \text{if } j_{u,0} = j_{u',0} \\ 0 & \text{otherwise.} \end{cases}$$

Using this new variable, we can quantify the security guarantee of our solution with enhanced security. Furthermore, we are able to show that our solution with enhanced security becomes ideally private if the table belongs to a broad class, which we call the *non-coinciding* tables. Intuitively, a table is non-coinciding if any two queries that test different attributes do not have exactly the same result. More formally, we have the following.

Definition 2. *A table T is* non-coinciding *if for any $j \neq j'$, any $v \in A_j$, $v' \in A_{j'}$,*

$$\{i : T_{i,j} = v\} \neq \{i : T_{i,j'} = v'\}.$$

Theorem 2. *Suppose that the block cipher $E()$ is a pseudorandom permutation (with the encryption key as the random seed). Then the query protocol with enhanced security reveals only $\epsilon_{u,u'}$ for $u, u' \in [1, j]$. When the table T is non-coinciding, the query protocol with enhanced security is ideally private.*

Proof. We construct a simulator $\mathcal{S}$ as follows. First, recall that $R_1(Q_u) = \{i : (i,j) \in R(Q_u)\}$ and $R_2(Q_u) = \{j : (i,j) \in R(Q_u)\}$. For $u = 1$ through t, if there exists $u' < u$ such that $\epsilon_{u,u'} = 1$ and that $R_1(Q_u) = R_1(Q_{u'})$, $\mathcal{S}$ sets $\overline{q}_u = \overline{q}_{u'}$ and $j'_{u,0} = j'_{u',0}$; if there exists $u' < u$ such that $\epsilon_{u,u'} = 1$ and that $R_1(Q_u) \neq R_1(Q_{u'})$, $\mathcal{S}$ chooses $\overline{q}_u$ from $\{0,1\}^{k_0} - \{\overline{q}_{u'} : u' < u \wedge \epsilon_{u,u'} = 1\}$ uniformly at random and sets $j'_{u,0} = j'_{u',0}$; otherwise, $\mathcal{S}$ chooses $\overline{q}_u$ from $\{0,1\}^{k_0}$ uniformly at random, and $j'_{u,0}$ from $[1, m] - \{j'_{u',0} : u' < u\}$ uniformly at random. Next, for $i = 1$ through n and $j = 1$ through m, $\mathcal{S}$ chooses $\overline{T'}_{i,j}\langle 1 \rangle$ uniformly and independently from $\{0,1\}^{k_0}$. For $u = 1$ through t, for each $i \in R_1(Q_u)$, $\mathcal{S}$ computes

$$\overline{T'}_{i,j'_{u,0}}\langle 2 \rangle = E_{\overline{q}_u}(\overline{T'}_{i,j'_{u,0}}\langle 1 \rangle).$$

For each pair (i,j) such that $\overline{T'}_{i,j}\langle 2 \rangle$ has not been defined, $\mathcal{S}$ chooses $\overline{T'}_{i,j}\langle 2 \rangle$ from $\{0,1\}^{k_0}$ uniformly and independently. Finally, $\mathcal{S}$ outputs $\overline{q}_1, \ldots, \overline{q}_t$, $\overline{T'}$. The indistinguishability by polynomial-size circuits follows from the pseudorandomness of $E()$.

When T is non-coinciding, in the above proof we can replace $\epsilon_{u,u'} = 1$ with $R_1(Q_u) = R_1(Q_{u'})$. Because these two conditions are equivalent, this replacement does not change the output of the simulator. On the other hand, because $\epsilon_{u,u'}$ is no longer needed by the simulator, we have shown the protocol is ideally private.

5 Query Speedup Using Metadata

In the two solutions we have presented, performing a query on the encrypted table requires testing each row of the table. Clearly, this is very inefficient in large-size databases. In this section, we consider a modification to the basic solution that drastically speeds up queries. It is easy to make a similar modification to the solution with enhanced security.

5.1 Solution Overview

We can significantly improve the efficiency if we are able to replace the sequential search in the basic solution with a binary search. However, our basic solution finds the appropriate rows by testing an equation, while a binary search cannot be used to find the items that satisfy an equation.

To sidestep this difficulty, we add some metadata to eliminate the need for testing an equation[2]. Specifically, for each cell in the column, we add a tag and a link. The tag is decided by the value of the cell; the link points to the cell. We sort the metadata according to the order of the tags. When there is a query on the attribute, the user sends the appropriate tag to the database so that the database can perform a binary search on the tags.

We illustrate the concept with a simple example, shown in Figure 3. Consider a column of four cells with values 977, 204, 403, 155. We compute four tag values based on the corresponding cell values. Suppose that the tags we get are 3, 7, 4, 8. We sort the tags to get a list: 3, 4, 7, 8. After we add links to the corresponding cells, we finally get: (3, link to cell "977"), (4, link to cell "403"), (7, link to cell "204"), (8, link to cell "155").

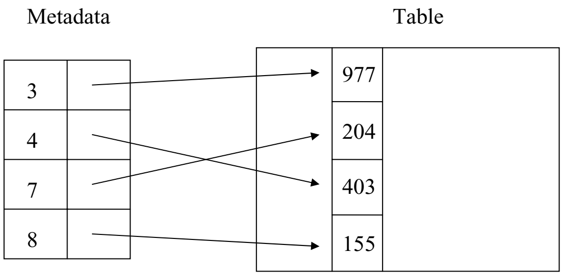


Fig. 3. Example of metadata

Nevertheless, there is a question of multiple occurrences of a single value: if multiple cells in the column have the same value, how do we choose the tags for these cells? Clearly, we cannot use the same tag for these cells; otherwise, when the database intruder looks at the tags, he can recognize cells with the same value. In fact, it should be hard for the intruder to find out which tags correspond to the same cell value. On the other hand, it should be easy for the user to derive the entire set of tags corresponding to a single value.

We resolve this dilemma using a two-step mapping. In the first step, we map a cell value $T_{i,j}$ to an intermediate value $H_{i,j}$ whose range is much larger. Then the $H_{i,j}$'s are sparse in their range, which means around each value of $H_{i,j}$ there is typically a large "blank" space. Consequently, to represent multiple occurrences of the same cell value $T_{i,j}$, we can use multiple points starting from $H_{i,j}$. In the second step, we map these intermediate points to tags such that the continuous intermediate points become random-looking tags. See Figure 4 for an illustration. In our design, the first step of mapping (from the cell value to the intermediate value) is implemented using an encryption of the cell value (appended with a k_2-bit 0 so that the input to the cipher is k_0

[2] The functionality of these metadata is analogous to indices in traditional database systems—to help speed up queries. However, since the structure and usage of these metadata are different from that of traditional indices (e.g., B+-trees), we do not call them indices. Note that indices like B+-trees cannot be used in our scenario.

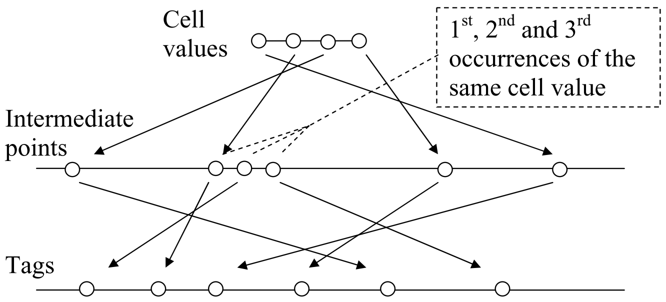


Fig. 4. Two-step mapping from cell values to tags

bits), where the encryption key is kept by the user. The second step of mapping (from the intermediate value to the tag) is implemented using another encryption, where the key is again kept by the user. Since the database intruder does not know the two encryption keys, he cannot figure out which cell value corresponds to which tag, or which of the tags correspond to the same cell value. On the other hand, when there is a query, the user can simply send the database the tags for the cell value in the query; then the database can easily locate the rows satisfying the condition of this query.

Note that, for the convenience of queries, we should keep a counter of the occurrences of each cell value; otherwise, when the user has a query, he cannot know how many intermediate values (and thus how many tags) he should compute. Clearly such counters should be encrypted and stored in the database, where the encryption key is kept by the user. Each encrypted counter should be kept together with the corresponding intermediate value (of the first occurrence of the cell value), so that it can be identified by the user. When the database intruder observes encrypted metadata, he does not know which cell value corresponds to which intermediate value and therefore does not know which cell value corresponds to the encrypted counter.

5.2 Solution Details

Metadata Format. To speed up queries on attribute A_j, the user picks keys $s_3, s_4, s_5 \in \{0,1\}^{k_0}$ independently and uniformly. For $i = 1, \ldots, n$, the user computes

$$H_{i,j} = E_{s_3}(T_{i,j}, 0).$$

For each value of each attribute, the user keeps a counter of the number of occurrences. If this is the $c_{i,j}$th occurrence of the value $T_{i,j}$ in the attribute A_j, the user computes

$$I_{i,j} = E_{s_4}((H_{i,j} + c_{i,j}) \bmod 2^{k_0}).$$

When $I_{i,j}$'s have been computed for all i's, suppose the final value of the counter is $c_j(v)$ for each value v. Then the user encrypts $c_j(v)$ using secret key k_5:

$$C_j(v) = E_{k_5}(c_j(v), 0).$$

The user stores $L = \{(I_{i,j}, \text{link to row } T'_i)\}_{i\in[1,n]}$ and

$$\begin{aligned} B &\triangleq \{(B_x\langle 1\rangle, B_x\langle 2\rangle)\}_{x\in[1,\ |\{T_{i,j}:i\in[1,n]\}|]} \\ &= \{(E_{s_3}(v,0), C_j(v))\}_{v\in\{T_{i,j}:i\in[1,n]\}} \end{aligned}$$

in the database as metadata for queries on attribute A_j. Note that L should be sorted in an increasing order of $I_{i,j}$. The user keeps s_3, s_4, and s_5 secret.

Query Protocol. Now suppose there is a query *select* A_{j_1},..., A_{j_ℓ} *from* T *where* $A_j = v$. To carry out this query, the user first computes $h = E_{s_3}(v, 0)$ and sends h to the database. The database finds x such that $B_x\langle 1\rangle = h$ and sends the corresponding $C = B_x\langle 2\rangle$ back to the user. The user then decrypts C (and discards the k_2-bit tail) to get $c_j(v)$, the overall number of occurrences of v. For $c = 1, \ldots, c_j(v)$, the user computes

$$I_c = E_{s_4}((h + c) \bmod 2^{k_0}),$$

and sends I_c to the database. Since L is sorted in the increasing order of I_c, the database can easily locate I_c and find the link corresponding to I_c. For each row T'_i pointed by these links, the database sends the encrypted cells $T'_{i,j_1}\langle 1\rangle, \ldots, T'_{i,j_\ell}\langle 1\rangle$ to the user. Finally, the user decrypts each received cell using secret key s_1 and discards the k_2-bit tail of the cleartext.

5.3 Performance Evaluation

To evaluate the speedup of our solution, we measured the query time on the same dataset used for testing the basic solution. Figure 5 compares the metadata generation time for four different attributes: ClassLabel, Finance, Parents, and ID. The metadata generation time depends on not only the number of rows in the table, but also the domain size of the attribute (more precisely, the number of the different values that actually appear in the attribute). In the attributes we experimented with, ClassLabel, Finance, and Parents have small domain sizes; the metadata generation time for each of them is about 6 seconds. In contrast, generating metadata on ID attribute needs about twice as much time because is the ID attribute has a large domain.

Figure 6 compares the query time of the basic solution and that of the solution with metadata. with the following four queries that are to select all records where Class=recommend, where Class=very_recom, where Parent=usual, and where ID=1000. The results of the first and the fourth queries have only 2 and 1 record, respectively. For such queries, the solution with metadata is so fast that the query time can hardly be seen in the figure. The other two queries have more records in their results: the second query has 328 records in its result and our solution with metadata saves about 94% of the query time; the third query has 4320 records in its result and our solution with metadata saves about 79% of the query time. Clearly, the trend is that the solution with metadata gains more in efficiency if there are fewer records in the query result. However, even for a query with a large number of records in the result, the solution with metadata is much faster than the basic solution.

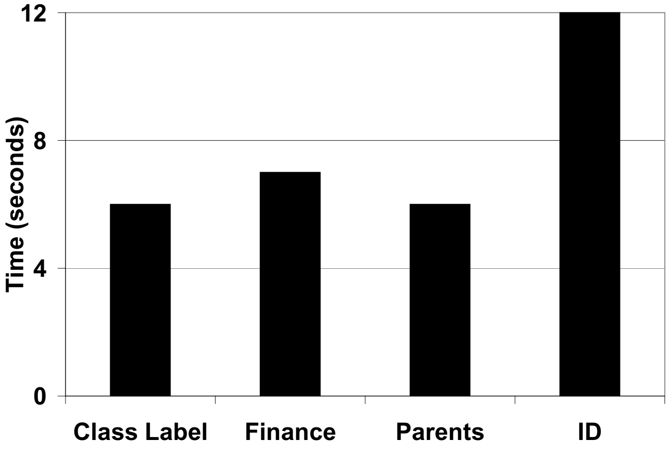


Fig. 5. Computational time to generate metadata

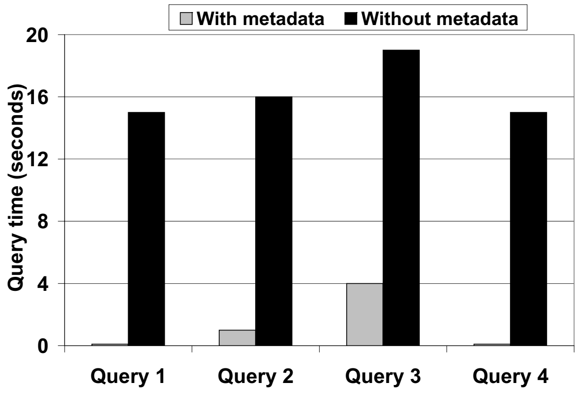


Fig. 6. Query time: solution with metadata vs. basic solution

6 Conclusions

In this paper, we have investigated privacy-preserving queries on encrypted data. In particular, we present privacy-preserving protocols for certain types of queries. Although the supported queries are limited, our main goal in this paper is to provide rigorous, quantitative (and cryptographically strong) security.

We note that, in general, it is difficult to evaluate the correctness and security of a new design if no quantitative analysis of information leakage is given. It is therefore beneficial to introduce quantitative measures of privacy such as those we have introduced. It is our hope that these measures may be useful elsewhere.

For many practical database applications, more complex queries than those considered in this paper must be supported. A future research topic is to extend the work in this paper to allow more complex queries. Ideally, the extension should maintain strong, quantifiable security while achieving efficiency for complex queries.

References

1. Oracle Corporation. Database Encryption in Oracle9i, 2001.
2. IBM Data Encryption for IMS and DB2 Databases, Version 1.1, 2003.
3. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *VLDB*, 2002.
4. Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *SIGMOD*, 2004.
5. C. Blake and C. Merz. UCI repository, 1998.
6. D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT*, 2004.
7. L. Bouganim and P. Pucheral. Chip-secured data access: Confidential data on untrusted servers. In *VLDB*, 2002.
8. C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT*, 1999.
9. Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. Cryptology ePrint Archive:2004/051, available at `http://eprint.iacr.org/2004/051.pdf`.
10. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, 1995.
11. Ernesto Damiani, S. De Capitani Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. In *CCS*, 2003.
12. Eric Dash. Lost credit data improperly kept, company admits. *New York Times*, June 20 2005.
13. G. I. Davida, D. L. Wells, and J. B. Kam. A database encryption system with subkeys. *ACM TODS*, 6(2):312–328, 1981.
14. J. Feigenbaum, M. Y. Liberman, and R. N. Wright. Cryptographic protection of databases and software. In *DIMACS Workshop on Distributed Computing and Cryptography*, 1990.
15. E. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216. `http://eprint.iacr.org/2003/216/`.
16. O. Goldreich. *Foundations of Cryptography*, volume 1. Cambridge University Press, 2001.
17. O. Goldreich. *Foundations of Cryptography*, volume 2. Cambridge University Press, 2004.
18. S. Goldwasser and M. Bellare. Lecture notes on cryptography. Summer Course Lecture Notes at MIT, 1999.
19. Hakan Hacigumus, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD*, 2002.
20. Hakan Hacigumus, Balakrishna R. Iyer, and Sharad Mehrotra. Providing database as a service. In *ICDE*, 2002.
21. Hakan Hacigumus, Balakrishna R. Iyer, and Sharad Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In *DASFAA*, 2004.
22. J. He and J. Wang. Cryptography and relational database management systems. In *Int. Database Engineering and Application Symposium*, 2001.
23. Bijit Hore, Sharad Mehrotra, and Gene Tsudik. A privacy-preserving index for range queries. In *VLDB*, 2004.
24. Bala Iyer, S. Mehrotra, E. Mykletun, G. Tsudik, and Y. Wu. A framework for efficient storage security in RDBMS. In *EDBT*, 2004.

25. J. Karlsson. Using encryption for secure data storage in mobile database systems. Friedrich-Schiller-Universitat Jena, 2002.
26. E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database computationally-private information retrieval. In *FOCS*, 1997.
27. Gultekin Ozsoyoglu, David Singer, and Sun Chung. Anti-tamper databases: Querying encrypted databases. In *Proc. of the 17th Annual IFIP WG 11.3 Working Conference on Database and Applications Security*, 2003.
28. Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, 2000.
29. David Stout. Veterans chief voices anger on data theft. *New York Times*, May 25 2006.
30. R. Vingralek. A small-footprint, secure database system. In *VLDB*, 2002.

Analysis of Policy Anomalies on Distributed Network Security Setups

J.G. Alfaro[1,2], F. Cuppens[1], and N. Cuppens-Boulahia[1]

[1] GET/ENST-Bretagne, 35576 Cesson Sévigné - France
{Frederic.Cuppens, Nora.Cuppens}@enst-bretagne.fr
[2] dEIC/UAB, Edifici Q, 08193, Bellaterra, Barcelona - Spain
Joaquin.Garcia-Alfaro@deic.uab.es

Abstract. The use of different network security components, such as *firewalls* and *network intrusion detection systems* (NIDSs), is the dominant method to survey and guarantee the security policy in current corporate networks. On the one hand, firewalls are traditional security components which provide means to filter traffic within corporate networks, as well as to police the incoming and outcoming interaction with the Internet. On the other hand, NIDSs are complementary security components used to enhance the visibility level of the network, pointing to malicious or anomalous traffic. To properly configure both firewalls and NIDSs, it is necessary to use several sets of filtering and alerting rules. Nevertheless, the existence of anomalies between those rules, particularly in distributed multi-component scenarios, is very likely to degrade the network security policy. The discovering and removal of these anomalies is a serious and complex problem to solve. In this paper, we present a set of algorithms for such a management.

1 Introduction

Generally, once a security administrator has specified a security policy, he or she aims to enforce it in the information system to be protected. This enforcement consists in distributing the security rules expressed in this policy over different security components of the information system – such as firewalls, intrusion detection systems (IDSs), intrusion prevention systems (IPSs), proxies, etc – both at application, system, and network level. This implies cohesion of the security functions supplied by these components. In other words, security rules deployed over the different components must be consistent, not redundant and, as far as possible, optimal.

An approach based on a formal security policy refinement mechanism (using for instance abstract machines grounded on set theory and first order logic) ensures cohesion, completeness and optimization as built-in properties. Unfortunately, in most cases, such an approach has not a wide follow and the policy is more often than not empirically deployed based on security administrator expertise and flair. It is then advisable to analyze the security rules deployed to detect and correct some policy anomalies – often referred in the literature as *intra- and inter-configuration anomalies* [4].

These anomalies might be the origin of security holes and/or heaviness of intrusion prevention and detection processes. Firewalls [6] and network intrusion detection systems (NIDSs) [12] are the most commonly used security components and, in this paper, we focus particularly on their security rules. Firewalls are prevention devices ensuring the access control. They manage the traffic between the public network and the private network zones on one hand and between private zones in the local network in the other hand. The undesirable traffic is blocked or deviated by such a component. NIDSs are detection devices ensuring a monitoring role. They are components that supervise the traffic and generate alerts in the case of suspicious traffic. The attributes used to block or to generate alerts are almost the same. The challenge, when these two kinds of components coexist in the security architecture of an information system is then to avoid inter-configuration anomalies.

In [7, 8], we presented an audit process to manage intra-firewall policy anomalies, in order to detect and remove anomalies within the set of rules of a given firewall. This audit process is based on the existence of relationships between the condition attributes of the filtering rules, such as coincidence, disjunction, and inclusion, and proposes a transformation process which derives from an initial set of rules – with potential policy anomalies – to an equivalent one which is completely free of errors. Furthermore, the resulting rules are completely disjoint, i.e., the ordering of rules is no longer relevant.

In this paper, we extend our proposal of detecting and removing intra-firewall policy anomalies [7, 8], to a distributed setup where both firewalls and NIDSs are in charge of the network security policy. This way, assuming that the role of both prevention and detection of network attacks is assigned to several components, our objective is to avoid intra and inter-component anomalies between filtering and alerting rules. The proposed approach is based on the similarity between the parameters of a filtering rule and those of an alerting rule. This way, we can check whether there are errors in those configurations regarding the policy deployment over each component which matches the same traffic.

The advantages of our proposal are threefold. First, as opposite to the related work we show in Section 6, our approach not only considers the analysis of relationships between rules two by two but also a complete analysis of the whole set of rules. This way, those conflicts due to the union of rules that are not detected by other proposals (such as [2, 3, 9]) are properly discovered by our intra- and inter-component algorithms. Second, after applying our intra-component algorithms the resulting rules of each component are totally disjoint, i.e., the ordering of rules is no longer relevant. Hence, one can perform a second rewriting of rules in a *close* or *open* manner, generating a configuration that only contains $deny$ (or $alert$) rules if the component default policy is open, and $accept$ (or $pass$) rules if the default policy is close (cf. Section 3.1). Third, we also present in this paper a network model to determine which components are crossed by a given packet knowing its source and destination, as well as other network properties. Thanks to this model, our approach better defines all the set of anomalies studied in the related work. Furthermore the lack of this model in other approaches, such as [2, 3], may lead to inappropriate decisions.

The rest of this paper is organized as follows. Section 2 starts by introducing a network model that is further used in Section 3 and Section 4 when presenting, respectively, our intra and inter-component anomaly's classifications and algorithms. Section 5 overviews the performance of our proposed algorithms. Section 6 shows an analysis of some related work. Finally Section 7 closes the paper with some conclusions and gives an outlook on future work.

2 Network Model

The purpose of our network model is to determine which components within the network are crossed by a given packet, knowing its source and destination. It is defined as follows. First, and concerning the traffic flowing from two different zones of the distributed policy scenario, we may determine the set of components that are crossed by this flow. Regarding the scenario shown in Figure 1, for example, the set of components crossed by the network traffic flowing from zone $external\ network$ to zone $private_3$ equals [C_1,C_2,C_4], and the set of components crossed by the network traffic flowing from zone $private_3$ to zone $private_2$ equals [C_4,C_2,C_3].

Let C be a set of components and let Z be a set of zones. We assume that each pair of zones in Z are mutually disjoint, i.e., if $z_i \in Z$ and $z_j \in Z$ then $z_i \cap z_j = \emptyset$. We then define the predicate $connected(c_1, c_2)$ as a symmetric and anti-reflexive function which becomes $true$ whether there exists, at least, one interface connecting component c_1 to component c_2. On the other hand, we define the predicate $adjacent(c, z)$ as a relation between components and zones which becomes $true$ whether the zone z is interfaced to component c. Referring to Figure 1, we can verify that predicates $connected(C_1, C_2)$ and $connected(C_1, C_3)$, as well as $adjacent(C_1, DMZ)$, $adjacent(C_2, private_1)$, $adjacent(C_3, DMZ)$, and so on, become $true$.

We then define the set of paths, P, as follows. If $c \in C$ then $[c] \in P$ is an atomic path. Similarly, if $[p.c_1] \in P$ (be "." a concatenation functor) and $c_2 \in C$, such that $c_2 \notin p$ and $connected(c_1, c_2)$, then $[p.c_1.c_2] \in P$. This way, we can notice that, concerning Figure 1, $[C_1, C_2, C_4] \in P$ and $[C_1, C_3] \in P$.

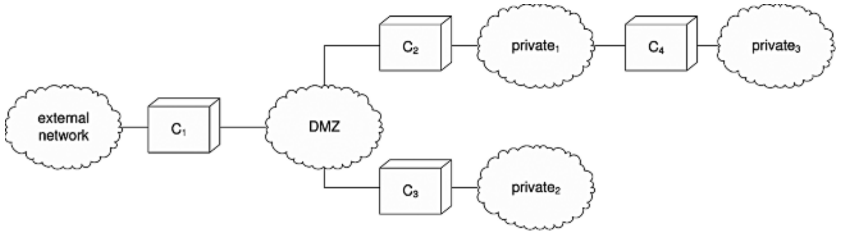


Fig. 1. Simple distributed policy setup

Let us now define a set of functions related with the order between paths. We first define functions $first$, $last$, and the order functor between paths. We define functions $first$ and $last$, respectively, from P in C, such that if p is a path, then $first(p)$ corresponds to the first component in the path, and $last(p)$ corresponds to the last component in the path. We then define the order functor between paths as $p_1 \leq p_2$, such that path p_1 is shorter than p_2, and where all the components within p_1 are also within p_2. We also define the predicates $isFirewall(c)$ and $isNIDS(c)$ which become $true$ whether the component c is, respectively, a firewall or a NIDS.

Two additional functions are $route$ and $minimal\_route$. We first define function $route$ from Z to Z in 2^P, such that $p \in route(z_1, z_2)$ iff the path p connects zone z_1 to zone z_2. Formally, we define that $p \in route(z_1, z_2)$ iff $adjacent(first(p), z_1)$ and $adjacent(last(p), z_2)$. Similarly, we then define $minimal\_route$ from Z to Z in 2^P, such that $p \in minimal\_route(z_1, z_2)$ iff the following conditions hold: (1) $p \in route(z_1, z_2)$; (2) There does not exist $p' \in route(z_1, z_2)$ such that $p' < p$. Regarding Figure 1, we can verify that the $minimal\_route$ from zone $private_3$ to zone $private_2$ equals $[C_4, C_2, C_3]$, i.e., $minimal\_route(private_3, private_2) = \{[C_4, C_2, C_3]\}$.

Let us finally conclude this section by defining the predicate *affects*(Z, A_c) as a boolean expression which becomes $true$ whether there is, at least, an element $z \in Z$ such that the configuration of z is vulnerable to the attack category $A_c \in V$, where V is a vulnerability set built from a vulnerability database, such as CVE/CAN [11] or OSVDB [13].

3 Intra-component Analysis

In this section we extend our previous work on analysis of network access control rules for the configuration of firewalls [7, 8], concentrating on anomalies that may also arise in NIDS setups. In particular, a new case of anomaly is pointed out (cf. Intra-Component Irrelevance) and the associated code of our intra-component algorithms has been properly revised[1].

For our work, we define the security rules of both firewalls and NIDSs as filtering and alerting rules, respectively. In turn, both filtering and alerting rules are specific cases of a more general configuration rule, which typically defines a $decision$ (such as $deny$, $alert$, $accept$, or $pass$) that applies over a set of $condition$ attributes, such as *protocol*, *source*, *destination*, *classification*, and so on. We define a general configuration rule as follows:

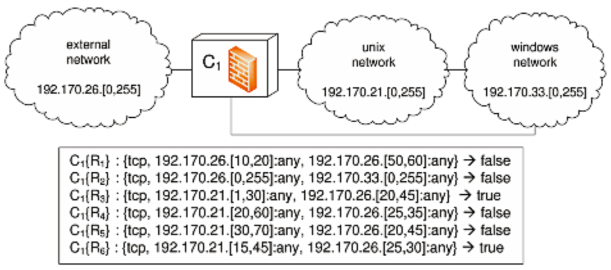
$$R_i : \{condition_i\} \rightarrow decision_i \tag{1}$$

where i is the relative position of the rule within the set of rules, $\{condition_i\}$ is the conjunctive set of condition attributes such that $\{condition_i\}$ equals $C_1 \wedge C_2 \wedge ... \wedge C_p$ – being p the number of condition attributes of the given rule – and $decision$ is a boolean value in $\{true, false\}$.

We shall notice that the decision of a filtering rule will be positive ($true$) whether it applies to a specific value related to *deny* (or *filter*) the traffic it matches, and will be negative ($false$) whether it applies to a specific value related to *accept* (or *ignore*) the traffic it matches. Similarly, the decision of an alerting rule will be positive ($true$) whether it applies to a specific value related to *alert* (or *warn*) about the traffic it matches, and will be negative ($false$) whether it applies to a specific value related to *pass* (or *ignore*) the traffic it matches.

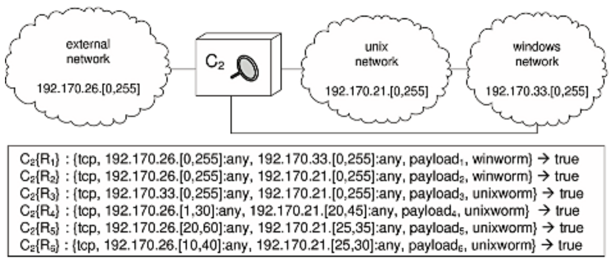
[1] Because of the similarity between the revision and the previous work already covered in [7, 8], we move our intra-component audit algorithms to Appendix A. Their correctness and complexity can also be found in [7, 8].

Let us continue by classifying the complete set of anomalies that can occur within a single component configuration. An example for each anomaly will be illustrated through the sample scenario shown in Figure 2.



$C_1\{R_1\}$: {tcp, 192.170.26.[10,20]:any, 192.170.26.[50,60]:any} → false
$C_1\{R_2\}$: {tcp, 192.170.26.[0,255]:any, 192.170.33.[0,255]:any} → false
$C_1\{R_3\}$: {tcp, 192.170.21.[1,30]:any, 192.170.26.[20,45]:any} → true
$C_1\{R_4\}$: {tcp, 192.170.21.[20,60]:any, 192.170.26.[25,35]:any} → false
$C_1\{R_5\}$: {tcp, 192.170.21.[30,70]:any, 192.170.26.[20,45]:any} → false
$C_1\{R_6\}$: {tcp, 192.170.21.[15,45]:any, 192.170.26.[25,30]:any} → true

(a) Example scenario of a filtering policy.



$C_2\{R_1\}$: {tcp, 192.170.26.[0,255]:any, 192.170.33.[0,255]:any, $payload_1$, winworm} → true
$C_2\{R_2\}$: {tcp, 192.170.26.[0,255]:any, 192.170.21.[0,255]:any, $payload_2$, winworm} → true
$C_2\{R_3\}$: {tcp, 192.170.33.[0,255]:any, 192.170.21.[0,255]:any, $payload_3$, unixworm} → true
$C_2\{R_4\}$: {tcp, 192.170.26.[1,30]:any, 192.170.21.[20,45]:any, $payload_4$, unixworm} → true
$C_2\{R_5\}$: {tcp, 192.170.26.[20,60]:any, 192.170.21.[25,35]:any, $payload_5$, unixworm} → true
$C_2\{R_6\}$: {tcp, 192.170.26.[10,40]:any, 192.170.21.[25,30]:any, $payload_6$, unixworm} → true

(b) Example scenario of an alerting policy.

Fig. 2. Example filtering and alerting policies

Intra-component Shadowing. A configuration rule R_i is shadowed in a set of configuration rules R whether such a rule never applies because all the packets that R_i may match, are previously matched by another rule, or combination of rules, with higher priority. Regarding Figure 2, rule $C_1\{R_6\}$ is shadowed by the union of rules $C_1\{R_3\}$ and $C_1\{R_5\}$.

Intra-component Redundancy. A configuration rule R_i is redundant in a set of configuration rules R whether the following conditions hold: (1) R_i is not shadowed by any other rule or set of rules; (2) when removing R_i from R, the security policy does not change. For instance, referring to Figure 2, rule $C_1\{R_4\}$ is redundant, since the overlapping between rules $C_1\{R_3\}$ and $C_1\{R_5\}$ is equivalent to the police of rule $C_1\{R_4\}$.

Intra-component Irrelevance. A configuration rule R_i is irrelevant in a set of configuration rules R if one of the following conditions holds:

(1) Both source and destination address are within the same zone. For instance, rule $C_1\{R_1\}$ is irrelevant since the source of its address, $external\ network$, as well as its destination, is the same.

(2) The component is not within the minimal route that connects the source zone, concerning the irrelevant rule which causes the anomaly, to the destination zone. Hence, the rule is irrelevant since it matches traffic which does not flow through this component. Rule $C_1\{R_2\}$, for example, is irrelevant since component C_1 is not in the path which corresponds to the minimal route between the source zone $unix\ network$ to the destination zone $windows\ network$.

(3) The component is a nids, i.e., the predicate $isNIDS(c)$ (cf. Section 2) becomes $true$, and, at least, one of the condition attributes in R_i is related with a classification of attack A_c which does not affect the destination zone of such a rule – i.e., the predicate affects(z_d, A_c) becomes $false$. Regarding Figure 2, we can see that rule $C_2\{R_2\}$ is irrelevant since the nodes in the destination zone $unix\ network$ are not affected by vulnerabilities classified as $winnuke$.

3.1 Default Policies

Each component implements a positive (i.e., close) or negative (i.e., open) default policy. In the positive policy, the default policy is to $alert$ or to $deny$ a packet when any configuration rule applies. Conversely, the negative policy will $accept$ or $pass$ a packet when no rule applies.

After rewriting the rules with the intra-component-audit algorithms (cf. Appendix A), we can actually remove every rule whose decision is *pass* or *accept* if the default policy of this component is negative (else this rule is redundant with the default policy) and, similarly, we can remove every rule whose decision is *deny* or *alert* if the default policy is positive. Thus, we can consider that our proposed *intra-component-audit* algorithm generates a configuration that only contains positive rules if the component default policy is negative, and negative rules if the default policy is positive.

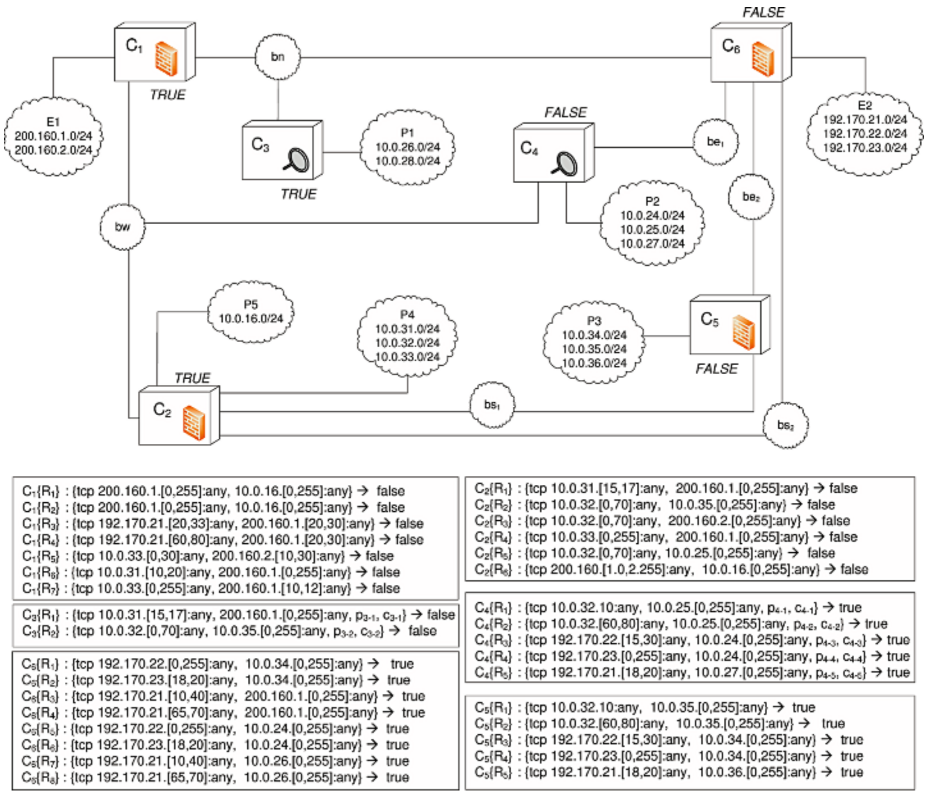
4 Inter-component Analysis

The objective of the inter-component audit algorithms is the complete detection of policy anomalies that could exist in a multi-component policy, i.e., to discover and warn the security officer about potential anomalies between policies of different components.

The main hypotheses to deploy our algorithms hold the following: (1) An upstream network traffic flows away from the closest component to the origin of this traffic (i.e., the most-upstream component [3]) towards the closest component to the remote destination (i.e., the most-downstream component [3]); (2) Every component's policy in the network has been rewritten using the intra-component algorithms defined in Appendix A, i.e., it does not contain intra-component anomalies and the rules within such a policy are completely independent between them.

4.1 Inter-component Anomalies Classification

In this section, we classify the complete set of anomalies that can occur within a multi-component policy. Our classification is based on the network model presented in Section 2. An example for each anomaly will be illustrated through the distributed multi-component policy setup shown in Figure 3.



$C_1\{R_1\}$: {tcp 200.160.1.[0,255]:any, 10.0.16.[0,255]:any} → false
$C_1\{R_2\}$: {tcp 200.160.1.[0,255]:any, 10.0.16.[0,255]:any} → false
$C_1\{R_3\}$: {tcp 192.170.21.[20,33]:any, 200.160.1.[20,30]:any} → false
$C_1\{R_4\}$: {tcp 192.170.21.[60,80]:any, 200.160.1.[20,30]:any} → false
$C_1\{R_5\}$: {tcp 10.0.33.[0,30]:any, 200.160.2.[10,30]:any} → false
$C_1\{R_6\}$: {tcp 10.0.31.[10,20]:any, 200.160.1.[0,255]:any} → false
$C_1\{R_7\}$: {tcp 10.0.33.[0,255]:any, 200.160.1.[10,12]:any} → false

$C_3\{R_1\}$: {tcp 10.0.31.[15,17]:any, 200.160.1.[0,255]:any, $p_{3\text{-}1}$, $c_{3\text{-}1}$} → false
$C_3\{R_2\}$: {tcp 10.0.32.[0,70]:any, 10.0.35.[0,255]:any, $p_{3\text{-}2}$, $c_{3\text{-}2}$} → false

$C_6\{R_1\}$: {tcp 192.170.22.[0,255]:any, 10.0.34.[0,255]:any} → true
$C_6\{R_2\}$: {tcp 192.170.23.[18,20]:any, 10.0.34.[0,255]:any} → true
$C_6\{R_3\}$: {tcp 192.170.21.[10,40]:any, 200.160.1.[0,255]:any} → true
$C_6\{R_4\}$: {tcp 192.170.21.[65,70]:any, 200.160.1.[0,255]:any} → true
$C_6\{R_5\}$: {tcp 192.170.22.[0,255]:any, 10.0.24.[0,255]:any} → true
$C_6\{R_6\}$: {tcp 192.170.23.[18,20]:any, 10.0.24.[0,255]:any} → true
$C_6\{R_7\}$: {tcp 192.170.21.[10,40]:any, 10.0.26.[0,255]:any} → true
$C_6\{R_8\}$: {tcp 192.170.21.[65,70]:any, 10.0.26.[0,255]:any} → true

$C_2\{R_1\}$: {tcp 10.0.31.[15,17]:any, 200.160.1.[0,255]:any} → false
$C_2\{R_2\}$: {tcp 10.0.32.[0,70]:any, 10.0.35.[0,255]:any} → false
$C_2\{R_3\}$: {tcp 10.0.32.[0,70]:any, 200.160.2.[0,255]:any} → false
$C_2\{R_4\}$: {tcp 10.0.33.[0,255]:any, 200.160.1.[0,255]:any} → false
$C_2\{R_5\}$: {tcp 10.0.32.[0,70]:any, 10.0.25.[0,255]:any} → false
$C_2\{R_6\}$: {tcp 200.160.[1.0,2.255]:any, 10.0.16.[0,255]:any} → false

$C_4\{R_1\}$: {tcp 10.0.32.10:any, 10.0.25.[0,255]:any, $p_{4\text{-}1}$, $c_{4\text{-}1}$} → true
$C_4\{R_2\}$: {tcp 10.0.32.[60,80]:any, 10.0.25.[0,255]:any, $p_{4\text{-}2}$, $c_{4\text{-}2}$} → true
$C_4\{R_3\}$: {tcp 192.170.22.[15,30]:any, 10.0.24.[0,255]:any, $p_{4\text{-}3}$, $c_{4\text{-}3}$} → true
$C_4\{R_4\}$: {tcp 192.170.23.[0,255]:any, 10.0.24.[0,255]:any, $p_{4\text{-}4}$, $c_{4\text{-}4}$} → true
$C_4\{R_5\}$: {tcp 192.170.21.[18,20]:any, 10.0.27.[0,255]:any, $p_{4\text{-}5}$, $c_{4\text{-}5}$} → true

$C_5\{R_1\}$: {tcp 10.0.32.10:any, 10.0.35.[0,255]:any} → true
$C_5\{R_2\}$: {tcp 10.0.32.[60,80]:any, 10.0.35.[0,255]:any} → true
$C_5\{R_3\}$: {tcp 192.170.22.[15,30]:any, 10.0.34.[0,255]:any} → true
$C_5\{R_4\}$: {tcp 192.170.23.[0,255]:any, 10.0.34.[0,255]:any} → true
$C_5\{R_5\}$: {tcp 192.170.21.[18,20]:any, 10.0.36.[0,255]:any} → true

Fig. 3. An example for a distributed network policy setup

Inter-component Shadowing. A shadowing anomaly occurs between two components whether the following conditions hold: (1) The most-upstream component is a firewall; (2) The downstream component, where the anomaly is detected, does not block or report (completely or partially) traffic that is blocked (explicitly, by means of positive rules; or implicitly, by means of its default policy), by the most-upstream component.

The explicit shadowing as result of the union of rules $C_6\{R_7\}$ and $C_6\{R_8\}$ to the traffic that the component C_3 matches by means of rule $C_3\{R_1\}$ is a proper example of *full shadowing* between a firewall and a NIDS. Similarly, the anomaly between rules $C_3\{R_2\}$ and $C_6\{R_8\}$ shows an example of an *explicit partial shadowing* anomaly between a firewall and a NIDS.

On the other hand, the implicit shadowing between the rule $C_1\{R_5\}$ and the default policy of component C_2 is a proper example of *implicit full shadowing* between two firewalls. Finally, the anomaly between the rule $C_1\{R_6\}$, $C_2\{R_1\}$, and the default policy of component C_2 shows an example of an *implicit partial shadowing* anomaly between two firewalls.

Inter-component Redundancy. A redundancy anomaly occurs between two components whether the following conditions hold: (1) The most-upstream component is a firewall; (2) The downstream component, where the anomaly is detected, blocks or reports (completely or partially) traffic that is blocked by the most-upstream component.

Rules $C_5\{R_3\}$ and $C_6\{R_1\}$ show a proper example of *full redundancy* between two firewalls, whereas rules $C_4\{R_3\}$ and $C_6\{R_5\}$ show an example of *full redundancy* between a firewall and a NIDS. Similarly, rules $C_5\{R_4\}$ and $C_6\{R_2\}$ show a proper example of *partial redundancy* between two firewalls, whereas rules $C_4\{R_4\}$ and $C_6\{R_6\}$ show an example of *partial redundancy* between a firewall and a NIDS.

Sometimes, this kind of redundancy is expressly introduced by network administrators (e.g., to guarantee the forbidden traffic will not reach the destination). Nonetheless, it is important to discover it since, if such a rule is applied, we may conclude that at least one of the redundant components is wrongly working.

Inter-component Misconnection. A misconnection anomaly occurs between two components whether the following conditions hold: (1) The most-upstream component is a firewall; (2) the most-upstream firewall permits (explicitly, by means of negative rules; or implicitly, through its default policy) all the traffic – or just a part of it – that is denied or alerted by a downstream component.

An explicit misconnection anomaly between two firewalls is shown through the rules $C_5\{R_1\}$ and $C_2\{R_2\}$ (*full misconnection*); and the rules $C_5\{R_2\}$ and $C_2\{R_2\}$ (*partial misconnection*). An implicit misconnection anomaly between two firewalls is also shown by the rule $C_1\{R_5\}$ and the default policy of firewall C_2 (*full misconnection*); and the rules $C_1\{R_6\}$ and $C_2\{R_1\}$, together with the default policy of C_2 (*partial misconnection*). Similarly, the pair of rules $C_4\{R_1\}$-$C_2\{R_5\}$ and the pair of rules $C_4\{R_2\}$-$C_2\{R_5\}$ show, respectively, an explicit example of full and partial misconnection anomaly between a firewall and a NIDS. Finally, the rule $C_4\{R_5\}$ together with the negative policy of the firewall C_2 shows an example of implicit misconnection anomaly between a firewall and a NIDS.

4.2 Inter-component Analysis Algorithms

For reasons of clarity, we split the whole analysis process in four different algorithms. The input for the first algorithm (cf. Algorithm 5) is the set of components C, such that for all $c \in C$, we note $c[rules]$ as the set of configuration rules of component c, and $c[policy] \in \{true, false\}$ as the default policy of such a component c. In turn, each rule $r \in c[rules]$ consists of a boolean expression over the attributes $szone$ (source zone), $dzone$ (destination zone), $sport$ (source port), $dport$ (destination port), $protocol$, and $decision$ (true or false).

Let us recall here the functions $source(r) = szone$ and $dest(r) = dzone$. Thus, we compute for each component $c \in C$ and for each rule $r \in c[rules]$, each one of the source zones $z_1 \in Z_s$ and destination zones $z_2 \in Z_d$ – whose intersection with respectively $szone$ and $dzone$ is not empty – which become, together with a reference to each component c and each rule r, the input for the second algorithm (i.e., Algorithm 6).

Once in Algorithm 6, we compute the minimal route of components that connects zone z_1 to z_2, i.e., $[C_1, C_2, \ldots, C_n] \in minimal\_route(z_1, z_2)$. Then, we decompose the set of components inside each path in downstream path ($path_d$) and upstream path ($path_u$). To do so, we use the implicit functions $head$ and $tail$. The first component $c_d \in path_d$, and the last component $c_u \in path_u$ are passed, respectively, as argument to the last two algorithms (i.e., Algorithm 7 and Algorithm 8) in order to conclude the set of necessary checks that guarantee the audit process[2].

Let us conclude by giving an outlook in Figure 4 to the set of warnings after the execution of Algorithm 5 over the scenario of Figure 3:

$C_1\{R_3\} - C_6\{R_3, R_4\}$: Full Shadowing
$C_1\{R_4\} - C_6\{R_4\}$: Partial Shadowing
$C_1\{R_5\} - C_2\{pol.\}$: Full Shadowing
$C_1\{R_6\} - C_2\{R_1, pol.\}$: Partial Shadowing
$C_2\{R_3\} - C_1\{pol.\}$: Full Misconnection
$C_2\{R_4\} - C_1\{R_7, pol.\}$: Partial Misconnection
$C_3\{R_1\} - C_6\{R_7, R_8\}$: Full Shadowing
$C_3\{R_2\} - C_6\{R_8\}$: Partial Shadowing
$C_4\{R_1\} - C_2\{R_5\}$: Full Misconnection

$C_4\{R_2\} - C_2\{R_5\}$: Partial Misconnection
$C_4\{R_3\} - C_6\{R_5\}$: Full Redundancy
$C_4\{R_4\} - C_6\{R_6\}$: Partial Redundancy
$C_4\{R_5\} - C_6\{pol.\}$: Full Misconnection
$C_5\{R_1\} - C_2\{R_2\}$: Full Misconnection
$C_5\{R_2\} - C_2\{R_2\}$: Partial Misconnection
$C_5\{R_3\} - C_6\{R_1\}$: Full Redundancy
$C_5\{R_4\} - C_6\{R_2\}$: Partial Redundancy
$C_5\{R_5\} - C_6\{pol.\}$: Full Misconnection

Fig. 4. Execution of Algorithm 5 over the scenario of Figure 3

Algorithm 5: inter-component-audit(C)

```
foreach $c \in C$ do
    foreach $r \in c[rules]$ do
        $Z_s \leftarrow \{z \in Z \mid z \cap$ source $(r) \neq \emptyset\}$;
        $Z_d \leftarrow \{z \in Z \mid z \cap$ dest $(r) \neq \emptyset\}$;
        foreach $z_1 \in Z_s$ do
            foreach $z_2 \in Z_d$ do
                audit $(c,r,z_1,z_2)$;
```

Algorithm 6: audit(c,r,z_1,z_2)

```
foreach $p \in$ minimal_route $(z_1,z_2)$ do
    $path_d \leftarrow$ tail $(p,c)$;
    $path_u \leftarrow$ header $(p,c)$;
    if $path_d \neq \emptyset$ and $r[decision] = "false"$
    and isFirewall $(c)$ then
        $c_d \leftarrow$ first$(path_d)$;
        downstream $(r,c,c_d)$;
    if $path_u \neq \emptyset$ then
        $c_u \leftarrow$ last$(path_u)$;
        if isFirewall $(c_u)$ then
            upstream $(r,c,c_u)$;
```

Algorithm 7: downstream(r,c,c_d)

```
if $c_d[policy] = true$ then
    $R_{df} \leftarrow \{r_d \in c_d \mid r_d \backsim r \wedge r_d[decision] = false\}$;
    if $R_{df} = \emptyset$ then warning
    ("Full Misconnection");
    else if $\neg$ testRedundancy $(R_{df},r)$ then
        warning ("Partial Misconnection");
```

Algorithm 8: upstream(r,c,c_u)

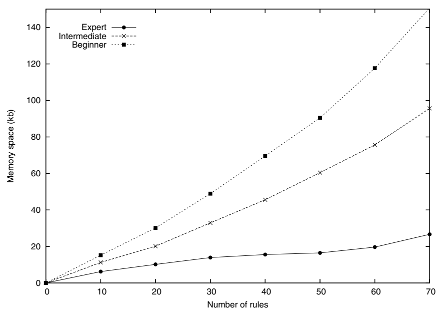
```
$R_{uf} \leftarrow \{r_u \in c_u \mid r_u \backsim r \wedge r_u[decision] = false\}$;
$R_{ut} \leftarrow \{r_u \in c_u \mid r_u \backsim r \wedge r_u[decision] = true\}$;
if $r[decision] =$ "true" then
    if testRedundancy $(R_{uf},r)$ then
        warning ("Full Spurious");
    else if $R_{ua} \neq \emptyset$ then
        warning ("Partial Spurious");
    else if testRedundancy $(R_{ut},r)$ then
        warning ("Full Redundancy");
    else if $R_{ut} \neq \emptyset$ then
        warning ("Partial Redundancy");
    else if $R_{uf} = \emptyset$ and $R_{ut} = \emptyset$
    and $c_u[policy] = false$ then
        warning ("Full Misconnection");
else
    if testRedundancy $(R_{ut},r)$ then
        warning ("Full Shadowing");
    else if $R_{ut} \neq \emptyset)$ then
        warning ("Partial Shadowing");
    else if $R_{uf} = \emptyset$ and $c_u[policy] = true$ then
        warning ("Full Shadowing");
    else if $\neg$ testRedundancy $(R_{uf},r)$
    and $c_u[policy] = true$ then
        warning ("Partial Shadowing");
```

[2] The operator "$\backsim$" within algorithms 7 and 8 denotes that two rules r_i and r_j are correlated if every attribute in R_i has a non empty intersection with the corresponding attribute in R_j.

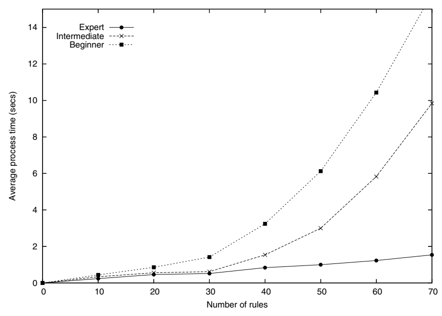
5 Performance Evaluation

In this section, we present an evaluation of the performance of MIRAGE (which stands for MIsconfiguRAtion manaGEr), a software prototype that implements the intra and inter-firewall algorithms presented in sections 3 and 4. MIRAGE has been developed using PHP, a scripting language that is especially suited for web services development and can be embedded into HTML for the construction of client-side GUI based applications [5]. MIRAGE can be locally or remotely executed by using a HTTP server and a web browser.

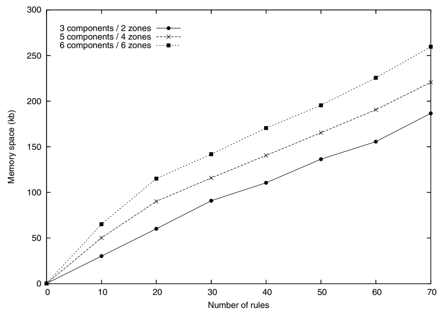
Inspired by the experiments done in [3], we evaluated our algorithms through a set of experiments over two different IPv4 real networks. The topology for the first network consisted of a single firewall based Netfilter [16], and a single NIDS based on Snort [15] – connected to three different zones with more than 50 hosts. The topology for the second network consisted of six different components – based on netfilter, ipfilter [14], and snort [15] – protecting six different zones with more than 200 hosts. The whole of these experiments were carried out on an Intel-Pentium M 1.4 GHz processor with



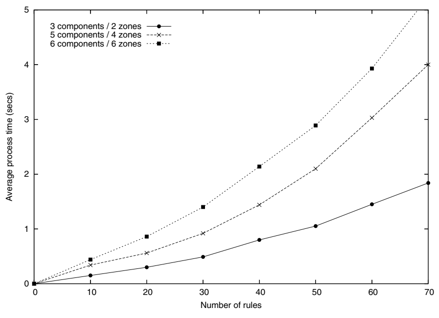
(a) Memory space evaluation.



(b) Processing time evaluation.



(c) Memory space evaluation.



(d) Processing time evaluation.

Fig. 5. Evaluation of our set of intra- and inter-component algorithms

512 MB RAM, running Debian GNU/Linux 2.6.8, and using Apache/1.3 with PHP/4.3 configured.

During a first phase, we measured the memory space and the processing time needed to perform Algorithm 4 over several sets of IPv4 policies for the first IPv4 network, according to the three following security officer profiles: beginner, intermediate, and expert – where the probability to have overlaps between rules increases from 5% to 90%. The results of these measurements are plotted in Figure 5(a) and Figure 5(b). Though those plots reflect strong memory and process time requirements, we consider they are reasonable for off-line analysis, since it is not part of the critical performance of a single component.

We conducted, in a second phase, similar experiments to measure the performance and scalability of Algorithm 5 through a progressive increment of auto-generated rules, firewalls and zones for the second network. The results of these measurements are plotted in Figure 5(c) and Figure 5(d). Similarly to the intra-component evaluation, we consider these requirements very reasonable for off-line inter-component analysis.

6 Related Work

Some related proposals to our work, such as [1, 9, 2, 10, 3, 4], provide means to directly manage the discovery of anomalies from the components' configuration. For instance, the authors in [1] consider that, in a configuration set, two rules are in conflict when the first rule in order matches some packets that match the second rule, and the second rule also matches some of the packets that match the first rule. This approach is very limited since it just detects a particular case of ambiguity within a single component configuration. Furthermore, it does not provide detection on multiple-component configurations.

In [9], two cases of anomalies are considered. First, a rule R_j is defined as backward redundant iff there exists another rule R_i with higher priority in order such that all the packets that match rule R_j also match rule R_i. Second, a rule R_i is defined as forward redundant iff there exists another rule R_j with the same decision and less priority in order such that the following conditions hold: (1) all the packets that match R_i also match R_j; (2) for each rule R_k between R_i and R_j, and that matches all the packets that also match rule R_i, R_k has the same decision as R_i. Although this approach seems to head in the right direction, we consider it as incomplete, since it does not detect all the possible cases of intra-component anomalies (as we define in this paper). For instance, given the set of rules shown in Figure 6(a), since R_2 comes after R_1, rule R_2 only applies over the interval $[51, 70]$ – i.e., R_2 is not necessary, since, if we remove this rule from the configuration, the filtering policy does not change. The detection proposal, as defined in [9], cannot detect the redundancy of rule R_2 within the configuration of such a given firewall. Furthermore, neither [9] nor [10] provide detection on multiple-component configurations.

To our best knowledge, the approach presented in [2, 3, 4] propose the most efficient set of techniques and algorithms to detect policy anomalies in both single and multi-firewall configuration setups. In addition to the discovery process, their approach also

$$\begin{aligned} R_1 &: s \in [10,50] \rightarrow \mathit{deny} \\ R_2 &: s \in [40,70] \rightarrow \mathit{accept} \\ R_3 &: s \in [50,80] \rightarrow \mathit{accept} \end{aligned}$$

(a) Set of rules A

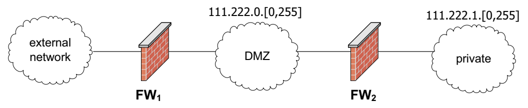
$$\begin{aligned} R_1 &: s \in [10,50] \rightarrow \mathit{accept} \\ R_2 &: s \in [40,90] \rightarrow \mathit{accept} \\ R_3 &: s \in [30,80] \rightarrow \mathit{deny} \end{aligned}$$

(b) Set of rules B

Fig. 6. Example of some firewall configurations

attempts an optimal insertion of arbitrary rules into an existing configuration, through a tree based representation of the filtering criteria. Nonetheless, and even though the efficiency of their proposed discovering algorithms and techniques is very promising, we also consider this approach as incomplete. First, their intra- and inter-component discovery approach is not complete since, given a single- or multiple-component security policy, their detection algorithms are based on the analysis of relationships between rules two by two. This way, errors due to the union of rules are not explicitly considered (as our approach does). The set of rules shown in Figure 6(b), for example, may lead their discovery algorithms to inappropriate decisions. The approach defined in [2] cannot detect that rule R_3 will be never applied due to the union of rules R_1 and R_2. Just a correlation signal – that is obviously a weaker signal than a shadowing one – would be labeled. Though in [3] the authors pointed out to this problematic, claiming that they break down the initial set of rules into an equivalent set of rules free of overlaps between rules, no specific algorithms have been provided for solving it in [2, 3, 4].

Second, their inter-component discovery approach considers as anomalies some situations that, from our point of view, must be suited to avoid inconsistent decisions between components used in the same policy to control or survey to different zones. For instance, given the following scenario:



$FW_1\{R_1\}$: p = tcp $\wedge$ s $\in$ any $\wedge$ d $\in$ 111.222.1.0/24 $\wedge$ dport = 80 $\rightarrow$ deny

$FW_2\{R_1\}$: p = tcp $\wedge$ s $\in$ 111.222.1.0/24 $\wedge$ d $\in$ 111.222.1.0/24 $\wedge$ dport = 80 $\rightarrow$ deny

their algorithms will inappropriately report a redundancy anomaly between filtering rules $FW_1\{R_1\}$ and $FW_2\{R_1\}$. This is because rule $FW_1\{R_1\}$ matches every packet that also $FW_2\{R_1\}$ does. As a consequence, [2] considers rule $FW_2\{R_1\}$ as redundant since packets denied by this rule are already denied by rule $FW_1\{R_1\}$. However, this conclusion is not appropriate because rule $FW_1\{R_1\}$ applies to packets from the external zone to the private zone whereas rule $FW_2\{R_1\}$ applies to packets from the DMZ zone to the private zone. So, rule $FW_2\{R_1\}$ is useful and cannot be removed. Though in [2, 3] the authors claim that their analysis technique marks every rule that is used on a network path, no specific algorithms have been provided for doing so. The main advantage of our proposal over their approach is that it includes a model of the traffic which flows through each component. We consider this is necessary to draw the right conclusion in this case.

Finally, although in [4] the authors consider their work as sufficiently general to be used for verifying many other filtering based security policies such as intrusion detection and prevention systems, no specific mechanisms have been provided for doing so.

7 Conclusions

In this paper we presented an audit process to set a distributed security scenario composed of both *firewalls* and *network intrusion detection systems* (NIDSs) free of anomalies. Our audit process has been presented in two main blocks. We first presented, in Section 3, a set of algorithms for intra-component analysis, according to the discovering and removal of policy anomalies over single-component environments. We then presented, in Section 4, a set of algorithms for inter-component analysis, in order to detect and warn the security officer about the complete existence of anomalies over a multi-component environment.

Some advantages of our approach are the following. First, our intra-firewall transformation process verifies that the resulting rules are completely independent between them. Otherwise, each rule considered as useless during the process is reported to the security officer, in order to verify the correctness of the whole process. Second, we can perform a second rewriting of rules, generating a configuration that only contains positive rules if the component default policy is negative, and negative rules if the default policy is positive. Third, the network model presented in Section 2 allows us to determine which components are crossed by a given packet knowing its source and destination, as well as other network properties. Thanks to this model, our approach better defines all the set of anomalies studied in the related work, and it reports, moreover, two new anomalies (*irrelevance* and *misconnection*) not reported, as defined in this paper, in none of the other approaches. Furthermore, and as pointed out in Section 6, the lack of this model in [2, 3, 4] leads to inappropriate decisions.

The implementation of our approach in a software prototype demonstrates the practicability of our work. We shortly discussed this implementation, based on a scripting language [5], and presented an evaluation of its performance. Although these experimental results show that our algorithms have strong requirements, we believe that these requirements are reasonable for off-line analysis, since it is not part of the critical performance of the audited component.

As future work, we are currently studying the anomaly problems of security rules in the case where the security architecture includes firewalls, IDS/IPS, and IPSec devices. Though there is a real similarity between the parameters of those devices' rules, more investigation has to be done in order to extend our proposal. In parallel to this work, we are also considering to extend our approach to the analysis of stateful policies.

Acknowledgements

This work was supported by funding from the French ministry of research, under the *ACI DESIRS* project, the Spanish Government project *TIC2003-02041*, and the Catalan Government grants *2003FI126* and *2005BE77*.

References

1. Adiseshu, H., Suri, S., and Parulkar, G. (2000). Detecting and Resolving Packet Filter Conflicts. *In 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, 1203–1212.
2. Al-Shaer, E. S., Hamed, H. H. (2004). Discovery of Policy Anomalies in Distributed Firewalls. In *IEEE INFOCOM'04*, March.
3. Al-Shaer, E. S., Hamed, H. H., and Masum, H. (2005). Conflict Classification and Analysis of Distributed Firewall Policies. In *IEEE Journal on Selected Areas in Communications*, 23(10).
4. Al-Shaer, E. S., Hamed, H. H. (2006). Taxonomy of Conflicts in Network Security Policies. In *IEEE Communications Magazine*, 44(3), March.
5. Castagnetto, J. et al. (1999). *Professional PHP Programming*. Wrox Press Inc, ISBN 1-86100-296-3.
6. Cheswick, W. R., Bellovin, S. M., Rubin A. D. (2003). *Firewalls and Internet security: repelling the wily hacker*. Addison-Wesley, second edition.
7. Cuppens, F., Cuppens-Boulahia, N., and Garcia-Alfaro, J. (2005). Detection and Removal of Firewall Misconfiguration. In *Proceedings of the 2005 IASTED International Conference on Communication, Network and Information Security*, 154–162.
8. Cuppens, F., Cuppens-Boulahia, N., and Garcia-Alfaro, J. (2005). Misconfiguration Management of Network Security Components. In *Proceedings of the 7th International Symposium on System and Information Security*, Sao Paulo, Brazil.
9. Gupta, P. (2000). *Algorithms for Routing Lookups and Packet Classification*. PhD Thesis, Department of Computer Science, Stanford University.
10. Gouda, M. G. and Liu, A. X. (2004). Firewall Design: Consistency, Completeness and Compactness. In *24th IEEE International Conference on Distributed Computing Systems (ICDCS-04)*, pages 320–327.
11. MITRE Corp. Common Vulnerabilities and Exposures. [Online]. Available from: `http://cve.mitre.org/`
12. Northcutt, S. (2002). *Network Intrusion Detection: An analyst's Hand Book*. New Riders Publishing, third edition.
13. Open Security Foundation. Open Source Vulnerability Database. [Online]. Available from: `http://osvdb.org/`
14. Reed, D. IP Filter. [Online]. Available from: `http://www.ja.net/CERT/Software/ipfilter/ip-filter.html`
15. Roesch, M. (1999), Snort: lightweight intrusion detection for networks. In *13th USENIX Systems Administration Conference*, Seattle, WA.
16. Welte, H., Kadlecsik, J., Josefsson, M., McHardy, P., and et al. The netfilter project: firewalling, nat and packet mangling for linux 2.4x and 2.6.x. [Online]. Available from: `http://www.netfilter.org/`

A Intra-component Algorithms

Our proposed audit process is a way to alert the security officer in charge of the network about these configuration errors, as well as to remove all the useless rules in the initial firewall configuration. The data to be used for the detection process is the following. A set of rules R as a list of initial size n, where n equals $count(R)$, and where each element is an associative array with the strings $condition$, $decision$, $shadowing$, $redundancy$, and $irrelevance$ as keys to access each necessary value.

For reasons of clarity, we assume one can access a linked-list through the operator R_i, where i is the relative position regarding the initial list size – $count(R)$. We also assume one can add new values to the list as any other normal variable does ($element \leftarrow value$), as well as to remove elements through the addition of an empty set ($element \leftarrow \emptyset$). The internal order of elements from the linked-list R keeps with the relative ordering of rules.

Each element $R_i[condition]$ is a boolean expression over p possible attributes. To simplify, we only consider as attributes the following ones: $szone$ (source zone), $dzone$ (destination zone), $sport$ (source port), $dport$ (destination port), $protocol$, and $attack\_class$ – or A_c for short – which will be empty whether the component is a firewall. In turn, each element $R_i[decision]$ is a boolean variable whose values are in $\{true, false\}$. Elements $R_i[shadowing]$, $R_i[redundancy]$, and $R_i[irrelevance]$ are boolean variables in $\{true, false\}$ – which will be initialized to $false$ by default.

We split the whole process in four different algorithms. The first algorithm (cf. Algorithm 1) is an auxiliary function whose input is two rules, A and B. Once executed, this auxiliary function returns a further rule, C, whose set of condition attributes is the exclusion of the set of conditions from A over B. In order to simplify the representation of this algorithm, we use the notation A_i as an abbreviation of the variable $A[condition][i]$, and the notation B_i as an abbreviation of the variable $B[condition][i]$ – where i in $[1, p]$.

The second algorithm (cf. Algorithm 2) is a boolean function in $\{true, false\}$ which applies the necessary verifications to decide whether a rule r is irrelevant for the configuration of a component c. To properly execute such an algorithm, let us define $source(r)$ as a function in Z such that $source(r) = szone$, and $dest(r)$ as a function in Z such that $dest(r) = dzone$.

The third algorithm (cf. Algorithm 3) is a boolean function in $\{true, false\}$ which, in turn, applies the transformation *exclusion* (Algorithm 1) over a set of configuration rules to check whether the rule obtained as a parameter is potentially redundant.

The last algorithm (cf. Algorithm 4) performs the whole process of detecting and removing the complete set of intra-component anomalies. This process is split in three different phases. During the first phase, a set of shadowing rules are detected and removed from a top-bottom scope, by iteratively applying Algorithm 1 – when the decision field of the two rules is different. Let us notice that this stage of detecting and removing shadowed rules is applied before the detection and removal of proper redundant and irrelevant rules.

The resulting set of rules is then used when applying the second phase, also from a top-bottom scope. This stage is performed to detect and remove proper redundant

Algorithm 1: exclusion(B,A)

```
$C[condition] \leftarrow \emptyset$;
$C[shadowing] \leftarrow false$;
$C[redundancy] \leftarrow false$;
$C[irrelevance] \leftarrow false$;
$C[decision] \leftarrow B[decision]$;
forall the elements of $A[condition]$ and $B[condition]$ do
    if $((A_1 \cap B_1) \neq \emptyset$ and $(A_2 \cap B_2) \neq \emptyset$
    and ... and $(A_p \cap B_p) \neq \emptyset)$ then
        $C[condition] \leftarrow C[condition] \cup$
        $\{(B_1 - A_1) \wedge B_2 \wedge ... \wedge B_p,$
        $(A_1 \cap B_1) \wedge (B_2 - A_2) \wedge ... \wedge B_p,$
        $(A_1 \cap B_1) \wedge (A_2 \cap B_2) \wedge (B_3 - A_3) \wedge ... \wedge B_p,$
        ...
        $(A_1 \cap B_1) \wedge ... \wedge (A_{p-1} \cap B_{p-1}) \wedge (B_p - A_p)\}$;
    else
        $C[condition] \leftarrow (C[condition] \cup B[condition])$;
return $C$;
```

Algorithm 2: testIrrelevance(c, r)

```
$z_s \leftarrow$ source ($r$);
$z_d \leftarrow$ dest ($r$);
if $(z_s = z_d)$ and $(\neg r[decision])$ then
    warning ("First case of irrelevance");
else if $z_s \neq z_d$ then
    $p \leftarrow$ minimal_route ($z_s$,$z_d$);
    if $c \notin p$ and $(\neg r[decision])$ then
        warning ("Second case of irrelevance");
    else if $(\neg$empty $(r[A_c]))$ and $(\neg affects(z_d, r[A_c]))$ then
        warning ("Third case of irrelevance");
    else return false;
return true;
```

Algorithm 3: testRedundancy(R,r)

```
$i \leftarrow 1$;
$temp \leftarrow r$;
while $\neg test$ and $(i \leq count(R))$ do
    $temp \leftarrow$ exclusion($temp$, $R_i$);
    if $temp[condition] = \emptyset$ then
        return true;
    $i \leftarrow (i + 1)$;
return false;
```

Algorithm 4: intra-component-audit(c, R)

```
begin
    $n \leftarrow count(R)$;
    /*Phase 1*/
    for $i \leftarrow 1$ to $(n - 1)$ do
        for $j \leftarrow (i + 1)$ to $n$ do
            if $R_i[decision] \neq R_j[decision]$ then
                $R_j \leftarrow$ exclusion ($R_j$,$R_i$);
                if $R_j[condition] = \emptyset$ then
                    warning ("Shadowing");
                    $R_j[shadowing] \leftarrow true$;
    /*Phase 2*/
    for $i \leftarrow 1$ to $(n - 1)$ do
        $R_a \leftarrow \{r_k \in R \mid n \geq k > i$ and
            $r_k[decision] = r_i[decision]\}$;
        if testRedundancy $(R_a, R_i)$ then
            warning ("Redundancy");
            $R_i[condition] \leftarrow \emptyset$;
            $R_i[redundancy] \leftarrow true$;
        else
            for $j \leftarrow (i + 1)$ to $n$ do
                if $R_i[decision]=R_j[decision]$ then
                    $R_j \leftarrow$exclusion ($R_j$,$R_i$);
                    if $(\neg R_j[redundancy]$ and
                    $R_j[condition] = \emptyset)$ then
                        warning ("Shadowing");
                        $R_j[shadowing] \leftarrow true$;
    /*Phase 3*/
    for $i \leftarrow 1$ to $n$ do
        if $R_i[condition] \neq \emptyset$ then
            if testIrrelevance $(c, R_i)$ then
                $R_j[irrelevance] \leftarrow true$;
                $r[condition] \leftarrow \emptyset$;
end
```

rules, through an iterative call to Algorithm 3 (i.e., *testRedundancy*), as well as to detect and remove all the further shadowed rules remaining during the latter process. Finally, during a third phase the whole set of non-empty rules is analyzed in order to detect and remove irrelevance, through an iterative call to Algorithm 2 (i.e., *testIrrelevance*).

Assessment of a Vulnerability in Iterative Servers Enabling Low-Rate DoS Attacks

Gabriel Maciá-Fernández, Jesús E. Díaz-Verdejo, and Pedro García-Teodoro

Dep. of Signal Theory, Telematics and Communications - University of Granada
c/ Daniel Saucedo Aranda, s/n - 18071 - Granada (Spain)
{gmacia, jedv, pgteodor}@ugr.es[*]

Abstract. In this work, a vulnerability in iterative servers is described and exploited. The vulnerability is related to the possibility of acquiring some statistics about the time between two consecutive service responses generated by the server under the condition that the server has always requests to serve. By exploiting this knowledge, an intruder is able to carry out a DoS attack characterized by a relatively low-rate traffic destined to the server. Besides the presentation of the vulnerability, an implementation of the attack has been simulated and tested in a real environment. The results obtained show an important impact in the performance of the service provided by the server to legitimate users (DoS attack) while a low effort, in terms of volume of generated traffic, is necessary for the attacker. Besides, this attack compares favourably with a naive (brute-force) attack with the same traffic rate. Therefore, the proposed attack would easily pass through most of current IDSs, designed to detect high volumes of traffic.

1 Introduction

The impact of Denial of Service (DoS) attacks in current networked systems is awesome, posing a very serious problem in many environments, both in economical and in performance sense [1]. The threat is specially overwhelming in Internet, with millions of interconnected systems and a practical lack of enforcement authorities. Furthermore, the possibility of performing the attack in a distributed way (DDoS, Distributed DoS) according to various methodologies [2], increases the risks and makes even more difficult the adoption of preventive and/or corrective measures. Recent incidents involving large-scale attacks that affected important Internet sites [3] [4] [5] demonstrate the vulnerability of the networks and services to this kind of attacks and its pernicious effects.

DoS attacks try to exhaust some resources in the target system with the aim of either reducing or subverting the availability of a service provided by the target. The intruders usually achieve their goal either by sending to the victim a stream of packets that exhausts its network bandwidth or connectivity, or exploiting a discovered vulnerability, causing an access denial to the regular clients [2].

[*] This work has been partially supported by Spanish government under project TSI2005-08145-C02-02 (70 % from FEDER funds).

Close to the evolution of the DoS attacks, many proposals have also appeared for preventing and detecting them. Many of the preventive measures are applicable to mitigate DoS attacks, like egress or ingress filtering [6] [7], disabling unused services [8], changing IP address, disabling IP broadcasts, load balancing, or honeypots [9]. However, although prevention approaches offer increased security, they can never completely remove the threat as the systems are always vulnerable to new attacks. On the other hand, it is advisable to establish an intrusion detection system (IDS) [10] capable of detecting the attacks. Although various approaches described in the bibliography [11] [12] [13] try to discover DoS attacks, most of them rely on the identification of the attack with techniques that are based on the hypothesis that a high rate flooding is going to be received from the intruder or intruders.

In this paper, a vulnerability in iterative servers is detected and exploited. This vulnerability allows an intruder to carry out an application level DoS attack [14] characterized by the use of a low-rate traffic against a server. Due to this fact, the attack would be capable of bypassing the detection mechanisms that rely on high-bandwidth traffic analysis. An attack with similar rate characteristics is described by Kuzmanovic et. al [15]. Although the attack presented here resembles in some aspects the previously cited one, there are key differences between them. First, both attacks take advantage of a vulnerability caused by the knowledge of a specific time value in the functioning of a protocol or application, allowing an ON/OFF attack that results in low-rate traffic but with high efficiency in service denial. However, the attack presented in [15] is TCP-targeted, while the proposed here threatens the application layer. Furthermore, Kuzmanovic's attack generates outages in a link, trying to trigger TCP's congestion control mechanism, while ours simply tries to overflow a single service running in a server. In other words, no noticeable effect on network traffic is expected. On the other hand, the proposed attack would not affect other services or users within the same network or host, as Kuzmanovic's does. There are also differences in the vulnerabilities exploited in both attacks. In the TCP-targeted low-rate case, the knowledge of the RTO timer for congestion control implemented in TCP is exploited, whilst in the iterative server case the inter-output times are the key to build the attack, as it will be presented in Section 3. But the main difference between both attacks lies in the fact that during the TCP-targeted attack, the link is only busy in the outages periods, while in the new proposal the server is always busy in processing service requests from the intruder, causing the legitimate users the perception that the server is not reachable. This last feature is similar to the behaviour of the *Naptha* attack [16], although the main difference is that *Naptha* is a brute-force attack executed with high rate traffic, while this attack uses low-rate traffic.

The rest of the article is structured as follows. Section 2 describes the scenario of the attack and the hypothesis about its behaviour enabling the vulnerability. Next, a validation of the hypothesis backing up the claimed vulnerability is presented. Section 3 describes the details and phases of the proposed attack, which is evaluated both by means of simulations and in a real environment in

Section 4. Besides, Section 4 shows the comparison of the proposed attack with a brute-force attack with the same rate of attack packets. Finally, some conclusions and further work to be carried out are compiled in Section 5.

2 Scenario and Vulnerability Analysis

The scenario to be considered for the analysis is a generic client-server configuration in which the server is going to receive aggregated traffic from legitimate users and from intruders. From our point of view, an iterative server is a black box system which acts in the usual way, that is, the server receives requests from the clients and responds them after some processing. Only after the processing of a request will the processor take a new petition (iterative operation). Thus, the existence of a finite length queue in which incoming requests are queued up while awaiting for the server to process them in a FIFO discipline is assumed. Therefore, the overall behaviour is that each request packet emitted by a client is first queued in the server and, after some time, processed and responded by the server. The possibility of rejecting an incoming request due to a queue overflow is also considered. Whether the rejection of a request packet is observable or not is irrelevant for our experiments.

The main objective of a DoS attack in this scenario is to keep the queue full of request from the attacker. This fact will avoid the acceptance of request packets from other (legitimate) users, thus causing a denial of service to these users. Usually, the DoS event is carried out by means of a so called brute-force attack, that is, the intruder or intruders send as many requests as they can with the aim of either obtaining the bulk of the queue positions or saturating the network. Our objective will be similar: to ”capture” all the queue positions, but emitting a reduced number of requests, what is done by choosing the key instants at which they should be sent.

Therefore, the key to succeed in our approach of attacking the service is by forecasting the optimum instants at which a request should be made in order to acquire a just freed position in a previously full queue. This way, the attack would eventually capture all the positions and generate a denial of the service to legitimate users. Obviously, a main question remains unanswered: is it possible to forecast the instants at which the server is going to get a request from the input queue and, therefore, generate a free position? Our hypothesis is that, under certain circumstances, and by simple inspection of the outputs generated by the server, this is possible. In fact, this is the argued vulnerability.

2.1 Timing of the Outputs

In order to predict the instants at which a position is freed and, therefore, is available for the first request received, we need to review the operation of an iterative server in a more formal way, as follows.

A service request enters the system. If the service queue has free positions, the request is queued. Otherwise, an overflow event occurs and a denial of service is perceived by the user. The request will stay in the service queue during a *queue time*, t_q, awaiting for its turn to be served. Afterwards, it will be processed by the service module during a *service time*, t_s. This time would have been employed in parsing the data, in a complex calculation or simply in building up the answer. Finally, once the processing is completed, the corresponding answer to the input request is generated and sent. Following, the next request in the queue is obtained to be processed. At this point, a free position in the queue is generated.

Our main hypothesis is that the service time is a random process, T_s that can be modeled by a distribution function. At this point, some studies suggest behaviours that depend upon the nature of the service [17]. Furthermore, various authors report different distributions even for the same service, depending on the network and other additional factors [18]. Nevertheless, it is not necessary to know the overall statistics of the service to carry out the proposed attack. The only needed knowledge concerns the statistics related to the processing and responses to the requests made by the intruder. In this context, if all the packets sent by the intruder contains the same request, it would be expectable to require the same service time. Obviously, many factors external to the service but related with the server itself will introduce some degree of variability. We lean on the central limit theorem [19] to characterize the model of this behaviour in a sufficiently complex system, e.g. a computer running a server, where a lot of variables are involved, as a normal distribution. Anyway, as the purpose of this paper is to show the existence of the vulnerability, which is solely based on the possibility of estimating the expected value for the service time, $E[t_s]$, and according to the central limit theorem, we use a normal distribution $\mathcal{N}(\overline{t_s}, var[t_s])$ to check our hypothesis. This approach allows us to consider the effects of the variance due to different CPU loads, occupation in the service network, etc. The behaviour of the queue time is irrelevant from our point of view, as will be argued hereafter. On the other hand, as it will be pointed later on, the attack procedure includes a mechanism to resynchronize its timing, which reduces the impact of the requests made by the legitimate users on the evolution of the attack and the relevance of the real distribution function of the service time. That is to say, we only need to model the distribution of the service time for the attack packets jointly with a mechanism to recover from mistakes.

The potential exploit that allows an intruder to carry out the attack is based on the knowledge about the statistics of the inter-output time of the server, τ, defined as the time elapsed between two consecutive outputs or answers provided by the server. With the knowledge of this time, the intruder can evaluate the timing at which free positions are generated.

For clarity, let us examine a case of study in which fixed service times are considered. Although, at first glance, this could seem a very restrictive case, it will be shown that the results are valid for more complex cases in which the service time is a random variable, as discussed above.

The scenario of interest is that in which the queue is always kept occupied with, at least, one pending request. Under this consideration, the behaviour of

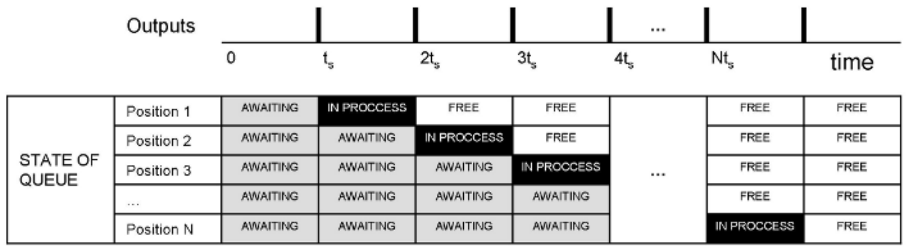


Fig. 1. Time diagram for the processing of the requests carried out by the server (bottom) and the associated timing of the generated outputs (top). In black, processing time; in gray, queue time.

the inter-output time, assuming fixed service times, can be described as follows (Fig. 1). Let us suppose a queue with N positions which is initially full of requests. At time $t = 0$, one of the pending requests is extracted from the queue and processed by the server. After t_s, an output is generated (vertical bar in the time diagram) and the next request is selected for its processing. Again, after t_s, a new answer is provided and the process is repeated while there are pending requests in the queue. Therefore, for this scenario, the time between two consecutive outputs from the server is equal to the service time, t_s. This rate of outputs will be maintained under the single condition that there always exists a pending request in the queue.

If, as previously hypothesized, the service time responds to a normal distribution, the inter-output time, τ, will behave as

$$\tau = \mathcal{N}(\overline{t_s}, var[t_s]) \tag{1}$$

As it can be seen, the queue time is not relevant, as it does not influence the timing of the outputs.

The distribution of the inter-output time could be estimated by a potential intruder by sending various requests close enough in time so as to occupy contiguous positions in the buffer. In a first approach, the time between two consecutive responses received by the attacker would provide the required information. By repeating this procedure, the potential attacker can collect enough information as to characterize the inter-output time distribution. Anyway, an effect that has not been previously considered appears in this mechanism. Both the requests and the answers must traverse the network to reach its destinations. Therefore, the round trip time (RTT) plays a role in the determination of the timings. Concretely, the variability of the RTT can produce deviations in the inter-output times. This way, the variance of the inter-output time perceived by a user, τ_{user}, will be affected by the variance of the RTT. Assuming that the service times and RTT are statistically independent variables, and that RTT can also be described by a normal distribution function, the perceived inter-output time will be:

$$\tau_{user} = \mathcal{N}\left(\bar{t_s}, var[t_s] + var[RTT]\right) \tag{2}$$

Experimental Validation by Simulation. In order to validate our hypothesis concerning inter-output time distribution, various simulation experiments have been carried out. For this purpose, Network Simulator (NS2) [20] have been used to check whether the assumption that inter-output time approximates to Eq. (2) is correct or not.

In a first set of experiments, an scenario with fixed service time has been considered. As expected, the distribution of the inter-output time, τ, behaves as predicted while there is at least one pending request in the queue.

In a second set of simulations, some scenarios in which the service time t_s and the round trip time RTT are modelled with normal distributions have been considered. The obtained results show low deviations in the behaviour of the system from that predicted theoretically.

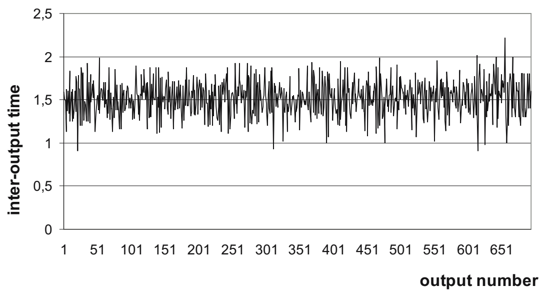
As an example, Fig. 2 shows the results of one of the simulations in which the queue is always full of requests, what is achieved by sending request packets at a higher rate than the service rate of the server. The service time and RTT are supposed to be $\mathcal{N}(1.5\ s,\ 0.02\ s)$ and $\mathcal{N}(0.6\,s, 0.02\ s)$, respectively, so that the inter-output time distribution is expected to be $\mathcal{N}(1.5\,s, 0.04)$ (see Fig. 2.a). The simulation results provide a mean value of 1.52 seconds and a variance of 0.041 seconds for the inter-output time, with a distribution that can be approximated by a normal distribution (Fig. 2.b). This fact have been tested through goodness of fit tests as the Kolmogorov-Smirnoff test [21].

On the other hand, Fig. 3 shows the results of another example in which the queue can become momentarily empty. The inter-output times and the occupation of the buffer are represented in the main axis and in the secondary axis (dashed lines), respectively, in Figure 3.a. For the same input parameters as in the previous example, the mean value obtained for the inter-output time is 1.536 s with variance 0.114. The deviation from theoretical results is greater than in the previous experiment due to the values generated when the buffer has no requests. In fact, the goodness of fit tests provide poor estimators for the obtained distribution when compared to a normal distribution. It is easily tested that the periods with no requests in the queue result in higher values for the inter-output times, greatly modifying the distribution of the samples. Therefore, the inter-output times becomes unpredictable if the queue becomes empty and the scenario of interest is that in which the queue has always at least one pending request, as previously stated.

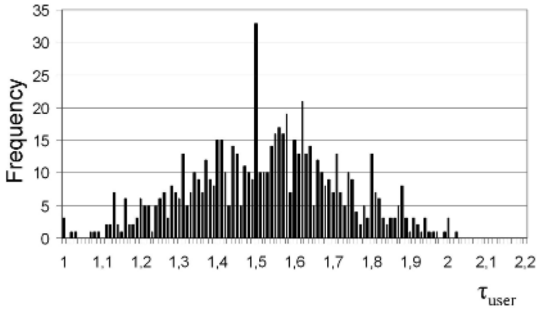
These experiments show that, even in simulated scenarios where the service time is variable, the inter-output time could still be predictable in some circumstances for a possible intruder to build up an attack based on this knowledge.

3 Low-Rate Attack Specification

As previously stated, the objective of the denial of service attack is to maintain the input queue of the target server full of requests coming from the attacker or attackers. This way, legitimate users are not going to be able to queue their requests, thus experimenting a denial of the service given by the server appli-



a)



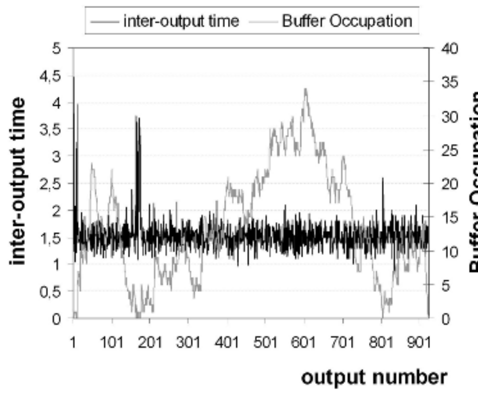
b)

Fig. 2. Simulation of inter-output time with flooded service buffer: a) inter-output time values, and b) histogram of the samples

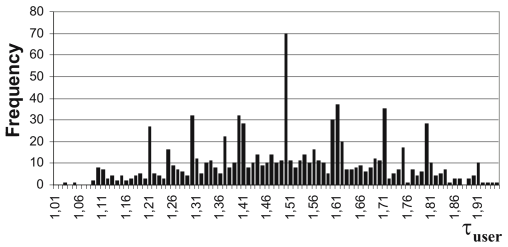
cation. This is achieved by making a prediction regarding the instants at which the server is going to answer the requests, that is, when an output is going to be generated. The intruder will flood the service queue only during a short period of time around the predicted output time, resulting in an overall low-rate flood experienced by the destination network. Therefore, the attack will follow an OFF/ON scheme, as described next.

The proposed attack strategy consists in the succession of consecutive periods composed by an interval of inactivity, *offtime*, followed by an interval of activity, *ontime*, as depicted in Fig. 4. The attack waveform is characterized by the following parameters:

– *Estimated mean inter-output time* ($E[\overline{\tau_{user}}]$): it is an estimation of τ_{user} made by the intruder.
– *Interval* (Δ): period of time elapsed between the sending of two consecutive attack packets during *ontime*.
– *Ontime time* (t_{ontime}): time during which an attempt to flood the service queue is made by emitting request packets, at a rate given by $1/\Delta$. The duration of ontime should be proportional to the variance of τ_{user}. It is centered around $E[\overline{\tau_{user}}]$.



a)



b)

Fig. 3. Simulation of inter-output time with the possibility of empty queue: a) Inter-output time values and buffer occupation level, b) histogram of the samples

– *Offtime time* ($t_{offtime}$): time during which there is no transmission of attack packets. Its duration should be

$$t_{offtime} = E[\overline{\tau_{user}}] - t_{ontime}/2 - \overline{RTT} \cdot \delta \tag{3}$$

where δ is equal to 0 if no response is received (a previous failure of the attacker in a seizure or loss of a packet) and 1 otherwise. This accounts for the delay among the emission of a packet and the reception of the response.

Both *offtime* and *ontime* are adjusted so that the generation of the output by the server and the reception of the requests from the intruder are synchronized. For this, the round trip time (RTT) has to be considered, as it represents the delay in the transmission from the server to the client and viceversa. Therefore, two points-of-view (server's and intruder's) should be considered in order to establish the timings and synchronization of the sendings. The descriptions made up to now have considered the server's point-of-view, i.e. all the events and sequences of events are timed according to the server's local clock. However, the communication among the server and the intruder will experience a delay due to the RTT which can make observation of the sequence of events slightly different. As an example, if an attack packet is due in the server at a given time,

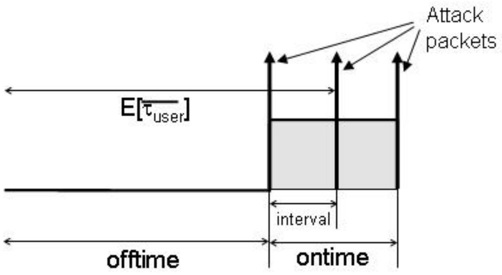


Fig. 4. Attack specification: Attack waveform and parameters

the intruder has to provide for it in advance (the packet should be sent RTT/2 time units before). Those effects will be considered in the following explanations concerning the attack execution.

The attack presents two phases. In the first phase, an attempt to capture all the positions in the queue is launched. After that, the attack tries to keep all the captured positions by sending a new request in such a way that it arrives at the server in the minimum time after the position becomes free, which is achieved by using the attack waveform in synchrony with the outputs from the server, as will be explained in the next paragraphs. For the first phase, a brute-force attack could be used although the same results can be achieved by using the attack mechanism proposed in the next paragraph. This alternative will achieve the aim in a longer time but reduces the chances of detection by an IDS rate-based mechanism.

The attack is composed by a continuous sequence of the attack waveform previously described, plus a mechanism to resynchronize the *ontime* periods or, equivalently, to restart the *offtime* period, just in case a response is received. Therefore, the execution of the attack can be explained as follows (Fig. 5). Just after the reception of an answer packet from the server (A1), the intruder sends a request packet (R1) and the *offtime* period starts. At the end of this period, the *ontime* period starts by emitting a new request packet (R2). While in the *ontime* period, another request packet is sent every *interval* (R3 and R4). At the end of the *ontime* period a new *offtime* starts inmediately, considering $\delta = 0$ in Eq. 3. On the reception of an answer packet during *offtime* (packet A2), a request packet is sent (R5) and the *offtime* period is restarted with the value given by using $\delta = 1$ in Eq. 3. If an answer packet had been received while in the *ontime* period, an additional request packet would have been sent, the *ontime* period would have been finished and the *offtime* period would have been started with $\delta = 1$ (not depicted in Fig. 5). This way, a request packet is sent whenever an answer is received. This is done to reduce the probability of losing the free position due to its capture by a legitimate user.

Finally, two comments about the behaviour of the attack should be pointed out. First, according to the described attack procedure, the intruder will flood the service queue only during a short period of time around the predicted output time, resulting in an overall low-rate flood experienced by the destination network. On the other hand, the behaviour during the flooding (*ontime* period) is

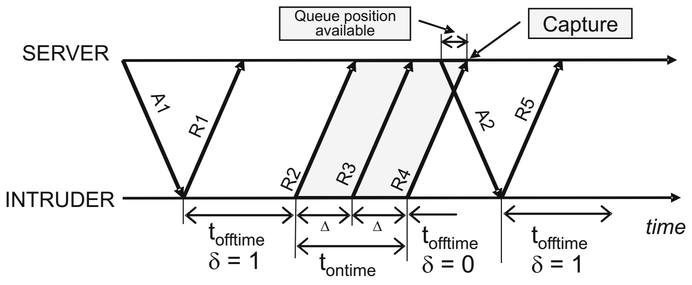


Fig. 5. Attack dynamics

designed to send packets that must arrive at the server at the precise time. Obviously, this can be made from a single attacker or in a distributed way, becoming a DDoS in the last case.

4 Experimental Results

In this section, the attack behaviour is evaluated and its impact analyzed. The attack has been tested in a simulated scenario, by using Network Simulator 2, as well as in a real environment.

Prior to the evaluation of the attack, some indicators are going to be defined in order to measure the attack performance.

4.1 Performance Indicators

The parameters of interest are:

- *Percentage of seizures* (S): percentage of the seizures in the server that corresponds to the attacker.
- *Effort of the attack* (E): percentage of request packets emitted by the intruder, related to the total number of packets that can be accepted by the server.
- *User success percentage* (U): percentage of seizures made by the legitimate clients related to the total number of requests generated by them.
- *Overflow percentage* (O): percentage of unattended requests due to full queue condition related to the total number of received requests.

It should be noticed that not all the parameters are observable by the agents in the scenario. For example, only the effort of the attack is observable by the intruder during the attack due to the fact that only the server knows the total number of packets and seizures generated in the observation period.

The aim of the attack should be to minimize the user perception of the availability of the server (U). This can be achieved by maximizing S, due to the fact that if the server is engaged more time with intruder requests, the user success percentage (U) will be lower. Besides, in order not to be detected by any active

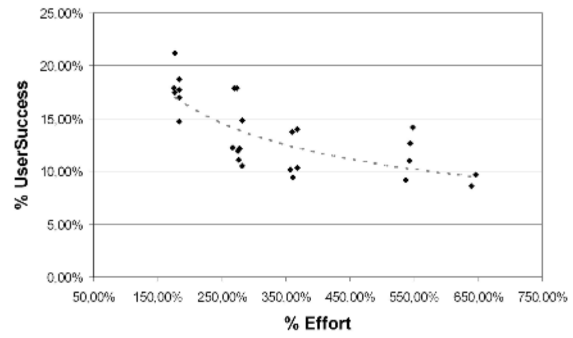


Fig. 6. User success and effective effort for 25 different configurations (t_{ontime} and Δ values) of the low-rate attack

intrusion detection system, the attack should also minimize its effort E. Minimizing E will contribute to a lower overflow O in the server, thus making the attack more undetectable.

4.2 Simulated Scenario

We are interested in discovering how effective the low-rate DoS attack can become. For that, a set of attacks has been analyzed in the simulator. The obtained results are really worrying due to the high effectiveness demonstrated.

As an example, in what follows the results obtained from one attack simulation composed by 1332 outputs are discussed. The attack has been launched against a server with $\overline{t_s} = 3.0$ seconds and $var(t_s) = 0.2$ seconds. The traffic generated by the user is enough by itself to keep the server busy all the time. The parameters of the attack for this example are: $t_{ontime} = 0.6$ s, $t_{offtime} = 2.7$ s, and $\Delta = 0.3$ s. Round trip time has been set to $\mathcal{N}(0.6\,s, 0.2)$. As expected, a very high efficiency is obtained: $S = 94\%$ and $U = 9\%$, which implies that only a 9.04 percent of the user requests are attended by the server. On the other hand, the overflow percentage $O = 77\%$ indicates that the traffic offered to the server by both the legitimate users and the intruders is about four times its capacity. In other words, only 22.9% of the requests are attended.

It is possible to adjust the effort of the attack (E) and, therefore, the ability to bypass an IDS system able to detect attacks on a given rate, by reducing *ontime* time and/or increasing *interval* at the expense of decreasing the effectiveness of the attack. In this context, Fig. 6 shows the variety of obtained vaules for the user success percentage and the effort for 25 possible settings for the attack to the previously defined server.

4.3 Comparison with a Naive Attack

The proposed attack strategy has to be measured not only in terms of effectiveness, according to the proposed indicators, but also in comparison with a *naive*

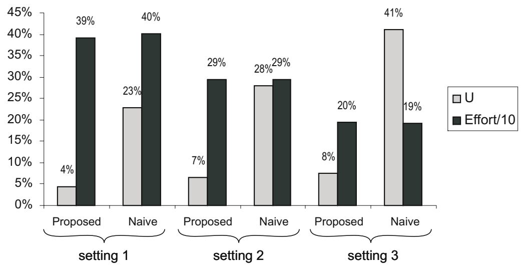


Fig. 7. Naive attack vs. proposed attack user success percentage values for three different efforts (packets rates)

attack. By *naive attack* we mean an attack with the same objective (to seize all the positions in the buffer) but carried out by simply sending packets in a fixed rate or randomly. Therefore, a naive attack is, in some sense, a brute-force attack, as it lacks any intelligence about the server's behaviour and is based in the exhaustion of the server's capabilities. However, we prefer the term "naive" as opposed to "brute-force" due to the fact that we are considering relatively low-rates for the attack.

The proposed attack would become useless if its figures does not improve those from the performance of a *naive attack* with a similar rate. The experimental results (Fig. 7) suggest an important impact in the expected behaviour when using the knowledge of the estimated inter-output time. As shown, the user success percentage of the proposed attack is about 20% lower, in absolute terms, when compared with the naive attack for the same effort of the attack. That's to say, the same rate of attack packets provides better results in denying requests from legitimate users when using the proposed dynamics for the attack. Furthermore, the difference for U between the naive and the proposed strategies increases as the attack rate (effort) decreases. This is an expected result; when using high rates both kinds of attacks should get the same performance due to the fact that the rate of arrivals at the server will be enough to saturate the capacity by itself (becoming, in this case, a brute-force attack).

4.4 Real Scenario

The proposed attack has been also tested in a controlled real environment to check its validity. The selected server is an Apache web server that keeps the condition of serving requests in an iterative way (`"ThreadsPerChild`= 1"). Although we positively know this is not a realistic scenario, as most of web servers are concurrent instead of being iterative, there exist some reasons for considering it. First, the argued vulnerability is present in every iterative server under the single condition of a predictable time distribution of the inter-output time or, equivalently, of the service time. In our opinion, this makes the iterative web server valid to test the behaviour of the proposed attack. Second, our interest is to extend this kind of study to concurrent servers, mainly to web servers, which

Table 1. Real and simulated attack performance

$\overline{t_s}$	$\overline{t_a}$		U	O	S	E
3	3.5	Simulated	10.4	71.4	90.5	260.8
		Real	9.8	69.4	91.4	239.7
5	6	Simulated	5.7	67.7	94.2	213.1
		Real	7.8	67.6	92.5	212.4
10	12	Simulated	3.0	64.4	97.2	198.3
		Real	6.4	65.5	94.3	201.6
15	17	Simulated	3.0	66.6	96.8	197.5
		Real	2.5	65.6	97.7	198.2
20	22	Simulated	3.0	65.0	97.2	197.5
		Real	4.3	65.1	96.0	196.2
25	28	Simulated	1.8	64.6	98.0	197.9
		Real	1.8	65.4	98.3	197.9

makes the iterative web server an interesting starting point. Furthermore, the next steps in our research, still in preliminary stages, confirm the existence of the vulnerability in concurrent servers.

We have considered that a client's petition consists in a connection request. The attack establishes connections and sends no messages on them, letting the web server to close the connection after a timeout period specified by the Apache directive `"Timeout"`, which corresponds to the service time, $\overline{t_s}$, in our model. Although it could be argued that there is no variance in the service time, it is not true due to two main reasons: there still exists some variability due to the processing of the connection request and, mainly, due to the variability in the RTT.

The real scenario is analogous to that one considered for the theoretical analysis. The user traffic has been generated following a Poisson process. A piece of software launches the attack from a single source. Both legitimate user and intruder traffic flows traverse a WAN network to reach the server, with a round trip time $\mathcal{N}(17\ ms, 0.05\ ms)$. Traces on the users and the intruder side have been issued for collecting the necessary data to calculate the attack indicators.

Table 1 shows some experimental results with a comparison among real and predicted values for different service times ($\overline{t_s}$) and user traffic arrival rates ($\overline{t_a}$). These rates have been selected in such a way that there is no congestion on the server if the attack is not carried out. The parameters of the attack have been tuned to $t_{ontime} = 0.4$ s and $\Delta = 0.4$ s for all the experiments.

We can even obtain better results in efficiency (lower U and higher S, with lower O) for the attack in a real environment in some cases. Two conclusions can be derived from these results: a) All the experiments we have made under simulation seem to provide results that are good approximations of the behaviour in real environments, and b) the real impact of the attack can be very high, showing that these vulnerabilities could be easily exploited in iterative servers.

5 Conclusions

In this work, a vulnerability present in iterative servers is described. It consists in the possibility that a potential attacker becomes aware about the statistics of the inter-output time of a given server. This vulnerability allows an intruder to perform a denial of service attack against an iterative server. The attack could be designed to nearly or completely saturate the capacity of the target system but, as a difference from the generalized brute force DoS attacks, it uses a relatively low-rate traffic to achieve its goals. Moreover, it is possible to tune the attack parameters in order to select the appropriate values for efficiency and load generated in the server. This distinctive characteristic could allow the attack to bypass, in many cases, existent IDS systems based on rate thresholds, becoming a non-detectable threat.

As a difference from other existent low-rate DoS attacks, this one threatens the application level, maintains the server engaged serving intruder requests and gets advantage of the knowledge of the inter-output time of the target server. This is opposed to the TCP-targeted low-rate attack defined in [15], that relies on selectively saturating the link in order to trigger TCP's congestion control mechanism. However, it has some common features with [15], what points out the existence of a new family of DoS attacks, characterized by the fact that they rely on vulnerabilities that consist in the a-priori knowledge of one timer of a protocol or end-system behaviour, and that allow the intruder to carry out the DoS attack with a low-rate traffic.

The fundamentals and details of the design of a possible exploit have been explained. We have demonstrated that this attack can be easily carried out and that it can obtain very efficient results. The potential risk presented by the attack is really worrying, due to the fact that it could behave very similar to legacy users, bypassing IDS systems and possibly affecting many services in a server.

The extension of this kind of attacks to concurrent servers is being researched jointly with a mathematical framework able to model the described behaviour. The preliminary experimental results obtained up to now show evidences of the existence of a analogous vulnerability in concurrent servers. On the other hand, the mathematical model under study points out some possible improvements in the proposed attack, which makes the associated threat more awesome.

References

1. Computer Security Institute and Federal Bureau of Investigation: CSI/FBI Computer crime and security survey 2001, CSI, March 2001. Available from <http://www.gocsi.com>.
2. Jelena Mirkovic and Peter Reiher: A taxonomy of DDoS attack and DDoS defense mechanisms. SIGCOMM Comput. Commun. Rev., 34(2):39–53, 2004.
3. M. Williams. Ebay, amazon, buy.com hit by attacks, 02/09/00. IDG News Service, 02/09/00, http://www.nwfusion.com/news/2000/0209attack.html - visited 18.10.2000.
4. CERT Coordination Center. *Denial of Service attacks.* Available at http://www.cert.org/tech_tips/denial_of_service.

5. D. Moore, G. Voelker, S. Savage: Inferring Internet Denial of Service activity, Proceedings of the USENIX Security Symposium, Washington, DC, USA, 2001, pp. 9-22.
6. P. Ferguson, D. Senie: Network ingress filtering: defeating Denial of Service attacks which employ IP source address spoofing, in: RFC 2827, 2001.
7. Global Incident analysis Center: Special Notice - Egress filtering. Available from <http://www.sans.org/y2k/egress.htm>.
8. X.Geng, A.B.Whinston: Defeating Distributed Denial of Service attacks, IEEE IT Professional 2(4)(2000) 36-42.
9. N. Weiler: Honeypots for Distributed Denial of Service, Proceedings of the Eleventh IEEE International Workshops Enabling Technologies: Infrastructure for Collaborative Enterprises 2002, Pitsburgh, PA, USA, June 2002, pp. 109-114.
10. S. Axelsson: Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Department of Computer Engineering, Chalmers Univ., Mar. 2000.
11. R. R. Talpade, G. Kim, and S. Khurana: NOMAD: Traffic-based network monitoring framework for anomaly detection. Proc. of IEEE Symposium on Computers and Communications, pages 442–451, 1999.
12. J. Cabrera et al.: Proactive detection of distributed denial of service attacks using MIB traffic variables - a feasibility study. Proc. of the IFIP/IEEE International Symposium on Integrated Network Management, 2001.
13. J. Mirkovic, G. Prier, and P. Reiher: Attacking DDoS at the source. Proc.of ICNP 2002, pages 312–321, 2002.
14. C. Douligeris and A. Mitrokotsa: DDoS attacks and defense mechanisms: classification and state-of-the-art. Comput. Networks, 44(5):643–666, 2004.
15. A. Kuzmanovic and E. Knightly: Low rate TCP-targeted denial of service attacks (The shrew vs. the mice and elephants). Proc. ACM SIGCOMM'03, pages 75–86, Aug. 2003.
16. SANS Institute: NAPTHA: A new type of Denial of Service Attack. Available at http://rr.sans.org/threats/naptha2.php.
17. A. Adas, Traffic models in broadband networks, IEEE commun. Mag. 35 (7) (1997) 82-89.
18. M. Izquierdo, D. Reeves, A survey of statistical source models for variable-bit-rate compressed video, in Multimedia systems, pp. 199-213, Springer Verlag, Berlin, 1999.
19. R.E. Walpole, R.H. Myers, and S. L. Myers, Probability and Statistics for Engineers and Scientists, Sixth Edition, Prentice Hall College Div, 1997. ISBN: 0138402086
20. K. Fall and K. Varadhan: The ns manual. Available at http://www.isi.edu/nsnam/ns/.
21. R. D'Agostino, M. Stephens, Goodness-of-Fit Techniques. Marcel Dekker, Inc. (1986).

Towards an Information-Theoretic Framework for Analyzing Intrusion Detection Systems

Guofei Gu[1], Prahlad Fogla[1], David Dagon[1], Wenke Lee[1], and Boris Skoric[2]

[1] Georgia Institute of Technology, USA
{guofei, prahlad, dagon, wenke}@cc.gatech.edu
[2] Philips Research Laboratories, Netherlands
boris.skoric@philips.com

Abstract. IDS research still needs to strengthen mathematical foundations and theoretic guidelines. In this paper, we build a formal framework, based on information theory, for analyzing and quantifying the effectiveness of an IDS. We firstly present a formal IDS model, then analyze it following an information-theoretic approach. Thus, we propose a set of information-theoretic metrics that can quantitatively measure the effectiveness of an IDS in terms of feature representation capability, classification information loss, and overall intrusion detection capability. We establish a link to relate these metrics, and prove a fundamental upper bound on the intrusion detection capability of an IDS. Our framework is a *practical* theory which is data trace driven and evaluation oriented in this area. In addition to grounding IDS research on a mathematical theory for formal study, this framework provides practical guidelines for IDS fine-tuning, evaluation and design, that is, the provided set of metrics greatly facilitates a static/dynamic fine-tuning of an IDS to achieve optimal operation and a fine-grained means to evaluate IDS performance and improve IDS design. We conduct experiments to demonstrate the utility of our framework in practice.

1 Introduction

As an essential component of the defense-in-depth strategy, intrusion detection systems (IDSs) have achieved more and more attention in both academic and industry. A number of IDSs have been developed in the last two decades [8]. Research work in the IDS field mainly focuses on how to construct a new detector based on some new idea so that the IDS can detect certain attacks with reasonable accuracy (in terms of false positives and false negatives). These are important topics, of course. However, very little work has been conducted on the fundamental theory. As a result, unlike cryptography, which now has a solid mathematical ground based on probability theory and the random oracle model, the IDS community still lacks a mathematical foundation that can be used to reason about the effectiveness of an IDS formally and practically. It is definitely necessary to base IDS research on a solid mathematical background [19] that can lead to a better understanding, evaluation, and design of an IDS. Such a theoretic framework should be mathematically sound, and useful in analyzing and quantifying the effectiveness of an IDS in both theory and practice.

In this paper, we investigate a novel theoretic framework for IDS research based on information theory. The basic observation is that, the intrusion detection process is actually a series of data processing and transformation procedures. This motivates us to use information theory, which is successfully applied in the area of communication (which is also a data/signal processing and transformation process), to study the efficiency of the intrusion detection process. We significantly extend our previous work on an information-theoretic measure of the intrusion detection capability [10], which only treats the IDS as a black box and measures the overall performance. In this paper, we further look into the basic components and architecture of an IDS, and apply information-theoretic analysis on the detailed intrusion detection procedure. Specifically, we make the following contributions in this paper:

1. We present **a formal model of an IDS** in Section 2 using an eight-tuple representation containing four data structures and four algorithms, which are used in three procedures, i.e., feature selection, profiling, and detection. This IDS model unifies signature-based and anomaly-based IDSs, thus, we can reason about all these IDSs using the same analytical approach. We also show how existing realistic IDSs such as PAYL [31] and Snort [26] fit into our model.

2. We perform **a fine-grained information-theoretic analysis on the IDS model** in Section 4. The detection procedure can be considered as a Markov transition chain in which two algorithms, i.e., a data reduction and representation algorithm $\mathcal{R}$ and a classification algorithm $\mathcal{C}$, are sequentially applied. This establishes a connection between intrusion detection and information theory. Further, we present **a series of information-theoretic metrics that can quantitatively measure the effectiveness of an IDS and its components**. We define the measures of feature representation capability (C_R), classification information loss (L_C), as well as the overall intrusion detection capability (C_{ID}, [10]). We establish a link among these metrics, and prove a fundamental upper bound of C_{ID}. The task of the IDS is to faithfully reflect the ground truth about intrusion information in observed data. If we assume the original ground truth information is 1 (normalized), when the data reduction and representation algorithm $\mathcal{R}$ is applied, this information is reduced to C_R. After the classification algorithm $\mathcal{C}$ is performed, there is further L_C amount of information loss. The end result is C_{ID}, the overall capability of the IDS. We also discuss how the metrics can be used in a robust way to tolerate uncertainties and possible estimation errors of parameters in practice.

3. This framework provides **practical guidelines for fine-tuning, evaluation and design of IDSs**. With the help of C_{ID}, one can select the optimal operating point (where C_{ID} is maximized) for an IDS, and we provide a concrete example for dynamically fine-tuning PAYL [31]. With the whole set of metrics, we provide a fine-grained analysis and quantification on the effectiveness of an IDS and its components. This yields a guideline for IDS design improvement, in particular, whether and how the feature representation or classification algorithm is (the bottleneck) to be improved. Experiments are conducted to show the utility of our framework in Section 5.

Note that in this paper we are not dealing with other important IDS performance issues, such as resilience to stress [25] and ability to resist attacks directed at the IDS [24,23]. These are different research topics beyond the scope of this paper. Also we are not trying to address cost related issues in IDS analysis because cost factor is subjective, but we are building an objective theoretic framework. Finally, we need to point out that although our technique/framework may be applicable to other domains (e.g., to analyze a general classifier), we focus on the intrusion detection (specifically network-based intrusion detection) field.

2 Modeling an Intrusion Detection System

In order to formally reason and analyze an IDS, we firstly present a formal model of the IDS. Briefly, an IDS is represented as an eight-tuple $(\mathbb{D}, \Sigma, \mathbb{F}, \mathbb{K}, \mathcal{S}, \mathcal{R}, \mathcal{P}, \mathcal{C})$, in which the first four items are data structures, and the last four are algorithms. Note that whenever we analyze and evaluate any IDS, we cannot talk about it without dealing with its data source. After all, our IDS model and framework are *data trace driven* and *evaluation oriented.*

$\mathbb{D}$: the data source that an IDS will examine and analyze. Essentially this is a stream of consecutive data units. Since each IDS has its own unit of analysis, e.g., packet level or flow level for a network-based IDS (NIDS), without loss of generality, we define $\mathbb{D} = (D_1, D_2, ...)$ where D_i is an analysis unit of data for the target IDS and $D_i \in \{d_1, d_2, ...\}$, d_j is the possible data unit. For example, an NIDS uses network traffic (packet stream), so the data source is a packet stream $\boldsymbol{P} = (P_1, P_2, ...)$. For a host-based IDS (HIDS) using system call sequence, the data source is a system call stream $\boldsymbol{C} = (C_1, C_2, ...)$. In this paper, we mainly take network data as our example, and packet as our data unit.

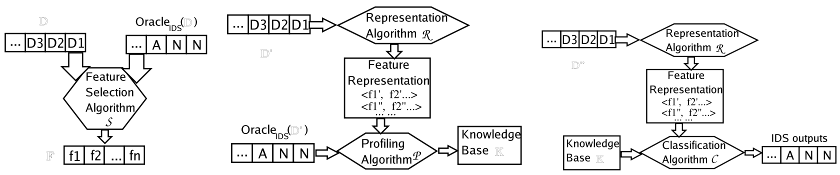
Σ: a finite set of data states indicating whether the data unit D_i is normal or anomalous (or further what type of intrusion). For convenience, we define an IDS oracle $Oracle_{IDS}$ which accepts any query with data unit D_i, and outputs an indication whether the unit is normal or anomalous. The IDS oracle knows the ground truth so it will always tell the truth[1]. Then for every data unit D_i, its state is $Oracle_{IDS}(D_i)$. The space of this state set is finite. For anomaly detection, $\Sigma = \{Normal, Anomalous\}$, or simply $\Sigma = \{N, A\}$, or $\Sigma = \{0, 1\}$ where 0 denotes normal and 1 denotes anomalous. For misuse detection, we can let $\Sigma = \{Normal, AttackType_1, AttackType_2, ...\}$, or $\Sigma = \{N, A_1, A_2, ...\}$.

$\mathbb{F}$: a feature vector contains a finite number of features, formally $F =< f_1, f_2, ..., f_n >$. Every feature is a meaningful attribute of a data unit. For example, f_1 could be the protocol type (TCP, UDP, ICMP, etc.), f_2 could be the port number. Each feature has its own meaningful domain (called feature space) which is a set of discrete or continuous values (either numerical or nominal). The full range of $\mathbb{F}$ is the product of the ranges of all the features. We denote it as $Range(F) = f_1 \times f_2 ... \times f_n$.

$\mathbb{K}$: the knowledge base about the profiles of normal/anomalous data. This knowledge base consists of profiling model (stored in some data structures) of normal and/or attack information. The detailed structure of $\mathbb{K}$ is possibly different for every IDS. It could be a tree, a Markov model, a Petri net, a rule set, a signature base, etc. For a signature-based NIDS, $\mathbb{K}$ is its rule set which contains only the attack profiling model (i.e., intrusion signatures). For an anomaly NIDS, $\mathbb{K}$ is mainly the profile of the normal traffic. Any activity that deviates the normal profile is considered as anomaly.

[1] In *real evaluation* of any IDS, since we should always know the ground truth of data, we are acting as the IDS oracle in these cases.

$\mathcal{S}$: feature selection algorithm. Given some $\mathbb{D}$ and the corresponding states $Oracle_{IDS}(D)$ (note sometimes only partial or even no such state information available), this algorithm should return several features f_i for the IDS to use. Although there is some preliminary effort to automatically generate worm signature [14,22] for misuse IDSs as part of their features, generally speaking $\mathcal{S}$ still highly depends on domain knowledge and is normally conducted manually. The automatic selection or generation of features for both anomaly and misuse IDSs remains a grand challenge. The quality of features is one of the most important factors that will affect the effectiveness of an IDS.



D
Oracle_IDS(D)
... D3 D2 D1
... A N N
Feature Selection Algorithm S
F
f1 f2 ... fn
... D3 D2 D1
Representation Algorithm R
D'
Feature Representation <f1', f2'...> <f1'', f2''...> ... ...
Oracle_IDS(D')
... A N N
Profiling Algorithm P
Knowledge Base K
... D3 D2 D1
Representation Algorithm R
D''
Feature Representation <f1', f2'...> <f1'', f2''...> ... ...
Knowledge Base K
Classification Algorithm C
IDS outputs
... A N N

(a) Feature selection procedure (b) Profiling/training procedure (c) Detection procedure

Fig. 1. Three IDS procedures

$\mathcal{R}$: data reduction and representation algorithm. When processing data, the IDS will firstly reduce the data and represent it in the feature space. This is a mapping/transition function, mapping the given data to a proper feature vector representation, namely $\mathcal{R} : \mathbb{D} \rightarrow \mathbb{F}$.

$\mathcal{P}$: profiling algorithm, which is the procedure of generating the profiling knowledge base $\mathbb{K}$. Given all the feature vector representations of data and their corresponding states, this algorithm will return the profiling knowledge base $\mathbb{K}$.

$\mathcal{C}$: classification algorithm. It is a mapping/transition function that maps the feature vector representation of given data to some states (it will also use the profiling base $\mathbb{K}$ in classification decision). Formally, $\mathcal{C} : \mathbb{F} \rightarrow \Sigma$.

Most IDSs work in three steps.

1. Feature selection procedure (Fig.1(a)). When we are developing an IDS, this is one of the first steps. Once the proper feature set is defined, it will be used in the following procedures. Normally, the feature selection procedure is conducted once, only during development.

2. Profiling procedure (Fig.1(b), sometimes also called training procedure[2]). We will run $\mathcal{P}$ (also involving $\mathcal{R}$) on a sufficiently large amount of training data and get the profiling knowledge base $\mathbb{K}$. Normally this procedure is performed once, only during development/training. In some situation, this procedure can be performed dynamically/periodically to update $\mathbb{K}$.

3. Detection procedure (Fig.1(c)). In this procedure, the IDS is used to detect intrusions in the data stream. This is the most important and frequently used procedure. We will perform an information-theoretic analysis on this procedure in Section 4.

[2] Some unsupervised learning based approach may skip this step.

Our IDS model unifies anomaly detection and misuse detection. In Appendix 7, we examine two representative IDSs, i.e., PAYL(Payload Anomaly Detection [31]) and Snort [26], to show how real world IDSs fit into our model.

3 Information Theory Background

Prior to introducing our information-theoretic framework for IDSs, we will first review a few basic concepts in information theory [6] to assist readers to follow our analysis.

Entropy: The entropy (or self-information) $H(X)$ of a discrete random variable X is defined as $H(X) = -\sum_x p(x)\log p(x)$. This definition, also known as Shannon entropy, measures the uncertainty of X. A smaller value of $H(X)$ indicates that X is less uncertain (more regular). The definition of entropy can also be easily extended to the case of jointly distributed random variables.

Conditional entropy: The conditional entropy $H(X|Y)$ is defined as $H(X|Y) = -\sum_y \sum_x p(x,y)\log p(x|y)$. It is the amount of information uncertainty of X after Y is seen. One can show that $H(X|Y)=0$ if and only if the value of X is completely determined by the value of Y. Conversely, $H(X|Y)=H(X)$ if and only if X and Y are completely independent. The conditional entropy $H(X|Y)$ has the following property: $0 \le H(X|Y) \le H(X)$.

Mutual information: Assume two random variables X and Y with a joint probability mass function $p(x,y)$ and marginal probability mass functions $p(x)$ and $p(y)$. The mutual information $I(X;Y)$ is defined as $I(X;Y) = \sum_x \sum_y p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$. It is the amount of *reduction* of uncertainty in X after Y is known, $H(X|Y)$ being the *remaining* uncertainty. It tells us the amount of information shared between two random variables X and Y. $I(X;Y)=0$ if and only if X and Y are independent. Obviously, $I(X;Y)=I(Y;X)$.

There is a nice relationship between entropy, conditional entropy and mutual information, i.e., $I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$. That is, $I(X;Y)$ corresponds to the intersection of the information in X with the information in Y. Clearly, $0 \le I(X;Y) \le H(X)$.

4 An Information-Theoretic Framework for Analyzing IDSs

The detection procedure (Fig.1(c)) of an IDS is the most important process for us to analyze. For simplicity, we will assume an anomaly NIDS with $\Sigma = \{N, A\}$ in all the following analysis (the analysis can be extended to an IDS with more than two states).

We firstly introduce three random variables X^o, Z^o, Y. X^o represents all possible input data units to the IDS. It can take value in $\{d_1, d_2, ...\}$ with some probability. $\boldsymbol{X^o}$ is the data stream $\mathbb{D} = (D_1, D_2, ...)$. Z^o (taking value in $Range(F)$ with some probability) is the intermediate representation of the data unit using the given feature set (performing $\mathcal{R}$). $\boldsymbol{Z^o}$ is the feature representation stream $(Z_1^o, Z_2^o, ...)$ where $Z_i^o = \mathcal{R}(D_i)$. Y (taking value in Σ with some probability) is the output alert of the IDS (the classification result of the IDS). $\boldsymbol{Y}$ is the alert stream $(Y_1, Y_2, ...)$ where $Y_i = \mathcal{C}(\mathcal{R}(D_i))$. Note that here we assume there is always an IDS output (decision)

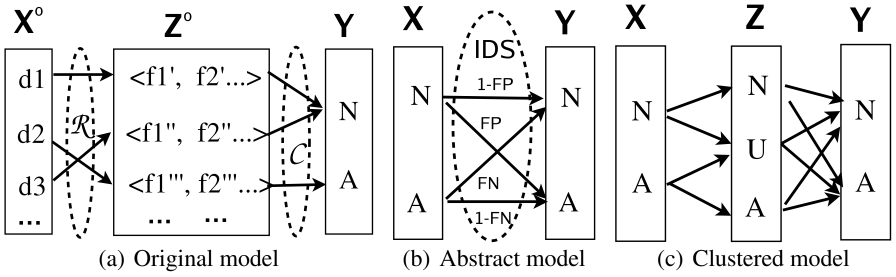


Fig. 2. Intrusion Detection Procedure: An Information-Theoretic View

corresponding to each input. Although a real IDS only needs to output alerts when there is an intrusion, this does not affect our analysis.

Thus, the detection process is the Markov chain of $X^o \rightarrow Z^o \rightarrow Y$ (data$\rightarrow$ representation$\rightarrow$alert) as shown in Fig.2(a), which we refer to as the original model. The input data are processed in sequence through two algorithms, $\mathcal{R}$ and $\mathcal{C}$. The mapping from X^o to Z^o is the result of $\mathcal{R}$. The mapping from Z^o to Y is the result of $\mathcal{C}$.

The simple observation of this Markov chain data processing procedure motivates us to use information theory to analyze the process. Intuitively, we can roughly consider $\mathcal{R}$ as an encoding algorithm that uses feature vector to encode the original data unit. And then, $\mathcal{C}$, as a decoding algorithm, decodes the feature representation to an output of the IDS. We should point out that although $\mathcal{R}$ and $\mathcal{C}$ resemble encoding and decoding procedures, they are not exactly the strict encoding and decoding schemes. In information theory, either encoding or decoding needs an encoding/decoding table containing all possible codewords for all possible source codes, so it can ensure a perfect encoding and decoding (without error or ambiguity). In the case of intrusion detection, we cannot enumerate all the possible input data units (source codes) and feature representations (code words), nor can we afford to store such a huge encoding/decoding table. As a result, both $\mathcal{R}$ and $\mathcal{C}$ algorithms can only work roughly correct, i.e., these algorithms may not guarantee errorless information transmission. We can analyze and quantify the effectiveness of this information transmission using information-theoretic metrics.

It is still a little hard to practically measure the effectiveness of the intrusion detection process based on the original model in Fig.2(a), because this model involves too many states in X^o and Z^o. We can hardly enumerate all the states and practically measure the transition probabilities. However, we notice that the purpose of an IDS is *not* to identify the original input data unit, but to identify the *state* of the data unit. That is, we are interested in only limited states of the data, i.e., Σ. We can group the input data to their states. This greatly simplifies the original model and our practical analysis. Similar idea can also be applied to the feature representation. Thus, we will introduce two simplified models step by step in the next paragraphs.

4.1 Abstract Model Analysis

First, we introduce a new random variable X to replace X^o in our analysis. X takes values in Σ, which represents the state of all possible input data unit to the IDS, with certain probabilities. $\boldsymbol{X}$ is the state stream $(X_1, X_2, ...)$ where $X_i = Oracle_{IDS}(D_i)$.

As the first step of our simplification, we ignore the intermediate feature representation process (that is, we ignore Z^o, and only consider X, Y). We treat an IDS as a black box and thus, introduce our first simplified model, i.e., the abstract model in Fig.2(b), as firstly shown in [10]. In this abstract model, $\Sigma = \{N, A\}$. We can denote transition probabilities between X and Y using false positive rate (FP, $P(Y = A|X = N)$, denoted as α) and false negative rate (FN,$P(Y = N|X = A)$, denoted as β). Thus, we have an abstract model of intrusion detection with a very simple Markov transition matrix between X and Y. The capability of an IDS to classify the input events correctly (i.e., faithfully reflect the "truth" about the input) can be measured using (normalized) mutual information, which captures the reduction of original uncertainty (intrusion or normal) given that the IDS alerts are known.

Definition 1. *Intrusion detection capability C_{ID} is defined as the normalized mutual information between X and Y [10], i.e., $C_{ID} = \frac{I(X;Y)}{H(X)}$.*

We can easily derive $C_{ID} = \frac{I(X;Y)}{H(X)} = \frac{H(X)-H(X|Y)}{H(X)} = 1 - \frac{H(X|Y)}{H(X)}$. Since $0 \leq H(X|Y) \leq H(X)$, we get $0 \leq C_{ID} \leq 1$.

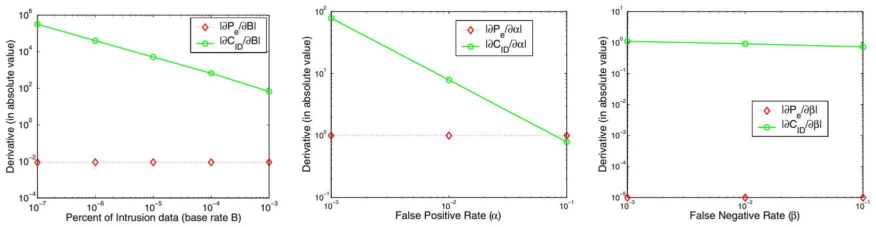
Intuitively, C_{ID} is interpreted as how much (normalized) ground truth information an IDS can identify. For example, $C_{ID} = 0.8$ means that the IDS identifies 0.8 bit of ground truth information assuming the original ground truth contains information 1. It indicates how well an IDS can distinguish normal from anomaly and distinguish anomaly from normal. In other words, it is an objective trade-off between FP and FN.

C_{ID} has several nice properties [10]: (1) it naturally takes into account all the important aspects of detection capability (if we expand the equation of C_{ID}), i.e., false positive rate, false negative rate, positive predictive value (PPV, or Bayes detection rate [3]), negative predictive value (NPV), and base rate (the probability of intrusion $P(X = A)$, denoted as B); (2) it objectively provides an *intrinsic* measure of intrusion detection capability; (3) C_{ID} yields a series of related information-theoretic metrics, which will be discussed soon. This gives a fine-grained measure of the basic architecture and components of an IDS; (4) it is very sensitive to IDS operation parameters such as α, β, which can demonstrate the effect of the subtle changes of an IDS.

[10] has showed that C_{ID} is more sensitive than some existing metrics (PPV, NPV), however, comparison with the probability of error $P_e = Pr(Y \neq X)$ (which is another metric to define how Y is different from X) is missing. Now we demonstrate that C_{ID} is also more sensitive to operation parameters than P_e in reasonable situations in which the base rate is very low [3]. For $\Sigma = \{N, A\}$, we can derive $P_e = B\beta + (1 - B)\alpha$. Similarly we can express the formula of C_{ID} using B, α, β. Since both P_e and C_{ID} have the same scale (value range [0,1]), it is fair to compare their sensitivities. To compare the sensitivities of C_{ID} and P_e, we perform a differential analysis of B, α, β to study the effect of changing these parameters on P_e and C_{ID}. For most IDSs and their operation environments, base rate and false positive rate are very low [3] so we can assume $B \ll 1$ and $\alpha \ll 1$.

Fig.3 shows the partial differential analysis (in absolute value) on different metrics. We only need to care about the absolute value of the derivatives. A larger derivative value shows more sensitivity to changes. For all the cases in Fig.3(a)(b)(c), a change in B, α or β results in very tiny change in P_e. Only when $\alpha > 0.1$, the derivative of P_e on

α begins to be greater than that of C_{ID}. But for real world IDSs, it is very unlikely to have a false positive higher than 10%. (For example, it is quite reasonable to have more than one million packets per day in an enterprise network. If a packet level IDS has a false positive rate of 10%, this will generate more than 100,000 false positives per day!) Clearly, from Fig.3 we can see that C_{ID} is more sensitive to changes in B, α, β than P_e (several orders of magnitude more sensitive).



(a) Partial differential on B ($\alpha = 0.001, \beta = 0.01$) (b) Partial differential on α ($B = 0.00001, \beta = 0.01$) (c) Partial differential on β ($B = 0.00001, \alpha = 0.001$)

Fig. 3. Partial differential analysis (in absolute value). In every situation C_{ID} has the highest sensitivity compared to P_e, except in (b) when $\alpha > 0.1$ (which is unlikely, in practice every IDS should have a much smaller false positive rate than 10%). For realistic situations, its derivative is always higher (several orders of magnitude) than P_e.

4.2 Clustered Model Analysis

In the previous abstract model, we have clustered numerous states in X^o into a smaller set of states in X. As a next step, we reconsider the intermediate feature representation Z^o. Using similar simplification techniques, we will cluster numerous states in Z^o into a smaller set. Specifically, we cluster the feature representation vectors to only three states, $\{N, U, A\}$. We can imagine that the IDS Oracle is labeling each feature representation vector F_i, denoted as $L(F_i) \in \{N, U, A\}$. State N means the feature representation vector is from and only from the data unit which is normal. If the feature vector is from and only from data unit which is anomaly, then this is labeled as A. Those feature vectors that can be from both normal and anomalous data have the state U (means *undistinguishable*). Formally,

$$\begin{array}{ll} L(F_i) = N & \Leftrightarrow \forall D_j, \mathcal{R}(D_j) = F_i, Oracle_{IDS}(D_j) = N \\ L(F_i) = A & \Leftrightarrow \forall D_j, \mathcal{R}(D_j) = F_i, Oracle_{IDS}(D_j) = A \\ L(F_i) = U & \Leftrightarrow \exists D_1 \neq D_2, \mathcal{R}(D_1) = F_i, \mathcal{R}(D_2) = F_i, Oracle_{IDS}(D_1) = N, Oracle_{IDS}(D_2) = A \end{array}$$

We use a new random variable Z to replace Z^o. Z denotes the clustered feature representation state, and $Z \in \{N, U, A\}$. Thus, we can slightly change the original transition model (Fig.2(a)) to a new one (Fig.2(c)).

In this clustered model, we can perform a fine-grained information-theoretic analysis on the intrusion detection procedure. Instead of viewing the IDS as a black box in the abstract model (Fig.2(b)) , we will analyze and reason about the basic architecture and components of an IDS. Here we have three random variables X, Z, Y, which form a Markov chain in the order $X \rightarrow Z \rightarrow Y$.

In this detailed model, we first consider the transition from X to Z. Here we can define a metric which measures the capability of feature representation. The definition is also the normalized mutual information, similar to the definition of C_{ID}.

Definition 2. *We define the feature representation capability* C_R *as the normalized mutual information between* X *and* Z*, i.e.,* $C_R = \frac{I(X;Z)}{H(X)}$.

Clearly C_R is also a measure of the capability of $\mathcal{R}$. Similar to C_{ID}, $0 \leq C_R \leq 1$.

A larger C_R means a better feature representation capability. If $C_R = 1$, then we say the IDS has an ideal feature representation capability. Intuitively this is saying that there is no information loss during the first transition from X to Z.

If there are some feature vectors with state U, it is hard to distinguish whether they are from normal or anomalous data only given the feature vectors (note that for $\mathcal{C}$, the feature representation vector is the only input). Intuitively, when transition from X to Z, we lose the "information". Information-theoretically, we will have a smaller C_R. Ideally, if the feature set has a perfect feature representation capability, we will have no feature vector with state U, which also means $P(Z = U|X = x) = 0$ for $\forall x \in \Sigma$. In this case, we get the identical distribution of X and Z, so we get $C_R = 1$. Then the model is much simplified as well as the abstract model in Fig.2(b).

Now let us consider the transition from Z to Y. In order to measure how good $\mathcal{C}$ is, we expect that there will be less information loss after the classification algorithm. Note here we do not simply use the normalized mutual information between Z and Y because actually in this transition, from Z to Y, we still need to involve X, otherwise we cannot know how good Y is (the classification result) according to X. Let us consider a new random variable Y^X which is the joint probability distribution of X and Y. Then the mutual information difference between (Y^X, Z) (i.e., $I(X,Y;Z)$) and (Y, Z) (i.e., $I(Y,Z)$) is the proper measure of classification information loss of $\mathcal{C}$. We will soon see this definition also yields another nice property stated in Theorem 1.

Definition 3. *We define the classification information loss* L_C *as the normalized information loss between* $I(X,Y;Z)$ *and* $I(Y;Z)$*, i.e.,* $L_C = \frac{I(X,Y;Z)-I(Y;Z)}{H(X)}$.

Because of the chain rule for information process, $I(X;Z|Y) = I(X,Y;Z) - I(Y;Z)$, we can also write L_C as $L_C = \frac{I(X;Z|Y)}{H(X)}$.

Since $I(X;Z|Y) = H(X|Y) - H(X|Y,Z) \leq H(X|Y) \leq H(X)$, we know that $0 \leq L_C \leq 1$.

We always expect that $\mathcal{C}$ does a good job so as to have less information loss. Thus, a smaller L_C means a better classification algorithm. If $L_C = 0$, then we say the IDS has an ideal classification algorithm $\mathcal{C}$ (so ideal classification information loss).

Now we have two new metrics C_R and L_C which can measure the feature representation capability and the classification information loss. The following theorem provides a nice relationship between these two metrics and C_{ID}.

Theorem 1. *The intrusion detection capability* C_{ID} *is equal to the feature representation capability* C_R *minus the classification information loss* L_C*, i.e.,* $C_{ID} = C_R - L_C$.

Proof. Since X, Z, Y form a Markov chain in the order $X \rightarrow Z \rightarrow Y$, the conditional distribution of Y depends only on Z and is conditionally independent on X. We can get $I(X;Y|Z) = 0$ in this case (because X and Y are conditionally independent given Z).

Using the chain rule, we can expand mutual information in two different ways.

$$\begin{aligned} I(X;Z,Y) &= I(X;Y) + I(X;Z|Y) \\ &= I(X;Z) + I(X;Y|Z) \end{aligned}$$

Applying the fact that $I(X;Y|Z) = 0$, we can get $I(X;Y) = I(X;Z) - I(X;Z|Y)$. Divided by $H(X)$, we get $C_{ID} = C_R - L_C$. □

We already know C_{ID} is the fraction of ground truth information identified by the IDS. If we assume the original ground truth information is 1 (normalized), when $\mathcal{R}$ is applied, this information will be reduced to C_R. After $\mathcal{C}$ is performed, we will further lose L_C amount of information due to the classification algorithm. So finally we can get C_{ID} amount of information. If both C_R and L_C are ideal, then the IDS has an ideal intrusion detection capability ($C_{ID} = 1$).

From this theorem, clearly we have $C_R \geq L_C$ because $C_{ID} \geq 0$. Also we can obtain the following corollary easily.

Corollary 1. *For an IDS, the intrusion detection capability is no more than its feature representation capability,i.e.,* $C_{ID} \leq C_R$.

This establishes an upper bound of C_{ID} for an IDS. For any given IDS, C_{ID} can never exceed C_R. Once the feature set and $\mathcal{R}$ are given, the upper bound of C_{ID} is also established no matter how good $\mathcal{C}$ is.

4.3 Implication and Discussion

Implication for Fine-Tuning, Fine-Grained Evaluation and Improvement of IDSs. Now we can perform a fine-grained evaluation of IDSs using a set of information-theoretic metrics, C_R, L_C, as well as C_{ID}. We can compare different IDSs, not only in terms of the overall performance, but also the performance of their specific components.

The overall measure, C_{ID}, is surely very useful. We can fine-tune an IDS to some configuration that maximizes the C_{ID} so that we have an optimal operation point. Section 5 will show a concrete example to demonstrate the static and dynamic fine-tuning of an IDS based on C_{ID}.

C_R can help us evaluate whether the features in use have a good representation capability or not, independent of the classification algorithm. An ideal feature set should have no information loss during the process, i.e., there should be no "undistinguishable" feature representation vector. Once there exist some, they will definitely be classified to one output category although they are actually from two different input category (normal and anomaly). Here we lose the information, and the lost information will never come back or be complemented by further process (classification algorithm).

If we do find some undistinguishable state (conflicts), we need to further reconsider/reselect the features (refine $\mathbb{F}$). For example, we can *carefully* add more features so that the existing undistinguishable state will become distinguishable. (The original distinguishable states are still distinguishable). Thus, we can improve C_R and avoid information loss in the first process ($X \rightarrow Z$). Note that only simply adding more features does not guarantee increasing accuracy (decreasing L_C) in the testing data, which is known as "overfitting" problem in machine learning literature, because the change of the feature set may also affect the accuracy of the classification algorithm. As a result, when adding more features, we increase the upper bound of C_{ID} (i.e., C_R), but we still need to do some possible adjustment/change on classification algorithm to make sure that L_C does not increase, so that we can improve the final C_{ID}.

In most cases when we compare two different IDSs, they can have different feature sets and different classification algorithms. With our framework, we can tell their fine-grained performance difference. For example, the reason why one IDS is less capable (lower C_{ID}) than another one can be mainly because its poor feature representation capability or classification information loss. Knowing the exact reason will point out future improvement direction (bottleneck) of IDS design. We have shown this using an example in experiment 4 of Section 5.

In practical evaluation, C_{ID} and C_R are easy to measure because the distribution, transition probabilities from X to Y in Fig.2(b) and the ones from X to Z in Fig.2(c) are easy to obtain in evaluation data. We may not need to directly calculate L_C, but simply apply Theorem 1 to compute $L_C = C_R - C_{ID}$.

Implication for IDS Design.

Feature* $\mathbb{F}$ *and algorithm* $\mathcal{R}$ *requirement: Feature selection is very important for any IDS. C_R is the first quantitative measure of its representation capability. If features are not carefully selected, the information will be lost when $\mathcal{R}$ is applied. Once C_R becomes lower, C_{ID} will also decrease no matter how good L_C is.

An IDS will not have a good representation capability if different types of data are represented in the same feature vector. It will misclassify some events because in the first transition process $(X \rightarrow Z)$, these different type of events cannot be distinguished from each other in terms of the feature vector representation (e.g., for Snort, some normal packets may match the same rule of some attack; for PAYL, the frequency vector of byte sequence for some attacks may be within the range of normal profile).

A lower feature representation capability C_R normally implies two possible reasons, either features are not well selected or $\mathcal{R}$ is not well designed. So we are left with two possible ways to improve C_R. (1) Re-select the feature set or at least carefully add more features (this implies a better feature selection algorithm $\mathcal{S}$). For example, a context-aware Bro [27] is better than the one without considering context because it essentially adds new features (about the context). (2) Well implemented data reduction and representation algorithm $\mathcal{R}$ will also improve C_R than poorly designed $\mathcal{R}$. For instance, in network intrusion detection, when using full assembling, protocol parsing $\mathcal{R}$, an IDS may achieve better C_R. Traffic normalization [11] is another good example of a better $\mathcal{R}$.

Knowledge base* $\mathbb{K}$ *requirement: Since knowledge base $\mathbb{K}$ is used in the procedure of $\mathcal{C}$, an accurate (complete and general) $\mathbb{K}$ is an important factor to improve the performance of classification algorithm $\mathcal{C}$, so as to improve L_C. For a signature based IDS, it is important to make sure the signature set is accurate and covering as many as possible known attacks. This directly affects the quality of $\mathbb{K}$, and $\mathcal{C}$. For anomaly detection, an exact profiling of large amount of normal data is the key to improve the quality of $\mathbb{K}$ and $\mathcal{C}$.

Realtime requirement on algorithms: The four algorithms in an IDS have different realtime requirement. $\mathcal{S}, \mathcal{P}$ are off-line algorithms, so there is fewer runtime speed requirement. However, for algorithm $\mathcal{R}$ and $\mathcal{C}$, they are mostly used online, so they should be efficiently implemented.

Finally, we should note that most of the implications are not surprising facts. They can be used as a sanity test for the correctness of any IDS model and theory. Our IDS model and information-theoretic framework nicely confirm them.

Prior and Transition Probabilities.

Static situation: When evaluating IDSs, we should always have the data set with detailed ground truth knowledge. Thus, from the evaluation data we can easily find out the base rate (fraction of intrusion) and measure all the transition probabilities (α, β, etc.) in Fig.2 (b) and (c).

Error bound and confidence: Machine learning researchers have given some bounds with certain confidence on the estimation of true error based on an observed error over a sample of data [21]. Given an observed sample error e_s, with approximately $N\%$ (e.g. 99%) probability, the true error e_t will lie in the interval $e_s \pm z_N\sqrt{\frac{e_s(1-e_s)}{n}}$, where n is the number of records in sample data, z_N is a constant related to the confidence interval $N\%$ we want to reach. For example, if we want approximately 99% confidence intervals then $z_N = 2.58$. Since the possible difference between testing data and real data is a general problem for every data-centric evaluation research, we are not trying to solve this problem in this paper. In practice, we can assume the transition probabilities are relatively stable (such as α, β) with reasonable high confidence, if the testing data is a representative sample of the real situation.

Base rate estimation: In the real world, the base rate may vary in different situations. Here we give a heuristic approach to estimate the base rate. Once we have the estimated FP (α), FN (β), we can approximately estimate the base rate in real traffic as follows. All we need is an alert rate (r_a) of the IDS (the fraction of generated alerts over total data). As we know this alert rate can be computed as $r_a = B(1-\beta) + (1-B)\alpha$. So we can approximately estimated the base rate as $B = \frac{r_a - \alpha}{1-\beta-\alpha}$, which provides us a good estimation of the real base rate. It is easy to prove that this is an unbiased estimator for B. In the next section, we will show how to use this estimation to dynamically fine-tune an IDS to keep it working on optimal operation points.

Towards a robust consideration: Our framework can also be easily analyzed with a robust consideration. For a robust evaluation with uncertain parameters in real world, we consider the real B, α, β can deviate from our estimation to some certain degree (a range). Thus, we release the assumptions in all above sections. Now instead of calculating C_{ID}, C_R, L_C with a static setting of B, α, β, we use a range of these parameters (to tolerate largest possible estimation error bound), and among all possible results, we take the worst values (stands for the worst cases with all possible situation of B, α, β as we expect) as the final resulting metric. By doing this, we are actually finding the best performing IDS against the worst situation (with the worst possible estimation error bound), instead of finding the best performing IDS on average (this is similar to the idea in [5]). Thus, we can make sure that this final measure (say, C_{ID}) is robust in sense that it is the low bound in all cases of possible estimated range of parameters. This will help if one is really concerned about the (large) estimation errors and uncertainties of the parameters in practice. The IDS is guaranteed to be better than this robust metric given the largest possible estimation error bound.

5 Experiments

In this section, we describe the experiments we conducted to show how our information-theoretic framework is useful for fine-tuning an IDS in the real world, and we also show how a fine-grained measurement of IDSs is helpful for improving IDS design.

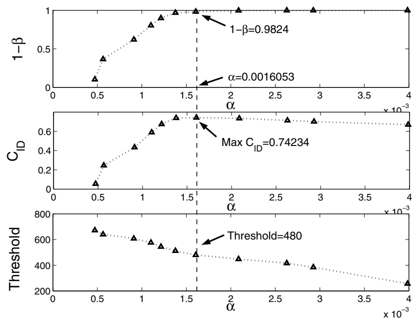
5.1 Dynamically Fine-Tuning an IDS

Fine-tuning an IDS is an important and non-trivial task, especially for anomaly-based IDSs. We can use C_{ID} as a yardstick to find an operation point yielding the best trade-off between FP and FN (best is in terms of the intrinsic ability of the IDS to classify input data). Specifically, we firstly change the threshold of the anomaly IDS so that we can achieve different FP and FN pairs, and create an ROC curve. Then, we can calculate corresponding C_{ID} for every point in the ROC. We select the point with the highest C_{ID}, and the threshold corresponding to this point provides the optimal threshold for use.

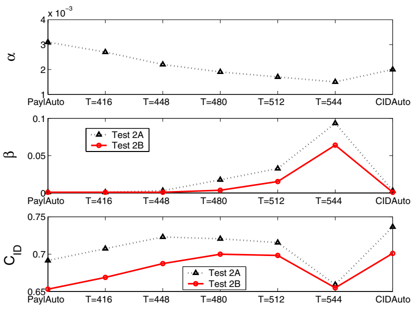
To demonstrate this, we select an anomaly IDS, PAYL [31], as our example. PAYL requires a threshold for determining whether the observed byte frequencies vary significantly from a trained model. For example, a threshold of 256 allows each character in an observed payload to vary within one standard deviation of the model. We collected a HTTP trace at a web server from our campus backbone network. Since PAYL only handles the HTTP requests from client to the server, we filter out all the outgoing HTTP responses. The trace data set only consists of incoming HTTP requests, approximately 7.5 million packets. We also filtered the trace set through to remove known attacks, and equally split the trace into three sets: training set, testing set 1, and testing set 2. We injected numerous HTTP attacks into the testing set, using tools such as Nikto [29].

In our first experiment, we train PAYL on the training set, and test it on testing set 1. The purpose is to choose an optimal threshold as the static operation point for PAYL in our testing environment. The base rate in testing set 1 is $B = 0.00081699$. The result is shown in Fig.4(a). We see that for the testing trace, as the threshold drops, C_{ID} reaches a peak and then drops, while the ROC curve (shown in the top graph) continues to slowly increase. The maximum point of C_{ID} corresponds to $<\alpha = 0.0016053, 1-\beta = 0.9824>$, and the corresponding threshold is 480. This tells us that PAYL works optimal in sense of intrusion detection capability at this threshold in our testing data. Without our information-theoretic guideline C_{ID}, it is not clear how to choose an optimal operation point from the ROC curve.

Experiment 1 finds optimal threshold in testing set 1. If the base rate in testing set 1 is representative to the real situation, then it is perfect. However, in real world situation, the base rate may vary from time to time. If we fix the operation point at a certain threshold, then in other testing data, we may not always achieve optimal C_{ID} when



(a) Experiment 1: Static tuning PAYL



(b) Experiment 2: Dynamic tuning PAYL using different schemes

Fig. 4. Fine-tuning PAYL in static and dynamic situations. In experiment 1, C_{ID} can tell the optimal operation point, while a pure ROC curve cannot. In experiment 2, $C_{ID}Auto$ outperforms other schemes in terms of achieving optimal operation points dynamically in different base rate situations.

the base rate varies. To address this problem, we introduce a new dynamic fine-tuning scheme that can be adaptive to any real situation. In previous section, we have discussed an unbiased estimation of base rate, i.e., $B = \frac{r_a - \alpha}{1-\beta-\alpha}$. If we divide the time series into many intervals, at each interval n, we estimate B_n, and then choose optimal operation point at this base rate to maximize C_{ID}. By dynamically fine-tuning the IDS, we ensure the IDS always operates on optimal points. Thus, we get a self-adaptive, self-tuning version of the IDS, which is very useful in practice.

We conduct a second experiment to investigate the effectiveness of dynamic fine-tuning. In experiment 2, we use the same training set in experiment 1. For the testing set, we inject different amount of attacks into testing set 2, and generate two new testing set $2A$ and $2B$. They contain the same normal data but different amount of attacks, so their base rates are different from testing set 1. Specifically, $B_{2A} = 0.00077256$, which is only slightly different from testing set 1, $B_{2B} = 0.00044488$, which is almost half of that in testing set 1. We modified PAYL to deploy a dynamic fine-tuning using C_{ID} as the guideline. And we denote this scheme as $C_{ID}Auto$ scheme. We compare the results to the cases when we fix the threshold at some certain values from 416 to 544. We also compare with the original automatic threshold adjusting scheme provided by PAYL (denoted as $PaylAuto$). This scheme is to adjust the threshold in testing to control the alarm rate below certain value (0.001 in PAYL's setting). Once the alarm rate is stable low for some time, then the threshold is fixed during the rest of the testing.

The results of experiment 2 are shown in Fig.4(b). $C_{ID}Auto$ outperforms all other schemes in all cases, i.e., it outputs the highest C_{ID} in both two testing sets $2A$ and $2B$. Fixing threshold at 480 as in experiment 1 still achieves satisfied result but not the optimal one because the base rate varies. Fixing threshold at 480 scheme gives the second highest C_{ID} in test $2B$, less than $C_{ID}Auto$. Fixing threshold at 448 achieves the second highest C_{ID} in test $2A$, still less than $C_{ID}Auto$. In both testing sets, the $PaylAuto$ scheme runs with a final stable threshold at 376, and this scheme has the highest FP and lowest C_{ID} (which is not good).

Our experiments clearly demonstrate the usefulness of C_{ID} in dynamic fine-tuning of IDSs.

5.2 Fine-Grained Evaluation and Design Improvement of IDSs

In this section, we will show how our framework can be used for a fine-grained evaluation of IDSs and how we can improve the design. As a motivating example, we will use several machine learning based IDSs in this experiment, because they have a clear architecture and we can easily manipulate them as we want to change features or classification algorithms. Thus, it is much easier for readers to understand the usefulness of our framework.

In [17], Lee and Stolfo proposed three categories of features to be used by IDSs, i.e., 9 basic features of individual TCP connections, 13 content features within a connection suggested by domain knowledge, and 19 traffic features computed using a two-second time window. Using these total 41 features, they processed data set from 1998 DARPA Intrusion Detection Evaluation program [18]. The processed data are available known as KDD cup 1999 data set [1]. Every connection record is labeled as N (normal) or A (anomalous). The training set has 494,020 connection records. The testing data set has 311,029 records, among which 250,436 are anomalous. Obviously we can see that the distribution (of normal and anomalous data) in the testing data is not good because the base rate is so high (about 0.8) and obviously not a reasonable operation environment. *Only for the purpose of providing a reasonable base rate*, we artificially duplicate the normal connections 4000 times, so that the base rate $B = 250436/(4000 * 60593 + 250436) \approx 0.001$ is more reasonable. Note that our duplicating normal data does not affect other parameters such as α, β.

We have noticed the critique [20] on the DARPA data set and the limit of the KDD data set. However, since our purpose is *not* to design a new IDS nor to conduct an official testing evaluation, we merely take them as a (public available) platform to demonstrate our framework. In this sense, we think the data are still valid to achieve our goal. We also plan to conduct more experiments using more real world IDSs on real world data to demonstrate our framework in the future.

In experiment 3, we use all 41 features (denoted as feature set 1). For the classification algorithm $\mathcal{C}$, we choose three different machine learning algorithms, i.e., decision tree (specifically, we use C4.5 [21]), Naive Bayes classifier, and SVM (Support Vector Machine [30]). All of them have been successfully applied to intrusion detection [2,13]. Since they are standard machine learning classification algorithms that are well documented in [21,30], we skip the details of these algorithms in this paper. The result of experiment 3 is shown in Table 1.

Form Table 1, we can see that in the transition $X \rightarrow Z$, these 41 feature set does *not* provide an ideal feature representation capability (i.e., $C_R = 0.9644 < 1$). Specifically, we measure the transition probabilities $P(Z = U|X = N) \approx 0.12$ and $P(Z = U|X = A) \approx 0.030319$. When further analyzing the state U in detail, we surprisingly find that all of the U states are caused by `snmpgetattack` (one kind of R2L attack), i.e., 7,593 (out of total 7,741) `snmpgetattack` connection records have the same feature representations with some normal data. In other words, only from these feature representation, one cannot distinguish these `snmpgetattack` from normal traffic. So we already have information loss at the data reduction and representation process. For the classification process, SVM has the lowest classification information loss ($L_C = 0.4002$, much lower than other two algorithms). Thus, SVM finally outputs

Table 1. Experiment 3: a fine-grained evaluation

IDS	α	β	C_{ID}	C_R	L_C
Feature set 1 with C4.5	0.017609	0.089676	0.4258	0.9644	0.5386
Feature set 1 with Naive Bayes	0.025713	0.099802	0.3756	0.9644	0.5888
Feature set 1 with SVM	0.0036473	0.12397	0.5642	0.9644	0.4002

$C_{ID} = 0.5642$, which means on average, SVM can achieve slightly more than half of the original ground truth 'information'. The other two algorithms get less than half.

Experiment 3 clearly shows that the feature set in use still has room to improve because the feature representation capability is not ideal (a simple possible solution to improve C_R is to add one more feature which can distinguish these `snmpgetattack` from normal traffic, e.g., SNMP protocol or not). We are the first to provide a quantitative measure of such capability. We also quantitatively compare the classification information loss of different machine learning algorithms such as SVM, decision tree and Naive Bayes. The result shows SVM has the least classification information loss in the data set.

In practice, more likely we will have IDSs with different feature sets and different classification algorithms. In these cases, we can first compare them using the overall intrusion detection capability (C_{ID}). Moreover, we can further (fine-grained) compare their feature representation capability and classification information loss. It will help us understand why an IDS is better, i.e., mainly due to its C_R or L_C. This not only helps us evaluate IDSs (especially when the IDSs have similar overall C_{ID}), but also indicates the direction for further improvement and tuning of IDS design. To demonstrate this point, we conduct experiment 4. We choose two different feature sets and two different classification algorithms to form two IDSs: one uses feature set 2 (including 9 basic features and 13 content features) and C4.5 classification algorithm, the other uses feature set 3 (including 9 basic features and 19 traffic features) and Naive Bayes classifier. For feature set 2, we get the transition probabilities $P(Z = U|X = N) \approx 0.2021$, $P(Z = U|X = A) \approx 0.1892$ in testing set, and $C_R = 0.8092$. For feature set 3, we get the transition probabilities $P(Z = U|X = N) \approx 0.12$, $P(Z = U|X = A) \approx 0.030319$, and $C_R = 0.9644$.

The experiment result is shown in Table 2. We can see that the two IDSs have similar C_{ID} (IDS1 is slightly better). But by further exploring the components of these IDSs, we find IDS1 has a much worse C_R but a better L_C. On the contrary, IDS2 has a better C_R but the classification algorithm is very poor (causing larger classification information loss). This fine-grained analysis indicates the bottleneck and further improvement direction for IDS2 is mainly on classification algorithm, while for IDS1 is primarily a better feature set (since its C_R is too low compared to that of IDS2). Following this direction, we do another experiment with indicated improvements. When IDS1 improves its feature set (classification algorithm unchanged) by simply adding more traffic features to become feature set 1, we can get a better $C_{ID} = 0.4258$, which is higher than the original 0.4002. By improving IDS2's classification algorithm (use c4.5 to substitute Naive Bayes in our experiment, feature set unchanged), we can improve the C_{ID} from 0.3875 to 0.4255.

Table 2. Experiment 4. The fine-grained analysis indicates the improvement direction for each IDS.

IDS	α	β	C_{ID}	C_R	L_C
IDS1(with feature set 2 and C4.5)	0.023699	0.079437	0.4002	0.8092	0.4090
IDS2(with feature set 3 and Naive Bayes)	0.022577	0.10329	0.3875	0.9644	0.5769
IDS1(after improving feature set)	0.017609	0.089676	0.4258	0.9644	0.5386
IDS2(after improving classification algorithm)	0.017576	0.090374	0.4255	0.9644	0.5389

The above example clearly demonstrates that fine-grained analysis can indicate our further (component) improvement direction for the design of IDSs.

6 Related Work

Intrusion detection is a field of active research for more than two decades. However, there is still little work on fundamental (theoretical) research, and there is still a huge gap between theory and practice.

For theoretical studies of intrusion detection, in 1987, Denning [9] was the first to systematically introduce an intrusion detection model, and also proposed several statistical models to build normal profiles. Later, Helman and Liepins [12] studied some statistical foundations of audit trail analysis for intrusion detection. Axelsson [4] pointed out that results from detection and estimation theory may be used in the IDS research. However, it is unclear how these similarities can benefit IDS evaluation and design. Song *et al.* [28] used ACL2 theorem prover for the analysis of IDSs that employ declarative rules for attack recognition by proving the specifications satisfy the policy with various assumptions. This approach is only useful for a certain type of IDSs, i.e., specification-based intrusion detection. In contrast, our framework is general to all types of IDSs. Recently, Di Crescenzo *et al.* [7] proposed a theory for IDSs based on both complexity-theoretic notions and well-known notions in cryptography (such as computational indistinguishability). Cardenas *et al.* [5] proposed a framework for IDS evaluation (not analysis) by viewing it as a multi-criteria optimization problem and gave two approaches: expected cost, and a new trade-off curve (IDOC) considering both the detection rate and the Bayes detection rate. Different from these existing work, our framework is an objective (without taking subject cost factors), natural and fine-grained approach with information-theoretic grounding. Besides, we established a clear and detailed IDS model, and provided an entire framework to analyze components inside an IDS and improve the design of IDSs.

Information-theoretic metrics have been widely applied in many fields. For instance, in the machine learning area, there are well-known algorithms (such as C4.5 [21]) that use information gain as a criterion to select features. [15] proposed to use entropy as a measure of distributions of packet features (IP addresses and ports) to identify and classify anomaly network traffic volumes. Lee *et al.* [16] applied information theoretic measurement to describe the characteristics of audit data set, suggested the appropriate anomaly detection model, and explained the performance of the models. This paper is a significant improvement and extension on our previous work [10], in which C_{ID} was firstly proposed as a measure of the overall intrusion detection capability by viewing the whole IDS as a black box. An overall measure of the IDS is useful, but it cannot measure the performance of each component of the IDS. In this paper, we looked into the detailed processes within an IDS and performed a white box information-theoretic analysis on the components of the IDS. Thus, we built a complete framework. In addition, we demonstrated fine-tuning an IDS in both static and dynamic cases. We also showed how to use our framework to evaluate IDSs in a fine-grained way and improve the design of IDSs with experiments.

7 Conclusion and Future Work

In the paper, we established a formal framework for analyzing IDSs based on information theory. As a *practical* theory, it is data trace driven and evaluation oriented. Within our framework, the analysis of anomaly based and signature based IDSs can be unified. In addition to providing a better understanding of IDSs grounded on information theory, the framework also facilitates a static/dynamic fine-tuning of an IDS to achieve optimal operation, a better or finer-grained means to evaluate IDS performance and improve IDS design. Our framework provided intrusion detection research a solid mathematic basis and opened the door for the study of many open problems.

This paper is only a preliminary start in the field. There are many topics for possible future work. One is to use more information theory, e.g., channel capacity models, to further study the effect of multiple processes/layers/sensors of the IDS architecture. Thus, we can analyze and improve *both internal and external* designs of the IDSs by extending our current framework. We will also further study robust ways of applying the framework.

Acknowledgements

This work is supported in part by NSF grant CCR-0133629 and Army Research Office contract W911NF0510139. The contents of this work are solely the responsibility of the authors and do not necessarily represent the official views of NSF and the U.S. Army.

References

1. Kdd cup 1999 data. Available at http://kdd.ics.uci.edu/databases/kddcup99/, 2006.
2. Nahla Ben Amor, Salem Benferhat, and Zied Elouedi. Naive bayes vs decision trees in intrusion detection systems. In *SAC '04*, 2004.
3. S. Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proceedings of ACM CCS'1999*, November 1999.
4. Stefan Axelsson. A preliminary attempt to apply detection and estimation theory to intrusion detection. Technical Report 00-4, Dept. of Computer Engineering, Chalmers Univerity of Technology, Sweden, March 2000.
5. Alvaro Cardenas, Karl Seamon, and John Baras. A Framework for the Evaluation of Intrusion Detection Systems. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, Oakland, California, May 2006.

6. Thomas Cover and Joy Thomas. *Elements of Information Theory*. John Wiley, 1991.
7. Giovanni Di Crescenzo, Abhrajit Ghosh, and Rajesh Talpade. Towards a theory of intrusion detection. In *ESORICS'05*, 2005.
8. Herve' Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999.
9. D. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2), Feb 1987.
10. Guofei Gu, Prahlad Fogla, David Dagon, Wenke Lee, and Boris Skoric. Measuring intrusion detection capability: An information-theoretic approach. In *Proceedings of ACM Symposium on InformAction, Computer and Communications Security (ASIACCS'06)*, March 2006.
11. Mark Handley, Vern Paxson, and Christian Kreibich. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proc. USENIX Security Symposium 2001*, 2001.
12. P. Helman and G. Liepins. Statistical foundations of audit trail analysis for the detection of computer misuse. *IEEE Transactions on Software Engineering*, 19(9), September 1993.
13. Wenjie Hu, Yihua Liao, and V. Rao Vemuri. Robust support vector machines for anomaly detection in computer security. In *Proc. 2003 International Conference on Machine Learning and Applications (ICMLA'03)*, 2003.
14. Hyang-Ah Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In *USENIX Security Symposium*, pages 271–286, 2004.
15. Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *SIGCOMM '05*, 2005.
16. W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, May 2001.
17. Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):p.227–261, 2000.
18. Massachusetts Institute of Technology Lincoln Laboratory. 1998 darpa intrusion detection evaluation data set overview. http://www.ll.mit.edu/IST/ideval/, 2005.
19. Teresa F. Lunt. Panel:foundations for intrusion detection. In *Proc. 13th Computer Security Foundations Workshop (CSFW 2000)*, 2000.
20. John McHugh. Testing intrusion detection systems: A critique of the 1998 and 1999 darpa off-line intrusion detection system evaluation as performed by lincoln laboratory. *ACM Transactions on Information and System Security*, 3(4), November 2000.
21. Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.
22. James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatically generating signatures for polymorphic worms. In *IEEE S&P '05*, 2005.
23. Vern Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, December 1999.
24. T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks Inc., January 1998.
25. Nicholas J. Puketza, Kui Zhang, Mandy Chung, Biswanath Mukherjee, and Ronald A. Olsson. A methodology for testing intrusion detection systems. *IEEE Transactions on Software Engineering*, 22(10):719–729, 1996.
26. M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of USENIX LISA'99*, 1999.
27. Robin Sommer and Vern Paxson. Enhancing byte-level network intrusion detection signatures with context. In *CCS '03*, 2003.

28. Tao Song, Calvin Ko, Jim Alves-Foss, Cui Zhang, and Karl N. Levitt. Formal reasoning about intrusion detection systems. In *Proceedings of RAID'2004*, September 2004.
29. Sullo. Nikto, 2006. Available at http://www.cirt.net/code/nikto.shtml.
30. V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.
31. Ke Wang and Salvatore J. Stolfo. Anomalous payload-based network intrusion detection. In *Proceedings of RAID'2004*, September 2004.

Appendix: How Real World IDSs Fit into Our Model

Table 3 shows how PAYL and Snort[3] fit into our model.

Table 3. Modeling PAYL and Snort

	PAYL model
$\mathbb{D}$	Packet sequence $(P_1, P_2, \ldots)$
Σ	$\{N, A\}$, only indicates normal or anomalous.
$\mathbb{F}$	A character frequency vector $< fre_0, fre_1, \ldots, fre_{255} >$, here fre_i is the frequency of char i in the payload of the packet.
$\mathbb{K}$	For each specific observed length i of each port j, M_{ij} stores the mean and standard deviation of the frequency for each distinct byte.
$\mathcal{S}$	Manually examines exploits and finds out the importance of byte frequency.
$\mathcal{R}$	Scans each incoming payload of the packet, computes its byte value distribution.
$\mathcal{P}$	Runs $\mathcal{R}$ on large normal data set to generate normal profile ($\mathbb{K}$) of the frequency.
$\mathcal{C}$	For each new payload distribution given by $\mathcal{R}$, compares against model M_{ij}. If their Mahalanobis distance significantly deviates from the normal threshold, flags the packet as anomalous and generates an alert.

	Snort model
$\mathbb{D}$	Packet sequence $(P_1, P_2, \ldots)$
Σ	$\{N, A_1, A_2, \ldots\}$. N is the normal state. A_i is the type of attack that can be detected by Snort, e.g., `WEB-IIS .asp HTTP header buffer overflow attempt`. Currently Snort can detect over three thousand attacks.
$\mathbb{F}$	Feature vector such as `<srcIP, dstIP, dstPort, payload containing '|3A|' or not, payload containing '|00|' or not, ...>`. Many of the features are boolean values to indicate whether the payload contains some substring (part of the intrusion signatures) or not.
$\mathbb{K}$	The rule set of Snort.
$\mathcal{S}$	Manually examines exploits and finds out most common strings in intrusions.
$\mathcal{R}$	Grinders the packet, preprocesses (defragmentation, assembling, etc.). If possible, uses string matching to explore feature space according to the rule set. Due to the implementation of Snort, $\mathcal{R}$ does not need to represent the packet into the whole feature space. Instead, $\mathcal{R}$ can stop when the packet is represented to some subset of features (matching a certain rule).
$\mathcal{P}$	Manually extracts signatures of intrusions and generates the rule set.
$\mathcal{C}$	Although $\mathcal{C}$ is not clearly separated from $\mathcal{R}$ in the implementation of Snort, we can consider it as a simple exact matching in knowledge base $\mathbb{K}$ (as 1-nearest neighbor searching in the rule set for the exact rule).

[3] There is no clear separation between $\mathcal{R}$ and $\mathcal{C}$ in the implementation of Snort. But we can still model the whole process into two algorithms. $\mathbb{K}$ is used in the whole process.

Author Index