Kil-Hyun Nam
Gwangsoo Rhee (Eds.)

Information Security and Cryptology – ICISC 2007

LNCS 4817

10th International Conference
Seoul, Korea, November 2007
Proceedings

Springer

Lecture Notes in Computer Science 4817

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Kil-Hyun Nam Gwangsoo Rhee (Eds.)

Information Security and Cryptology – ICISC 2007

10th International Conference
Seoul, Korea, November 29-30, 2007
Proceedings

Springer

Volume Editors

Kil-Hyun Nam
National Defense University
122-875 Susaek-dong, Eunpyung-gu, Seoul 122-875, Korea
E-mail: khnam@kndu.ac.kr

Gwangsoo Rhee
Sookmyung Women's University
52 Hyochangwon-gil, Yongsan-Ku, Seoul 140-742, Korea
E-mail: rhee@sookmyung.ac.kr

Library of Congress Control Number: 2007939824

CR Subject Classification (1998): E.3, G.2.1, D.4.6, K.6.5, F.2.1, C.2, J.1

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-540-76787-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-76787-9 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12192021 06/3180 5 4 3 2 1 0

Preface

ICISC 2007, the Tenth International Conference on Information Security and Cryptology, was held in Seoul, Korea, during November 29–30, 2007. It was organized by the Korea Institute of Information Security and Cryptology (KIISC) in cooperation with the Ministry of Information and Communication (MIC), Korea. The aim of this conference was to provide a forum for the presentation of new results in research, development, and applications in the field of information security and cryptology. It also intended to be a place where research information can be exchanged.

The conference received 123 submissions from 24 countries, covering all areas of information security and cryptology. The review and selection processes were carried out in two stages by the Program Committee (PC) of 57 prominent researchers via online meetings through the iChair Web server. First, each paper was blind reviewed by at least three PC members, and papers co-authored by the PC members were reviewed by at least five PC members. Second, individual review reports were revealed to PC members, and detailed interactive discussion on each paper followed. Through this process the PC finally selected 28 papers from 14 countries. The authors of selected papers had a few weeks to prepare final versions of their papers, aided by comments from the reviewers. The proceedings contained the revised versions of the accepted papers. However, most of these final revisions were not subject to any further editorial review.

The conference program included two invited talks from eminent researchers in information security and cryptology. The invited speakers were Daniel J. Bernstein from University of Illinois at Chicago and Mitsuru Matsui from Mitsubishi Electric Corporation.

We would like to thank everyone who contributed to the success of this conference. First, thanks to all the authors who submitted papers to this conference. Second, thanks to all 57 members of the PC listed overleaf. It was a truly nice experience to work with such talented and hard-working researchers. Third, thanks to all the external reviewers for assisting the PC in their particular areas of expertise. Fourth, we would like to thank all the participants of the conference who made this event an intellectually stimulating one through their active contribution. We would also like to thank the iChair developers in EPFL for allowing us to use their software. Finally, we are delighted to acknowledge the partial financial support provided by CIST, KISIA, NICS Tech, NITGEN, STG Security, and TSonNet.

November 2007

Kil-Huyn Nam
Gwangsoo Rhee

ICISC 2007

The 10th International Conference on
Information Security and Cryptology

November 29–30, 2007
Olympic Parktel, Seoul, Korea

Korea Institute of Information Security and Cryptology (KIISC)
(http://www.kiisc.or.kr)

Ministry of Information and Communication (MIC), Korea
(http://www.mic.go.kr)

Organization

General Chair

Min Surp Rhee	Dankook University, Korea

Program Co-chairs

Kil-Hyun Nam	National Defense University, Korea
Gwangsoo Rhee	Sookmyung Women's University, Korea

Program Committee

Michel Abdalla	ENS and CNRS, France
Joonsang Baek	Institute for Infocomm Research, Singapore
Alex Biryukov	University of Luxembourg, Luxembourg
Bill Caelli	Queensland University of Technology, Australia
Kyo-il Chung	ETRI, Korea
Jean-Sebastien Coron	University of Luxembourg, Luxembourg
Frederic Cuppens	ENST Bretagne, France
Ed Dawson	Queensland University of Technology, Australia
Bart De Decker	Katholieke Universiteit Leuven, Belgium
Orr Dunkelman	Katholieke Universiteit Leuven, Belgium
Eduardo B. Fernandez	Florida Atlantic University, USA
Pierre-Alain Fouque	Ecole Normale Superieure, France
Mario Marques Freire	University of Beira Interior, Portugal
Marc Girault	Orange Labs, France

Philippe Golle	Palo Alto Research Center, USA
Dieter Gollmann	Hamburg University of Technology, Germany
Goichiro Hanaoka	AIST, Japan
Hiroaki Kikuchi	Tokai University, Japan
Kwangjo Kim	ICU, Korea
Christopher Kruegel	Technical University Vienna, Austria
Chi Sung Laih	National Cheng Kung University, Taiwan
Kwok-Yan Lam	Tsinghua University, China
Kristin E. Lauter	Microsoft Research, USA
Dong Hoon Lee	Korea University, Korea
Pil Joong Lee	POSTECH, Korea
Arjen K. Lenstra	EPFL, Switzerland
Yingjiu Li	Singapore Management University, Singapore
Javier Lopez	University of Malaga, Spain
Masahiro Mambo	University of Tsukuba, Japan
Mark Manulis	Ruhr University of Bochum, Germany
Keith Martin	Royal Holloway, University of London, UK
Mitsuru Matsui	Mitsubishi Electric Corporation, Japan
Atsuko Miyaji	JAIST, Japan
SangJae Moon	Kyungpook National University, Korea
Yi Mu	University of Wollongong, Australia
Jesper Buus Nielsen	University of Aarhus, Denmark
DaeHun Nyang	Inha University, Korea
Rolf Oppliger	eSECURITY Technologies, Switzerland
D'Arco Paolo	University of Salerno, Italy
Kunsoo Park	Seoul National University, Korea
Sangwoo Park	National Security Research Institute, Korea
Raphael Chung-Wei Phan	EPFL, Switzerland
Rei Safavi-Naini	University of Calgary, Canada
Kouichi Sakurai	Kyushu University, Japan
Palash Sarkar	Indian Statistical Institute, India
Dongkyoo Shin	Sejong University, Korea
Willy Susilo	University of Wollongong, Australia
Tsuyoshi Takagi	Future University, Hakodate, Japan
Jozef Vyskoc	VaF s.r.o., Slovakia
Guilin Wang	Institute for Infocomm Research, Singapore
Dongho Won	Sungkyunkwan University, Korea
Sung-Ming Yen	National Central University, Taiwan, ROC
Yongjin Yeom	National Security Research Institute, Korea
Fangguo Zhang	Sun Yat-sen University, China
Alf Zugenmaier	DoCoMo Euro-Labs, Germany

Organizing Chair

Dong-gue Park	Soonchunhyang University, Korea

Organizing Committee

Hoon Ko	Information and Communications University(ICU), Korea
DaeHun Nyang	Inha University, Korea
Jintae Oh	Electronics and Telecommunications Research Institute(ETRI), Korea
Kangbin Yim	Soonchunhyang University, Korea
Joongcheol Moon	National Security Research Institute(NSRI), Korea
Changho Seo	Kongju National University, Korea
Jaesung Kim	Korea Information Security Agency(KISA), Korea
Sangjin Kim	Korea University of Technology and Education, Korea
Sehyun Park	Chung-Ang University, Korea

External Reviewers

Imad Aad	Ernest Foo	Wen-Chung Kuo
Sultan Zayid Al-Hinai	Georg Fuchsbauer	Eun Jeong Kwon
Man Ho Au	Soichi Furuya	Yunho Lee
Jean-Philippe Aumasson	D.J. Guan	HoonJae Lee
Vicente Benjumea	JaeCheol Ha	Tieyan Li
Jean-Luc Beuchat	Keisuke Hakuta	Wanqing Li
Annalisa De Bonis	Kyusuk Han	Vo Duc Liem
Reinier Broker	Dong-Guk Han	Wei-Chih Lien
Sebastien Canard	Chao-Chih Hsu	Hsi-Chung Lin
Ku-Young Chang	Xinyi Huang	Breno de Medeiros
Ee-Chien Chang	Sebastiaan Indesteege	Anton Mityagin
Jiun-Ming Chen	Toshiyuki Isshiki	Peter Montgomery
Chien-Ning Chen	Tetsu Iwata	Daesung Moon
Wang Chih-Hung	Tetsuya Izu	Yusuke Naito
S.Y. Chiou	Ik Rae Jeong	Toshiya Nakajima
Kuo-Zhe Chiou	Dimitar Jetchev	Cedric Ng
Dickson K.W. Chiu	Shaoquan Jiang	Mototsugu Nishioka
Yong-je Choi	Marcelo Kaihara	Masao Nonaka
Celine Coma	Nathan Keller	Katsuyuki Okeya
Nora Cuppens	Dmitry Khovratovich	Dag Arne Osvik
Rennie deGraff	Jin Ho Kim	Ranjan Pal
Holger Dreger	Shinsaku Kiyomoto	Tae Jun Park
Dang Nguyen Duc	Hiroki Koga	Maura Paterson
Chun-I Fan	Yuichi Komano	Kun Peng
Gerardo Fernandez	Divyan M. Konidala	Geong Sen Poh
Umberto Ferraro	Noboru Kunihiro	Deike Priemuth-Schmid

Roberto De Prisco
Havard Raddum
Matthieu Rivain
Rodrigo Roman
Chun Ruan
Akashi Satoh
Sven Schaege
Scarlet Schwiderski-Grosche
Jae Woo Seo
Siamak F. Shahandashti
Ning Shang
Jong Hoon Shin
Masaaki Shirase
Leonie Simpson
Claudio Soriente
Chunhua Su
Hongwei Sun
Kenichi Takahashi
Terry Lam Vinh The
Julien Thomas
Jacques Traore
Jheng-Hong Tu
Yoshifumi Ueshige
Masashi Une
Ivan Visconti
Camille Vuillaume
Shuhong Wang
Baodian Wei
Christopher Wolf
Bo-Ching Wu
Yeon-Hyeong Yang
Bo-Yin Yang
Shenglin Yang
Fan Zhang
Chang-An Zhao
Xingwen Zhao
Sebastien Zimmer

Sponsoring Institutions

CIST, Korea	http://cist.korea.co.kr/
KISIA, Korea	http://www.kisia.or.kr/
NICS Tech, Korea	http://www.nicstech.com/
NITGEN, Korea	http://www.nitgen.com/
STG Security, Korea	http://www.stgsecurity.com/
TSonNet, Korea	http://www.tsonnet.co.kr/

Table of Contents

Biometrics

Cryptographic Protocols

Hash Functions – I

Block and Stream Ciphers

Copyright Protection

Smart Cards

Elliptic Curve Cryptosystems

Hash Functions – II

Authentication and Authorization

Cryptanalysis of a Hash Function Proposed at ICISC 2006

Willi Geiselmann[1] and Rainer Steinwandt[2]

[1] Institut für Algorithmen und Kognitive Systeme, Fakultät für Informatik, Universität Karlsruhe (TH), Am Fasanengarten 5, 76128 Karlsruhe, Germany
geiselma@ira.uka.de
[2] Department of Mathematical Sciences, Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431, USA
rsteinwa@fau.edu

Abstract. A simple method for constructing collisions for Shpilrain's polynomial-based hash function from ICISC 2006 is presented. The attack relies on elementary linear algebra and can be considered as practical: For the parameters suggested, we give a specific collision, computed by means of a computer algebra system.

Keywords: cryptanalysis, hash function.

1 Introduction

In [Shp06] Shpilrain proposes a hash function H which builds on the Merkle-Damgård construction [Dam90, Mer90] and relies on computations in the quotient of a polynomial ring. In [Cha06] Chang reports that the underlying compression function is easy to invert and that a meet-in-the-middle attack enables a preimage attack on H. According to Chang's complexity estimate, for the specific parameters proposed in [Shp06] the computational effort for mounting such a preimage attack appears to be in the magnitude of 2^{80} operations.

The collision attack we describe below can be considered as practical—for the specific parameters proposed in [Shp06] we give a collision of two equal length bitstrings with about 10.2 KByte each. Shpilrain's proposed hash function H does not involve padding, but the collision given below remains valid if the usual Merkle-Damgård strengthening is applied to H.

2 The Proposal from ICISC 2006

Let $p(x) \in \mathbb{F}_2[x]$ be a univariate polynomial of degree n over the finite field with two elements. Moreover, let α be the residue class of x in the quotient $R := \mathbb{F}_2[x]/(p(x))$, thus $p(\alpha) = 0$. We remark that [Shp06] writes "$R = \mathbb{F}_{2^n} = \mathbb{F}_2[x]/(p(x))$" which suggests $p(x)$ to be irreducible, but the specific polynomial $p(x)$ proposed is reducible.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 1–10, 2007.

2.1 General Construction

To define the hash function H, two elements $h_0, h_1 \in R$ are fixed, and the hash value of an individual bit is defined as

$$\begin{array}{l} H(0) := h_0, \\ H(1) := h_1 \end{array} \quad . \tag{1}$$

Next, a triple $(u_0, u_1, u_2) \in R^3$ is used to fix a binary operation $\circ$ on R:

$$\begin{array}{rccc} \circ: & R^2 & \longrightarrow & R \\ & (r_1, r_2) & \longmapsto & r_1 \circ r_2 := u_0 + r_1 \cdot r_2 + r_1^2 \cdot u_1 + r_2^2 \cdot u_2 \end{array} \tag{2}$$

To hash a bitstring M, the following procedure is used:

1. Going from left to right, the bitstring M is split into 32-bit blocks $M = B_1 \parallel B_2 \parallel \cdots \parallel B_\ell$, where the last block B_ℓ has less than 32 bit, if the length of M is not a multiple of 32. There is no padding.
2. The hash value of each single 32-bit block $B_i = B_{i,0} \parallel \cdots \parallel B_{i,31}$ is computed by applying the above operation $\circ$ one bit at a time, going from left to right:

$$H(B_i) := (\ldots((H(B_{i,0}) \circ H(B_{i,1})) \circ H(B_{i,2}))\ldots) \circ H(B_{i,31})$$

   (where the hash value $H(B_{i,j})$ of a single bit $B_{i,j}$ is given by (1)).
3. The hash value $H(M)$ of M is computed by applying the operation $\circ$ one block at a time, going from left to right:

$$H(M) := (\ldots((H(B_0) \circ H(B_1)) \circ H(B_2))\ldots) \circ H(B_\ell)$$

The value $H(M)$ is the output of the hash function for input M.

2.2 Suggested Parameters

As specific parameter choice, [Shp06] suggests the following:

$$\begin{aligned} p(x) &:= x^{163} + x^7 + x^6 + x^5 + x^4 + x + 1 \\ h_0 &:= \alpha^7 + 1 \\ h_1 &:= \alpha^8 + 1 \\ (u_0, u_1, u_2) &:= (1, \alpha^2, \alpha) \end{aligned}$$

To demonstrate the practicality of the attack proposed below, in Section 3.3 we construct a specific collision for this parameter choice.

3 Finding Collisions

As already indicated above, the notation "$R = \mathbb{F}_{2^n} = \mathbb{F}_2[x]/(p(x))$" in [Shp06] suggests the considered polynomial $p(x)$ to be irreducible. However, with a

computer algebra system like Magma [BCP97] one easily checks that the proposed polynomial splits into four irreducible factors from $\mathbb{F}_2[x]$. Namely, for $p(x) = x^{163} + x^7 + x^6 + x^5 + x^4 + x + 1$ we have $p(x) = q_1(x) \cdot q_2(x) \cdot q_3(x) \cdot q_4(x)$, where

$$\begin{aligned}
q_1(x) &:= x^9 + x^7 + x^5 + x + 1, \\
q_2(x) &:= x^{18} + x^{14} + x^{12} + x^{11} + x^6 + x^4 + 1, \\
q_3(x) &:= x^{38} + x^{36} + x^{33} + x^{31} + x^{30} + x^{28} + x^{24} + x^{22} + x^{21} + x^{20} + x^{19} \\
&\quad + x^{17} + x^{16} + x^{12} + x^{10} + x^8 + x^7 + x^4 + x^3 + x^2 + 1, \\
q_4(x) &:= x^{98} + x^{94} + x^{93} + x^{91} + x^{90} + x^{88} + x^{87} + x^{84} + x^{82} + x^{73} + x^{69} \\
&\quad + x^{68} + x^{67} + x^{65} + x^{64} + x^{61} + x^{58} + x^{55} + x^{54} + x^{53} + x^{46} \\
&\quad + x^{45} + x^{44} + x^{43} + x^{42} + x^{41} + x^{39} + x^{37} + x^{31} + x^{29} + x^{28} \\
&\quad + x^{26} + x^{25} + x^{24} + x^{20} + x^{18} + x^{17} + x^{14} + x^{13} + x^9 + x^8 + x^7 \\
&\quad + x^6 + x^5 + x^3 + x^2 + 1 \quad .
\end{aligned}$$

Thus, before discussing the core part of our attack, it is worth discussing briefly how to exploit such a factorization for a collision search.

3.1 Using the Chinese Remainder Theorem

According to the Chinese Remainder Theorem, any factorization of the polynomial $p(x)$ into coprime factors $q_1(x) \ldots, q_s(x)$ yields a decomposition of the ring $R = \mathbb{F}_2[x]/(p(x))$ into a direct product of rings $R_i := \mathbb{F}_2[x]/(q_i(x))$:

$$R \simeq R_1 \times \cdots \times R_s$$

As the hash function H composes the hash values of the individual 32-bit blocks with simple ring operations, it looks tempting to exploit this isomorphism of rings to perform the collision search "one R_i at a time". Suppose we have found two bitstrings M_1, M_2 whose lengths are multiples of 32 and which satisfy

$$H(M_1) \equiv H(M_2) \pmod{q_s(x)} \quad ,$$

i. e., we have a collision in the R_s-component. Owing to the Merkle-Damgård structure of H, we then have

$$H(M_1 \parallel T) \equiv H(M_2 \parallel T) \pmod{q_s(x)}$$

for arbitrary bitstrings T appended to M_1 and M_2. Thus, if we heuristically (though actually incorrectly) take the values $H(M_1 \parallel T)$ and $H(M_2 \parallel T)$ as being uniformly and independently distributed modulo $q_{s-1}(x)$, we would expect that within $\mathrm{O}(2^{\deg(q_{s-1}(x))})$ random attempts for T, we encounter a pair of messages $M_1 \parallel T_{s-1}$, $M_2 \parallel T_{s-1}$ whose hash values coincide in the $R_{s-1} \times R_s$-component of R. If the degree of q_{s-1} is small, this approach can be efficient enough. In our experiments we used the linear algebra technique described in the next section to reduce the computational effort for finding a matching T_{s-1}.

Now assume we have found a matching "tail" T_{s-1} and that the length of T_{s-1} is a multiple of 32. Then we can apply the same reasoning as before to extend the collision

$$H(M_1 \parallel T_{s-1}) \equiv H(M_2 \parallel T_{s-1}) \pmod{q_{s-1}(x) \cdot q_s(x)}$$

from $R_{s-1} \times R_s$ to $R_{s-2} \times R_{s-1} \times R_s$: Analogously as before, now we test bitstrings T_{s-2} until

$$H(M_1 \parallel T_{s-1} \parallel T_{s-2}) \equiv H(M_2 \parallel T_{s-1} \parallel T_{s-2}) \pmod{q_{s-2} \cdot q_{s-1} \cdot q_s}$$

holds. In this way, we can process the components $R_s, R_{s-1}, \ldots, R_1$ one by one, starting from a collision in a single component.

For the specific parameters from Section 2.2 we have $s = 4$, and the degrees of $q_1(x)$, $q_2(x)$ and $q_3(x)$ are rather small—namely 9, 18 and 38. Thus, once we know a pair of messages colliding in the larger R_4-component (of size 2^{98}), deriving a full collision that is valid in R should be straightforward. Indeed, in our actual computations this worked as expected.

3.2 Using Linear Algebra

In view of the above discussion, the parameter choice in [Shp06] does not seem to offer an adequate security level, and constructing a collision in the component R_4 (of size 2^{98}) seems to be the most time-consuming task for mounting such an attack. In this section we show that such a collision can be found easily, without implementing a full birthday attack in R_4.

We describe the attack for an irreducible polynomial $p(x)$ of degree n, i. e., for $R \simeq \mathbb{F}_{2^n}$. For the specific parameter set from Section 2.2, this linear algebra based part is exploited for R_4 and R_3 only, but the attack technique as such does not rely on the described shortcut via the Chinese Remainder Theorem. In particular, simply imposing $p(x)$ to be irreducible of degree 163 does not appear to be an adequate countermeasure to rule out the attack.

Let $R' \subseteq R$ be the image of H when being restricted to messages whose length is a multiple of 32 (i. e., we have no incomplete last blocks). To each 32-bit block B, we can assign the following map ϕ_B, which captures the update of H's internal state when appending B to a message whose length is a multiple of 32.

$$\begin{aligned} \phi_B : R' &\longrightarrow R' \\ h &\longmapsto h \circ H(B) \end{aligned}$$

The map ϕ_B, is affine in the sense that it splits into the sum of the $\mathbb{F}_2$-linear map $h \mapsto h \cdot H(B) + h^2 \cdot u_1$ and the constant shift $H(B)^2 \cdot u_2 + u_0$. If we consider a sequence of blocks $B_1, \ldots, B_t$, then the composition

$$\phi_{B_1 \| B_2 \| \cdots \| B_t}(h) := \phi_{B_t}(\phi_{B_{t-1}}(\ldots \phi_{B_1}(h)) \ldots)$$

computes the hash value obtained by appending $B_1 \parallel B_2 \parallel \cdots \parallel B_t$ to a preimage of $h \in R'$. As each of the ϕ_{B_i} is affine in the sense just described, the same holds for $\phi_{B_1\|B_2\|\cdots\|B_t}$—with the constant shift depending on $B_1, \ldots, B_t$. The linear part of $\phi_{B_1\|B_2\|\cdots\|B_t}$ is just the functional composition of the linear parts of the ϕ_{B_i}s.

Once we know a sequence of 32-bit blocks $B_1, \ldots, B_t$ and two different values $h_1, h_2 \in R'$ with

$$\phi_{B_1\|B_2\|\cdots\|B_t}(h_1) = \phi_{B_1\|B_2\|\cdots\|B_t}(h_2) \quad ,$$

or equivalently

$$\phi_{B_1\|B_2\|\cdots\|B_t}(h_1) + \phi_{B_1\|B_2\|\cdots\|B_t}(h_2) = 0 \quad , \tag{3}$$

we have a collision for H—provided we know preimages of h_1 and h_2. As the left-hand side of Equation (3) is $\mathbb{F}_2$-linear in $h_1 + h_2$—the constant shifts cancel out in the summation—we can rewrite (3) in the form

$$\overline{(h_1 + h_2)} \cdot \mathcal{M}_{B_1\|B_2\|\cdots\|B_t} = 0 \quad .$$

Here $\mathcal{M}_{B_1\|B_2\|\cdots\|B_t}$ is an $n \times n$ matrix over $\mathbb{F}_2$, and $\overline{(h_1 + h_2)} \in \mathbb{F}_2^n$ is comprised of the coefficients of $h_1 + h_2$ when being expressed in the appropriate $\mathbb{F}_2$-vector space basis. Now, if we can find $B_1, \ldots B_t$ such that $\mathcal{M}_{B_1\|B_2\|\cdots\|B_t}$ is of low rank (i. e., has a large kernel) we can simply try to choose messages $M_1 \neq M_2$ at random until the sum of their hash values $(H(M_1) + H(M_2))$ yields a vector $\overline{(H(M_1) + H(M_2))}$ in the kernel of $\mathcal{M}_{B_1\|B_2\|\cdots\|B_t}$.

It is worth noting that there is no particular requirement on the messages M_1, M_2. This seems a useful feature when aiming at meaningful collisions: Suppose we have a message/file format of interest, where it is possible to append "garbage" at the end of a valid message (up to some fixed end-of-message delimiter).

Then we could fix two meaningful messages M_1', M_2' which we want to collide and choose our candidates as $M_1 := M_1' \parallel N_1$, $M_2 := M_2' \parallel N_2$ with random bitstrings N_1, N_2. The final colliding messages then had the form

$$M_1 = M_1' \parallel N_1 \parallel B_1 \parallel \cdots \parallel B_t \parallel E$$
$$M_2 = M_2' \parallel N_2 \parallel B_1 \parallel \cdots \parallel B_t \parallel E$$

where E can be a message-independent (possibly empty) end-of-message delimiter.

Expediting the computation of a kernel element. In our experiments with the parameters from Section 2.2, finding a small, say ≈16, number t of blocks $B_1, \ldots, B_t$ such that $\mathcal{M}_{B_1\|B_2\|\cdots\|B_t}$ has a rank defect of $\approx t$ required no particular effort. Already a trivial enumeration of some 32-bit blocks B_1 quickly yields a candidate where choosing all t blocks equal to B_1 results in a matrix

$\mathcal{M}_{B_1\|B_1\|\cdots\|B_1}$ with rank defect t. For larger rank defects, however, the heuristics we used required a significantly larger number of blocks (see below). Aiming at collisions of moderate length, it seems worthwhile to improve the simple guessing strategy for finding kernel elements:

Suppose our $n \times n$ matrix $\mathcal{M}_{B_1\|B_2\|\cdots\|B_t}$ over $\mathbb{F}_2$ has rank defect d. Taking the candidate vectors $\overline{(H(M_1) + H(M_2))}$ for independently and uniformly at random chosen elements from $\mathbb{F}_2^n$, we could expect that after $\mathrm{O}(2^{n-d})$ attempts a kernel vector is found. If we do not mandate M_1 and M_2 to have a particular form, we can easily improve on this as follows:

1. Using a computer algebra system, we can easily find a vector space basis of the (d-dimensional) kernel $\ker(\mathcal{M}_{B_1\|B_2\|\cdots\|B_t})$ of $\mathcal{M}_{B_1\|B_2\|\cdots\|B_t}$.
2. Using a birthday attack we search for messages M_1, M_2 such that the projections of $\overline{(H(M_1))}$, $\overline{(H(M_2))}$ on $\ker(\mathcal{M}_{B_1\|B_2\|\cdots\|B_t})$ coincide. In other words we want $\overline{(H(M_1))}$ and $\overline{(H(M_2))}$ to be in the same residue class of $\mathbb{F}_2^n / \ker(\mathcal{M}_{B_1\|B_2\|\cdots\|B_t})$. Then

$$\overline{(H(M_1))} + \overline{(H(M_2))} = \overline{(H(M_1) + H(M_2))} \in \ker(\mathcal{M}_{B_1\|B_2\|\cdots\|B_t})$$

   as desired.

Taking the $\overline{(H(M_i))}$ for independently and uniformly at random chosen elements from $\mathbb{F}_2^n$, we expect to find the desired messages M_1 and M_2 after $\mathrm{O}(2^{(n-d)/2})$ attempts.

For Shpilrain's specific parameter proposal (see Section 2.2), in the largest component obtained from the Chinese Remainder Theorem, we have $n = 98$. Here we used a matrix with a rank defect of $d = 42$, constructed from $t = 2882$ blocks B_i.

Finding a low rank matrix. By construction, we have

$$\mathcal{M}_{B_1\|B_2\|\cdots\|B_t} = \mathcal{M}_{B_1} \cdot \mathcal{M}_{B_2} \cdot \cdots \cdot \mathcal{M}_{B_{t-1}} \cdot \mathcal{M}_{B_t},$$

with $\mathcal{M}_{B_i}$ being the $n \times n$ matrix over $\mathbb{F}_2$ representing the linear part of ϕ_{B_i}. Thus, the task of finding a matrix $\mathcal{M}_{B_1\|B_2\|\cdots\|B_t}$ of low rank reduces to finding 32-bit blocks B_i such that we can form products of the respective matrices $\mathcal{M}_{B_i}$ with the product having low rank. Also, from a practical perspective it seems desirable that the number t of blocks is not too large, so that the resulting collision fits into, say, a few KByte.

In our experiments with the parameter set from Section 2.2, simple heuristics turned out to yield adequate blocks $B_1, \ldots, B_t$, and we did not attempt a thorough theoretical analysis or optimization of the task:

– For small values of t, say $t \approx 16$, already by just enumerating some 32-bit blocks B_i we quickly obtain candidates such that t identical blocks B_i yield a matrix $\mathcal{M}_{B_i\|B_i\|\cdots\|B_i}$ with rank defect t.

– Knowing a product $\mathcal{M}_{B_1} \cdot \cdots \cdot \mathcal{M}_{B_{t'}}$ of low rank, one can try to exhaust 32-bit blocks $B_{t'+1}$ until multiplying $\mathcal{M}_{B_1} \cdot \cdots \cdot \mathcal{M}_{B_{t'}}$ with $\mathcal{M}_{B_{t'+1}}$ reduces the rank further. Experimentally, this worked nicely for up to around $t' \approx 20$ blocks.
– If we have found a small number of matrix products $\mathcal{P}_1, \ldots, \mathcal{P}_v$ with a certain rank defect, we can try to form short products of these $\mathcal{P}_i$s and hope that the multiplication reduces the rank.
This procedure can be applied repeatedly and in our experiments worked quite nicely. The main drawback is that each $\mathcal{P}_i$ can already be derived from a number of 32-bit blocks B_j: if we form a product

$$\mathcal{P}' := \mathcal{P}_1 \cdot \cdots \cdot \mathcal{P}_{n_1}$$

of n_1 matrices $\mathcal{P}_i$ where each $\mathcal{P}_i$ is a product of n_2 matrices $\mathcal{M}_{B_j}$, then $\mathcal{P}'$ corresponds to $n_1 \cdot n_2$ 32-bit blocks B_j.

The next section shows that the above attack can be considered as practical: We use it to derive a collision for the parameter choice proposed in [Shp06] (see Section 2.2).

3.3 A Collision for the Proposed Parameters

As already mentioned, the specific polynomial $p(x)$ suggested by Shpilrain in [Shp06] splits into a product $p(x) = q_1(x) \cdot q_2(x) \cdot q_3(x) \cdot q_4(x)$ as specified in Section 3. Therefore we made use of the Chinese Remainder Theorem as discussed in Section 3.1.

A collision in $\mathbb{F}_{2^{98}}$ To construct a collision in $\mathbb{F}_2[x]/(q_4(x)) \simeq \mathbb{F}_{2^{98}}$ we applied the techniques from the previous section: Using a sequence

$$T_4' := B_1 \parallel \cdots \parallel B_{2882}$$

of 2882 suitably chosen 32-bit blocks, we derived a matrix $\mathcal{M}_{B_1 \parallel \cdots \parallel B_{2882}}$ of rank 56, i. e., with rank defect $d = 98 - 56 = 42$. To specify T_4', we define the following bitstrings (to be read from left to right, line by line):

$A_1 :=$ ‘003FF003 06B80000 06B20000 06B20000 06BA0000 06B0C000
06BA6800 06B4F400 06B6F400 06B52A00 06BB9600 06B9DC80
06BD1180 06B6AB20 06BEF3B0 06B2B470 06BDCAF0 06B11ACC
06B90F3C 06B3B432 06B49CCA 06BB6E03’ $(22 \cdot 32$ bit$)$

$A_2 :=$ ‘003FF003 06A80000 06AA0000 06A50000 06A84000 06A24000
06A22000 06A16800 06AE8400 06AE1C00 06ADAE00 06A9D500
06A3B780 06AC29C0 06AD93C0 06A7E260 06A874C2 06A85DCA
06A7A3B9 06ABAF95 06A84DFD’ $(21 \cdot 32$ bit$)$

$A_3 :=$ ‘003FF003 06CC0000 06C20000 06CA8000 06CF8000 06C84000
06C64000 06C17000 06CF4800 06C98400 06CB2900 06CE8080
06C79080 06C95080 06C2A948 06CBCE28 06C00214 06CC572C
06C70021’ $(19 \cdot 32$ bit)

$A_4 :=$ ‘003FF003 06F80000 06F80000 06F80000 06F88000 06F98000
06FBA000 06F13000 06F04800 06FE9C00 06F32E00 06FEEE00
06FA9180 06F4CDC0 06F88EB0 06F0BEF0 06FE26A8 06FB3B78’
$(18 \cdot 32$ bit)

$A_5 :=$ ‘003FF003 00000000 00010000 00038000 0009C000 000CE000
00065000 0007D000 00033000 000D1400 00033C00 000D0900
00008080 000CD020 000A9FA0 0009EEF0 000BDE0C 000A944C
00031A4A 0007A5FE 001F97E7 004081C9 006AC9DC 008039BD
01C1E775 031A68F0 0E217B84’ $(27 \cdot 32$ bit)

At this each hexadecimal digit represents a sequence of 4 bits (‘0’ – ‘0000’, ‘1’ – ‘0001’,..., ‘E’ – ‘1110’, ‘F’ – ‘1111’). Using $A_1, \ldots, A_5$ as building blocks, we define eight more bitstrings:

$$A_6 := A_5 \parallel A_5 \parallel A_4 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_3 \parallel A_5 \quad (226 \text{ blocks})$$
$$A_7 := A_5 \parallel A_5 \parallel A_4 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_3 \parallel A_5 \quad (199 \text{ blocks})$$
$$A_8 := A_5 \parallel A_5 \parallel A_4 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_3 \parallel A_5 \parallel A_5 \quad (226 \text{ blocks})$$
$$A_9 := A_5 \parallel A_5 \parallel A_4 \parallel A_5 \parallel A_2 \parallel A_1 \parallel A_5 \parallel A_5 \quad (196 \text{ blocks})$$
$$A_{10} := A_5 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_3 \parallel A_5 \parallel A_3 \parallel A_3 \parallel A_5 \parallel A_5 \quad (246 \text{ blocks})$$
$$A_{11} := A_5 \parallel A_3 \parallel A_3 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_3 \parallel A_3 \parallel A_5 \parallel A_5 \quad (265 \text{ blocks})$$
$$A_{12} := A_5 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_3 \parallel A_3 \parallel A_2 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_5 \quad (275 \text{ blocks})$$
$$A_{13} := A_5 \parallel A_4 \parallel A_5 \parallel A_5 \parallel A_5 \parallel A_3 \parallel A_3 \parallel A_5 \parallel A_5 \quad (218 \text{ blocks})$$

In terms of $A_6, \ldots, A_{13}$, the bitstring T_4' can be described as follows:

$$T_4' := A_{11} \parallel A_{12} \parallel A_7 \parallel A_{11} \parallel A_{13} \parallel A_{10} \parallel A_9 \parallel A_6 \parallel A_{11} \parallel A_{12} \parallel A_8 \parallel A_6$$

Next, with a birthday attack as described we found two 32-bit blocks

$$M_1 := \text{‘2B99EF46’ and } M_2 := \text{‘02B6CF84’}$$

with $\overline{(H(M_1) + H(M_2))}$ being in the kernel of $\mathcal{M}_{B_1 \parallel \cdots \parallel B_{2882}}$. Consequently we obtain

$$H(M_1 \parallel T_4') \equiv H(M_2 \parallel T_4') \pmod{q_4(x)} \quad . \tag{4}$$

Pruning T_4'. Inspecting $M_1 \parallel T_4'$ and $M_2 \parallel T_4'$ more closely, it turns out that (4) remains valid, if we remove the last 300 blocks from T_4'. We write T_4 for the bitstring of length $2582 \cdot 32 = 2882 \cdot 32 - 300 \cdot 32$ resulting from pruning T_4' accordingly. In particular, we have

$$H(M_1 \parallel T_4) \equiv H(M_2 \parallel T_4) \pmod{q_4(x)} \quad . \tag{5}$$

Applying the Chinese Remainder Theorem. Next, we want to identify bitstrings T_3, T_2, T_1 such that

$$H(M_1 \parallel T_4 \parallel \cdots \parallel T_i) \equiv H(M_2 \parallel T_4 \parallel \cdots \parallel T_i) \pmod{q_1(x) \cdot \cdots \cdot q_i(x)}$$

holds for $1 \leq i \leq 4$.

The polynomial $q_3(x)$ is of degree 38. To extend the "$\mathbb{F}_{2^{98}}$-collision" in (5) accordingly, the linear algebra approach from before can be reused: First, we identify a short bitstring

$T_3' :=$ `'003FF003 06300000 06320000 063E8000 06394000 0638C000 0639A000 063C6000 0633D000 063A3400 063DBA00 0633BC80 06395B80 0637AC40 0635AF10 0636CB38 063CF824 063EEE8C'` (18 blocks)

which, when "hashing modulo $q_3(x)$", corresponds to a matrix $\mathcal{M}_{T_3'}$ of low rank. Then we enumerate short bitstrings, until a candidate

$$T_3'' := \texttt{'00171999'}$$

is found such that $H(M_1 \parallel T_4 \parallel T_3'') + H(M_2 \parallel T_4 \parallel T_3'') \pmod{q_3(x)}$ yields a vector in the kernel of $\mathcal{M}_{T_3'}$. Defining T_3 as $T_3 := T_3'' \parallel T_3'$, we have

$$H(M_1 \parallel T_4 \parallel T_3) \equiv H(M_2 \parallel T_4 \parallel T_3) \pmod{q_4(x) \cdot q_3(x)} \tag{6}$$

as desired. Extending the collision in (6) to the complete quotient ring $\mathbb{F}_2[x]/(q_1(x) \cdot q_2(x) \cdot q_3(x) \cdot q_4(x))$ turns out to be straightforward: Appending one more 32-bit block

$$T_2 := \texttt{'0008D718'}$$

already yields the desired collision

$$H(M_1 \parallel T_4 \parallel T_3 \parallel T_2) = H(M_2 \parallel T_4 \parallel T_3 \parallel T_2) \quad .$$

Thus, we have found two different bitstrings of size $2603 \cdot 32$ bit (i. e., ≈10.2 KByte), both of which hash to the same value.

For computing this collision we used the computer algebra system Magma [BCP97] on a number of different hardware platforms. We estimate our computational effort to be in the magnitude of one CPU day on a standard PC with about 8 GByte RAM.

4 Conclusion

As explained in the above discussion and demonstrated through a specific collision, the hash function proposed in [Shp06] does not offer strong collision resistance. Consequently, for applications that rely on collision resistance, the use of this hash function does not seem to be advisable.

Acknowledgments

We would like to thank Markus Grassl, Kenneth Matheis and Viktória Ildikó Villányi for interesting discussions.

References

[BCP97] Bosma, W., Cannon, J.J., Playoust, C.: The Magma Algebra System I: The User Language. Journal of Symbolic Computation 24, 235–265 (1997)

[Cha06] Chang, D.: Preimage Attack on Hashing with Polynomials proposed at ICISC 2006. Cryptology ePrint Archive: Report 2006/411(2006), available at http://eprint.iacr.org/2006/411

[Dam90] Damgård, I.B.: A Design Principle for Hash Functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, Heidelberg (1990)

[Mer90] Merkle, R.C.: A Certified Digital Signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (1990)

[Shp06] Shpilrain, V.: Hashing with Polynomials. In: Rhee, M.S., Lee, B. (eds.) ICISC 2006. LNCS, vol. 4296, pp. 22–28. Springer, Heidelberg (2006)

Cryptanalysis of Reduced Versions of the HIGHT Block Cipher from CHES 2006*

Jiqiang Lu

Information Security Group, Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
lvjiqiang@hotmail.com

Abstract. HIGHT is a 32-round block cipher with a 64-bit block size and a 128-bit user key, which was proposed at CHES '06 for low-resource applications like RFID. In this paper, we present an impossible differential attack on 25-round HIGHT, a related-key rectangle attack on 26-round HIGHT, and finally a related-key impossible differential attack on 28-round HIGHT. Our result suggests that the safety margin of HIGHT decreases from the originally expected thirteen rounds to about four rounds now.

Keywords: Block cipher, HIGHT, Impossible differential cryptanalysis, Rectangle attack, Related-key attack.

1 Introduction

Recently, cryptography for embedded and ubiquitous computing systems receives an extensive research attention. At CHES '06, Hong et al. [9] presented a 32-round block cipher with a 64-bit block size and a 128-bit user key, known as HIGHT. Due to the simple byte-oriented operations involved, HIGHT is especially efficient in hardware implementations, much faster than those [7,8] of AES [19], and it is most suitable for various real-life resource-constrained application environments, such as RFID (Radio Frequency Identification) systems. The HIGHT proposers also analysed its security against various existing cryptanalytic attacks; they described a differential attack [6], a linear attack [18] and a boomerang attack [20] on 13-round HIGHT, a truncated differential attack [14] and a saturation attack [17] on 16-round HIGHT, an impossible differential attack [2,15] on 18-round HIGHT, and finally a related-key [1,12] boomerang attack [5] on 19-round HIGHT.

In this paper, we further analyse the security of HIGHT. We exploit 16-round impossible differentials such that we can devise an impossible differential attack on 25-round HIGHT; we also exploit 18-round related-key rectangle distinguishers with probability $2^{-92.4}$, which can enable us to mount a related-key rectangle

* This work as well as the author was supported by a British Chevening / Royal Holloway Scholarship and the European Commission under contract IST-2002-507932 (ECRYPT).

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 11–26, 2007.

attack on 26-round HIGHT. Finally, we exploit 19-round related-key impossible differentials that can be used to mount a related-key impossible differential attack on 28-round HIGHT.

The rest of this paper is organised as follows. In the next section, we briefly describe some notation and the HIGHT block cipher. In Sections 3 and 4, we present our cryptanalytic results. Section 5 concludes this paper.

2 Preliminaries

2.1 Notation

We will use the following notation throughout this paper.

- $\oplus$: bitwise logical exclusive OR (XOR)
- $\boxplus$: addition modulo 2^8
- $\lll i$: left rotation by i bits
- e_j : a byte with zeros in all positions but bit j, $(0 \leq j \leq 7)$
- $e_{i_1,\cdots,i_j}$: $e_{i_1} \oplus \cdots \oplus e_{i_j}$, $(0 \leq i_1, \cdots, i_j \leq 7)$
- $e_{j,\sim}$: a byte that has zeros in bits 0 to $j-1$, a one in bit j and indeterminate values in bits $(j+1)$ to 7
- $e_{\overline{j},\sim}$: a byte that has zeros in bits 0 to j and indeterminate values in bits $(j+1)$ to 7
- ? : an arbitrary byte, where two bytes represented by the ? symbol may be different

The notion of difference used throughout this paper is with respect to the $\oplus$ operation. It is assumed that in a byte the rightmost bit is the least significant bit and referred as the 0-th bit, and the leftmost bit is the most significant bit and referred as the 7-th bit.

2.2 The HIGHT Block Cipher

HIGHT [9] takes as an input a 64-bit plaintext P, represented as eight bytes $(P_7, \cdots, P_1, P_0)$, and it has a total of 32 rounds. Let $(X_{i-1,7}, X_{i-1,6}, X_{i-1,5}, X_{i-1,4}, X_{i-1,3}, X_{i-1,2}, X_{i-1,1}, X_{i-1,0})$ denote the eight-byte input to Round i, and $(X_{i,7}, X_{i,6}, X_{i,5}, X_{i,4}, X_{i,3}, X_{i,2}, X_{i,1}, X_{i,0})$ denote the eight-byte output of Round i, $(1 \leq i \leq 32)$. The encryption procedure can be described as follows.

1. Perform the Initial Transformation: the eight-byte output $(X_{0,7}, X_{0,6}, X_{0,5}, X_{0,4}, X_{0,3}, X_{0,2}, X_{0,1}, X_{0,0}) = (P_7, P_6 \oplus \mathrm{WK}_3, P_5, P_4 \boxplus \mathrm{WK}_2, P_3, P_2 \oplus \mathrm{WK}_1, P_1, P_0 \boxplus \mathrm{WK}_0)$.
2. For $i = 1$ to 32:
   $X_{i,0} = X_{i-1,7} \oplus (F_0(X_{i-1,6}) \boxplus \mathrm{SK}_{4i-1})$,
   $X_{i,1} = X_{i-1,0}$,
   $X_{i,2} = X_{i-1,1} \boxplus (F_1(X_{i-1,0}) \oplus \mathrm{SK}_{4i-2})$,
   $X_{i,3} = X_{i-1,2}$,
   $X_{i,4} = X_{i-1,3} \oplus (F_0(X_{i-1,2}) \boxplus \mathrm{SK}_{4i-3})$,

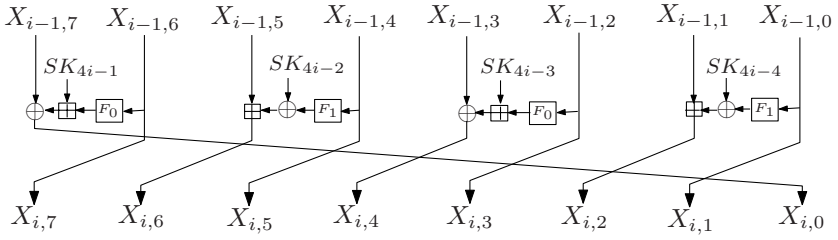


Fig. 1. The i-th encryption round of HIGHT

$X_{i,5} = X_{i-1,4}$,
$X_{i,6} = X_{i-1,5} \boxplus (F_1(X_{i-1,4}) \oplus \mathrm{SK}_{4i-4})$,
$X_{i,7} = X_{i-1,6}$.

3. Perform the Final Transformation: the ciphertext $C = (C_7, C_6, C_5, C_4, C_3, C_2, C_1, C_0) = (X_{32,0}, X_{32,7} \oplus \mathrm{WK}_7, X_{32,6}, X_{32,5} \boxplus \mathrm{WK}_6, X_{32,4}, X_{32,3} \oplus \mathrm{WK}_5, X_{32,2}, X_{32,1} \boxplus \mathrm{WK}_4)$.

In the above description, SK_i $(0 \le i \le 127)$ are the round subkeys, WK_j $(0 \le j \le 7)$ are the whitening subkeys used in the initial and final transformation, the functions $F_0(\cdot)$ and $F_1(\cdot)$ are defined as $F_0(x) = (x \quad 1) \oplus (x \quad 2) \oplus (x \quad 7)$, and $F_1(x) = (x \quad 3) \oplus (x \quad 4) \oplus (x \quad 6)$. Note that the first round is referred as Round 1. Fig. 1 depicts one encryption round of HIGHT.

The key schedule of HIGHT only accepts a 128-bit user key MK, represented as sixteen bytes $(\mathrm{MK}_{15}, \cdots, \mathrm{MK}_1, \mathrm{MK}_0)$. The whitening subkeys WK_j are generated as follows: $\mathrm{WK}_j = \mathrm{MK}_{j+12}$ for $j = 0, 1, 2, 3$, and $\mathrm{WK}_j = \mathrm{MK}_{j-4}$ for $j = 4, 5, 6, 7$. The round subkeys are generated as follows: $\mathrm{SK}_{16 \cdot i+j} = \mathrm{MK}_{j-i \bmod 8} \boxplus \delta_{16 \cdot i+j}$, or $\mathrm{SK}_{16 \cdot i+j+8} = \mathrm{MK}_{(j-i \bmod 8)+8} \boxplus \delta_{16 \cdot i+j+8}$ $(0 \le i, j \le 7)$, where $\delta_{16 \cdot i+j}$ and $\delta_{16 \cdot i+j+8}$ are public constants.

3 Impossible Differential Attack on 25-Round HIGHT

3.1 16-Round Impossible Differentials

We exploit certain 16-round impossible differentials: $(e_{i,\sim}, 0, 0, 0, 0, 0, 0, 0) \nrightarrow (e_{0,3,5,6,7}, 0, 0, 0, 0, 0, 0, e_7)$, where $1 \le i \le 7$. Note that the 16-round differentials $(e_7, e_{0,3,5,6,7}, 0, 0, 0, 0, 0, 0) \rightarrow (0, e_{i,\sim}, 0, 0, 0, 0, 0, 0)$ are also impossible. These 16-round impossible differentials are mainly because the following general property.

Property 1. [illegible]

The 16-round impossible differentials are built in a miss-in-the-middle manner [3]: a 8-round differential $(e_{i,\sim}, 0, 0, 0, 0, 0, 0, 0) \rightarrow (e_{i,\sim}, ?, ?, ?, ?, ?, ?, ?)$ with

probability 1 is concatenated with another 8-round differential $(e_{0,\sim}, 0, ?, ?, ?, ?, ?, ?) \leftarrow (e_{0,3,5,6,7}, 0, 0, 0, 0, 0, 0, e_7)$ with probability 1, but the leftmost bytes of the intermediate differences of these two differentials contradict one another. The input difference $(e_{i,\sim}, 0, 0, 0, 0, 0, 0, 0)$ of the first 8-round differential propagates to a difference $(0, 0, 0, 0, 0, 0, 0, e_{i,\sim})$ after one round of HIGHT, which then propagates to a difference $(0, 0, 0, 0, 0, ?, e_{i,\sim}, 0)$ after another round. As a result, the difference $(0, 0, 0, 0, 0, ?, e_{i,\sim}, 0)$ finally propagates to a difference $(e_{i,\sim}, ?, ?, ?, ?, ?, ?, ?)$ after the following six rounds. On the other hand, when we roll back the difference $(e_{0,3,5,6,7}, 0, 0, 0, 0, 0, 0, e_7)$ through one round of HIGHT in the reverse direction, then we will definitely get the difference $(0, e_{0,3,5,6,7}, 0, 0, 0, 0, 0, 0)$, as the difference $e_{0,3,5,6,7}$ becomes $(e_{0,3,5,6,7} \quad 1) \oplus (e_{0,3,5,6,7} \quad 2) \oplus (e_{0,3,5,6,7} \quad 7) = e_{0,1,4,6,7} \oplus e_{0,1,2,5,7} \oplus e_{2,4,5,6,7} = e_7$ after the F_0 function. The difference $(0, e_{0,3,5,6,7}, 0, 0, 0, 0, 0, 0)$ propagates to a difference $(e_{0,\sim}, 0, ?, ?, ?, ?, ?, ?)$ when we roll it back through seven more rounds. Now a contradiction occurs if $i \neq 0$, as the leftmost byte difference of one of the two intermediate differences is $e_{i,\sim}$ while the leftmost byte difference of the other is $e_{0,\sim}$.

3.2 Attacking Rounds 6–30

HIGHT has a Feistel structure with four branches, which can be efficiently implemented. However, we observe this round structure is much weaker than a regular Feistel structure, in terms of security.

Property 2. [illegible] $(i+1)$ [illegible] $(i+2)$ $(1 \leq i \leq 29)$

Property 2 suggests that to get a byte value (or difference) of the input to a round we need not guess all the twelve 8-bit subkeys in its following three rounds; and we can determine whether a candidate pair is a right pair byte by byte, and even bit by bit, due to the round struture and the operations involved. This plays an important role in our attacks.

We can use the 16-round impossible differentials to break 25-round HIGHT. Here, we attack Rounds 6 to 30 of HIGHT with the final transformation only. The attack procedure is as follows.

1. Choose 2^{13} structures of 2^{47} plaintexts, where the two bytes (0,1) and the least significant bits of the third bytes of the 2^{47} plaintexts in a structure are fixed to certain values, and the other 47 bit positions take all the possible values. Obviously, a structure proposes $2^{47\times 2}/2 = 2^{93}$ plaintext pairs (P^i, P^j) with difference $(?, ?, ?, ?, ?, e_{\overline{0},\sim}, 0, 0)$, $(i, j = 1, 2, \cdots, 2^{47})$, thus the 2^{13} structures propose a total of 2^{106} plaintext pairs.
2. In a chosen-plaintext attack scenario, obtain all the ciphertexts C^i of the plaintexts P^i. Choose only the ciphertext pairs (C^i, C^j) with difference $(?, ?, ?, ?, ?, e_{0,\sim}, 0, 0)$.
3. Guess the two key bytes $(\mathrm{MK}_0, \mathrm{MK}_3)$, compute the subkeys $(\mathrm{WK}_7, \mathrm{SK}_{119})$ in the final transformation and Round 30, and do the following.

(a) Partially decrypt every remaining ciphertext pair (C^i, C^j) with (WK_7, SK_{119}) to get the two bytes (7,6) of their intermediate values just before Round 30, and check if they have a difference $(0, ?)$. Keep only the qualified pairs.

(b) Guess the two key bytes $(\mathrm{MK}_2, \mathrm{MK}_7)$, compute the subkeys $(\mathrm{WK}_6, \mathrm{SK}_{118})$ in the final transformation and Round 30, and compute the subkey SK_{114} in Round 29 with MK_3 guessed above. Partially decrypt every remaining (C^i, C^j) with $(\mathrm{WK}_6, \mathrm{SK}_{114}, \mathrm{SK}_{118})$ to get the two bytes (5,4) of their intermediate values just before Round 29.[1] Check if they have a difference $(0, ?)$. Keep only the qualified pairs.

(c) Guess the 8 key bits MK_1, compute the subkey WK_5 in the final transformation, and do as follows.

i. Guess the least significant bit $\mathrm{MK}_{6,0}$ of the key byte MK_6, and compute the least significant bit $\mathrm{SK}_{117,0}$ of the subkey SK_{117} in Round 30. Partially decrypt every remaining (C^i, C^j) with $(\mathrm{WK}_5, \mathrm{SK}_{117,0})$ to get their intermediate values $(X^i_{29,3,0}, X^i_{29,2})$ and $(X^j_{29,3,0}, X^j_{29,2})$ just before Round 30. Keep the pairs such that $X^i_{29,3,0} \oplus X^j_{29,3,0} = 1$.

ii. Guess the other seven bits $\mathrm{MK}_{6,1-7}$ of MK_6, and compute the subkey SK_{117} (together with $\mathrm{MK}_{6,0}$ guessed above). Partially decrypt every remaining (C^i, C^j) with $(\mathrm{WK}_5, \mathrm{SK}_{117})$ to get the two bytes (2,3) of their intermediate values just before Round 30.

(d) Compute the subkey SK_{113} in Round 29 with MK_2 guessed above. For every remaining (C^i, C^j), partially decrypt the two bytes (4,3) of their intermediate values just before Round 30 with SK_{113} to get the two bytes (3,2) of their intermediate values just before Round 29. Check if they have a difference $(e_{2,\sim}, e_{0,\sim})$. Keep only the qualified pairs.

(e) For $l = 0$ to 7, do as follows.

- Guess the l-th bit $\mathrm{MK}_{15,l}$ of the key byte MK_{15}, and compute the $(l+1)$ bits $\mathrm{SK}_{109,0-l}$ of the subkey SK_{109} in Round 28.
- For every remaining (C^i, C^j), partially decrypt the two bytes (4,3) of their intermediate values just before Round 29 with $\mathrm{SK}_{109,0-l}$ to get their intermediate values $(X^i_{27,3,0-l}, X^i_{27,2})$ and $(X^j_{27,3,0-l}, X^j_{27,2})$ just before Round 28. Keep the pairs such that $X^i_{27,3,0-l} = X^j_{27,3,0-l}$.

(f) Guess the 8 key bits MK_5, compute the subkey SK_{116} in Round 30, and compute the subkeys $(\mathrm{WK}_4, \mathrm{SK}_{112})$ in the final transformation and Round 29 with $(\mathrm{MK}_0, \mathrm{MK}_1)$ guessed above. Partially every remaining (C^i, C^j) with $(\mathrm{WK}_4, \mathrm{SK}_{112}, \mathrm{SK}_{116})$ to get the two bytes (1,0) of their intermediate values just before Round 29. Check if they have the difference $(e_{0,3,5,6,7}, 0)$. Keep only the qualified pairs.

(g) Guess the least significant bit $\mathrm{MK}_{14,0}$ of the key byte MK_{14}; for $l = 1$ to 7, do as follows.

[1] The other required intermediate values have been obtained in the previous steps. Same for some following steps as well as the attacks in the next section, without explicit statement.

- Guess the l-th bit $\mathrm{MK}_{14,l}$ of the key byte MK_{14}, and compute the $(l+1)$ bits $\mathrm{SK}_{108,0-l}$ of the subkey SK_{108} in Round 28.
- For every remaining (C^i, C^j), partially decrypt the two bytes (1,2) of their intermediate values just before Round 29 with $\mathrm{SK}_{108,0-l}$ to get their intermediate values $(X^i_{27,1,0-l}, X^j_{27,1,0-l})$ just before Round 28. If $l \neq 7$, keep the pairs such that $X^i_{27,1,0-l} = X^j_{27,1,0-l}$; if $l = 7$, keep the pairs $X^i_{27,1,0-l} \oplus X^j_{27,1,0-l} = e_7$.

(h) Guess the least significant 3 bits $\mathrm{MK}_{10,0-2}$ of the key byte MK_{10}; for $l = 3$ to 7, do as follows.

- Guess the l-th bit $\mathrm{MK}_{10,l}$ of the key byte MK_{10}, and compute the $(l+1)$ bits $\mathrm{SK}_{104,0-l}$ of the subkey SK_{104} in Round 27.
- For every remaining (C^i, C^j), partially decrypt the two bytes (1,2) of their intermediate values just before Round 28 with $\mathrm{SK}_{104,0-l}$ to get their intermediate values $(X^i_{26,1,0-l}, X^j_{26,1,0-l})$ just before Round 27. Keep the pairs such that $X^i_{26,1,0-l} = X^j_{26,1,0-l}$.

4. Compute the subkey SK_{23} with MK_6 guessed in Step 3, and do the following.
   (a) Partially encrypt every plaintext pair (P^i, P^j) corresponding to a remaining ciphertext pair (C^i, C^j), with SK_{23} to get the two bytes (7,0) of their intermediate values just after Round 6. Check if they have a difference $(?, 0)$. Keep only the qualified pairs.
   (b) Compute the subkeys $(\mathrm{SK}_{22}, \mathrm{SK}_{27})$ with $(\mathrm{MK}_5, \mathrm{MK}_{10})$ guessed in Step 3. Partially encrypt every remaining (P^i, P^j) with $(\mathrm{SK}_{22}, \mathrm{SK}_{27})$ to get the two bytes (7,0) of their intermediate values just after Round 7. Check if they have a difference $(?, 0)$. Keep only the qualified pairs.
   (c) Guess the two key bytes $(\mathrm{MK}_4, \mathrm{MK}_9)$, compute the subkeys $(\mathrm{SK}_{21}, \mathrm{SK}_{26})$ in Rounds 6 and 7, and compute the subkey SK_{31} with MK_{14} guessed in Step 3. Partially encrypt every remaining (P^i, P^j) with $(\mathrm{SK}_{21}, \mathrm{SK}_{26}, \mathrm{SK}_{31})$ to get the two bytes (7,0) of their intermediate values just after Round 8. Check if they have a difference $(?, 0)$. Keep only the qualified pairs.
   (d) Guess the two key bytes $(\mathrm{MK}_8, \mathrm{MK}_{13})$, compute the subkeys $(\mathrm{SK}_{25}, \mathrm{SK}_{30})$ in Rounds 7 and 8, and compute the subkeys $(\mathrm{SK}_{20}, \mathrm{SK}_{35})$ with $(\mathrm{MK}_1, \mathrm{MK}_3)$ guessed in Step 3. Partially encrypt every remaining (P^i, P^j) with $(\mathrm{SK}_{20}, \mathrm{SK}_{25}, \mathrm{SK}_{30}, \mathrm{SK}_{35})$ to get the two bytes (7,0) of their intermediate values just after Round 9. Check if they have a difference $(?, 0)$. Keep only the qualified pairs.
   (e) Guess the key byte MK_{12}, compute the subkey SK_{29}, and compute the subkeys $(\mathrm{SK}_{24}, \mathrm{SK}_{34}, \mathrm{SK}_{39})$ with $(\mathrm{MK}_0, \mathrm{MK}_5, \mathrm{MK}_{15})$ guessed in Step 3. Partially encrypt every remaining (P^i, P^j) with $(\mathrm{SK}_{24}, \mathrm{SK}_{29}, \mathrm{SK}_{34}, \mathrm{SK}_{39})$ to get the two bytes (7,0) of their intermediate values just after Round 10. Check if they have a difference $(e_{\overline{0},\sim}, 0)$. If none of the plaintext pairs satisfies this test, record the guessed 120 key bits $(\mathrm{MK}_0, \cdots, \mathrm{MK}_{10}, \mathrm{MK}_{12}, \cdots, \mathrm{MK}_{15})$, and execute Step 5; otherwise, discard this guess and try another.
5. For a recorded $(\mathrm{MK}_0, \cdots, \mathrm{MK}_{10}, \mathrm{MK}_{12}, \cdots, \mathrm{MK}_{15})$, exhaustively search for the remaining 8 key bits with three known pairs of plaintexts and ciphertexts.

If a 128-bit key is suggested, output it as the user key of the 25-round HIGHT; otherwise, go to Step 3.

There are 17-bit, 8-bit, 8-bit, 1-bit, 3-bit and 7-bit filtering conditions over the ciphertext pairs in Steps 2 and 3-(a)∼(d) and (f), respectively, and a 1-bit filtering condition in every iteration of Steps 3-(e), (g) and (h). Thus, it is expected about $2^{106} \cdot 2^{-64} = 2^{42}$ ciphertext pairs remain after Step 3. There is a 8-bit filtering condition in each of Steps 4-(a)∼(e), so it follows that about $2^{120} \cdot (1-2^{-8})^{2^{10}} \approx 2^{120} \cdot e^{-2^2} \approx 2^{114.24}$ guesses of the 120 key bits are recorded in Step 4-(e). The probability that a wrong key is suggested in Step 5 is approximately $2^{-64\times 3} = 2^{-192}$, thus, the expected number of wrong keys in Step 5 is about $2^{-192} \cdot 2^{114.24+8} = 2^{-73.76}$. It is very likely that we can find the correct key.

The attack requires 2^{60} chosen plaintexts. Step 3 has about $2 \cdot 2^{89} \cdot 2^{16} \cdot \frac{1}{4} \cdot \frac{1}{25} + 2 \cdot 2^{81} \cdot 2^{32} \cdot \frac{1}{4} \cdot \frac{2}{25} + 2 \cdot 2^{73} \cdot 2^{41} \cdot \frac{1}{4} \cdot \frac{1}{25} + 2 \cdot 2^{72} \cdot 2^{48} \cdot \frac{1}{4} \cdot \frac{1}{25} + 2 \cdot 2^{72} \cdot 2^{48} \cdot \frac{1}{4} \cdot \frac{1}{25} + \sum_{l=0}^{7}(2 \cdot 2^{69-l} \cdot 2^{48+l+1} \cdot \frac{1}{4} \cdot \frac{1}{25}) + 2 \cdot 2^{61} \cdot 2^{64} \cdot \frac{1}{4} \cdot \frac{1}{25} + \sum_{l=0}^{6}(2 \cdot 2^{54-l} \cdot 2^{64+2+l} \cdot \frac{1}{4} \cdot \frac{1}{25}) + \sum_{l=0}^{4}(2 \cdot 2^{47-l} \cdot 2^{72+4+l} \cdot \frac{1}{4} \cdot \frac{1}{25}) \approx 2^{120.73}$ computations. Step 4 has about $2 \cdot 2^{42} \cdot 2^{80} \cdot \frac{1}{4} \cdot \frac{1}{25} + 2 \cdot 2^{34} \cdot 2^{80} \cdot \frac{1}{4} \cdot \frac{2}{25} + 2 \cdot 2^{26} \cdot 2^{96} \cdot \frac{1}{4} \cdot \frac{3}{25} + 2 \cdot 2^{18} \cdot 2^{112} \cdot \frac{1}{4} \cdot \frac{4}{25} + 2 \cdot 2^{120}[1 + (1-2^{-8}) + \cdots + (1-2^{-8})^{2^{10}}] \cdot \frac{1}{4} \cdot \frac{4}{25} \approx 2^{126.68}$ computations. Step 5 has about $2^{122.24}$ computations. Therefore, the attack has a total time complexity of about $2^{126.75}$ 25-round HIGHT computations.

4 Related-Key Cryptanalysis of Reduced HIGHT

A related-key attack [1,12] assumes that the attacker knows the differences between one or more pairs of unknown keys. In this section, we present a related-key rectangle attack on 26-round HIGHT, and a related-key impossible differential attack on 28-round HIGHT.

4.1 Related-Key Rectangle Attack on 26-Round HIGHT

A related-key rectangle attack [5,10,13] is a combination of a related-key attack and a rectangle attack [4]; it is based on a related-key rectangle distinguisher, which treats a block cipher $E : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$ as a cascade of two sub-ciphers $E = E^1 \circ E^0$.

18-Round Related-Key Rectangle Distinguishers of HIGHT. Let E^0 denote Rounds 3 to 12, and E^1 denote Rounds 13 to 20. The first related-key differential for this 18-round distinguisher is the following 10-round related-key differential $\alpha \to \beta$ with probability 2^{-12} for E^0: $(e_{1,3,5}, e_{0,1,6}, e_7, 0, 0, 0, 0, 0) \to (0, e_{1,5,6}, e_{0,6,7}, e_7, 0, 0, 0, 0)$[2], where the user key difference $K_A \oplus K_B = K_C \oplus K_D = (\Delta \mathrm{MK}_{15}, \cdots, \Delta \mathrm{MK}_1, \Delta \mathrm{MK}_0)$ is $(0, \cdots, 0, e_7, 0, 0)$. The second related-key

[2] $(e_{1,3,5}, e_{0,1,6}, e_7, 0, 0, 0, 0, 0) \xrightarrow{(0,0,0,0)} (e_{0,1,6}, e_7, 0, 0, 0, 0, 0, 0) \xrightarrow{(0,0,0,0)} (e_7, 0, 0, 0, 0, 0, 0, 0) \xrightarrow{(e_7,0,0,0)} (0, 0, 0, 0, 0, 0, 0, 0) \xrightarrow{(0,0,0,0)} \cdots \xrightarrow{(0,0,0,0)} (0, 0, 0, 0, 0, 0, 0, 0) \xrightarrow{(0,0,0,e_7)} (0, 0, 0, 0, 0, 0, e_7, 0, 0) \xrightarrow{(0,0,0,0)} (0, 0, 0, e_{0,6,7}, e_7, 0, 0, 0) \xrightarrow{(0,0,0,0)} (0, e_{1,5,6}, e_{0,6,7}, e_7, 0, 0, 0, 0)$.

differential is the following 8-round related-key differential $\gamma \to \delta$ with probability 2^{-9} for E^1: $(0,0,0,0,e_{2,5,6},e_{0,6,7},e_7,0) \to (e_7,0,0,0,0,0,0,e_{0,1,6})$[3], where the user key difference $K_A \oplus K_C = K_B \oplus K_D = (0,e_7,0,\cdots,0)$.

We can compute a square sum of at least $6\cdot(2^{-12})^2+20\cdot(2^{-13})^2+20\cdot(2^{-14})^2+72\cdot(2^{-15})^2 \approx 2^{-19.98}$ for the probabilities of all the possible 10-round related-key differentials $\alpha \to \beta'$ for E^0, as there are at least 6 (10-round related-key differential characteristics) with probability 2^{-12}, at least 20 with probability 2^{-13}, at least 20 with probability 2^{-14}, and at least 72 with probability 2^{-15}. We can also compute a square sum of at least $5\cdot(2^{-9})^2+18\cdot(2^{-10})^2+40\cdot(2^{-11})^2 \approx 2^{-14.42}$ for the probabilities of all the possible 8-round related-key differentials $\gamma' \to \delta$ for E^1, as there are at least 5 (8-round related-key differential characteristics) with probability 2^{-9}, at least 18 with probability 2^{-10}, and at least 40 with probability 2^{-11}.

Therefore, we can learn that this 18-round related-key rectangle distinguisher has a probability of at least $2^{-19.98}\cdot 2^{-14.42}\cdot 2^{-64} = 2^{-98.4}$ for the correct key, while it has a probability of 2^{-128} for a wrong key. We can further improve it by counting many possible 8-round related-key differentials $\gamma' \to \delta'$ for every $\gamma' \to \delta$ for E^1. We count those that only have the output difference $(e_7,0,0,0,0,0,0,\Delta X_{21,0})$ different from the 8-round differential $\gamma' \to \delta$; an analysis of this one-round differentials reveals that there are 4 possible $\Delta X_{21,0}$ (i.e., $e_{0,1,6}, e_{0,6}, e_{0,6,7}, e_{0,1,6,7}$) with probability 2^{-3}, 4 possible $\Delta X_{21,0}$ with probability 2^{-4}, 4 possible $\Delta X_{21,0}$ with probability 2^{-5}, 4 possible $\Delta X_{21,0}$ with probability 2^{-6}, and 8 possible $\Delta X_{21,0}$ with probability 2^{-7}. Actually, these are all the 24 possible output differences of the last one-round differentials; we denote them by the set $\mathcal{S}$. As a result, the distinguisher now has a probability of at least $2^{-19.98}\cdot(4\cdot 2^{-7.21}+4\cdot 2^{-8.21}+4\cdot 2^{-9.21}+4\cdot 2^{-10.21}+8\cdot 2^{-11.21})^2\cdot 2^{-64} = 2^{-92.4}$ for the correct key, while it has a probability of $(24\cdot 2^{-64})^2 \approx 2^{-118.83}$ for a wrong key. Similar related-key rectangle distinguishers exist for some other series of 18 rounds.

Attacking Rounds 1–26. To get the difference $(e_{1,3,5},e_{0,1,6},e_7,0,0,0,0,0)$ just before Round 3, the input difference to Round 1 must have the form $(?,e_{0,\sim},?,e_{0,\sim},e_7,0,0,0)$, with 31 bits definitely being zero differences. On the other hand, the output difference $(e_7,0,0,0,0,0,0,x)$ of this distinguisher will propagate to a difference $(0,0,0,0,0,?,x,e_7)$ just after Round 21, where $x \in \mathcal{S}$, which will then propagate to a difference $(0,0,0,?,?,e_{0,\sim},e_7,0)$ just after Round 22, to a difference $(0,?,?,?,e_{0,\sim},e_7,0,e_7)$ just after Round 23 (due to the subkey difference in Round 23), and a difference $(?,?,?,e_{\overline{0},\sim},e_7,e_{2,\sim},e_7)$ just after Round 24. This property allows us to use the early abort technique [16] to break Rounds 21 and 24; the main idea of the early abort technique is to partially determine whether or not a candidate quartet in a rectangle attack is valid earlier than usual; if not, we can discard it immediately, which results in less

[3] $(0,0,0,0,e_{2,5,6},e_{0,6,7},e_7,0) \xrightarrow{(0,0,0,0)} (0,0,0,0,e_{0,6,7},e_7,0,0) \xrightarrow{(0,0,0,0)} (0,0,0,0,e_7,0,0,0) \xrightarrow{(0,0,e_7,0)} (0,0,0,0,0,0,0,0) \xrightarrow{(0,0,0,0)} \cdots \xrightarrow{(0,0,0,0)} (0,0,0,0,0,0,0,0) \xrightarrow{(0,e_7,0,0)} (0,e_7,0,0,0,0,0,0) \xrightarrow{(0,0,0,0)} (e_7,0,0,0,0,0,0,e_{0,1,6})$.

computations in the subsequent steps and may allow us to break more rounds by guessing the subkeys involved, depending on how many candidates are remaining.

The above analysis enables us to give a related-key rectangle attack on the first 26 rounds of HIGHT with the final transformation only, after noting that the same 64 user key bits are used in Rounds 1, 2, 25 and 26 as well as the final transformation. With a success probability of 80%, the attack requires $2^{49.7}$ chosen plaintexts, and has a time complexity of $2^{121.37}$ 26-round HIGHT computations. See the Appendix A for the detailed attack procedure.

4.2 Related-Key Impossible Differential Attack on 28-Round HIGHT

19-Round Related-Key Impossible Differentials. We exploit certain 19-round related-key impossible differentials: $(e_7, 0, 0, 0, 0, 0, 0, 0) \nrightarrow (0, 0, 0, 0, 0, 0, 0, 0, e_{0,\sim})$, where the user key difference $(\Delta \mathrm{MK}_{15}, \cdots, \Delta \mathrm{MK}_1, \Delta \mathrm{MK}_0)$ is $(0, e_7, 0, \cdots, 0)$, which start from Round 8 and end at Round 26. They are also built in a miss-in-the-middle manner: a 12-round related-key differential with probability 1 is concatenated with a 7-round related-key differential with probability 1, where the second right byte of the output difference of the 12-round related-key differential is $e_{\overline{0,\sim}}$, and the second right byte of the difference of the 7-round related-key differential is $e_{0,\sim}$, which contradict with each other.

Attacking Rounds 3–30. Similar to that given in Section 3.2, the 19-round related-key impossible differentials can be used to break the 28 rounds from Rounds 3 to 30 of HIGHT with only the final transformation; the main difference between them lies in that here we compute the related-key difference between a pair of data. The attack procedure is as follows.

1. Choose 2^{19} structures of 2^{40} plaintexts, where the two bytes (0,1) and the least significant seven bits of the third bytes and the least significant bits of the fourth bytes of the 2^{40} plaintexts in a structure are fixed to certain values, and the other 40 bit positions take all the possible values. A structure proposes 2^{79} plaintext pairs $(P^i, \widetilde{P}^j)$ with difference $(?, ?, ?, ?, e_{\overline{0,\sim}}, e_7, 0, 0)$, thus the 2^{19} structures propose a total of 2^{98} plaintext pairs with difference $(?, ?, ?, ?, e_{\overline{0,\sim}}, e_7, 0, 0)$.
2. In a chosen-plaintext attack scenario, obtain all the ciphertexts of the plaintexts P^i encrypted with K_A; we denote them by C^i, respectively; obtain all the ciphertexts of the plaintexts $\widetilde{P}^j$ encrypted with K_B; we denote them by $\widetilde{C}^j$, respectively, where $K_A \oplus K_B = (0, e_7, 0, \cdots, 0)$. Choose only the ciphertext pairs $(C^i, \widetilde{C}^j)$ with difference $(?, ?, ?, ?, e_{0,\sim}, 0, 0, 0)$.
3. Guess the two key bytes $(\mathrm{MK}_0, \mathrm{MK}_3)$, compute the subkeys $(\mathrm{WK}_7, \mathrm{SK}_{119})$ in the final transformation and Round 30, and do the following.
   (a) Partially decrypt every remaining ciphertext pair $(C^i, \widetilde{C}^j)$ with $(\mathrm{WK}_7, \mathrm{SK}_{119})$ to get the two bytes (7,6) of their intermediate values just before Round 30. Check if they have a difference $(0, ?)$. Keep only the qualified pairs.

(b) Guess the two key bytes $(\mathrm{MK}_2, \mathrm{MK}_7)$, compute the subkeys $(\mathrm{WK}_6, \mathrm{SK}_{118})$ in the final transformation and Round 30, and compute the subkey SK_{114} in Round 29 with MK_3 guessed above. Partially decrypt $(C^i, \widetilde{C}^j)$ with $(\mathrm{WK}_4, \mathrm{SK}_{114}, \mathrm{SK}_{118})$ to get the two bytes (5,4) of their intermediate values just before Round 29. Check if they have a difference $(0, ?)$. Keep only the qualified pairs.

(c) Guess the three key bytes $(\mathrm{MK}_1, \mathrm{MK}_6, \mathrm{MK}_{15})$, compute the subkeys $(\mathrm{WK}_5, \mathrm{SK}_{95}, \mathrm{SK}_{117})$ in the final transformation and Rounds 27 and 30, and compute the subkey SK_{113} in Round 29 with MK_2 guessed above. Partially decrypt $(C^i, \widetilde{C}^j)$ with $(\mathrm{WK}_5, \mathrm{SK}_{95}, \mathrm{SK}_{113}, \mathrm{SK}_{117})$ to get the two bytes (3,2) of their intermediate values just before Round 28. Check if they have a difference $(0, ?)$. Keep only the qualified pairs.

(d) Guess the two key bytes $(\mathrm{MK}_5, \mathrm{MK}_{14})$, compute the subkeys $(\mathrm{SK}_{108}, \mathrm{SK}_{116})$ in Rounds 28 and 30, and compute the subkeys $(\mathrm{WK}_4, \mathrm{SK}_{112})$ in the final transformation and Round 29 with $(\mathrm{MK}_0, \mathrm{MK}_1)$ guessed above. For $l = 0$ to 7, do as follows.
- Guess the l-th bit $\mathrm{MK}_{10,l}$ of the key byte MK_{10}, and compute the $(l+1)$ bits $\mathrm{SK}_{104,0-l}$ of the subkey SK_{104} in Round 27.
- For every remaining $(C^i, \widetilde{C}^j)$, Partially decrypt C^i with $(\mathrm{WK}_4, \mathrm{SK}_{116}, \mathrm{SK}_{112}, \mathrm{SK}_{108}, \mathrm{SK}_{104,0-l})$ to get its intermediate value $X^i_{26,1,0-l}$ just before Round 27, and partially decrypt $\widetilde{C}^j$ with $(\mathrm{WK}_4, \mathrm{SK}_{116}, \mathrm{SK}_{112}, \mathrm{SK}_{108} \oplus e_7, \mathrm{SK}_{104,0-l})$ to get its intermediate value $\widetilde{X}^j_{26,1,0-l}$ just before Round 27. Keep only the pairs such that $X^i_{26,1,0-l} = \widetilde{X}^j_{26,1,0-l}$.

4. For all the plaintext pairs $(P^i, \widetilde{P}^j)$ corresponding to remaining ciphertext pairs $(C^i, \widetilde{C}^j)$, do the following.

(a) For $l = 0$ to 7, do as follows.
- Guess the l-th bit $\mathrm{MK}_{11,l}$ of the key byte MK_{11}, and compute the $(l+1)$ bits $\mathrm{SK}_{11,0-l}$ of the subkey SK_{11} in Round 3.
- Partially decrypt every remaining $(P^i, \widetilde{P}^j)$ with $\mathrm{SK}_{11,0-l}$ to get their intermediate values $(X^i_{3,7}, X^i_{3,0,0-l})$ and $(\widetilde{X}^j_{3,7}, \widetilde{X}^j_{3,0,0-l})$ just after Round 3. Keep the pairs such that $X^i_{3,0,0-l} = \widetilde{X}^j_{3,0,0-l}$.

(b) Compute the subkeys $(\mathrm{SK}_{10}, \mathrm{SK}_{15})$ in Rounds 3 and 4 with $(\mathrm{MK}_{10}, \mathrm{MK}_{15})$ guessed in Step 3. Partially decrypt $(P^i, \widetilde{P}^j)$ with $(\mathrm{SK}_{10}, \mathrm{SK}_{15})$ to get the two bytes (7,0) of their intermediate values just after Round 4. Check if they have a difference $(?, 0)$. Keep only the qualified pairs.

(c) Guess the key byte MK_9, compute the subkey SK_9 in Round 3, and compute the subkeys $(\mathrm{SK}_{14}, \mathrm{SK}_{19})$ in Rounds 4 and 5 with $(\mathrm{MK}_2, \mathrm{MK}_{14})$ guessed in Step 3. Partially decrypt P^i with $(\mathrm{SK}_9, \mathrm{SK}_{14}, \mathrm{SK}_{19})$ to get the two bytes (7,0) of its intermediate value just after Round 5, and partially decrypt $\widetilde{P}^j$ with $(\mathrm{SK}_9, \mathrm{SK}_{14} \oplus e_7, \mathrm{SK}_{19})$ to get the two bytes (7,0) of its intermediate value just after Round 5. Check if they have a difference $(?, 0)$. Keep only the qualified pairs.

(d) Guess the two key bytes $(\mathrm{MK}_8, \mathrm{MK}_{13})$, compute the subkeys $(\mathrm{SK}_8, \mathrm{SK}_{13})$ in Rounds 3 and 4, and compute the subkeys $(\mathrm{SK}_{18}, \mathrm{SK}_{29})$ in Rounds

5 and 6 with $(\mathrm{MK}_1, \mathrm{MK}_6)$ guessed in Step 3. Partially decrypt $(P^i, \widetilde{P}^j)$ with $(\mathrm{SK}_8, \mathrm{SK}_{13}, \mathrm{SK}_{18}, \mathrm{SK}_{29})$ to get the two bytes (7,0) of their intermediate values just after Round 6. Check if they have a difference $(e_{0,\sim}, 0)$. Keep only the qualified pairs.

(e) Guess the key byte MK_{12}, compute the subkey SK_{12} in Round 4, and compute the subkeys $(\mathrm{SK}_{17}, \mathrm{SK}_{22}, \mathrm{SK}_{27})$ in Rounds 5, 6 and 7 with $(\mathrm{MK}_0, \mathrm{MK}_5, \mathrm{MK}_{10})$ guessed in Step 3. For every remaining $(P^i, \widetilde{P}^j)$, partially decrypt the two bytes (1,0) of their intermediate values just after Round 3 with $(\mathrm{SK}_{12}, \mathrm{SK}_{17}, \mathrm{SK}_{22}, \mathrm{SK}_{27})$ to get the two bytes (7,0) of their intermediate values just after Round 7. Check if they have a difference $(e_7, 0)$. If none of the plaintext pairs satisfies this test, then record the guessed 120 key bits $(\mathrm{MK}_0, \cdots, \mathrm{MK}_3, \mathrm{MK}_5, \cdots, \mathrm{MK}_{15})$, and execute Step 5; otherwise, discard this guess and try another.

5. For a recorded $(\mathrm{MK}_0, \cdots, \mathrm{MK}_3, \mathrm{MK}_5, \cdots, \mathrm{MK}_{15})$, exhaustively search for the remaining 8 key bits with three known pairs of plaintexts and ciphertexts. If a 128-bit key is suggested, output it as the user key of the 28-round HIGHT; otherwise, go to Step 3.

There is a 25-bit filtering condition on the ciphertext pairs in Step 2, and a 8-bit filtering condition in each of Steps 3-(a)∼(d) and Steps 4-(a)∼(e). Hence, for every key guess, it is expected about $2^{98} \cdot 2^{-25-8\times 8} = 2^9$ plaintext pairs remain after Step 4-(d), and about $2^{120} \cdot (1-2^{-8})^{2^9} \approx 2^{120} \cdot e^{-2} \approx 2^{117.12}$ guesses of the 120 key bits are recorded in Step 4-(e). Thus, the expected number of suggested wrong keys in Step 5 is about $2^{-192} \cdot 2^{117.12+8} = 2^{-66.88}$. Thus, the correct key can be determined.

Step 3 has about $2 \cdot 2^{73} \cdot 2^{16} \cdot \frac{1}{4} \cdot \frac{1}{28} + 2 \cdot 2^{65} \cdot 2^{32} \cdot \frac{1}{4} \cdot \frac{2}{28} + 2 \cdot 2^{57} \cdot 2^{56} \cdot \frac{1}{4} \cdot \frac{3}{28} + \sum_{l=0}^{7} (2 \cdot 2^{49-l} \cdot 2^{72+l+1} \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{4}{28}) \approx 2^{120.19}$ computations, where $\frac{1}{2}$ means the average fraction of the guessed keys that are tested in a step. Step 4 has about $\sum_{l=0}^{7} (2 \cdot 2^{41-l} \cdot 2^{80+l+1} \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{1}{28}) + 2 \cdot 2^{33} \cdot 2^{88} \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{2}{28} + 2 \cdot 2^{25} \cdot 2^{96} \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{3}{28} + 2 \cdot 2^{17} \cdot 2^{112} \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{4}{28} + 2 \cdot 2^{120} \cdot [1 + (1-2^{-8}) + \cdots + (1-2^{-8})^{2^9}] \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{4}{28} \approx 2^{124.79}$ computations. Step 5 has about $2^{125.12}$ computations. Therefore, the attack has a total time complexity of about $2^{125.99}$ 28-round HIGHT computations.

5 Conclusions

The HIGHT block cipher was proposed for low-resource devices at CHES '06. In this paper, we present an impossible differential attack on 25-round HIGHT, a related-key rectangle attack on 26-round HIGHT and a related-key impossible differential attack on 28-round HIGHT. Like most cryptanalytic attacks on block ciphers, the presented attacks are theoretical, but they suggest that the reduced versions of HIGHT are less secure than they should be. These are better than any previously known cryptanalytic results on HIGHT in terms of the numbers of attacked rounds.

Acknowledgments

The author is very grateful to his supervisor Prof. Chris Mitchell for his editorial comments.

References

1. Biham, E.: New types of cryptanalytic attacks using related keys. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 398–409. Springer, Heidelberg (1994)
2. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 12–23. Springer, Heidelberg (1999)
3. Biham, E., Biryukov, A., Shamir, A.: Miss in the middle attacks on IDEA and Khufu. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 124–138. Springer, Heidelberg (1999)
4. Biham, E., Dunkelman, O., Keller, N.: The rectangle attack — rectangling the Serpent. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 340–357. Springer, Heidelberg (2001)
5. Biham, E., Dunkelman, O., Keller, N.: Related-key boomerang and rectangle attacks. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 507–525. Springer, Heidelberg (2005)
6. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1991)
7. Feldhofer, M., Dominikus, S., Wolkerstorfer, J.: Strong authentication for RFID systems using the AES algorithm. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 357–370. Springer, Heidelberg (2004)
8. Feldhofer, M., Wolkerstorfer, J., Rijmen, V.: AES implementation on a grain of sand. IEE Proceedings on Information Security 152(1), 13–20 (2005)
9. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B.-S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: HIGHT: a new block cipher suitable for low-resource device. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)
10. Hong, S., Kim, J., Lee, S., Preneel, B.: Related-key rectangle attacks on reduced versions of SHACAL-1 and AES-192. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 368–383. Springer, Heidelberg (2005)
11. Kelsey, J., Kohno, T., Schneier, B.: Amplified boomerang attacks against reduced-round MARS and Serpent. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 75–93. Springer, Heidelberg (2001)
12. Kelsey, J., Schneier, B., Wagner, D.: Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 237–251. Springer, Heidelberg (1996)
13. Kim, J., Kim, G., Hong, S., Lee, S., Hong, D.: The related-key rectangle attack — application to SHACAL-1. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 123–136. Springer, Heidelberg (2004)
14. Knudsen, L.R.: Trucated and higher order differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995)

15. Knudsen, L.R.: DEAL— a 128-bit block cipher, Technical report, Department of Informatics, University of Bergen, Norway (1998)
16. Lu, J., Kim, J., Keller, N., Dunkelman, O.: Related-key rectangle attack on 42-round SHACAL-2. In: Katsikas, S.K., Lopez, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC 2006. LNCS, vol. 4176, pp. 85–100. Springer, Heidelberg (2006)
17. Lucks, S.: The saturation attack — a bait for Twofish. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 1–15. Springer, Heidelberg (2002)
18. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
19. National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES), FIPS-197 (2001)
20. Wagner, D.: The boomerang attack. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 156–170. Springer, Heidelberg (1999)

A Procedure of the Related-Key Rectangle Attack on 26-Round HIGHT

1. Choose $2^{14.7}$ structures S_i of 2^{33} plaintexts $P_{i,l}$ each, $i = 1, 2, \cdots, 2^{14.7}$, $l = 1, 2, \cdots, 2^{33}$, where in each structure the 31 bit positions (0–31) of $P_{i,l}$ are fixed, and the remaining 33 bit positions take all the possible values. In a chosen plaintext attack scenario, obtain the ciphertexts $C_{i,l}$, $C^*_{i,l}$, $C'_{i,l}$ and $C'^*_{i,l}$ of $P_{i,l}$ encrypted with K_A, K_B, K_C and K_D, respectively, where $K_A \oplus K_B = K_C \oplus K_D = (0, \cdots, 0, e_7, 0, 0)$ and $K_A \oplus K_C = K_B \oplus K_D = (0, e_7, 0, \cdots, 0)$.
2. Guess the 8 key bytes $(\mathrm{MK}_0, \cdots, \mathrm{MK}_7)$, compute the subkeys $(\mathrm{SK}_0, \cdots, \mathrm{SK}_7)$ in Rounds 1 and 2, and do as follows.
   (a) Partially encrypt each plaintext $P_{i,l}$ through Rounds 1 and 2 with $(\mathrm{SK}_0, \cdots, \mathrm{SK}_7)$ to get its intermediate value $x_{i,l}$ just after Round 2. Then, partially decrypt $x_{i,l} \oplus (e_{1,3,5}, e_{0,1,6}, e_7, 0, 0, 0, 0, 0)$ through Rounds 1 and 2 with $(\mathrm{SK}_0, \mathrm{SK}_1, \mathrm{SK}_2 \oplus e_7, \mathrm{SK}_3, \cdots, \mathrm{SK}_7)$ to get its plaintext, denoted by $\widetilde{P}_{i,l}$. Find $\widetilde{P}_{i,l}$ in S_i. We denote by $\widetilde{C}_{i,l}, \widetilde{C}^*_{i,l}$, $\widetilde{C}'_{i,l}$ and $\widetilde{C}'^*_{i,l}$ the corresponding ciphertexts for $\widetilde{P}_{i,l}$ encrypted under K_A, K_B, K_C and K_D, respectively. This process generates $2^{14.7} \cdot 2^{33} = 2^{47.7}$ plaintext pairs for every key guess, which can produce the difference $(e_{1,3,5}, e_{0,1,6}, e_7, 0, 0, 0, 0, 0)$ just before Round 3.
   (b) Compute the subkeys $(\mathrm{SK}_{96}, \cdots, \mathrm{SK}_{99})$, $(\mathrm{SK}_{100}, \cdots, \mathrm{SK}_{103})$ and $(\mathrm{WK}_0, \cdots, \mathrm{WK}_3)$ with $(\mathrm{MK}_0, \cdots, \mathrm{MK}_7)$. Then, partially decrypt all the $C_{i,l}$ and $C'_{i,l}$ with these subkeys to get their respective intermediate values $T_{i,l}$ and $T'_{i,l}$ just before Round 25, and partially decrypt all the $\widetilde{C}^*_{i,l}$ and $\widetilde{C}^{*'}_{i,l}$ with the related subkeys $(\mathrm{SK}_{96} \oplus e_7, \mathrm{SK}_{97}, \mathrm{SK}_{98}, \mathrm{SK}_{99})$, $(\mathrm{SK}_{100}, \cdots, \mathrm{SK}_{103})$ and $(\mathrm{WK}_0, \mathrm{WK}_1, \mathrm{WK}_2 \oplus e_7, \mathrm{WK}_3)$ to get their respective intermediate values $\widetilde{T}^*_{i,l}$ and $\widetilde{T}^{*'}_{i,l}$ just before Round 25. Store $(T_{i,l}, T'_{i,l}, \widetilde{T}^*_{i,l}, \widetilde{T}^{*'}_{i,l})$ in a hash table. Finally, check if both $T_{i_1,l_1} \oplus T'_{i_2,l_2}$ and $\widetilde{T}^*_{i_1,l_1} \oplus \widetilde{T}'^*_{i_2,l_2}$ have the form $(?, ?, ?, e_{\overline{0},\sim}, e_7, e_{2,\sim}, e_7, ?)$, for $1 \le i_1 \le i_2 \le 2^{14.7}$ and $1 \le l_1, l_2 \le 2^{33}$. If 6 or more quartets $(T_{i_1,l_1}, \widetilde{T}^*_{i_1,l_1}, T'_{i_2,l_2}, \widetilde{T}^{*'}_{i_2,l_2})$ pass this test, record them, and go to Step 3; otherwise, repeat Step 2 with another guess.

3. For $l = 0$ to 7:
   (a) Guess the l-th bit $\mathrm{MK}_{10,l}$ of the key byte MK_{10}, and compute the $(l+1)$ bits $\mathrm{SK}_{95,0-l}$ of the subkey SK_{95} in Round 24.
   (b) Partially decrypt the two bytes (0,7) of every remaining $(T_{i_1,l_1}, \widetilde{T}^*_{i_1,l_1}, T'_{i_2,l_2}, \widetilde{T}^{*'}_{i_2,l_2})$ with $\mathrm{SK}_{95,0-l}$ to get the least significant $(l+1)$ bits of the bytes (7) of their intermediate values just before Round 24, and check if the intermeidate $(l+1)$ bits of T_{i_1,l_1} and T'_{i_2,l_2} have a zero difference, and the intermeidate $(l+1)$ bits of $\widetilde{T}^*_{i_1,l_1}$ and $\widetilde{T}^{*'}_{i_2,l_2}$ also have a zero difference. If 6 or more quartets pass this test, record them; otherwise, repeat Step 3-(a) with another guess.
4. Guess the key byte MK_9, and compute the subkey SK_{94} in Round 24; for $l = 0$ to 7, do as follows.
   (a) Guess the l-th bit $\mathrm{MK}_{13,l}$ of MK_{13}, and compute the $(l+1)$ bits $\mathrm{SK}_{90,0-l}$ of the subkey SK_{90} in Round 23.
   (b) Partially decrypt the two bytes (5,6) of every remaining $(T_{i_1,l_1}, \widetilde{T}^*_{i_1,l_1}, T'_{i_2,l_2}, \widetilde{T}^{*'}_{i_2,l_2})$ with $(\mathrm{SK}_{94}, \mathrm{SK}_{90,0-l})$ to get the least significant $(l+1)$ bits of the bytes (5) of their intermediate values just before Round 23, and check if the intermeidate $(l+1)$ bits of T_{i_1,l_1} and T'_{i_2,l_2} have a zero difference, and the intermeidate $(l+1)$ bits of $\widetilde{T}^*_{i_1,l_1}$ and $\widetilde{T}^{*'}_{i_2,l_2}$ have a zero difference as well. If 6 or more quartets pass this test, record them; otherwise, repeat Step 4-(a) with another guess (if all the guesses of $\mathrm{MK}_{13,l}$ are tested, repeat Step 4 with another guess of MK_9).
5. Guess the least significant 3 bits $\mathrm{MK}_{15,0-2}$ of the key byte MK_{15}; for $l = 3$ to 7, do as follows.
   (a) Guess the l-th bit $\mathrm{MK}_{15,l}$ of MK_{15}, and compute the $(l+1)$ bits $\mathrm{SK}_{92,0-l}$ of the subkey SK_{92} in Round 24.
   (b) Partially decrypt the two bytes (1,2) of every remaining $(T_{i_1,l_1}, \widetilde{T}^*_{i_1,l_1}, T'_{i_2,l_2}, \widetilde{T}^{*'}_{i_2,l_2})$ with $\mathrm{SK}_{92,0-l}$ to get the least significant $(l+1)$ bits of the bytes (1) of their intermediate values just before Round 24, and check if the intermeidate $(l+1)$ bits of T_{i_1,l_1} and T'_{i_2,l_2} have a zero difference, and the intermeidate $(l+1)$ bits of $\widetilde{T}^*_{i_1,l_1}$ and $\widetilde{T}^{*'}_{i_2,l_2}$ have a zero difference as well. If 6 or more quartets pass this test, record them; otherwise, repeat Step 5-(a) with another guess.
6. Guess the key bytes $(\mathrm{MK}_8, \mathrm{MK}_{12})$, compute the subkeys $(\mathrm{SK}_{93}, \mathrm{SK}_{89})$, and compute the subkey SK_{85} with MK_0 guessed in Step 2. Partially decrypt the two bytes (3,4) of every remaining $(T_{i_1,l_1}, \widetilde{T}^*_{i_1,l_1}, T'_{i_2,l_2}, \widetilde{T}^{*'}_{i_2,l_2})$ with $(\mathrm{SK}_{93}, \mathrm{SK}_{89}, \mathrm{SK}_{85})$ to get the two bytes (3,2) of their intermediate values just before Round 22, and check if the intermeidate values of T_{i_1,l_1} and T'_{i_2,l_2} have a difference $(0, ?)$, and the intermeidate values of $\widetilde{T}^*_{i_1,l_1}$ and $\widetilde{T}^{*'}_{i_2,l_2}$ have a difference $(0, ?)$ as well. If 6 or more quartets pass this test, execute Step 7 with them; otherwise, repeat this step with another guess. Now, for every remaining $(T_{i_1,l_1}, \widetilde{T}^*_{i_1,l_1}, T'_{i_2,l_2}, \widetilde{T}^{*'}_{i_2,l_2})$, we obtain their intermediate values just before Round 24; we denote them by $(Q_{i_1,l_1}, \widetilde{Q}^*_{i_1,l_1}, Q'_{i_2,l_2}, \widetilde{Q}^{*'}_{i_2,l_2})$, respectively.

7. Guess the key byte MK_{11}, compute the subkey SK_{88} in Round 23, and compute the subkey SK_{84} with MK_7 guessed in Step 2. Partially decrypt the two bytes (1,2) of $(Q_{i_1,l_1}, \widetilde{Q}^*_{i_1,l_1}, Q'_{i_2,l_2}, \widetilde{Q}^{*'}_{i_2,l_2})$ with $(\mathrm{SK}_{88}, \mathrm{SK}_{84})$ to get the two bytes (1,0) of their intermediate values just before Round 22, and check if the intermediate values of $Q_{i_1,l_1}, Q'_{i_2,l_2}$ have a difference belonging to the set $\{(x, e_7)|x \in \mathcal{S}\}$, and the intermediate values of $\widetilde{Q}^*_{i_1,l_1}$ and $\widetilde{Q}^{*'}_{i_2,l_2}$ also have a difference belonging to the set $\{(x, e_7)|x \in \mathcal{S}\}$. If 6 or more quartets $(Q_{i_1,l_1}, \widetilde{Q}^*_{i_1,l_1}, Q'_{i_2,l_2}, \widetilde{Q}^{*'}_{i_2,l_2})$ pass this test, execute Step 8 with them; otherwise, repeat this step with another guess of MK_{11}.
8. Compute the subkey SK_{80} with MK_3 guessed in Step 2. For every remaining $(Q_{i_1,l_1}, \widetilde{Q}^*_{i_1,l_1}, Q'_{i_2,l_2}, \widetilde{Q}^{*'}_{i_2,l_2})$, since we already obtain the two bytes (1,2) of their intermediate values just before Round 22, we can partially decrypt them with SK_{80} to check if the bytes (1) of the intermediate values just before Round 21 of $(Q_{i_1,l_1}, Q'_{i_2,l_2})$ have a zero difference, and the bytes (1) of the intermediate values just before Round 21 of $(\widetilde{Q}^*_{i_1,l_1}, \widetilde{Q}^{*'}_{i_2,l_2})$ have a zero difference as well. If 6 or more $(Q_{i_1,l_1}, \widetilde{Q}^*_{i_1,l_1}, Q'_{i_2,l_2}, \widetilde{Q}^{*'}_{i_2,l_2})$ pass this test, record the guessed 120 key bits $(\mathrm{MK}_0, \cdots, \mathrm{MK}_{13}, \mathrm{MK}_{15})$, and go to Step 9; otherwise, repeat Step 7 with another guess of MK_{11}.
9. For a recorded $(\mathrm{MK}_0, \cdots, \mathrm{MK}_{13}, \mathrm{MK}_{15})$, exhaustively search for the remaining 8 key bits with a known plaintext/ciphertext pair. If a 128-bit key is suggested, output it as the user key of the 26-round HIGHT; otherwise, go to Step 2 (If all the guesses are tested during any of Steps 3–8, repeat its previous steps with other guesses).

The related-key rectangle distinguisher involves four different keys, thus about $2^{47.7\times 2} = 2^{95.4}$ candidate quartets are constructed for every guess in Step 2. To produce the output difference δ', the two pairs in a right quartet must have differences $(?, ?, ?, e_{\overline{0},\sim}, e_7, e_{2,\sim}, e_7, ?)$ just before Round 25, so a candidate quartet that does not meet this filtering condition is an incorrect quartet. Therefore, it is expected that almost all the 2^{64} guesses of $(\mathrm{MK}_0, \cdots, \mathrm{MK}_7)$ will pass Step 2-(b), and for every guess about $2^{95.4} \cdot 2^{-20\times 2} = 2^{55.4}$ candidate quartets remain after 2-(b).

For every iteration in Step 3-(b), the probability that a quartet meets the filtering condition is $(2^{-1})^2 = 2^{-2}$, so it follows that all the 2^{72} guesses of $(\mathrm{MK}_0, \cdots, \mathrm{MK}_7, \mathrm{MK}_{10})$ will past Step 3, and for a wrong guess it is expected about $2^{55.4} \cdot 2^{-2\times 8} = 2^{39.4}$ quartets remain after Step 3. For every iteration in Step 4-(b), the probability that a quartet meets the filtering condition is also 2^{-2}, so it is expected that all the 2^{88} guesses of $(\mathrm{MK}_0, \cdots, \mathrm{MK}_7, \mathrm{MK}_9, \mathrm{MK}_{10}, \mathrm{MK}_{13})$ will past this step, and for a wrong guess about $2^{39.4} \cdot 2^{-2\times 8} = 2^{23.4}$ quartets remain after Step 4. For every iteration in Step 5-(b), the probability that a quartet meets the filtering condition is 2^{-2}, so for a wrong guess about $2^{23.4} \cdot 2^{-2\times 5} = 2^{13.4}$ quartets remain after Step 5. In Step 6, the probability that a quartet meets the filtering condition is also $2^{-8\times 2} = 2^{-16}$, so for a wrong guess about $2^{13.4} \cdot 2^{-16} = 2^{-2.6}$ quartets remain after Step 6, and the probability that 6 or more quartets pass the test for a wrong guess is approximately

$\sum_{i=6}^{2^{13.4}} [\binom{2^{13.4}}{i} \cdot (2^{-16})^i \cdot (1-2^{-16})^{2^{13.4}-i}] \approx 2^{-25.09}$, thus it is expected that about $2^{112} \cdot 2^{-25.09} = 2^{86.91}$ guesses of $(\mathrm{MK}_0, \cdots, \mathrm{MK}_{10}, \mathrm{MK}_{12}, \mathrm{MK}_{13}, \mathrm{MK}_{15})$ pass Step 6. In Step 7, the probability that a quartet meets the filtering condition is $(\frac{24}{2^7})^2 = 2^{-4.83}$, and the probability that 6 or more quartets pass the test for a wrong guess is approximately $(2^{-4.83})^6 \approx 2^{-28.98}$, so it is expected about $2^{86.91+8} \cdot 2^{-28.98} = 2^{65.93}$ guesses of $(\mathrm{MK}_0, \cdots, \mathrm{MK}_{13}, \mathrm{MK}_{15})$ pass Step 7. In Step 8, the probability that 6 or more quartets pass the test for a wrong guess is approximately $(2^{-8\times 2})^6 = 2^{-96}$, thus it is expected about $2^{65.93} \cdot 2^{-96} = 2^{-30.07}$ guesses of $(\mathrm{MK}_0, \cdots, \mathrm{MK}_{13}, \mathrm{MK}_{15})$ pass Step 8. Therefore, it is expexcted that we can find the correct user key with 2^8 trials in Step 9.

The attack requires $2^{49.7}$ related-key chosen plaintexts. Step 2-(a) has about $2 \cdot 2^{47.7} \cdot 2^{64} \cdot \frac{1}{2} \cdot \frac{2}{26} \approx 2^{108}$ 26-round HIGHT computations, where $\frac{1}{2}$ means the average fraction of the guessed keys that are tested in the step. The time complexity of Step 2-(b) is dominated by the partial decryptions, which is about $4 \cdot 2^{47.7} \cdot 2^{64} \cdot \frac{1}{2} \cdot \frac{2}{26} \approx 2^{109}$ computations. Step 3 has about $\sum_{l=0}^{7}(4 \cdot 2^{55.4-2\cdot l} \cdot 2^{65+l} \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{1}{26}) \approx 2^{115.69}$ computations. Step 4 has about $\sum_{l=0}^{7}(4 \cdot 2^{39.4-2\cdot l} \cdot 2^{81+l} \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{2}{26}) \approx 2^{116.69}$ computations. Step 5 has about $\sum_{l=0}^{4}(4 \cdot 2^{23.4-2\cdot l} \cdot 2^{92+l} \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{1}{26}) \approx 2^{110.65}$ computations. Step 6 has about $4 \cdot 2^{13.4} \cdot 2^{112} \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{3}{26} \approx 2^{121.28}$ computations. Step 7 has about $4 \cdot 6 \cdot 2^{94.91} \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{2}{26} \approx 2^{92.79}$ computations. Step 8 has about $4 \cdot 6 \cdot 2^{65.93} \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{1}{26} \approx 2^{62.81}$ computations. Therefore, the attack has a total time complexity of about $2^{121.37}$ 26-round HIGHT computations.

In Step 8, it is expected that about $2^{95.4} \cdot 2^{-92.4} = 8$ quartets pass the filtering condition for the correct key, and the probability that 6 or more quartets pass the test for the correct key guess is approximately $\sum_{i=6}^{2^{95.4}} [\binom{2^{95.4}}{i} \cdot (2^{-92.4})^i \cdot (1-2^{-92.4})^{2^{95.4}-i}] \approx 0.8$, so this related-key rectangle attack can break the 26-round HIGHT with a success probability of 80%.

A Cryptanalysis of the Double-Round Quadratic Cryptosystem

Antoine Scemama

Johann Wolfgang Goethe University
Frankfurt am Main, Germany
scemama@math.uni-frankfurt.de

Abstract. In the 80's Matsumoto and Imai [8] proposed public key cryptosystems based on the difficulty of solving systems of polynomials in several variables. Although these first schemes were broken, many others followed, leading to a very active field known as Multivariate cryptography. In this paper, we show how to break one of these schemes, the Double-Round Quadratic cryptosystem from [12]. We stress that this cryptosystem has, in practice, already been cryptanalysed in [5]. However their attack uses several "non-standard" heuristics, they provide experimental evidence, but no proof is given, as opposed to this present article. Our attack uses a very general technique introduced in [9] to break the cryptosystem.

Keywords: Multivariable Cryptography, Double-Round Quadratic Cryptosystem, 2R Cryptosystem.

1 Introduction

The introduction of multivariate cryptography corresponded to the need for primitives based on new problems. Indeed, today's widely used (cryptographic) schemes are based on the factorisation or on the discrete logarithm problem, and we can not be sure (although it seems very unlikely) that no efficient algorithm to solve these problems exists. Thus, it is natural to look for alternatives.

Considering also that both factorisation and discrete logarithm problems are easily solved on quantum computers, even if operational ones won't exist for decades, one must be prepared.

Multivariate cryptography could be a credible alternative in the sense that the underlying problem (solving a system of polynomial equations over some finite field) is NP-hard, seems to stay hard "on average" and is believed to remain hard even with a quantum computer.

An additional advantage is that it allows the building of efficient schemes that are particularly well suited to low-cost devices such as smartcard.Unfortunately, nowadays, most of the multivariate encryption schemes have been broken. However, several modifications have enabled the creation of a large variety of very efficient signature algorithms. Even if these schemes are quite recent, in 2003 a scheme from Patarin et al. [3] was considered strong enough to be selected by the NESSIE project [1] for standardization. This last scheme was recently broken in [6].

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 27–36, 2007.

After his cryptanalysis of the Imai-Matsumoto cryptosystem [8], Patarin et al. proposed in a series of articles ([12] [7]), a collection of new multivariate schemes which the author called $2R$ (for "2 Round"), one of which is also called the "Double-Round Quadratic cipher" in [10], and which is at the center of the present paper.

A cryptanalysis of a large class of the $2R$ cipher (particularly of the Double-Round Quadratic cipher) was published in 1999 in [5], followed a year later by another cryptanalysis [2] which, however does not apply to the Double-Round Quadratic cipher.

In this article we present another Cryptanalysis of the Double-Round Quadratic Cryptosystem that differs significantly from [5]. Based on the work from [9], we show that the scheme can be described differently (in another mathematical structure) and that in this other formalism, relinearization techniques enable the computing, in polynomial time, of the private keys from the public one. Our proof is also heuristic in the sense that we assume the linearized equations to be independent. This assumption is very general and has been used and verified many times in similar cases. The author feels that the formalism used here is preferable when describing Multivariate ciphers, because it allows us to present it in a unified (and cleaner) way.

Organisation of the paper
In the first section, we describe the Double-Round Quadratic cryptosystem. In the second, we introduce the results from [9] which establish an equivalence between a system of polynomials in n variables over a finite field $\mathbb{F}_q$ and an univariate polynomial in the degree n extension of $\mathbb{F}_q$. Then, we recall results based on so called re-linearization technique (mainly from [9]) in section 3. Finally we present the cryptanalysis in section 4, 5, 6 and 7.

2 Description of the Double-Round Quadratic Cryptosystem

2.1 Notation

$\mathbb{K} = \mathbb{F}_{q^n}$ denotes the field with q^n elements. Moreover, q must satisfy $q = 3 \mod 4$.

Let $(\beta_1, \cdots, \beta_n)$ be a basis of $\mathbb{K}$ seen as a $\mathbb{F}_q$-vector space.

For an element x of $\mathbb{K}$, we will use both vector and field representations to refer to x.

We note $\mathbf{x} = x_1\beta_1 + \cdots + x_n\beta_n \in \mathbb{K}$ (with $(x_1, \cdots, x_n) \in (\mathbb{F}_q)^n$) to refer to the field element. We also refer to x according to the vector notation : $\bar{x} = (x_1, \cdots, x_n)$.

2.2 Idea of the Cryptosystem

The public key consist of a system of n degree 4 polynomials (in n variables) $P_1(x_1, \cdots, x_n), \cdots, P_n(x_1, \cdots, x_n)$ defined over $\mathbb{F}_q$.

Given a message $\bar{m} = (m_1, \cdots, m_n) \in (\mathbb{F}_q)^n$, the corresponding ciphertext is the vector corresponding to the image of the polynomial system at the point $\bar{m}$. Hence the ciphertext is $\bar{c} = (P_1(m_1, \cdots, m_n), \cdots, P_n(m_1, \cdots, m_n))$.

To produce the public key (i.e. the system of public polynomials) one chooses 3 nonsingular matrices in $M_n(\mathbb{F}_q)$ which are kept secret. Given a vector $\bar{x} = (x_1, \cdots, x_n) \in (\mathbb{F}_q)^n$ one alternatively applies a private linear transformations (with the help of the private matrices) and a public non-linear (invertible) transformation. The result of this mix of (private) linear, and (public) non-linear transformations, gives rise to a system of n polynomials which form the public key.

Finally, to decrypt the vector $\bar{c}$, one has to apply the inverse linear transformations (with the private matrices) and the inverse non-linear transformation, in the reverse order.

The general idea here (and it is the case for all multivariate cryptosystems) is that if a malicious adversary wants to recover the plaintext from the ciphertext $(c_1, \cdots, c_n)$ he has to inverse a system of polynomials of the following form:

$$\begin{cases} P_1(x_1, \cdots, x_n) = c_1 \\ \vdots \\ P_n(x_1, \cdots, x_n) = c_n \end{cases}$$

The hope is that, even if he knows the machinery used to produce the system (except the private keys), he can not effectively inverse it other than by directly solving the public system of equations, which is in general a NP-hard problem (even when the polynomials are quadratic over the finite field $\mathbb{F}_2$).

2.3 How It Works

ff

Public	*Private*
q, n	
$P_1(x_1, \cdots, x_n)$	
$\vdots$	$A, B, C \in M_n(\mathbb{F}_q)$ nonsingular
$P_n(x_1, \cdots, x_n)$	

Production of the public key

Let $\bar{x} = (x_1, \cdots, x_n)$. First, one chooses the parameters n and q along with the (nonsingular) matrices A, B and C. To produce the public system of equations, one computes the following:

$$\bar{u} = A\bar{x} \Rightarrow \mathbf{v} = \mathbf{u}^2 \Rightarrow \bar{w} = B\bar{v} \Rightarrow \mathbf{z} = \mathbf{w}^2 \Rightarrow \bar{c} = C\bar{z} = \begin{pmatrix} P_1(x_1, \cdots, x_n) \\ \vdots \\ P_n(x_1, \cdots, x_n) \end{pmatrix}.$$

fi

Encryption
Given the message $\bar{m} = (m_1, \cdots, m_n)$, the ciphertext is $\bar{c} = (P_1(m_1, \cdots, m_n), \cdots, P_n(m_1, \cdots, m_n))$ in $(\mathbb{F}_q)^n$.

Decryption
To decrypt, one must be able to inverse all the transformations which led to the public key. The linear transformations are easy to inverse, given the matrix A, B and C. The squaring transformation is also easily "almost" invertible as follows:

$$(\mathbf{x}^2)^{\frac{q^n+1}{4}} = \mathbf{x}^{\frac{q^n-1}{2}}\mathbf{x} = \pm\mathbf{x} \text{ (remember that } q = 3 mod 4)$$

Hence, given the ciphertext $\bar{c}$, the decryption algorithm work as follows:

$$C^{-1}\bar{c}=\bar{z} \Rightarrow \mathbf{z}^{\frac{q+1}{4}} = \pm\mathbf{w} \Rightarrow B^{-1}(\pm\bar{w}) = \pm\bar{v} \Rightarrow (\pm\mathbf{v})^{\frac{q+1}{4}} = \pm\mathbf{u} \Rightarrow A^{-1}(\pm\bar{u}) = \pm\bar{x}$$

We can not exactly inverse the squaring transformation, but it is not really a problem. Hence, if $\bar{m}$ is the original message and $\bar{x} = (x_1, \cdots, x_n)$ is the decrypted one, we know that either $\bar{m} = (x_1, \cdots, x_n)$ or $\bar{m} = (q-x_1, \cdots, q-x_n)$. We can take, for instance, the convention that each message $\bar{m}$ to be encrypted must have the first non zero component m_i in the interval $[1, \frac{q-1}{2}]$, so that we know which of $\bar{x}$ and $-\bar{x}$ correspond to m.

The cryptosystem needs two rounds, i.e. two applications of the (non-linear) squaring transformation. Indeed, the one round version can be attacked, see [12] or the book [10].

3 The Kipnis-Shamir Formalism

In the description of the cryptosystem, we alternate between transformations on elements of $(\mathbb{F}_q)^n$ and transformations in $\mathbb{F}_{q^n}$. This different framework makes it difficult to analyse and to attack the cryptosystem. In [9], the authors showed that one could have a unified framework, with only transformations in $\mathbb{F}_{q^n}$. In section 5, we will show that in this framework, the Double-Round Quadratic cryptosystem is easily breakable.

The next two theorems are taken directly from [9]. We add the proof for completeness.

Theorem 1 (Kipnis, Shamir 99). *M n n $\mathbb{F}_q$ ffi $a_1, \cdots, a_n$ $\mathbb{F}_{q^n}$ n $\mathbb{F}_q$ $(x_1, \cdots, x_n)$ $\mathbf{x} = \sum_{i=1}^n x_i\beta_i$ $\mathbb{F}_{q^n}$ $(y_1, \cdots, y_n)$ $\mathbf{y} = \sum_{i=1}^n y_i\beta_i$ $\mathbb{F}_{q^n}$ $(y_1, \cdots, y_n) = M(x_1, \cdots, x_n)$ $\mathbf{y} = \sum_{i=1}^n a_i\mathbf{x}^{q^i}$*

Proof
It is well known that the application $\mathbf{F}$ with $\mathbf{F}(\mathbf{x}) = \mathbf{x}^{q^i}$ is a linear application in $\mathbb{F}_{q^n}$ (seen as a $\mathbb{F}_q$-vector space). Hence, every polynomial $P(\mathbf{x}) = \sum_{i=0}^{n-1} a_i\mathbf{x}^{q^i} \in \mathbb{F}_{q^n}[X]$ is also a linear application.

It follows that there exists a matrix M in $M_n(\mathbb{F}_q)$ such that for any two n-tuples over $\mathbb{F}_q$, $(x_1, \cdots, x_n)$ (which represents $\mathbf{x} = \sum_{i=1}^{n} x_i\beta_i$ in $\mathbb{F}_{q^n}$) and $(y_1, \cdots, y_n)$ (whith $\mathbf{y} = \sum_{i=1}^{n} y_i\beta_i$ in $\mathbb{F}_{q^n}$), $(y_1, \cdots, y_n) = M(x_1, \cdots, x_n)$ if $\mathbf{y} = \sum_{i=1}^{n} a_i\mathbf{x}^{q^i}$.

Moreover there are q^{n^2} different matrices in $M_n(\mathbb{F}_q)$ and also q^{n^2} different Polynomials of the form $P(\mathbf{x}) = \sum_{i=0}^{n-1} a_i\mathbf{x}^{q^i}$ in $\mathbb{F}_{q^n}[X]$. Finally, it is clear that two different polynomials can not lead to the same matrix, since otherwise it would mean that a non-zero polynomial is represented by a zero matrix.

Given a linear mapping M, it is clear that we can find the coefficients a_i of the corresponding polynomial by identification. This can be done in time roughly $O(n^5)$.

Now, we can state an even stronger result. Indeed, starting from the previous theorem, once can easily show that any system of n equations in n variables over $\mathbb{F}_q$, can be represented by an equivalent polynomial in $\mathbb{F}_{q^n}[X]$ of a special shape. The next theorem (and its proof) is important for us, as it provides a method of building such an equivalent polynomial (in $\mathbb{F}_{q^n}[X]$).

Theorem 2 (Kipnis, Shamir 99). $P_1(x_1, \cdots, x_n), \cdots, P_n(x_1, \cdots, x_n)$ n n $\mathbb{F}_q$ ffi $a_1, \cdots, a_{q^n}$ $\mathbb{F}_{q^n}$ n $(x_1, \cdots, x_n)$ $(y_1, \cdots, y_n)$ $(\mathbb{F}_q)^n$ $y_j = P_j(x_1, \cdots, x_n)$ $1 \leq j \leq n$ $\mathbf{y} = \sum_{i=1}^{q^n} a_i\mathbf{x}^i$ $\mathbf{x} = \sum_{i=1}^{n} x_i\beta_i$ $\mathbf{y} = \sum_{i=1}^{n} y_i\beta_i$ $\mathbb{F}_{q^n}$ $\mathbb{F}_q$

We also reproduce the proof, as it leads to a lemma, which we will use in the cryptanalysis.

Proof

The mapping $\overline{\phi_{1,i}} : (x_1, \cdots, x_n) \rightarrow (x_i, 0, \cdots, 0)$ is $\mathbb{F}_q$-linear. Thus, from the first theorem, there exists a corresponding polynomial over $\mathbb{F}_{q^n}$, ie there exists $P_{1,i} \in \mathbb{F}_{q^n}[X]$ such that $\overline{P_{1,i}(\mathbf{x})} = \overline{\phi_{1,i}}(\bar{x})$.

To represent the mapping $(x_1, \cdots, x_n) \rightarrow (\prod_{i=1}^{k} x_i^{c_i}, 0, \cdots, 0)$ we can simply multiply the polynomials $P_{1,i}$ corresponding to each linear transformation.

The polynomial corresponding to the following mapping, $(x_1, \cdots, x_n) \rightarrow (0, \cdots, 0, \prod_{i=1}^{k} x_i^{c_i}, 0, \cdots, 0)$ (where the non-zero component of the image is the component of β_j) is simply the product of all the polynomials $P_{1,i}^{c_i}$ $(i = 1, \cdots, n)$ multiplied with $\beta_1^{-1}\beta_j$.

This proof leads to the following lemma:

Lemma 1. n d n $\mathbb{F}_q$ x ffi $G(x)$ $\mathbb{F}_{q^n}$ d q, $q^{i_1} + q^{i_2} + \cdots + q^{i_n}$ d $G(x)$ ffi

. . Hence, given a system of n polynomials (in n variables) of maximum total degree d over $\mathbb{F}_q$, we can find the coefficients of the corresponding polynomial over $\mathbb{F}_{q^n}$ easily. Indeed, one can compute all the polynomials $P_{1,i}$ $(i = 1, \cdots, n)$ and then compute their product (as mentioned in the later proof). Another strategy to find the polynomial is by using simple interpolation based on sufficiently many Input/Output pairs.

4 Relinearization Technique

In this section, we are interested in solving an overdefined system of quadratic equations in some finite field.

Let us say that this system has m variables, and many more equations, i.e. ϵm^2 for $\epsilon > 0$. In [9] the authors demonstrate an easy algorithm to solve this system, if ϵ is not too small.

We briefly recall this technique, which will be used in the cryptanalysis.

First, if $\epsilon \gtrsim \frac{1}{2}$ the system is easily solvable because one can set new variables $y_{ij} = x_i x_j$. The equations becomes linear in the $\approx \frac{n^2}{2}$ variables y_{ij}, and when $\epsilon \gtrsim \frac{1}{2}$ we have more linear equations than variables, we can solve the system using standard linear algebra.

. . We assume that all (or most) of the linearized equations are linearly independent. Moreover, we assume that if the number of equations is (much) larger than $\frac{n^2}{2}$ we will not have (or will only have very few) parasitic solutions for the $y_{i,j}$ which do not correspond to the solution for x that we are looking for. In [3] the authors have led extensive experiments in the same context, and this heuristic always turned out to be right.

Essentially, if $\epsilon < \frac{1}{2}$ we can still set the new variables y_{ij}, but we will have fewer (linear) equations than variables. The set of solutions to this new system is a vector space of dimension $(\frac{1}{2} - \epsilon)m^2$, and we can easily find a basis $(b_1, \cdots, b_{(\frac{1}{2}-\epsilon)m^2})$ of such a space. The particular solution $(y_{11}, \cdots, y_{nn})$ we are looking for can be expressed as a linear combination of the basis element, i.e. $(y_{11}, \cdots, y_{nn}) = \sum_{i=1}^{(\frac{1}{2}-\epsilon)m^2} z_i b_i$. So we have in fact ϵm^2 equations and $(\frac{1}{2} - \epsilon)m^2$ variables (the z_i).

Now we notice that the linearization step also produces new equations, indeed:

$$(x_a x_b)(x_c x_d) = (x_a x_c)(x_b x_d) = (x_a x_d)(x_c x_b) \Rightarrow y_{ab} y_{cd} = y_{ac} y_{bd} = y_{ad} y_{bc}$$

For each sorted list (a, b, c, d) we get 2 equations (for simplicity we neglect the case where a, b, c, d are not distinct), hence we get in total $\approx \frac{m^4}{12}$ new quadratic equations in y_{ij}, which lead to the same amount of quadratic equations in the z_i.

Now we can linearize these quadratic equations again in the $(\frac{1}{2} - \epsilon)m^2$ variables z_k, we get $\frac{m^4}{12}$ linear equations in $\frac{((\frac{1}{2}-\epsilon)m^2)^2}{2}$ variables.

This system is uniquely solvable if $\frac{m^4}{12} \geq \frac{((\frac{1}{2}-\epsilon)m^2)^2}{2}$ which correspond to $\epsilon \gtrsim 0.1$.

Hence we see that for $\epsilon \gtrsim 0.1$ we can solve the system. For a detailed and precise analysis of the method, see [11]. Of course, the method presented above can be generalised in many ways (see [9]) and in [4] it is shown that the heuristical argument works very well in practice. It is also conjectured that one can solve the system for every $\epsilon > 0$ in time $n^{O(\frac{1}{\sqrt{\epsilon}})}$. So, even for ϵ much smaller than 0.1 this type of method could work.

5 Cryptanalysis

5.1 Recovering the C Matrix

With the help of the theorems from section 3, we are able to present the Double-Round Quadratic cryptosystem in a unified framework.

The private values (the matrix A, B and C) are now the polynomials P_A, P_B, P_C in $\mathbb{F}_{q^n}[X]$ which, according to the first theorem of section 3 are of the following form: $P_A = \sum_{i=0}^{n-1} a_i \mathbf{x}^{q^i}$, $P_B = \sum_{i=0}^{n-1} b_i \mathbf{x}^{q^i}$, $P_C = \sum_{i=0}^{n-1} c_i \mathbf{x}^{q^i}$.

The public system of equations is also represented by a polynomial ($P_{public}(\mathbf{x})$) in $\mathbb{F}_{q^n}[X]$ and satisfies:

$$P_C(P_B(P_A(\mathbf{x})^2)^2) = P_{public}(\mathbf{x}) \tag{1}$$

Moreover, we see from the shape of P_A, P_B, P_c that P_{public} has the form.

$$P_{public} = \sum_{0 \le i_1 \le i_2 \le i_3 \le i_4 \le n-1} p_{i_1,i_2,i_3,i_4} \mathbf{x}^{q^{i_1}+q^{i_2}+q^{i_3}+q^{i_4}}$$

The shape of the public polynomial in $\mathbb{F}_{q^n}[X]$ can also be viewed as a direct consequence of the lemma from section 3. Moreover, the coefficients p_{i_1,i_2,i_3,i_4} can be found by the constructive method presented in the proof of the theorem (from section 3).

As P_C is the polynomial corresponding to the linear transformation with matrix C, it follows that $(P_C)^{-1}$ corresponds to the linear transformation with matrix C^{-1}. From the first theorem we know that $(P_C)^{-1}$ is of the form:

$$(P_C)^{-1} = \sum_{t=0}^{n-1} c_t^{'} \mathbf{x}^{q^t}$$

We can write the equation (1) in the following way:

$$P_B(P_A(\mathbf{x})^2)^2 = (P_C)^{-1}(P_{public}(\mathbf{x})) = \sum_{\mathbf{t=0}}^{\mathbf{n-1}} \sum_{\mathbf{0 \le i_1 \le i_2 \le i_3 \le i_4 \le n-1}} \mathbf{c_t^{'} p_{i_1,i_2,i_3,i_4}^{q^t} x^{q^{i_1+t}+q^{i_2+t}+q^{i_3+t}+q^{i_4+t}}} \tag{2}$$

Moreover $P_A(\mathbf{x}) = \sum_{i=0}^{n-1} a_i \mathbf{x}^{q^i} \Rightarrow (P_A(\mathbf{x}))^2 = \sum_{0 \le i \le j \le n} a_{i,j}^{'} \mathbf{x}^{q^i+q^j}$
so $P_B(P_A(\mathbf{x})^2) = \sum_{k=0}^{n-1} b_k [\sum_{0 \le i \le j \le n} a_{i,j}^{'} \mathbf{x}^{q^i+q^j}]^{q^k}$

It follows that there exist $b_{i,j}$ in $\mathbb{F}_{q^n}$ so that $P_B(P_A(\mathbf{x})^2)$ satisfies:

$$P_B(P_A(\mathbf{x})^2) = \sum_{0 \le i_1 \le i_2 \le n-1} b_{i_1,i_2} \mathbf{x}^{q^{i_1}+q^{i_2}} \tag{3}$$

Which means that the following equation must hold:

$$\Big(\sum_{0 \le i_1 \le i_2 \le n-1} b_{i_1 i_2} \mathbf{x}^{q^{i_1}+q^{i_2}}\Big)^2 = \sum_{t=0}^{n-1} \sum_{0 \le i_1,i_2,i_3,i_4 \le n-1} c_t' p_{i_1,i_2,i_3,i_4}^{q^t} \mathbf{x}^{q^{i_1+t}+q^{i_2+t}+q^{i_3+t}+q^{i_4+t}} \tag{4}$$

Hence, we have an equality between two polynomials, which leads to as many equations as the number of different terms in the polynomials. All the terms of the form $\mathbf{x}^{q^{i_1}+q^{i_2}+q^{i_3}+q^{i_4}}$ with $0 \le i_1, i_2, i_3, i_4 \le n-1$ are present.

And clearly if (i_1, i_2, i_3, i_4) (with $i_1 \le i_2 \le i_3 \le i_4$), and (i_1', i_2', i_3', i_4') (with $i_1' \le i_2' \le i_3' \le i_4'$) are different, then $q^{i_1} + q^{i_2} + q^{i_3} + q^{i_4} \ne q^{i_1'} + q^{i_2'} + q^{i_3'} + q^{i_4'}$. So the number of different terms is $\binom{n+3}{4} = \frac{1}{24}n^4 + o(n^4)$, and the number of unknowns (the b_{ij} and c_k') is $\frac{n(n+1)}{2} + n$.

We obtain another system of quadratic equations, but instead of having n equations in n unknown (over $\mathbb{F}_q$) we now have $\approx \frac{1}{24}n^4$ equations, in $\approx \frac{n^2}{2}$ variables (over $\mathbb{F}_{q^n}$). Using the relinearization technique, we can easily solve this system, as in our case $\epsilon \approx \frac{4}{24} = \frac{1}{6} > 0.1$. We recover the matrix C' via the coefficients c_i', and we can compute $C = (C')^{-1}$.

It makes sense to look at the asymptotical values for n, because it is the security parameter of the cryptosystems. Which means that n is the parameter to be increased if one wants to keep the overall security for the system with regards to the growth in terms of computational power to perform attacks.
Hence the system is theoretically broken, if it is broken for $n \to \infty$.

In practice, the proposed values were $q = 251$ and $n = 9$, an exact computation leads to $\epsilon = 0.17$ which means that the system is also practically solvable for any values of n (as ϵ increases with n).

5.2 Recovering the *B* and *A* Matrices

The coefficients $b_{i,j}$ which were found in the previous paragraph lead us to the polynomials $Q(\mathbf{x})$ with:
$Q(\mathbf{x}) = P_B(P_A(\mathbf{x})^2) = \sum_{0 \le i_1 \le i_2 \le n-1} b_{i_1 i_2} \mathbf{x}^{q^{i_1}+q^{i_2}}$.

Now we can use exactly the same method as above to find the matrices B and A.

We know that $P_B(\mathbf{x}) = \sum_{i=0}^{n-1} b_i \mathbf{x}^{q^i}$, Hence $P_B^{-1}(\mathbf{x}) = \sum_{i=0}^{n-1} b_i' \mathbf{x}^{q^i}$.
$P_A(\mathbf{x})^2 = P_B^{-1}(Q(\mathbf{x}))$.
Hence, we get (remember $P_A(\mathbf{x}) = \sum_{i=0}^{n-1} a_i \mathbf{x}^{q^i}$):

$$\Big(\sum_{i=0}^{n-1} a_i \mathbf{x}^{q^i}\Big)^2 = \sum_{i=0}^{n-1} b_i (Q(\mathbf{x}))^{q^i}$$

We have a quadratic system of $2n$ variables and $\binom{n+2}{2} = \frac{n^2}{2} + o(n^2)$ equations over $\mathbb{F}_{q^n}$. In this case again $\epsilon = \frac{1}{8} > 0.1$, we can recover the variables (so also the matrices A and B) with the relinearization technique mentioned above.

As for the recovering of the matrix C the attack works as well in practice, as for $n = 9$ we find $\epsilon = 0.17$.

6 The Affine Case

It is common in Multivariate Cryptography that the private transformations are chosen to be affine (and not linear), because it does not cost more (at least asymptotically), and may enhance the security of the scheme. Here, instead of using only the matrix A, B and C, we would also have vectors $A^{'}, B^{'}$ and $C^{'}$ to make these three transformations affine.

It is easy to see that even when the transformations are affine, the same cryptanalysis would work. Indeed, in the Shamir-Kipnis formalism the only change would be to add a constant term to P_A, P_B and P_C.

So we would have $P_A(\mathbf{x}) = \sum_{i=0}^{n-1} a_i \mathbf{x}^{q^i} + a_c$, $P_B(\mathbf{x}) = \sum_{i=0}^{n-1} b_i \mathbf{x}^{q^i} + b_c$ and $P_C(\mathbf{x}) = \sum_{i=0}^{n-1} c_i \mathbf{x}^{q^i} + c_c$.

If we use the same technique as above, the number of unknowns and the number of equations changes. Obviously, we have 3 more unknowns (in F_{q^n}) in the affine case. On the other hand we have many more equations to be completed. i.e. in the linear case each equation corresponded to one monom of the form $\mathbf{x}^{q^{i_1}+q^{i_2}+q^{i_3}+q^{i_4}}$, whereas in the affine case we also have to take into account all the monoms of the form $\mathbf{x}^{q^{i_1}+q^{i_2}+q^{i_3}}$,$\mathbf{x}^{q^{i_1}+q^{i_2}}$, $\mathbf{x}^{q^{i_1}}$ and the constant term.

Overall, there are many new equations and only 3 new variables, so the same technique (as in the linear case) will also work.

7 Complexity Analysis

The relinearization technique has polynomial time complexity. Moreover, we use it for quadratic systems of $O(n^2)$ variables, where n is our security parameter. Hence, our attack is clearly polynomial time, so the system is theoretically broken.

Now let us take a deeper look at the actual complexity. It is known that one can solve a linear system of dimension m over a finite field using Coppersmith/Winograd method in $O(m^{2.4})$. The relinearization seeks to solve a system of dimension roughly m^4 hence an overall complexity of $O(m^{10})$.

In the cryptanalysis we use the relinearization with $m = n^2$, hence having a complexity of roughly $O(n^{20})$. The proposed value was $n = 9$, hence the attack is practically feasible.

Acknowledgements. We would like to thank the referees for their many helpful comments.

References

1. Nessie project (2003), https://www.cosic.esat.kuleuven.be/nessie/
2. Biham, E.: Cryptanalysis of Patarin 2-Round Public Key System with S Boxes (2R). In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, Springer, Heidelberg (2000)
3. Courtois, N., Goubin, L., Patarin, J.: SFLASH, a fast asymmetric signature scheme (2003), available at http://eprint.iacr.org/2003/211/
4. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient Algorithms for solving Overdefined Systems of Multivariate Polynomial Equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, Springer, Heidelberg (2000)
5. Din-Feng, Y., K-Yan, L., Zong-Duo, D.: Cryptanalysis of $2R$ Schemes. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, Springer, Heidelberg (1999)
6. Dubois, V., Fouque, P., Shamir, A., Stern, J.: Practical Cryptanalysis of SFLASH. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, Springer, Heidelberg (2007)
7. Goubin, L., Patarin, J.: Asymmetric Cryptography with S-Boxes. In: Han, Y., Quing, S. (eds.) ICICS 1997. LNCS, vol. 1334, Springer, Heidelberg (1997)
8. Imai, H., Matsumoto, T.: Algebraic Methods for Constructing Asymmetric Cryptosystems. In: Calmet, J. (ed.) Algebraic Algorithms and Error-Correcting Codes. LNCS, vol. 229, Springer, Heidelberg (1986)
9. Kipnis, A., Shamir, A.: Cyptanalysis of the HFE Public Key Cryptosystem. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, Springer, Heidelberg (1999)
10. Koeblitz, N.: Algebraic Aspects of Cryptography. Springer, Heidelberg (1998)
11. Moh, T.: The Method of Relinearization of Kipnis and Shamir and its Applications to TTM (1999), available at http://citeseer.ist.psu.edu/371723.html
12. Patarin, J., Goubin, L.: Trapdoor One-Way Permutations and Multivariate Polynomials. In: Han, Y., Quing, S. (eds.) ICICS 1997. LNCS, vol. 1334, Springer, Heidelberg (1997)

A Lightweight Privacy Preserving Authentication and Access Control Scheme for Ubiquitous Computing Environment

Jangseong Kim, Zeen Kim, and Kwangjo Kim

School of Engineering, Information and Communications University
{withkals,zeenkim,kkj}@icu.ac.kr

Abstract. In Ubiquitous Computing Environment (UCE), service provider wants to provide its service to only legitimate users. Some users who belong to same service provider do not want to reveal their identities while using some privacy-related services such as location information, printing, browsing web pages, *etc.* In addition, we should consider lightweight cryptographic protocols because UCE can be constructed by lots of resource and energy constrained devices. In this paper we propose a lightweight privacy-preserving authentication and access control scheme for UCE. Compared to the previous schemes [13,14], our proposed scheme which was designed to reduce the number of public key operations and to improve non-linkability feature is found to be more secure and requires less memory on the user's device. Moreover the proposed scheme provides mutual authentication, accountability and differentiated access control.

1 Introduction

Ubiquitous Computing Environment (UCE) with their interconnected devices and abundant services promise great integration of digital infrastructure into all aspects of our lives [1,2]. User authentication, authorization and access control are also basic requirements for various services in UCE such as Auction, e-Learning, GPS, accessing wireless LAN, e-Government, . . . However we cannot adapt the traditional mechanisms since they do not consider unique characteristics of UCE [3].

Especially user privacy is one of the big challenges due to the limited communication range of ubiquitous computing devices [6,11]. Also there are many "invisible" computing devices in UCE that can collect and analyze the identities, locations and personal information of users without their prior agreement or recognition. Typical approach for dealing with user privacy protection is to provide anonymity based on blind signature scheme. Double spending problem of an authorized credential [13,14] can happen if there is no mechanism for verification that the user is actual holder of the authorized credential. In this case a malicious user can reuse a previous credential of a legitimate user on requesting a special service. Therefore we should consider accountability for an authorized credential.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 37–48, 2007.

Energy management is also another big challenge in UCE because proactivity and self-tuning for providing adaptation capability increases the energy demand on software of a mobile computer in personal computing space. Consequently we should consider lightweight cryptographic protocol for reducing energy demand while providing proper security level.

There are many approaches to solve user privacy and security challenges in UCE [4,5,7,8,9,10,11,12,13,14,15,16]. However, most of these results fall in the scope of establishing general security framework and identifying general security requirements, without providing concrete security protocols. Some work [4,5,7,9,10,15] focused on designing specific security infrastructures to protect user context privacy like location information from service providers. Creese . . [8] and Wu . . [11] revised authentication and privacy requirements and Zugenmaier . . [12] showed that the use of a combination of devices using incompatible anonymizing mechanisms can compromise the anonymity, which is achieved when each device is used seperately. Recent researches [13,14,16] focused on designing concrete security protocols. Characteristics and limitations of these protocols are discussed in Section 2.

In this paper we propose a novel scheme for lightweight privacy-preserving authentication and access control in ubiquitous computing environment. The scheme reduces computation overhead and storage overhead on the user side, provides accountability, improves non-linkability, enhances security level by sharing a selected number set between the user and the authentication server, and does not rely on underlying system infrastructure. Also differentiated service access control is feasible by arranging users in different service groups.

The rest of this paper is organized as follows: In Section 2 we review related work, describe the system architecture of a campus UCE and mention requirements of the system. We present our proposed scheme in Section 3. Then we discuss the security features and the performance analysis of the proposed scheme in Section 4. Finally we conclude the paper in Section 5.

2 Background and Related Work

2.1 Related Work

Jendricke . . [15] introduced an identity management system in UCE. A user can issue multiple identities and use them depending on the applications. Based on these virtual identities the scheme can protect user privacy while providing access control and user authentication. However there is no concrete protocol. He . . [16] presented a simple anonymous ID scheme for UCE but the scheme cannot prevent the double spending problem since it is a direct application of Chaum's blind signature technique [17]. More recently Ren . . [13,14] proposed new scheme which can satisfy the requirements in UCE and prevent double spending problem by combining two cryptographic primitives, blind signature and hash chain. It reduces the number of signature verifications on an authentication server side. Additionally the scheme provides non-linkability and differentiated service access control, prevents double spending problem of an

authorized credential, and does not rely on underlying system infrastructure such as the "lighthouse" or "mist routers" [5]. However a mobile user should store all hash chains of his/her anchor to increase performance aspect and perform public key operation whenever the user sends a service access request message even if the computation can be done off-line.

Gruteser and Grunwald [18] offered a method for hiding user's MAC address with anonymous IDs so that the user cannot be tracked in a wireless LAN environment.

2.2 System Architecture for a Campus UCE

Lots of researchers use a campus UCE to illustrate their example scenarios for UCE. second scenario in [3] and its system architecture usually consists of three major components, , **U**, SP and AS. For supporting lots of mobile users, database server (DS) is considered as a component for the target environment. Figure 1 illustrates the typical system architecture.

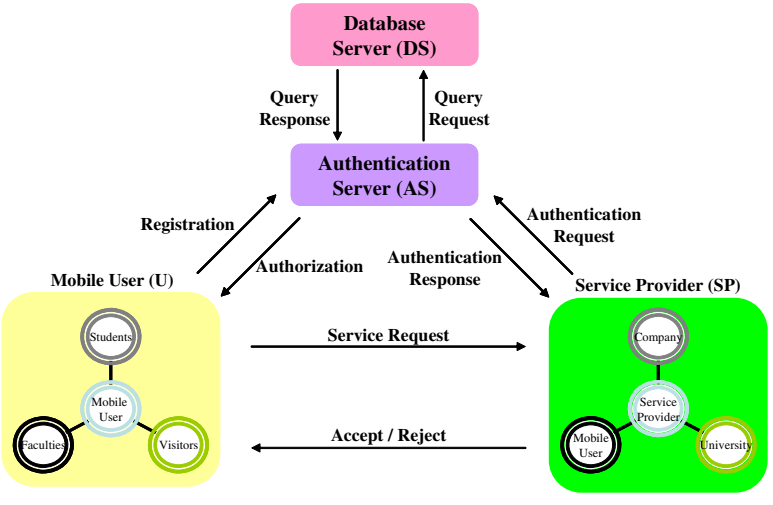


Fig. 1. System architecture

Usually the student is modeled like **U** in the system architecture. Wireless LAN, campus map, a student's time table, a class web page and printer can belong to SP. We assume that DS has all related or partial information for user authentication and has proper security methods to protect the information from the adversaries. However the methods are out of our scope of this paper.

2.3 System Requirements

To support second scenario in [3], basically the system should provide user authentication and access control. Since the scenario assumes its target environment as the UCE, we also consider the characteristics, , energy efficiency and user privacy, of the UCE. Typical approach for protecting user privacy is to provide anonymous identity based on blind signature technique [17]. If there is no verification step to check that a **U** is an actual holder of the authorized credential then a malicious user can reuse the authorized credential [13,14].

Based on these considerations Ren , states the system requirements for the UCE [13,14]. The system should 1) provide explicit mutual authentication between **U** and SP; 2) allow the mobile users to anonymously interact with a SP; 3) enable differentiated service access control among different users; 4) provide flexibility, scalability to both **U** and SP; 5) generate fresh session key to secure the interaction if necessary; 6) have high efficiency with respect to communication, computation costs and management overheads; 7) provide easy accountability.

3 Our Proposed Scheme

We assume that a **U** can control the source addresses of the outgoing Medium Access Control (MAC) frames since it is a prerequisite for anonymous communications. Gruteser , [18] touched one of the detailed methods for this kind of modification and detailed of which is out of scope of this paper. Also all users' public keys, all SID and public key corresponding to each SID are stored in a DS. By sending a query message to the DS, an AS can get proper information and the mobile user knows the mapping between SID and its corresponding public key. Additionally the **U** determines n based on his/her service access frequency. The SP defines the scope and the meaning of service type, associates each user with a particular service type, assigns a unique public key to each service type and provides this information to the AS for further enforcement of authorization rules. Table 1 illustrates the notation used in this paper.

Our proposed scheme consists of two main phases. The first phase is to generate and authorize a user's credential information. Second phase is to establish a fresh session key based on the user's authenticated credential information. Our proposed scheme can hide the relationship between the authorized credential and the mobile user's real identity through blind signature technique based on the first phase. Moreover our scheme can provide non-repudiation because an anchor value contains a user's signature which consists of access frequency n, his/her identity and a fresh nonce. To provide accountability of the authorized credential we adopt selected number set. The selected number set is expressed as l-bit array. **U** only once generates it randomly during the first access request. For example if the i-th element of the array is 1, it means that i is already selected.

Table 1. Notation

U	A mobile user
AS	Authentication server
SID	A service type identifier is identified by a unique public key and it describes a selected subset of the available service pool that can be accessed by a mobile user
SP	Service provider or service access point
K_{AB}	Shared secret key between entities A and B
m, X_m	A message m and its corresponding ciphertext
(m_0, m_1)	Concatenation of two messages m_0 and m_1
$\{m\}_{K_A}$	A message m is encrypted by K_A
$\{m\}_{PriK_A}$	A message m is signed by private key of entity A
$H(m)$	Hash message m
n	A user's access frequency
S	A selected number set and its length should be larger than $2n$
ID_A	An identifier of entity A
$C^i, i = 0, 1, \cdots$	A series of authorized credentials
$j^i, i = 1, 2, \cdots$	A series of a user's number selections
$R_A^i, i = 1, 2, \cdots$	A series of nonce generated by entity A and it is usually a 64-bit pseudo random number.
$CertA$	A certificate which binds entity A with A's public key $PubK$
$Credential$	A ticket for authentication
$Anchor$	An initial credential C^0

3.1 Credential Generation

The **U** generates two fresh nonces and signs his/her identity together with one fresh nonce R'_U using own private key $PriK_U$. Then the **U** computes an anchor value C^0 with the signature. Note that the procedure can be done off-line. We summarize it as:

1. Generate two fresh nonces: R'_U and R''_U
2. Sign user's own ID with a fresh nonce $R'_U and n$:

$$\{ID_U, n, R'_U\}_{PriK_U}$$

3. Compute the anchor value C^0 of credential chain as:

$$C^0 = h(ID_U, n, R'_U, \{ID_U, n, R'_U\}_{PriK_U})$$

4. Blind C^0 as $C_U = \{R''_U\}_{PubK_{SID}} \times C^0$

3.2 Credential Authorization

The **U** sends own identity, blinded credential C_U and SID with own certificate. Next the AS checks validation of the received certification and verifies whether the **U** has access permission on the service. Note that the proposed scheme

can provide the differentiated service access control through this verification procedure. If the result is valid and the requestor has access permission, the AS signs on C_U and sends C_S, ID_{AS} and the received identity to the **U**. Then the **U** verifies ID_U and ID_{AS}. Only if the information is valid, the **U** can get valid credential information by unblinding the received C_S. Otherwise the **U** discards it and retries. We illustrate this procedure in Figure 2.

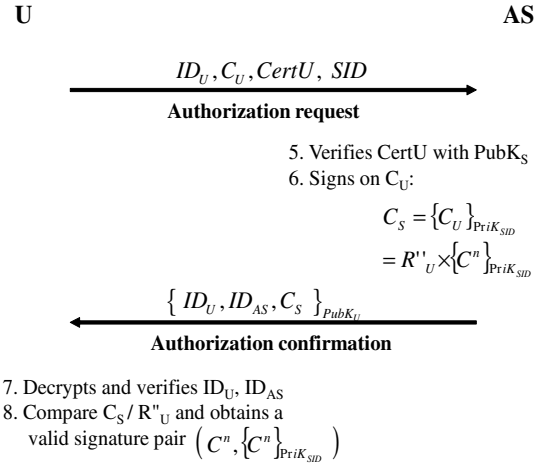


Fig. 2. Authorization of credential information

3.3 Verification of Credential and Session Key Establishment

Only if the **U** has the legitimacy of the target service, the **U** sends correct C^0, C^{i-1} and S^i to the AS. Also the AS store C^0, C^{i-1} and S^i to the DS only if it verifies that the credential is authorized. The AS authenticates **U** based on these facts. Moreover both entities **U** and AS can easily generate a fresh session key $K_{U,AS}$ since they shared the anchor value and S. Also the **U** discloses her previous credential information C^{i-1}.

When the **U** sends an i-th access request to the SP, the **U** generates a fresh nonce R_U^i and selects one random number j between 0 to $l-1$. Next the **U** verifies j is not in the list S. If j is in S then the **U** should select unused random number. Then she generates one time credential as $C^i = h(C^0, j^i, R_U^i)$. Also both entities **U** and AS share a secret key $K_{U,AS}$ by computing as:

$$K_{U,AS} = \begin{cases} h(C^0, PubK_{AS}, R_U^1, j_U^1, SID) & \text{if } i = 1 \\ h(C^0, C^{i-1}, SID) & \text{otherwise} \end{cases}$$

The SP forwards the request message to the AS with a fresh nonce. After decrypting the request message, the AS checks duplication and validation of the secret information, C^{i-1} and S. There are two cases in the verification procedure:

1. When type is 0: It means that the received message is the user's "first access request". After decrypting it, the AS checks whether the requestor has an authorized credential. So the AS signs C^0 with the private key of the SID and compares the result $\{C^0\}_{PriK_{SID}}$ with the received signature. Only if the result is same, then the AS computes $C^1 = h(C^0, j^1, R_U^1)$ and stores SID, S^1, C^0 and C^1 in the DS. Otherwise the AS discards it.
2. When type is 1: To get proper C^0 and S^{i-1}, the AS sends a query message to the DS by setting C^{i-1} and SID as searching condition. If the AS can finds C^0 and S^{i-1}, then the received message can be decrypted by $K_{U,AS}$. After decrypting the request message, the AS verifies that the j-th index of the stored S^{i-1} is 0 and the stored S^{i-1} is the same as S^i except the j-th index. Only the verification result is correct, then the AS believes the **U** has legitimacy of the requested service and stores C^i and S^i in the DS. Otherwise the AS discards it. If there are several verification failure on series of the authorized credentials, the AS can request the mobile user to change his/her credential or notify that there is an impersonation attack on the **U**. Note that C^i and S^i are stored as the authorized credential and the selected number list respectively.

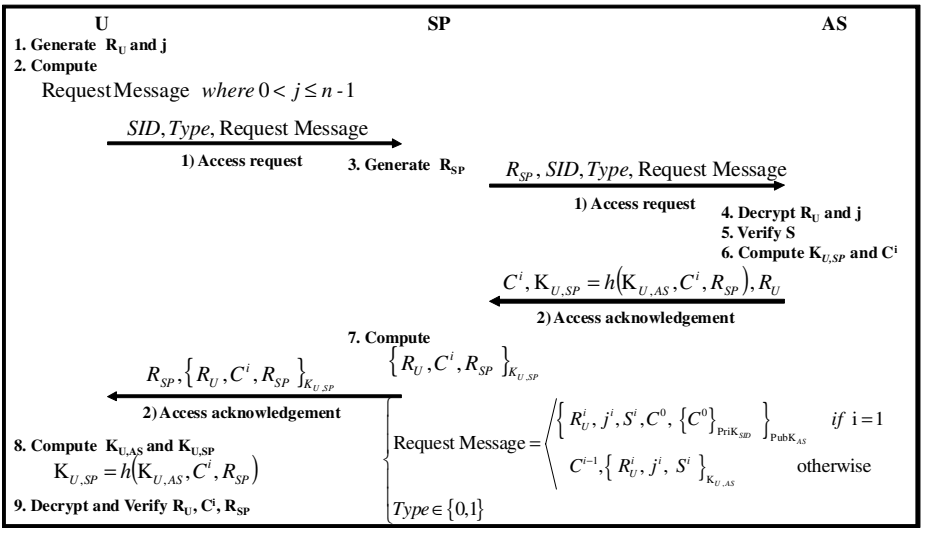


Fig. 3. Verification of credential information and Key generation when the **U** sends the i-th access request

After verifying validation and duplication of the request message, the AS computes $K_{U,SP}$ which is used to secure communication between SP and **U**. Next the AS sends $C^i, K_{U,SP}$ and R_U^i to the SP. The SP encrypts received information with a fresh nonce R_{SP} by using $K_{U,SP}$. Through this activity our proposed scheme provides explicit key authentication between SP and **U**. This

processes are shown in Figure 3. Since to provide a secure tunnel is not our interesting point we simply assume that there is a secure tunnel (e.g., IPsec ESP mode [19]) between SP and AS.

After computing $K_{U,AS}$ and $K_{U,SP}$, the **U** decrypts the received access acknowledgement and verifies C^i,R_U^i and R_{SP}. If the verification result is correct, the **U** can access the target service of the SP. Otherwise the **U** resend the access request to the SP.

3.4 Extension for Out-of-Order Requests

Sometimes the **U** might want to request multiple services simultaneously. If the multiple concurrent sessions are handled by a single server, it is possible to happen that the access request messages arrive out of order at the AS due to unexpected network problems. To deal with this problem we adapt a sliding-window-based extension to the credential verification and key generation procedure on the AS side. We assume that the DS has the stored credential list, the selected number list, the nonce list and the encrypted message to deal with our-of-order requests. There are two cases to deal with out-of-order requests:

1. When the AS cannot find C^{i-1} in the authorized credential and the stored credential list
   (a) Store C^{i-1} and $\{R_U^i, j^i, C^0\}_{K_{U,AS}}$ to the stored credential list and the encrypted message respectively.
2. When the AS find C^{i-1} in the authorized credential and the stored credential
   (a) Send a query message to the DS by setting C^{i-1} and SID as searching condition for getting proper S^{i-1} and C^0.
   (b) Compute $K_{U,AS}$ and decrypt the received message.
   (c) Flip the j-th index of the stored S^{i-1} only if the index is set as 0. Otherwise, discard it.
   (d) Update C^i in the authorized credential and generate C^{i+1}. Next search the generated credential in the stored credential list. If the C^{i+1} are found in the stored credential list then repeat 2.(a)- 2.(d) steps until the searching has failed or the stored credential list has empty.

4 Analysis of Our Proposed Scheme

In this section we analyze the performance and security related features of our proposed scheme.

4.1 Performance

– ***Storage overhead:*** **U** is only required to save C^0, R^i, j^i, n and S. Since all credential information except the anchor value can be generated directly from the anchor value, it means that our proposed scheme does not require to store all credential information. Although **U** in [13,14] should store C^0, to

avoid repeated hash operation all credential information should be stored. In this point our proposed scheme requires less storage capability. Additionally our proposed scheme is more flexible in the view of access frequency since the information which should be stored is fixed even if the user's access frequency is increased.

– ***Computation overhead:*** Except first access request encrypted with a public key of an AS, all messages between **U** and AS are encrypted using a shared symmetric key. Therefore our proposed scheme is computationally efficient since symmetric key operation is lightweight than public key operation, We compare computation overhead of the proposed scheme with the scheme in [13,14] in Table 2. Note that in Table 2 if we do not append the term "off-line", then the communication entity such as **U**, AS and SP, needs online computation.

Table 2. Computation overheads comparison

		# of Pub. Key	Sig.Veri.	Nonce Gen.	Hash Oper.	# of Sym. Key
[13,14]	User	1(off-line)	0	1	2	3
	SP	0	0	1	2	3
	AS	1	1/n	0	0	0
Ours	User	1/n(off-line)	0	1	1	1(off-line)+1
	SP	0	0	1	0	1
	AS	1/n	1/n	0	2	1

– ***Communication overhead:*** The proposed scheme only requires two rounds to achieve the authenticated key establishment. Note that two rounds in authenticated key establishment protocol are minimum rounds to satisfy its goal. Let's compare the message size in the proposed scheme with the scheme in [13,14]. When we assume that the nonce in the both scheme is 64 bits and the hash function in the both scheme is SHA-1, we can calculate the increased size of the message during n sessions:

$$\text{Increased size} = (n-1) \times (2 \times n + \log n - 159) + (2 \times n + \log n + 1).$$

If the **U**'s access frequency is 80, then the increased size of the message is 392.247. It means that 4.903 bits is increased per each session. However the message size is not critical factor in the campus UCE, the proposed scheme is efficient from the point of communication overhead.

4.2 Security

– ***Mutual authentication:*** In the proposed scheme, the **U** authenticates himself/herself to the AS using own authorized credential, so that the AS knows that the **U** is legal and authorized. The AS also authenticates itself to the **U** through its public key and by showing its knowledge of the corresponding private key.

- ***User context privacy:*** Our proposed scheme protects the **U**'s context privacy against insiders and outsiders. Note that all communication channels are well protected. The AS can only know the **U**'s SID. Also the SP can not imagine who sends the service access request.
- ***Non-linkability:*** Non-linkability means that, for insiders(i.e., SP) and outsiders, 1) neither of them could ascribe any session to a particular **U**, and 2) neither of them could link two different sessions to the same **U** [20]. In the proposed scheme non-linkability is achieved with respect to both of insiders and outsiders. Firstly the authorized credential combined with the fresh nonce is never transmitted in plaintext form. Hence the outsiders can't associate a session with a particular user and ascribe two sessions to the same user. Secondly the **U**'s all authorized credentials are derived from an anchor value and it is only known to AS and **U**. Even if the SP can get all authorized credentials except the anchor value, the SP can't link two different sessions to the same user. Moreover the authorized credential combined with the fresh nonce is never transmitted in plaintext form. Therefore the insiders can't associate a session with a particular user. Note that the AS is regarded as trusted third party.
- ***Accountability and nontransferability equivalency:*** In the proposed scheme the credentials are authorized only when the **U** is explicitly authenticated. By adapting selected set the proposed scheme can provide one-time usage of the authorized credentials. Hence the proposed scheme can prevent double spending problems. Also the proposed scheme can provide good accounting capability feature by incorporating accounting function. Furthermore the proposed scheme provides equivalent nontransferability from the service point of view. Because the credentials are delegated among users, no harm is done to the SP in the sense that the authorized user is responsible for all the service received by own authorized credentials. This property greatly reduces the service abuse problem worried by the SPs.
- ***Data confidentiality and integrity:*** Both entities **U** and SP generate a fresh session key to protect their communications during verification and session key establishment process . Hence data confidentiality and integrity can be easily achieved using symmetric cryptography.
- ***Differentiated service access control:*** Our proposed scheme can provide differentiated service access control by classifying users into different service types. Different users are authorized accordingly based on the service types to which they belong. Hence "User authorization" is accomplished in a differentiated way. Moreover, it is possible to combine usage of the different credentials for high-level differentiated service access control. But it is beyond the scope of this paper.
- ***Enhanced security level:*** The **U**'s every access request message contains S used to prove the actual holder of the message since it is randomly generated by the **U** and only known to **U** and AS. To impersonate the target user, the adversary should present S even if the adversary knows the user's anchor value. Therefore the proposed scheme enhances security level.

– ***No additional key management:*** **U** and AS can generate a shared symmetric key based on the anchor value. Also it is used only one-time. So there is no additional key management overhead by replacing the reduced public key operations with the symmetric key operations.

In table 3 we compare our proposed scheme with other similar approaches whose goal is to provide anonymous interaction between **U** and SP. Note that the SP in our proposed scheme can't link two different sessions to the same user even if the SP can get all authorized credentials except the anchor value. However the SP in the scheme which was proposed by Ren , , [13,14] can link two different sessions to the same user if the SP can get all authorized credentials except the anchor value.

Table 3. Security-related features comparison

	Our scheme	Ren *et al.*[13, 14]	He *et al.*[16]
Concrete protocol	Yes	Yes	Yes
Mutual authentication	Yes	Yes	Yes
User context privacy	Yes	Yes	Yes
Non-linkability	Yes to outsiders, yes to SP	Yes to outsiders partially yes to SP	No
Non-transferability	Almost yes	Almost yes	No
Data confidentiality	Yes	Yes	Easy to obtain
Message integrity	Easy to obtain	Yes	Yes
Differentiated service access control	Yes	Yes	No

5 Conclusion

In this paper we have proposed a lightweight privacy-preserving access control for UCE which can be used as a component in middleware. Our proposed scheme is efficient in solving the conflict between user privacy and user authentication. Because user authentication requires the user identity information while user privacy needs to hide the user identity information. Additionally the proposed scheme improves non-linkability on SP's side, enhances security level and consumes less storage burden on the user's device. Moreover the proposed scheme also provides mutual authentication, accountability and differentiated access control.

In the near future we would like to extend our scheme to deal with privacy and security in the service discovery protocol which is an essential element to access network services. Also we try to show the correctness of the proposed scheme by formal verification method.

References

1. Easy Living, Microsoft Research, http://research.microsoft.com/easyliving
2. Weiser, M.: The Computer for the 21st Century. Scientific of American, 265 (September 1991)

3. Satyanarayanan, M.: Pervasive computing: Vision and Challenges. IEEE Personal Communications 8(4), 10–17 (2001)
4. Al-Muhtadi, J., Ranganathan, A., Campbell, R., Mickunas, M.: A Flexible, Privacy-Preserving Authentication Framework for Ubiquitous Computing Environments. In: Proc. 22nd International Conference on Distributed Computing Systems (ICDCS), pp. 771–776 (2002)
5. Al-Muhtadi, J., Campbell, R., Kapadia, A., Mickunas, D., Yi, S.: Routing Through the Mist: Privacy Preserving Communication in Ubiquitous Computing. In: Proc. ICDCS, Vienna, Austria, pp. 65–74 (2002)
6. Al-Muhtadi, J., Ranganathan, A., Campbell, R., Mickunas, M.: Cerberus: A Context-Aware Security Scheme for Smart Spaces. In: Proc. the First IEEE International Conference on Pervasive Computing and Communications (PerCom), pp. 489–496. IEEE Computer Society Press, Los Alamitos (2003)
7. Burnside, M., et al.: Proxy-Based Security Protocols in Networked Mobile Devices. In: Proc. ACM SAC, Madrid, Spain, pp. 265–272. ACM Press, New York (2002)
8. Creese, S., Goldsmith, M., Roscoe, B., Zakiuddin, I.: Authentication for Pervasive Computing. In: Hutter, D., Müller, G., Stephan, W., Ullmann, M. (eds.) Security in Pervasive Computing. LNCS, vol. 2802, pp. 116–129. Springer, Heidelberg (2004)
9. Langheinrich, M.: A Privacy Awareness System for Ubiquitous Computing Environments. In: Borriello, G., Holmquist, L.E. (eds.) UbiComp 2002. LNCS, vol. 2498, pp. 237–245. Springer, Heidelberg (2002)
10. Nahanishi, K., Nakazawa, J., Tokuda, H.: LEXP: Preserving User Privacy and Certifying Location Information. In: Proc. 2nd Workshop Security Ubicomp (2003)
11. Wu, M., Friday, A.: Integrating Privacy Enhancing Services in Ubiquitous Computing Environments. In: Borriello, G., Holmquist, L.E. (eds.) UbiComp 2002. LNCS, vol. 2498, Springer, Heidelberg (2002)
12. Zugenmaier, A., Hohl, A.: Anonymity for Users of Ubiquitous Computing. In: Proc. Security Workshop in Ubicomp, Seattle, Washington (October 2003)
13. Ren, K., Lou, W.: Privacy Enhanced Access Control in Ubiquitous computing Environments. In: 2nd International Conference of Broadband Networks 2005, 3-7 October 2005, vol. 1, pp. 356–365 (2005)
14. Ren, K., Lou, W., Kim, K., Deng, R.: A Novel Privacy Preserving Authentication and Access Control Scheme for Pervasive Computing Environments. IEEE Transactions on Vehicular Technology 55(4), 1373–1384 (2006)
15. Jendricke, U., Kreutzer, M., Zugenmaier, A.: Pervasive Privacy with Identity Management. In: Proc. 1st Workshop Security, Ubicomp (2002)
16. He, Q., Wu, D., Khosla, P.: Quest for Personal Control over Mobile Location Privacy. IEEE Commun. Mag. 42(5), 130–136 (2004)
17. Chaum, D.: Untraceable Electronic Mail, Return Address, and Digital Pseudonyms. Communications of the ACM 24(2), 84–88 (1981)
18. Gruteser, M., Grunwald, D.: Enhancing Location Privacy in Wireless LAN Through Disposable Interface Identifiers: A Quantitative Analysis. Mobile Networks and Applications 10(3), 315–325 (2003)
19. Kent, S., Atkinson, R.: Security Architecture for the Internet Protocol, IETF RFC 2401 (1998)
20. Xu, S., Yung, M.: K-anonymous Secret Handshakes with Reusable Credentials. In: Proc. ACM Conf. CCS, pp. 158–167. ACM Press, New York (2004)

Establishing RBAC-Based Secure Interoperability in Decentralized Multi-domain Environments

Jinwei Hu, Ruixuan Li, and Zhengding Lu

College of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, China
{jwhu,rxli,zdlu}@hust.edu.cn

Abstract. Establishing interoperability is the first and foremost problem of secure interoperation in multi-domain environments. In this paper, we propose a framework to facilitate the establishment of secure interoperability in decentralized multi-domain environments, which employ Role-Based Access Control (RBAC) policies. In particular, we propose a method for setting up interoperating relationships between domains by combining role mappings and assignments of permissions to foreign roles. A key challenge in the establishment of secure interoperability is to guarantee security of individual domains in presence of interoperation. We present rules which regulate the interoperability. These rules ensure that constraints of RBAC policies are respected when cross-domain accesses are allowed.

1 Introduction

Due to the extensive use of the Role-Based Access Control (RBAC) [13,6] model and its variants, interoperation based on RBAC is of theoretical and practical importance. A typical method is to create cross-domain role mappings between domains and resolve security breaches arising from the interoperation. Entities in one domain are permitted to access resources of other domains through these mappings.

Generally speaking, existing methods of specifying role mappings can be categorized into two types: (1) role mappings manually selected by domains' security administrators [9,16,15], and (2) role mappings automatically generated by a trusted third-party [14,12]. Simple as it is, the first method provides no convenient tool for interoperability management and some conflicts may be too complicated for administrators to detect and resolve manually. These conflicts can result in security breaches such as unexpected authorizations and contradictions to constraints of RBAC policies.

The second approach depends on a trusted third-party to integrate domains' access control policies. It enables automatic generation of role mappings and resolution of conflicts in a centralized way. Though convenient, some of its assumptions are at odds with the requirements of secure interoperation in decentralized multi-domain environments. For example, it requires that domains should expose all their access control policies to the third-party. In practice, this assumption seems unreasonable in some scenarios. Because those policies often contain sensitive information, domains may be unable to or unwilling to expose them.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 49–63, 2007.

Establishing interoperability in decentralized multi-domain environments raises some nontrivial questions. Firstly, domains cooperate in an ad-hoc manner. Interoperability between two domains should be independent of other domains, and not influence interoperation among others. Second, a domain may have interoperability with an unpredictable number of domains. Hence, the establishing method should be flexible and not incur heavy burden on the management of domains' access control policies. Third, since Separation of Duty (SoD) policies are regarded as a fundamental principle in computer security [10,2,3], by no means can they be violated despite the advantages of interoperation. Some mechanism should be in place to regulate cross-domain accesses. However, no central party can vouch for it in decentralized environments. Additionally, as a domain has no global knowledge of the environment, conflicts with SoD policies may occur as a result of the interoperation across multiply domains. Hence, it is challenging to preserve the security of individual domains in such a decentralized environment. SoD policies in RBAC, however, are not enforced directly but by Statically Mutually Exclusive Roles (SMER) constraints. We focus on ensuring SMER constraints intact, thereby keeping SoD policies valid.

In this study, we develop such a framework to address these issues. We assume that every domain employs the RBAC models to specify their security policies. The framework distinguishes between domains based on their functions in interoperation. A server domain, written `s-domain`, acts as a provider by sharing its resources, whereas a client domain, written `c-domain`, applies for establishing interoperability with `s-domain`. Each domain can be `s-domain` in some scenarios, and `c-domain` in others. The interoperability is setup in a pair-wise manner between `c-domain` and `s-domain`. We present an approach for specifying interoperability, which combines role mappings and assignments of permissions to roles of `c-domain`. We also study how `s-domain` may enforce SMER constraints in decentralized multi-domain environments. Firstly, a set of cross-domain link rules is derived from the interoperability and SMER constraints of `s-domain`; secondly, one check whether access requests from `c-domain` conform to the rules. We prove that accesses complying with the rules are also consistent with SMER constraints. Therefore, SMER constraints of `s-domain` are honored in the presence of interoperation and, consequently, so are SoD policies.

The rest of the paper is organized as follows. A brief review of RBAC models is given in Section 2. We present an overview of the proposed framework in Section 3, followed by the method of establishing interoperability in Section 4 and the cross-domain link rules in Section 5. We discuss related work in Section 6 and conclude in Section 7.

2 Preliminaries

We assume that all domains in a decentralized multi-domain environment form a set $\mathcal{D}$, and that each domain in $\mathcal{D}$ employs the RBAC model [13,6] to specify security policies. An RBAC state is a 7-tuple $\langle \mathcal{U}, \mathcal{P}, \mathcal{R}, UA, PA, RH, \mathcal{C} \rangle$, where $\mathcal{U}$ is a set of users, $\mathcal{P}$ is a set of permissions, $\mathcal{R}$ is a set of roles, $UA \subseteq \mathcal{U} \times \mathcal{R}$ assigns roles to users, $PA \subseteq \mathcal{R} \times \mathcal{P}$ assigns permissions to roles, $RH \subseteq \mathcal{R} \times \mathcal{R}$ enables permission inheritance between roles, and $\mathcal{C}$ is a set of SMER constraints stating what assignments are not allowed in the state. Let RH^* the reflexive and transitive closure of RH. The

relation RH^* is often called *role hierarchies*. We denote $(r_i, r_j) \in RH^*$ by $r_i \succcurlyeq r_j$ and say that r_i is senior to r_j in the sense that r_i inherits all permissions of r_j. We borrow the expressions of constraints in [10]. A statically mutually exclusive role (SMER) t-m constraint $\texttt{smer}\langle\{r_1, r_2, \cdots, r_m\}, t\rangle$, where $1 < t \leq m$, forbids a user from being assigned to t or more roles.

The RBAC components of a domain are identified by the domain identity. For example, $\mathcal{R}$ in domain α is written as $\mathcal{R}^\alpha$. Components of `s-domain` are identified by the superscript s such as $\mathcal{P}^s$, RH^s and the role hierarchies $\succcurlyeq^s$, and components of `c-domain` by c such as PA^c.

3 Framework Overview

The goal of our framework is to support specifying interoperability in a decentralized multi-domain environment. Prior to the establishment of interoperability, `s-domain` designates the resources available to other domains by means of a *sharing policy*. A *sharing policy* in `s-domain`, $SP : \mathcal{D} \rightarrow \mathfrak{P}(\mathcal{P}^s)$, where $\mathfrak{P}(\mathcal{P}^s)$ is the power set of $\mathcal{P}^s$, is a function mapping each domain in $\mathcal{D}$ to a set of permissions. In our framework, we regard accesses to resources as permissions. However, interoperability is not enabled simply by the sharing policy, because that would result in the loss of all attractive traits of RBAC in the first place; and interoperability in this way is prone to security breaches as well. We illustrate the framework through the interoperability establishing process between a dummy pair of `s-domain` and `c-domain`.

As shown in Fig. 1, `c-domain` initiates a setup process by issuing an interoperation request. An *interoperation request* is a set of request-specifications. A *request-specification* is of the form $[r^c, RPS]$, where $r^c \in \mathcal{R}^c$ and $RPS \subseteq \mathcal{P}^s$. r^c is referred to as an *applying role*. The request-specification $[r^c, RPS]$ states that the applying role r^c wishes to obtain the permissions in RPS. An interoperation request is of form $\{[r_1^c, RPS_1], \cdots, [r_k^c, RPS_k]\}$.

After receiving a request, `s-domain` may map each such r^c to local roles or directly assign it local permissions. We refer to both role mappings and direct permission assignments as *cross-domain links*. A *role mapping* is of the form $r^c \rhd r^s$, where $r^c \in \mathcal{R}^c$ and $r^s \in \mathcal{R}^s$. The role mapping $r^c \rhd r^s$ enables users assigned to r^c in `c-domain` to assume r^s in `s-domain`. A *direct permission assignment* is of the form $r^c \rhd p^s$, where

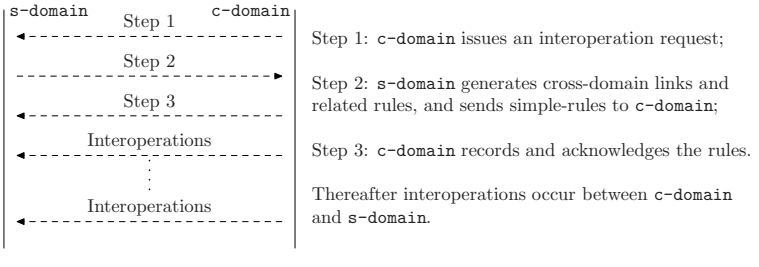


Fig. 1. The overall process of interoperability establishment

$r^c \in \mathcal{R}^c$ and $p^s \in \mathcal{P}^s$. The assignment $r^c \rhd p^s$ grants users, being member of r^c in c-domain, the permission p^s.

By associating a set $RM[r_i^c]$ of role mappings and a set $DPA[r_i^c]$ of direct permission assignments with any request-specification $[r_i^c, RPS_i]$, $1 \leq i \leq k$, s-domain finishes the generation of cross-domain links for c-domain. Let CDL[c-domain] be the set $\bigcup_{i=1}^{k} (RM[r_i^c] \cup DPA[r_i^c])$. However, allowing users in c-domain to use all links in CDL[c-domain] may result in security breaches in s-domain (as discussed in section 5.1). Some restrictions must be imposed on their usage. s-domain deduces from the links and its own RBAC policies a set of *cross-domain link rules*. Then, cross-domain accesses are made conforming to these rules. s-domain also informs c-domain of the interoperability by one kind of rules (i.e., simple-rules). Finally, c-domain will record and acknowledge the simple-rules, indicating that, interoperability has been established and thus users of c-domain have accesses to s-domain.

We illustrate the generation of the set CDL[c-domain] and cross-domain link rules in Section 4 and 5 respectively. Before going into the details, we state the assumptions that the framework relies on. First, in a decentralized multi-domain environment, a domain is only aware of its own RBAC policies, and the interoperability concerning itself (i.e., links from and to the domain). Second, both c-domain and s-domain have some knowledge of the permissions being shared by some means (e.g., by trust negotiation or by contract). Finally, messages delivered to setup interoperability are signed by domains' private keys. And domains' public keys are available to each other.

4 Interoperability Establishment

4.1 The Method of Setting Interoperability

Given a request-specification $[r^c, RPS]$, this section shows how to obtain the sets $RM[r^c]$ and $DPA[r^c]$. A top-down search on s-domain's role hierarchies is performed to generate role mappings. Let $MAPS$ be SP(c-domain) $\cap\, RPS$. The mapping $r^c \rhd r^s$ is added to the role mapping set $RM[r^c]$ if the following conditions hold:

B1 $PermSet(r^s) \subseteq MAPS$, and
B2 $\neg\exists r_1 \in \mathcal{R}^s\ [\ r_1 \succcurlyeq^s r^s\ \wedge\ PermSet(r_1) \subseteq MAPS\ ]$,

where $PermSet(r)$ denotes the permission set of the role r.

Let $PermSet(RM[r^c])$ be $\bigcup_{r^c \rhd r^s \in RM[r^c]} PermSet(r^s)$. Obviously, there is a possibility that the maximum interoperability (i.e., $PermSet(RM[r^c]) = MAPS$) is not achieved. To this aim, each permission in $MAPS \backslash PermSet(RM[r^c])$ is directly assigned to the applying role r^c. Formally,

$$DPA[r^c] = \{r^c \rhd p | p \in MAPS \setminus PermSet(RM[r^c])\}.$$

We use an example shown in Fig. 2 to illustrate the ideas mentioned above. Consider that s-domain shares the permissions $\{p1, p3, p6, p7, p8\}$ with c-domain, which in turn submits the request $\{[r_1^c, \{p1, p3, p8\}], [r_2^c, \{p4, p6, p7\}]\}$. Upon receiving the request, s-domain embarks on the creation of cross-domain links. As for the applying

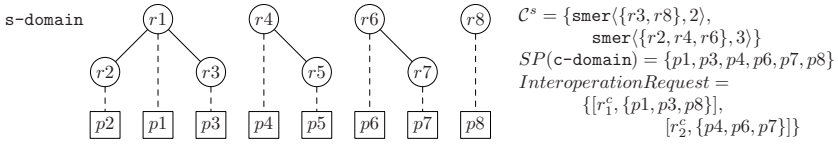


Fig. 2. Roles are show in circles and permissions in boxes. Solid lines represent role hierarchies, and dashed lines represent role-permission assignments.

role r_1^c, $r3$ and $r8$ can be mapped to r_1^c. Although r_1^c asks for $p1$, $r1$ cannot be mapped, because, if so, that would enable r_1^c to obtain the permission $p2$ which `c-domain` is prohibited from. Instead, we directly assign $p1$ to r_1^c to achieve maximum interoperability. Similarly, for r_2^c we create the links $r_2^c \rhd r6$ and $r_2^c \rhd p4$. To further show the necessity of direct permission assignments for the maximum interoperability, consider a request specification $[r_3^c, \{p1, p4\}]$. Then $RM[r_3^c] = \emptyset$, that is, no interoperability is enabled by role mappings. Thus, the assignments $r_3^c \rhd p1$ and $r_3^c \rhd p4$ are necessary for the expected interoperation.

4.2 Comparison with Existing Works

The most of manual approaches to establishing interoperability in literature [9,16,15] do not discuss how to fulfill a request, but simply create mappings related to existing roles. They can be viewed as our method without direct permission assignments. As discussed earlier, this may result in less or none interoperability for some requests. On the other hand, a representative work in [14], called MDRBAC05 in this paper, made an important step towards automatic generation of role mappings. We make some comparisons between our method and MDRBAC05.

The essential difference between the proposed method and MDRBAC05, is that we allow direct permission assignments as a complement of role mappings. This leads to several further distinctions. When pursuing maximal interoperability, often there does not exist suitable roles for mappings. In this case, MDRBAC05 may resort to altering role hierarchies (e.g., splitting roles) to satisfy the request. Instead, we leave these permissions to direct permission assignments; thus role hierarchies in `s-domain` remain the same in presence of interoperation. Keeping `s-domain`'s role hierarchies unchanged has an impact on the applicability of role mapping based interoperability.

Firstly, role hierarchies reflect, to some extent, applications' organizational structures, which is one important characteristic of RBAC. With role hierarchies not conforming to organizational structures, RBAC would be less attractive.

Secondly, in MDRBAC05, roles may be split many times and many new roles would appear. When a domain interoperates with a large number of domains, this may make role hierarchies too complicated to be understood. This directly leads to less manageable RBAC policies in `s-domain` and larger administrative overhead. Worse of all, a new role created simply for mapping may be split again in other subsequent interoperability establishments. This would result in the loss of independence among interoperabilities involving different `c-domains`. The complexity in role hierarchies and dependencies among interoperability are obstacles for large-scale multi-domain

interoperation. By contrast, our method depends solely on roles that exist not because of interoperability establishment. No dependencies between interoperabilities concerning different c-domains exist. The only plus is some role-permission assignments.

Last but not least, when interoperation between s-domain and c-domain ceases, it is expected that s-domain's RBAC policies recover the state prior to the establishment of interoperability. In the case of changing role hierarchies for interoperability, the recovery (e.g., merging split roles, deleting roles and role hierarchies; and readjusting SMER constraints) would incur large administration overhead. Furthermore, new roles created for previous interoperability may not be removed, because it is likely they are mapped or even split in other interoperability establishments. Moreover, as the generation of SMER constraints needs to consider role hierarchies and other SMER constraints, SMER constraints have to be readjusted again for the recovery. By comparison, the recovery in our method only consists of deleting role mappings, direct permission assignments and constraints generated for them, and cross-domain link rules.

However, setting interoperability merely by direct permission assignments is cumbersome and makes it difficult for security administrators to manage cross-domain accesses. Therefore, we propose to combine role mappings with direct permission assignments, and make these assignments as a complementary part of role mappings.

5 Cross-Domain Link Rules

To keep SoD policies intact, restrictions should be imposed on the usage of cross-domain links. In this section, we propose to use cross-domain link rules to regulate these links. Given the set $CDL[\texttt{c-domain}]$ of cross-domain links, we first show how s-domain may deduce link rules, and then illustrate how to enforce them dynamically.

5.1 Constraint Violation Caused by Cross-Domain Links

Improper specification of interoperability can easily lead to conflicts with constraints. For instance, if u_2^c is permitted to use both $r_1^c \rhd r3$ and $r_1^c \rhd r8$ in Fig. 2, then the constraint $\texttt{smer}\langle\{r3, r8\}, 2\rangle$ is violated.

In case SoD policies are undermined by direct permission assignments, we define a constraint analogous to SMER constraints. An SMEP (*Statically Mutually Exclusive Permission*) constraint is expressed as $\texttt{smep}\langle\{p_1, \cdots, p_m\}, t\rangle$, where $1 < t \leq m$. The constraint means that at most t-1 permissions in $\{p_1, \cdots, p_m\}$ can be assigned to a role. The SMEP constraints can be deduced from SoD policies in the same way as SMER constraints, if we simply regard a permission as a role assigned only this permission. Hereafter, we denote the set of SMER constraints and SMEP constraints as CON.

As far as interoperation is concerned, the difference between SMEP and SMER is that, SMEP constraints have impacts on direct permission assignments, whereas SMER constraints only influence role mappings. While rules regarding direct permission assignments are deduced from SMEP constraints and these assignments, rules for role mappings are based on SMER constraints and these mappings. The generation and enforcement mechanisms, however, are of no difference. Unless stated otherwise, we do not explicitly distinguish between them in the following discussion about link rules.

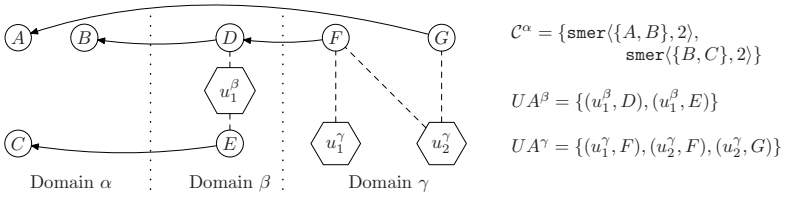


Fig. 3. Two kinds of disagreements with constraints. A line with an arrow across domains represents a role mapping. For example, the mapping $G \rhd A$ is denoted as a line from γ to α.

We simply discuss SMER constraints, while all results applies to SMEP constraints. As shown in [2,10], each t-m SMER constraint is equivalent to a set of t-t SMER constraints. Therefore, we only consider rules with respect to t-t SMER constraints.

We refer to a sequence of role mappings $\{r^{c_1} \rhd r^{s_1}, r^{c_2} \rhd r^{s_2}, \cdots, r^{c_n} \rhd r^{s_n}\}$ as a *path* if $r^{s_i} \succcurlyeq r^{c_{i+1}}$ for all $1 \leq i < n$. By a path, a user may cross several domains in a *session*. We assume that, for each session, the user has an *access history* [15] recording a sequence of roles the user is assigned to in each domain before the current access request. An access history is expressed as a sequence of pairs of a role and a domain to which the role belongs. The first pair represents the user's origin. For example, $\{(\alpha, r_1), (\beta, r_2), (\gamma, r_3)\}$ means a user, belonging to domain α as a member of r_1, had entered into domain β with the role r_2, followed by the access to γ with r_3.

Requests to enter `s-domain` may be classified into two types by users' access histories. We say a request is a *simple-access* if the access history contains only the origin, and a *domain-access* if the access history includes more than one item. For example, as shown in Fig. 3, u_2^γ's entrance into α by $G \rhd A$ is a simple-access; in contrast, if u_2^γ accesses α by the path $\{F \rhd D, D \rhd B\}$, then this is a domain-access because the access history is $\{(\gamma, F), (\beta, D)\}$.

Both kinds of accesses could induce violations of constraints. Correspondingly, there are two kinds of disagreements with constraints. The first kind, called *simple-violations*, consists only of simple-accesses; the second one, named *domain-violations*, results from a combination of simple-accesses and domain-accesses or simply domain-accesses. For example, permitting u_1^β to utilize both $D \rhd B$ and $E \rhd C$ is against $\texttt{smer}\langle\{B, C\}, 2\rangle$; this is a simple-violation. If u_2^γ assumes A in α by $G \rhd A$ and also acquires B by the path $\{F \rhd D, D \rhd B\}$, $\texttt{smer}\langle\{A, B\}, 2\rangle$ is violated; this is a domain-violation.

5.2 Cross-Domain Link Rules

Definition 1. *(Simple-rules) A simple-rule is expressed as:* $\texttt{sr}\langle\tau, L, t\rangle$*, where* $\tau \in \mathcal{D}$*,* $L \subseteq CDL[\tau]$*, and* $1 \leq t \leq |L| + 1$*. This rule forbids the domain* τ *from making use of* t *or more cross-domain links in* L*.*

Accesses constituting a simple-violation all originates from the same `c-domain`. Therefore, it is relatively easy to detect and prevent simple-violations. We simply restrict `c-domain`'s usage of cross-domain links to a certain number. Simple-rules capture this requirement. For example, u_1^β is entitled to only one of $D \rhd B$ and $E \rhd C$, but not both. The simple-rule $\texttt{sr}\langle\beta, \{D \rhd B, E \rhd C\}, 2\rangle$ serves this purpose.

On the other hand, domain-violations are more complicated. Because s-domain lacks knowledge of c-domain's RBAC policies and the interoperability among others, it is more difficult for s-domain to decide whether or not users' accesses violate its constraints.

For instance, consider a user u with the access history $\{(\gamma, F), (\beta, D)\}$ in Fig. 3. Suppose u has obtained the role A by $G \rhd A$. When u requests to access α by $D \rhd B$, α cannot determine whether this access would result in any conflict with constraints. If u happens to be u_1^γ, then no conflict occurs, and u's access should be permitted. Unfortunately, if incidentally u is u_2^γ, it is likely that u assumes A and B, which definitely runs counter to $\mathtt{smer}\langle\{A, B\}, 2\rangle$. In this case, s-domain should decline the request for security reasons. Domain-rules are used to describe this situation.

Definition 2. *A domain-rule is expressed as:* $\mathtt{dr}\langle\tau, Q, t\rangle$*, where* $\tau \in \mathcal{D}$*,* $Q \in \mathfrak{P}(\mathcal{R}^s) \cup \mathfrak{P}(\mathcal{P}^s)$ *and* $1 \leq t \leq |Q|$*. This rule means that the domain* τ *as a whole can obtain at most* $t-1$ *roles or permissions in* Q *of* s-domain*.*

For example, a domain-rule $\mathtt{dr}\langle\gamma, \{B\}, 1\rangle$ may be constructed for the domain-violation mentioned above. Since only t-t SMER constraints are considered, the practical form of a domain-rule is $\mathtt{dr}\langle\tau, Q, |Q|\rangle$. So are simple-rules of the form $\mathtt{sr}\langle\tau, L, |L|\rangle$. Hence, we simply write $\mathtt{dr}\langle\tau, Q\rangle$ for $\langle\tau, Q, |Q|\rangle$ and $\mathtt{sr}\langle\tau, L\rangle$ for $\langle\tau, L, |L|\rangle$. An exception is $\mathtt{sr}\langle\tau, \{l\}, 2\rangle$, which is introduced in Section 5.3.

5.3 Generation of Cross-Domain Link Rules

After the generation of cross-domain links, s-domain associates a set of rules with c-domain. We give the basic idea of the rule generation algorithm. Please see Appendix A for the pseudo-code. Denote CDL[c-domain] as CDL.

The basic idea is to replace roles or permissions in constraints with links. Given $b \in \mathcal{R}^s \cup \mathcal{P}^s$, we write $b \simeq q$ if $b \succcurlyeq^s q$ in the case q is a role, and if $b = q$ when q is a permission. Initially, for a constraint $\langle\{q_1, \cdots, q_t, \}, t\rangle$ in s-domain, where $\{q_1, \cdots, q_t\} \in \mathfrak{P}(\mathcal{R}^s) \cup \mathfrak{P}(\mathcal{P}^s)$, replace q_1 with $l = r^c \rhd b$ if $b \simeq q_1$ holds. Continue this process for the set CDL of links. This creates at most $|CDL|$ many items. Repeat this procedure for each q_i $(2 \leq i \leq t)$ by testing $b \simeq q_i$ with each item after the replacement of q_{i-1}. Perform this process with all constraints in CON. Finally, we would obtain at most $|CON| \cdot |CDL|^T$ items in a set, say $UFRules$ (short for UnFormattedRules), where T is the maximal t among all t-t constraints. Then rules are deduced from $UFRules$. For each item I in $UFRules$, if $I = \langle\{l_1, \cdots, l_k\}, t\rangle$, then add to $SimpleRules$ a simple-rule $\mathtt{sr}\langle\text{c-domain}, L\rangle$, where $L = \{l_1, \cdots, l_k\}$; if $I = \langle\{l_1, \cdots, l_i, q_1, \cdots, q_j\}, t\rangle$, where $i + j = t$ and $1 \leq j \leq t-1$, then we add to $DomainRules$ a rule: $\mathtt{dr}\langle\text{c-domain}, Q\rangle$, where $Q = \{q_1, \cdots, q_j\}$. If a link l is not involved in any simple-rule, we create a special simple-rule: $\mathtt{sr}\langle\text{c-domain}, \{l = r^c \rhd q\}, 2\rangle$. This rule indicates q is available to c-domain with no limitation if used in simple-accesses, but not in domain-accesses.

Continuing the example in Fig. 2. The cross-domain links are $\{l_1 = r_1^c \rhd r3, l_2 = r_1^c \rhd r8, l_3 = r_1^c \rhd p1, l_4 = r_2^c \rhd r6, l_5 = r_2^c \rhd p4\}$, and the set CON in `s-domain` is $\{\texttt{smer}\langle\{r3, r8\}, 2\rangle, \texttt{smer}\langle\{r2, r4, r6\}, 3\rangle\}$. The constraints evolve as follows:

$$\langle\{r3, r8\}, 2\rangle \xrightarrow{\text{replace r3}} \langle\{l_1, r8\}, 2\rangle \xrightarrow{\text{replace r8}} \langle\{l_1, l_2\}, 2\rangle$$

$$\langle\{r2, r4, r6\}, 3\rangle \xrightarrow{\text{replace r2 or r4}} \langle\{r2, r4, r6\}, 3\rangle \xrightarrow{\text{replace r6}} \langle\{r2, r4, l_4\}, 3\rangle$$

Finally, $UFRules = \{\langle\{l_1, l_2\}, 2\rangle, \langle\{r2, r4, l_4\}, 3\rangle\}$. Rules $\texttt{sr}\langle\texttt{c-domain}, \{l_1, l_2\}\rangle$ and $\texttt{dr}\langle\texttt{c-domain}, \{r2, r4\}\rangle$ are deduced. As l_3, l_4, and l_5 do not appear in simple-rules, $\texttt{sr}\langle\texttt{c-domain}, \{l_i\}, 2\rangle$, where $i \in \{3, 4, 5\}$, are also constructed.

5.4 Enforcement of Cross-Domain Link Rules

`s-domain` enforces cross-domain link rules to guard against conflicts with constraints that result from accesses through these links. We assume that, when making requests to `s-domain`, users are required to manifest access histories [15]. `s-domain` checks rules with users' access histories to decide whether the request should be allowed. If the request is a simple-access, only simple-rules are checked. Otherwise, both simple-rules and domain-rules take effect; only positive decisions from both types of rules could lead to an approval.

Enforcement of Simple-Rules. To enforce simple-rules, `s-domain` associates with each domain two lists, *approve-list*, denoted as $\mathcal{L}_{app}$, and *check-list*, denoted as $\mathcal{L}_{chk}$.

$$\mathcal{L}_{app} = \{l \in CDL[\texttt{c-domain}] \mid \exists \texttt{sr}\langle\texttt{c-domain}, \{l\}, 2\rangle \in SimpleRules\}$$
$$\mathcal{L}_{chk} = \{sr \in SimpleRules \mid sr \text{ is of the form } \texttt{sr}\langle\texttt{c-domain}, L\rangle\}.$$

Consider a user in `c-domain` applies for roles or permissions in `s-domain` with a the link l. Obviously, if $l \in \mathcal{L}_{app}$, the request is permitted. Otherwise, `s-domain` checks whether there are some rules concerning l. We define $\Omega_{sr}(l) = \{e \in \mathcal{L}_{chk} | e = \texttt{sr}\langle\texttt{c-domain}, L\rangle \wedge l \in L\}$. $\Omega_{sr}(l)$ contains all the rules regarding l. If $\Omega_{sr}(l) = \emptyset$, then a negative decision is announced, because this implies that l is a nonexistent link. Otherwise we check whether l is forbidden by simple-rules. Note that a rule $\texttt{sr}\langle\texttt{c-domain}, \{l\}\rangle$, i.e., $\texttt{sr}\langle\texttt{c-domain}, \{l\}, 1\rangle$, denies all requests for l by users in `c-domain`. Therefore, if there exists $e \in \Omega_{sr}(l)$ such that $e = \texttt{sr}\langle\texttt{c-domain}, \{l\}\rangle$, the request is refused. Otherwise `s-domain` permits the required access.

Enforcement of Domain-Rules. Each domain is associated with one list, where domain-rules regarding the domain are stored.

Consider a user with an access history $\Sigma = \{(\sigma_1, r_1), \cdots, (\sigma_j, r_j)\}$, $j \geq 2$, issues a request with a link $l = (r^c \rhd b)$. Definitely we have $r^c = r_j$. Denote the associated list for domain σ_i as $\mathcal{L}_i$ $(1 \leq i \leq j)$. The link l is first checked by simple-rules. If a negative decision is given, it is unnecessary for domain-rules to examine the request.

Due to the access history, domain-rules may make roles or permissions inaccessible for the sake of `s-domain`'s security. We check whether the requested one is such a role or permission. Define $\Omega_{dr}(l, \Sigma) = \{e \in \bigcup_{i=1}^{j-1} \mathcal{L}_i | e = \texttt{dr}\,\langle\sigma, Q\rangle \wedge \exists q \in Q\,[b \simeq q]\}$,

where $\sigma \in \{\sigma_1, \cdots, \sigma_{j-1}\}$. $\Omega_{dr}(l, \Sigma)$ includes all the rules that might prohibit this user from using b. If there exists an $e \in \Omega_{dr}(l, \Sigma)$ such that $e = \mathtt{dr}\,\langle \sigma_i, Q\rangle \wedge \forall q \in Q\,[b \simeq q]$, where $1 \leq i \leq j-1$, then the request is denied. Otherwise a positive answer is given. This means that the request is denied if and only if the user has ever entered a domain that is forbidden from b.

5.5 Evolution of Cross-Domain Link Rules

Rules created in Section 5.3 provide a basis for the enforcement of constraints in presence of interoperation. However, these rules do not rule out all the accesses which may cause contradictions to constraints.

For example, consider the user u_1^c in Fig. 2. Though the rule $\mathtt{sr}\langle \mathtt{c\text{-}domain}, \{r_1^c \rhd r3, r_1^c \rhd r8\}\rangle$ is enforced, it is still possible for u_1^c to violate $\mathtt{smer}\langle\{r3, r8\}, 2\rangle$. u_1^c may start a session in which $r3$ is activated. After some operations, he or she could close the session and start another session being member of $r8$. The rule $\mathtt{sr}\langle \mathtt{c\text{-}domain}, \{r_1^c \rhd r3, r_1^c \rhd r8\}\rangle$ dose not forbid this behavior.

Evolution Strategy of Simple-Rules. Define a function $\overline{L}(l^*, L^*)$ as follows:

$$\overline{L}(l^*, L^*) = \begin{cases} \{r^{c*} \rhd p^* \mid & (l^* = r^{c*} \rhd b^*) \wedge (b^* \in \mathcal{R}^s) \\ \quad p^* \in PermSet(b^*)\} & \quad \wedge (\forall l_1^* \in L[(l_1^* = r_1^{c*} \rhd b_1^*) \wedge (b_1^* \in \mathcal{P}^s)]) \\ \{l^*\} & otherwise \end{cases}$$

Both simple-accesses and domain-accesses may influence simple-rules. After a simple-access utilizes a link $l_i \in L$, for each $\mathtt{sr}\langle \mathtt{c\text{-}domain}, L\rangle$ in $\Omega_{sr}(l)$, the rule is changed as $\mathtt{sr}\langle \mathtt{c\text{-}domain}, L \backslash \overline{L}(l_i, L)\rangle$. Each time a link is used, rules are adjusted accordingly. Recursively, the rule would finally become $\mathtt{sr}\langle \mathtt{c\text{-}domain}, \{l_k\}\rangle$, which means l_k is accessible under no circumstances. Hence, only $t-1$ links in L are admitted, which complies with the rule. This also implies that users can never acquire all the roles or permissions related to L as a whole. On the other hand, a domain-access utilizing q incurs the following adjustment: for each $\mathtt{sr}\langle \mathtt{c\text{-}domain}, L\rangle$ in $\mathcal{L}_{chk}$ such that $\exists l \in L[l = r^c \rhd b \wedge b \simeq q]$, replace it with $\mathtt{sr}\langle \mathtt{c\text{-}domain}, L \backslash \overline{L}(l, L)\rangle$; add each link in $\overline{L}(l, L)$ to $\mathcal{L}_{app}$.

Evolution Strategy of Domain-Rules. Define a function $\overline{Q}(b^*, Q^*)$ as follows:

$$\overline{Q}(b^*, Q^*) = \begin{cases} PermSet(b^*) & b^* \in \mathcal{R}^s \wedge Q^* \subseteq \mathcal{P}^s \\ \{q^* \in Q^* \mid b^* \simeq q^*\} & otherwise \end{cases}$$

Suppose that a link $l = r^c \rhd b$ is used in a domain-access by a user with an access history $\Sigma = \{(\sigma_1, r_1), \cdots, (\sigma_j, r_j)\}$, $j \geq 2$. The domain-rule $\mathtt{dr}\langle \sigma, Q\rangle$ in $\Omega_{dr}(l, \Sigma)$, where $\sigma \in \{\sigma_1, \cdots, \sigma_{j-1}\}$, turns into $\mathtt{dr}\langle \sigma, Q \backslash \overline{Q}(b, Q)\rangle$. For a domain $\sigma' \in \{\sigma_1, \cdots, \sigma_{j-1}\}$, users in σ' are somehow able to enter into $\mathtt{s\text{-}domain}$. If there is no domain-rule concerning σ' and q, it is not unlikely that σ' violates some constraint $\langle Q_c = \{q, q_1, \cdots, q_m\}, m+1\rangle$, where $Q_c \in \mathfrak{P}(R^s) \cup \mathfrak{P}(P^s)$. Hence, for each domain

σ' such that $\neg\exists e \in \Omega_{dr}(l, \Sigma)\ [e = \mathtt{dr}\langle\sigma', Q'\rangle]$, where $Q' \subseteq Q_c$, we need to construct more rules: add the domain-rule $\mathtt{dr}\langle\sigma', Q_c \backslash \overline{Q}(b, Q_c)\rangle$ so that at most m entities in $\{q, q_1, \cdots, q_m\}$ are available to σ'.

When a link $l = r^c \rhd b$ is used by a simple-access from σ, domain-rules may be adjusted or added to prevent domain-violations. For each rule $\mathtt{dr}\langle\sigma, Q = \{q_1, \cdots, q_n\}\rangle$ in the associated list $\mathcal{L}_\sigma$, change it to $\mathtt{dr}\langle\sigma, Q \backslash \overline{Q}(b, Q)\rangle$. For each constraint $\langle Q_c = \{q, q_1, \cdots, q_m\}, m + 1\rangle$ such that $\neg\exists e \in \mathcal{L}_\sigma\ [e = \mathtt{dr}\langle\sigma, Q\rangle]$, where $Q \subseteq Q_c$ and $b \simeq q$, add the domain-rule $\mathtt{dr}\langle\sigma, Q_c \backslash \overline{Q}(b, Q_c)\rangle$.

Let $\mathcal{A}$ be a set of cross-domain access events in `s-domain`, and $RULES$ the set of cross-domain link rules enforced in `s-domain`. By an access event, we mean that a cross-domain access request is permitted by the rules and that one and only one link l is employed by the access. We write $\mathcal{A} \nsim CON$, if the access events violate any constraint c in `s-domain`, and $\mathcal{A} \nsim RULES$ if the events break any rule. Then, the following theorem ensures that constraints are always observed.

Theorem 1. *If $\mathcal{A} \nsim CON$, then $\mathcal{A} \nsim RULES$.*

Proof. See Appendix B for the proof.

The theorem indicates that the rules are able to preserve the effects of constraints in presence of interoperation. The rules would deny all the accesses that do not comply with constraints. Note that rules might deny accesses that actually conform to constraints. Formally, $\mathcal{A} \nsim RULES \not\Rightarrow \mathcal{A} \nsim CON$. This is because no domain has knowledge of other domains' RBAC policies and the interoperability among others. Unfortunately, it is unreasonable to assume that these information are available in a decentralized environment. Hence, cross-domain link rules are useful when interoperability is needed in a decentralized multi-domain environment.

6 Related Work

Recent advances in the field have produced some approaches to the secure interoperation of multi-domain environments. Mechanisms based on multilevel security (MLS) model were proposed by Gong and Qian [7] , Dawson et al. [4] and Bonatti et al. [1]. Unfortunately, these models are not suitable for decentralized multi-domain environments because of the inherent characteristic of being static of the MLS model.

More recently, B.Shafiq et al. [14] proposed a centralized RBAC-based framework for building up secure interoperability. This framework is capable of composing a secure interoperation policy from RBAC policies of multiple domains. However, as discussed in Section 1 and Section 4.2, it relies on a trusted third-party and does not handle the problems faced in decentralized multi-domain environments.

M.Shehab et al. [15,16] presented an important decentralized framework for secure interoperation without a third party. The framework enables domains to make localized access control decisions based on users' access histories. Besides, that framework could cope with several attacks. However, few emphasis has been put on interoperability establishment. The framework assumes that there already exist role mappings manually selected by security administrators of collaborating domains. That is, there are no mechanisms for domains' security administrators to set up interoperability. This is not in line

with requirements of decentralized multi-domain environments. Second, by comparing the roles in users' access histories and roles involved in an SMER constraint, constraints were expected to be respected in the framework. However, it appears that only DMER constraints are protected in this way because users can activate multiple sessions to circumvent the verification as noted in [10]. That is to say, violations of SMER constraints by either simple-accesses or domain-accesses are not forbidden.

Du and Joshi [5] studied the problem of mapping a request for a set of permissions to a minimal set of roles in presence of hybrid hierarchies. But they did not consider the enforcement of constraints in presence of role mappings.

Other works that highlight the importance of inter-domain role mappings include [8,11]. Jin and Ahn [8] presented a role-based access management framework for secure digital information sharing in collaborative environments. The framework depends on mappings between collaborator roles and normative collaboration roles. Pan et al. [11] proposed semantic access control for interoperation based on RBAC and employed a role mapping table.

7 Conclusion

A framework for establishing secure interoperability in decentralized multi-domain environments has been presented in this paper. We presented a method of setting up interoperability which comprises role mappings and direct permission assignments. Further, we defined cross-domain link rules to capture the requirement of enforcing SMER constraints in presence of interoperation, thus observing SoD policies. Interoperation through the proposed method preserves the security and autonomy principles [7]. Cross-domain link rules, together with their enforcements and evolvements, guarantee the security of individual domains. Access control within a domain is not influenced by interoperation, and thus the autonomy principle is respected.

Acknowledgement. This work was partially supported by National Natural Science Foundation of China under Grant 60403027, Natural Science Foundation of Hubei Province under Grant 2005ABA258, Open Foundation of State Key Laboratory of Software Engineering under Grant SKLSE05-07. We thank the anonymous reviewers for their helpful comments.

References

1. Bonatti, P., Sapino, M., Subrahmanian, V.: Merging heterogeneous security orderings. In: Proceedings of the 4th European Symposium on Research in Computer Security, Rome, Italy, pp. 183–197 (September 1996)
2. Chen, H., Li, N.: Constraint generation for separation of duty. In: ACM Symposium on Access Control Models and Technologies, Lake Tahoe, California, USA, pp. 130–138. ACM Press, New York (2006)
3. Clark, D.D., Wilson, D.R.: A comparison of commercial and military computer security policies. In: IEEE Symposium on Security and Privacy, pp. 184–195. IEEE Computer Society Press, Los Alamitos (1987)

4. Dawson, S., Qian, S., Samarati, P.: Providing security and interoperation of heterogeneous systems. Distributed and Parallel Databases 8(1), 119–145 (2000)
5. Du, S., Joshi, J.B.D.: Supporting authorization query and inter-domain role mapping in presence of hybrid role hierarchy. In: ACM Symposium on Access Control Models and Technologies, pp. 228–236. ACM Press, New York (2006)
6. Ferraiolo, D.F., Sandhu, R.S., Gavrila, S.I., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur. 4(3), 224–274 (2001)
7. Gong, L., Qian, X.: Computational issues in secure interoperation. Software Engineering, IEEE Transactions on 22(1), 43–52 (1996)
8. Jin, J., Ahn, G.-J.: Role-based access management for ad-hoc collaborative sharing. In: ACM Symposium on Access Control Models and Technologies, pp. 200–209. ACM Press, New York (2006)
9. Kapadia, A., Al-Muhtadi, J., Campbell, R.H., Mickunas, M.D.: IRBAC 2000: Secure interoperability using dynamic role translation. In: Proceedings of the 1st International Conference on Internet Computing, pp. 231–238 (2000)
10. Li, N., Bizri, Z., Tripunitara, M.V.: On mutually-exclusive roles and separation of duty. In: ACM Conference on Computer and Communications Security, pp. 42–51. ACM Press, New York (2004)
11. Pan, C.-C., Mitra, P., Liu, P.: Semantic access control for information interoperation. In: ACM Symposium on Access Control Models and Technologies, pp. 237–246. ACM Press, New York (2006)
12. Piromruen, S., Joshi, J.B.D.: An RBAC framework for time constrained secure interoperation in multi-domain environments. In: the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, pp. 36–45. IEEE Computer Society Press, Los Alamitos (2005)
13. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Computer 29(2), 38–47 (1996)
14. Shafiq, B., Joshi, J., Bertino, E., Ghafoor, A.: Secure interoperation in a multidomain environment employing rbac policies. IEEE Trans. Knowl. Data Eng. 17(11), 1557–1577 (2005)
15. Shehab, M., Bertino, E., Ghafoor, A.: Secure collaboration in mediator-free environments. In: ACM Conference on Computer and Communications Security, pp. 58–67. ACM Press, New York (2005)
16. Shehab, M., Bertino, E., Ghafoor, A.: SERAT: SEcure Role mApping Technique for decentralized secure interoperability. In: ACM Symposium on Access Control Models and Technologies, pp. 159–167. ACM Press, New York (2005)

APPENDIX

A Pseudo-code for the Generation of Cross-Domain Link Rules

CREATERULES(CON, CDL)

1 $UFRules \leftarrow \emptyset$; $WC \leftarrow \emptyset$; $tmpWC \leftarrow \emptyset$
2 **for** each (SMER or SMEP) constraint $c = \langle\{q_1, q_2, \cdots, q_t\}, t\rangle \in CON$
3 **do** $WC \leftarrow \{c\}$
4 **for** $i \leftarrow 1, t$
5 **do for** each item $w = \langle\{\cdots, l_{i-1}, q_i, \cdots, q_t\}, t\rangle \in WC$
6 **do for** each link $l = (r^c \rhd b) \in CDL$
7 **do if** $b \simeq q_i$
8 **then** $tmpWC \leftarrow tmpWC \cup \{\langle\{\cdots, l, q_{i+1}, \cdots, q_t\}, t\rangle\}$
9 $WC \leftarrow tmpWC$; $tmpWC \leftarrow \emptyset$
10 $UFRules \leftarrow UFRules \cup WC$
11 $SimpleRuls \leftarrow \emptyset$; $DomainRules \leftarrow \emptyset$
12 **for** each item $I \in UFRules$
13 **do if** $I = \langle\{l_1, l_2, \cdots, l_k\}, t\rangle$
14 **then** $SimpleRules = \{\mathtt{sr}\langle\mathtt{c\text{-}domain}, \{l_1, l_2, \cdots, l_k\}\rangle\} \cup SimpleRules$
15 **elseif** $I = \langle\{l_1, l_2, \cdots, l_i, q_1, \cdots, q_j\}, t\rangle \wedge \langle\mathtt{c\text{-}domain}, \{q_1, \cdots, q_j\}, j\rangle \notin DomainRules$
16 **then** $DomainRules = \{\mathtt{dr}\langle\mathtt{c\text{-}domain}, \{q_1, \cdots, q_j\}\rangle\} \cup DomainRules$
17 **for** each link $l \in CDL$
18 **do** $flag \leftarrow$ **false**
19 **for** each simple-rule $\langle\mathtt{c\text{-}domain}, L\rangle \in SimpleRules$
20 **do if** $(l \in L)$
21 **then** $flag \leftarrow$ **true**; $BREAK$
22 **if** $flag =$ **false**
23 **then** $SimpleRules = \{\mathtt{sr}\langle\mathtt{c\text{-}domain}, \{l\}, 2\rangle\} \cup SimpleRules$

The algorithm CREATERULES takes as arguments the set CON of constraints enforced in `s-domain`, and CDL[`c-domain`]. CREATERULES has a worst-case complexity of $O(|CON| \cdot |CDL|^T)$, where T is the maximal t among all t-t SMER or SMEP constraints.

B Proof for Theorem 1

Proof. Suppose that the violated constraint is $c = \langle\{q_1, q_2, \cdots, q_t\}, t\rangle$ and each access event occupies a different q_i. It takes at most t events to violate c. There exists a link $l = r^c \rhd b$ such that $\left|\{q_i | b \simeq q_i\}\right| \geq 2$, where $1 \leq i \leq t$, if and only if it takes less than t events to violate c. However, the proof is similar when either t or less than t events constitute the violation. We assume that the violation is caused by a subset A of $\mathcal{A}$: $\{a_1, a_2, \cdots, a_t\}$, the access event a_i occurs before a_{i+1}, where $1 \leq i \leq t-1$, and, without loss of generality, that a_i corresponds to the usage of q_i. That is, each link enables users to employ only one q. Assume that there are k $(0 \leq k \leq t)$ domain-accesses in A. We proceed with discussions on k.

($k = 0$): This means that all $a_i \in A$ are simple-accesses. Suppose all the employed links by the user are $\{l_1, l_2, \cdots, l_t\}$. Since c is infracted, $l_i = r^c \rhd b$ holds, where $b \simeq q_i$. Thus, a rule $\mathtt{sr}\langle \mathtt{c\text{-}domain}, \{l_1, l_2, \cdots, l_n\}, n\rangle$ ($n \leq t$) exists in $RULES$ from algorithm CREATERULES. By the evolution strategy of simple-rules and the order of events, the rule becomes $\mathtt{sr}\langle \mathtt{c\text{-}domain}, \{l_{i+1}, \cdots, l_n\}, n - i\rangle$ after the event a_i. Finally, we have the rule $\mathtt{sr}\langle \mathtt{c\text{-}domain}, \{l_n\}, 1\rangle$ enforced after a_{n-1}. Obviously, the access event a_n runs counter to the rule. Consequently, $\mathcal{A} \nsim RULES$.

($1 \leq k < t$): There are k domain-accesses and $t - k$ simple-accesses in A. When a_1 is a domain-access, a domain-rule, $\mathtt{dr}\langle \sigma, \{q_2, \cdots, q_t\}\rangle$, would be created after a_1, according to the first part of domain-rules' evolution strategy. In the case that a_1 is a simple-access, the same domain-rule is generated from the second part of domain-rules' evolution strategy; and a simple-rule concerning l_1 would change as needed. Each time a_i occurs, the domain rule and the simple rule evolve as specified in the evolution strategies. If a_t is a domain-accesses, the domain rule should have become $\mathtt{dr}\langle \sigma, \{q_t\}\rangle$ due to the occurrences of $\{a_2, \cdots, a_{t-1}\}$. Hence, a_t would not be allowed by the domain-rule. Likewise, if a_t is a simple-access, the simple rule would be $\mathtt{sr}\langle \mathtt{c\text{-}domain}, \{l_t\}\rangle$ after a_{t-1}, where $l_t = r^c \rhd q_t$. Therefore a_t should also have been denied.

($k = t$): All events are domain-accesses. If there is already a domain-rule concerning σ and c, the rule would evolve with each occurrence of a_i, $1 \leq i \leq t - 1$. From evolution strategies, a_t could not have been permitted to occur by the final version of the rule. On the other hand, suppose that there is no rule regarding σ prior to the access a_1. However, as a_1 exploits q_1, a new domain-rule $\mathtt{dr}\langle \sigma, \{q_2, \cdots, q_t\}\rangle$ is constructed from the constraint $c = \langle \{q_1, q_2, \cdots, q_t\}, t\rangle$, according to the evolution strategy of domain-rules. Similar to the case of ($1 \leq k < t$), the rule finally becomes $\mathtt{dr}\langle \sigma, \{q_t\}\rangle$, which would refuse the access a_t. Hence, we also have $\mathcal{A} \nsim RULES$ in this case. In conclusion, if $\mathcal{A} \nsim CON$, we have $\mathcal{A} \nsim RULES$. □

Handling Dynamic Information Release

Li Jiang, Lingdi Ping, and Xuezeng Pan

College of Computer Science and Technology, Zhejiang University, China
ljiang@zju.edu.cn, ldping@cs.zju.edu.cn, xzpan@cs.zju.edu.cn

Abstract. Information flow and in particular noninterference ensure that sensitive information does not affect public information. But noninterference is too restrictive: real computing systems sometimes need to dynamically release certain amount of sensitive information. In this paper, we propose a new security property that requires the decision to perform information release have high integrity, and permits low integrity data which comes from untrusted sources to dynamically affect information release by upgrading (or endorsing) its integrity. To control such integrity upgrading, we introduce an endorsement mechanism that takes the form of a local integrity endorsing policy declaration. So the programmer can express more precise ways of endorsing, by specifying the integrity levels from which information may be endorsed. In addition, we show a new type system to enforce the security property.

Keywords: Computer security, information flow, languages, security-type systems.

1 Introduction

Language-based information-flow control is a promising approach for enforcing security properties such as data confidentiality and integrity. In this area (cf. [4] for a survey), the security policy is often formalized as noninterference properties, requiring that it be impossible to deduce any information about the program's secret inputs from its public outputs. But this requirement is too restrictive because real systems often intend to release (or *declassify*) certain amount of confidential information. To allow declassification that can be externally justified, many researchers have proposed various ideals [5-8, 11, 13-17]. See Sabelfeld and Sands [12] for a detailed survey.

However, little work has been published on the case in which some confidential information release conditionally depends on other factors. It means information can be released dynamically. A standard example is the one of a service selling electronic information, like articles in a journal for instance. The contents of an article have to be kept secret from the client until he has paid for enough money. Moreover, information-flow control must provide integrity guarantees, where important data in the system is not allowed to be affected by untrusted sources of information. The money cannot directly affect information (i.e, *the contents of journal*) declassification, because it paid by the client that may be an attacker is untrusted. We must upgrade (or *endorse*) the integrity of such untrusted data in a controlled way to give it a limited ability to affect information release. Our goal here is to provide the programmer with such a support.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 64–76, 2007.

This paper introduces *intransitive endorsement*, a security property that unifies declassification and endorsement policies. Like *robust declassification* [7, 8], our property requires that the decision to perform declassification have high integrity and thus controls *who* initiates the act of declassification. On the other hand, to support integrity endorsing, this paper provides a richer policy language which enables a more precise characterization of integrity upgrading. Each data has its local endorsing policy which is declared by a programmer. The integrity of data is upgraded according to the declared policy. So the programmer can express more precise ways of endorsing, by specifying the integrity levels from which information may be endorsed.

We also borrow some philosophy from *intransitive noninterference* [9, 10, 11], which says that flows from the high level to a *declassifier* level and flows from the *declassifier* level to the low level are admissible while a direct flow from high to low is not allowed. So it limits *where* declassification can occur in a program. In our model, endorsing the integrity of data from the low level to the high level is managed by a specific intransitive endorsing policy. This is achieved by annotating variables with policies of the form $p \rightsquigarrow p'$, which means that the policy p must be enforced on that variable, and the policy p' must be enforced on the endorsed variable after endorsement. Thus, such endorsing policies can control tightly *where* information may be endorsed relative to the integrity levels of the system.

A similar policy for introducing intransitive declassifying relations exists in [11], but with an important difference: there it is a global security policy, whereas in our case the policy is *local*. Intuitively, the global policy can not distinguish the security policies for two items of data which are at the same security level but may be used in different ways. Contrasted to the lack of precision in the global policy, the local policy is expressive enough to differentiate between data that can be endorsed and data that cannot.

The paper is organized as follows. In Section 2, we present a core language λ_{endorse} and recall two security properties *noninterference* and *robust declassification*. Section 3 defines local endorsing policy and the *intransitive endorsement* property. Section 4 gives a security type system that provably enforces intransitive endorsement and discusses the relation to declassification principles [12]. Section 5 presents a journal-selling example to illustrate how our security property can be applied to real applications. Finally, Section 6 discusses related work and Section 7 concludes.

2 Security Policy

2.1 Security Labels

In our information-flow control policy, each piece of data is annotated by a label that describes the security level of the data. Such labels form a security lattice $\mathcal{L}$, which states confidentiality and integrity policies applied to the labeled data. Confidentiality policies prevent private data from being leaked to the public, while integrity policies restrict the use of untrusted data. The use of high-confidentiality data is more restricted than that of low-confidentiality data. By contrast, the use of low-integrity data which comes from untrusted sources is more restricted than that of high-integrity data. So confidentiality and integrity can be viewed as duals [2]. For any label $\ell \in \mathcal{L}$, it is a

pair $(\mathrm{C}(\ell), \mathrm{I}(\ell))$. The notions $\mathrm{C}(\ell)$ and $\mathrm{I}(\ell)$ refer respectively to the confidentiality and integrity parts of ℓ. The bottom element of the lattice denoted $\bot$ is a most public one which has the lowest confidentiality and the highest integrity parts of $\mathcal{L}$, whereas the top element $\top$ indicates a secret reference which is most restrictive in usage and has the highest confidentiality and the lowest integrity parts of $\mathcal{L}$.

The lattice forms a pre-order written as $\sqsubseteq$. Consider a simple security lattice $\mathcal{L}$ with two levels *high* and *low* for confidentiality and integrity. For two labels ℓ_1 with level (*low*, *high*) and ℓ_2 with level (*high*, *low*), we write $\ell_1 \sqsubseteq \ell_2$, because ℓ_1 specifies less confidentiality than ℓ_2 and more integrity than ℓ_2. Moreover, we write $\mathrm{C}(\ell_1) \sqsubseteq_C \mathrm{C}(\ell_2)$ and $\mathrm{I}(\ell_1) \sqsubseteq_I \mathrm{I}(\ell_2)$ respectively for the confidentiality and integrity parts of security levels. According to this relation, if a variable x is labeled ℓ_1, the value of x can be transferred to a variable y labeled ℓ_2. The lattice join operation is $\sqcup$, which combines the restrictions on how data may be used. For example, the expression $x+y$ has label $\ell_1 \sqcup \ell_2$, which is at least as restrictive as ℓ_1 and ℓ_2: $\ell_1 \sqcup \ell_2$ = (*low*, *high*) $\sqcup$ (*high*, *low*) = (*high*, *low*).

2.2 Language Syntax and Semantics

In this section, we present a core language λ_{endorse}, a call-by-value λ-calculus extended with security types, the constructs of declassifying and endorsing. We will use it to define some security properties later. The language syntax is given in Figure 1.

$\tau ::=$	β_ℓ	Secure types
$\beta ::=$	Nat | Unit | Bool | $\tau \to \tau'$ | ref τ	Base types
$v ::=$	$x \mid n \mid unit \mid true \mid false \mid \lambda x:\tau.\, e \mid m^\tau$	Values
$e ::=$	$v \mid e\ e \mid ref^\tau e \mid !e \mid e := e \mid e;e \mid$ if e then e else $e \mid$ declassify$(e, \mathrm{C}(\ell)) \mid$ endorse$(e, \mathrm{I}(\ell))$	Expressions

Fig. 1. Syntax of λ_{endorse}

Meta variable x ranges over variables *Var,* and m ranges over memory locations. Expressions e and types β are largely standard, except for the constructs of $declassify(e, \mathrm{C}(\ell))$ and $endorse(e, \mathrm{I}(\ell))$, which are not standard language expressions and have no computational effect. Whenever a value is declassified or endorsed from one security level to another, the value is not modified in any way. The security type β_ℓ is the base type β annotated with security label ℓ. We use the notation $label(\beta_\ell) = \ell$ to obtain the security label of a type. The base types include natural numbers, unit, booleans, functions and references. A function $\lambda x:\tau.\, e$ with type $\tau \to \tau'$ has one argument x with type τ and return results of type τ'. A typed memory location m^τ can only store values of type τ.

The small-step operational semantics of λ_{endorse} are presented in Figure 2. A configuration $\langle e, M \rangle$ is a pair, consisting of an expression e and a local memory M. The memory M is a mapping from a finite set dom(M) of typed memory locations to values.

$$\frac{\langle e_1, M\rangle \to \langle e_1', M'\rangle}{\langle e_1 e_2, M\rangle \to \langle e_1' e_2, M'\rangle} \quad \frac{\langle e_2, M\rangle \to \langle e_2', M'\rangle}{\langle v_1 e_2, M\rangle \to \langle v_1 e_2', M'\rangle} \quad \frac{\langle e, M\rangle \to \langle e', M'\rangle}{\langle \mathrm{ref}^{\tau} e, M\rangle \to \langle \mathrm{ref}^{\tau} e', M'\rangle}$$

$$\langle \mathrm{ref}^{\tau} v, M\rangle \to \langle m^{\tau}, M[m^{\tau} \mapsto v]\rangle \quad \langle m^{\tau} := v, M\rangle \to \langle unit, M[m^{\tau} \mapsto v]\rangle$$

$$\frac{M(m^{\tau}) = v}{\langle !m^{\tau}, M\rangle \to \langle v, M\rangle} \quad \frac{\langle e_1, M\rangle \to \langle e_1', M'\rangle}{\langle e_1 := e_2, M\rangle \to \langle e_1' := e_2, M'\rangle} \quad \frac{\langle e_2, M\rangle \to \langle e_2', M'\rangle}{\langle v_1 := e_2, M\rangle \to \langle v_1 := e_2', M'\rangle}$$

$$\frac{\langle e, M\rangle \to \langle true, M'\rangle}{\langle \text{if } e \text{ then } e_1 \text{ else } e_2, M\rangle \to \langle e_1, M'\rangle} \quad \frac{\langle e, M\rangle \to \langle false, M'\rangle}{\langle \text{if } e \text{ then } e_1 \text{ else } e_2, M\rangle \to \langle e_2, M'\rangle}$$

$$\langle (\lambda x : \tau. e) v, M\rangle \to \langle [v/x]e, M\rangle \quad \langle unit; e, M\rangle \to \langle e, M\rangle$$

$$\langle \mathrm{declassify}(e, \mathrm{C}(\ell)), M\rangle \to \langle e, M\rangle \quad \langle \mathrm{endorse}(e, \mathrm{I}(\ell)), M\rangle \to \langle e, M\rangle$$

Fig. 2. Operational semantics of $\lambda_{\mathrm{endorse}}$

The notation $M(m^{\tau})$ denotes the value of location m^{τ} in M. The operation of updating the value of a location in the memory is denoted by $M[m^{\tau} \mapsto v]$. We use the notation $[v/x]e$ to indicate capture-avoiding substitution of value v for variable x in expression e.

The small-step semantics are given by transitions between configurations. The general form of a transition is either $\langle e, M\rangle \to \langle\langle\rangle, M'\rangle$, which means termination with the final memory M' denoted by $\langle e, M\rangle \Downarrow M'$, or $\langle e, M\rangle \to \langle e', M'\rangle$. Given a configuration $\langle e_1, M_1\rangle$, a trace of $\langle e_1, M_1\rangle$ is a sequence of configurations $\langle e_1, M_1\rangle ... \langle e_n, M_n\rangle$, such that $\langle e_{i-1}, M_{i-1}\rangle \to \langle e_i, M_i\rangle$ for all $i \in 2..n$. We also define the memory trace of $\langle e_1, M_1\rangle$ to be the sequence $T_m(\langle e_1, M_1\rangle) = M_1 ... M_n$.

2.3 Security Specification

This section recalls two security policies *noninterference* and *robust declassification*. First, it is necessary to define what the attacker can observe. The attacker at security level ℓ can observe the value at security level ℓ' if and only if $\ell' \sqsubseteq \ell$. We write $M_1 =_{\ell} M_2$ if memories M_1 and M_2 agree on locations whose labels are $\sqsubseteq \ell$, which is formalized as $\forall m^{\beta_{\ell'}}.(\ell' \sqsubseteq \ell) \Rightarrow M_1(m^{\beta_{\ell'}}) = M_2(m^{\beta_{\ell'}})$. We call M_1 and M_2 are ℓ-equal.

To define our security properties we need to consider how to characterize the power of possible attackers. We use trace equivalence relation $\approx_{\ell}$ to model attacker knowledge. The equivalence relation corresponds to an ability to distinguish different traces. Given two memory traces $T_m(\langle e_1, M_1\rangle)$ and $T_m(\langle e_2, M_2\rangle)$ for configurations $\langle e_1, M_1\rangle$ and $\langle e_2, M_2\rangle$, $T_m(\langle e_1, M_1\rangle) \approx_{\ell} T_m(\langle e_2, M_2\rangle)$ if $M_1 =_{\ell} M_2$ and the subsequences of memories resulting from e_1 and e_2 are also ℓ-equal. For this semantic model, we formalize noninterference in the following definition.

Definition 2.1 (Noninterference). Expression e is noninterference if

$$\forall \ell, M_1, M_2.\ M_1 =_{\ell} M_2 \Rightarrow T_m(\langle e, M_1\rangle) \approx_{\ell} T_m(\langle e, M_2\rangle)$$

Intuitively, noninterference says that if two input memories are indistinguishable for an attacker at a certain level, then the behavior of the program on these memories is also indistinguishable at that level. Yet many practical programs do release information. For example, the encryption function of a cryptographic library takes confidential data and makes it public. Consequently, realistic systems need a means of declassification. Unfortunately the noninterference property does not hold in the presence of declassification. So the cryptosystem violating noninterference would be rejected by the type systems of most current security-typed languages.

We use the construct $declassify(e, \mathrm{C}(\ell))$ to downgrade the confidentiality of the result of the expression e to $\mathrm{C}(\ell)$. Unregulated use of declassifying can easily result in unexpected release of confidential information. To control declassification, Zdancewic and Myers have proposed a security condition called *robust declassification* [7, 8], which requires that the decision to declassify must be at high integrity level. For example, in a two-level setting, a program exhibits robust declassification if decisions about downgrading high-confidentiality data cannot be affected by low-integrity data which may be controlled by an attacker. In other words, low-integrity data cannot influence what data is declassified.

Definition 2.2 (Robust Declassification). Suppose that an attacker is at security level ℓ_a. Then for any variable x with type τ_x in expression e, if $\mathrm{I}(\ell_a) \sqsubseteq_I \mathrm{I}(label(\tau_x))$,

$$\forall M_1, M_2, v_1, v_2.\ T_m(\langle [v_1 / x]e, M_1\rangle) \approx_{\ell_a} T_m(\langle [v_1 / x]e, M_2\rangle)$$
$$\Rightarrow T_m(\langle [v_2 / x]e, M_1\rangle) \approx_{\ell_a} T_m(\langle [v_2 / x]e, M_2\rangle).$$

Intuitively, attacker-controlled computation is not allowed to increase observations about secrets by causing misuse of the declassification mechanism. So the above definition requires that any change of low-integrity variables in the expression cannot increase what the attacker can observe. According to this, the attacker cannot control information release. However, robust declassification does not support dynamical information release, such as a journal-selling service mentioned in Section 1.

3 Intransitive Endorsement

3.1 Local Endorsing Policy

To fix the aforementioned weakness of *robust declassification*, we extend each data to be annotated with not only a security label ℓ but also an intransitive endorsing policy p. We also use the construct $endorse(e, \mathrm{I}(\ell))$ and add an endorsing relation $\rightsquigarrow$ between integrity levels of a given lattice structure for exceptions to the standard flow relation $\sqsubseteq$. Then we define policy $p ::= I \rightsquigarrow p$, where I refers to the integrity part of a security level $\ell \in \mathcal{L}$. Such a policy specifies a sequence of integrity levels through which a data item may be endorsed. The syntax for policies is given in Figure 3.

In the grammar, endorsing policies range over elements of an endorsing policy lattice $\mathcal{L}_p$ with a top element $\top_p$ and a bottom element $\bot_p$. The lattice order and join are written $\leq_p$ and $\sqcup_p$ respectively. The ordering $\leq_p$ is defined as the least relation over endorsing policies such that $p \leq_p p'$ if policy p' is at least as restrictive as policy p.

For two policies $p ::= I \rightsquigarrow p_1$ and $p' ::= I' \rightsquigarrow p'_1$, we write $p \leq_p p'$ if the integrity level I' is lower than I written as $I \sqsubseteq_I I'$ and the policy p'_1 is at least as restrictive as policy p_1. Whenever data labeled with p can be endorsed to p_1, data labeled with p' can also be endorsed to p'_1. Roughly speaking, p has more power to affect endorsement than p'.

$\ell \in \mathcal{L}$	Lattice element
$I ::= \mathrm{I}(\ell)$	The integrity part of lattice-level
$p ::=$	Policies
I	Integrity-level policy
$I \rightsquigarrow p'$	Endorsing policy

Fig. 3. Syntax for endorsing policies

The join operation $\sqcup_p$ is useful to compute the least upper bound on the endorsing policy of an expression that combines sub-expressions with different endorsing policies. We present the operation rules and axioms of $\leq_p$ and $\sqcup_p$ in Figure 4, using ϕ to denote the null endorsing policy which has no power to affect endorsement.

$$\frac{}{I \leq_p I \rightsquigarrow I \cdots \rightsquigarrow I} \qquad \frac{}{I \rightsquigarrow I \cdots \rightsquigarrow I \leq_p I} \qquad \frac{}{p \leq_p \phi}$$

$$\frac{I \sqsubseteq_I I'}{I \leq_p I'} \qquad \frac{I \sqsubseteq_I I' \quad p \leq_p p'}{I \rightsquigarrow p \leq_p I' \rightsquigarrow p'} \qquad \frac{p \leq_p p' \quad p' \leq_p p''}{p \leq_p p''}$$

$$\frac{I \sqcup I' = I'' \quad p \sqcup_p p' = p''}{(I \rightsquigarrow p) \sqcup_p (I' \rightsquigarrow p') = (I'' \rightsquigarrow p'')} \qquad \frac{}{\phi \sqcup_p p = \phi}$$

Fig. 4. Ordering $\leq_p$ and join operation $\sqcup_p$ for endorsing policies

Because endorsement to the same integrity level is harmless, we have that the integrity-level policy I is equivalent to the policy $I \rightsquigarrow I \cdots \rightsquigarrow I$. Based on this truth, we can compare and join two policies which are not the same for the length of the sequence of integrity levels. For example, give two policies $p ::= I_1 \rightsquigarrow I_2 \rightsquigarrow I_3$ and $p' ::= I'_1 \rightsquigarrow I'_2$ where $I_1 \sqsubseteq_I I'_1$, $I'_2 \sqsubseteq_I I_2$ and $I_3 \sqsubseteq_I I'_2$. Since $I'_2 \equiv I'_2 \rightsquigarrow I'_2$, we have $p' \equiv I'_1 \rightsquigarrow I'_2 \rightsquigarrow I'_2$. Due to this, the endorsing policy $p \sqcup_p p' = (I_1 \sqcup_p I'_1) \rightsquigarrow (I_2 \sqcup_p I'_2) \rightsquigarrow (I_3 \sqcup_p I'_2) = I'_1 \rightsquigarrow I_2 \rightsquigarrow I'_2$. Based on the definitions of $\leq_p$ and $\sqcup_p$, the top element of the endorsing lattice $\top_p$ is ϕ and the bottom element $\bot_p$ is $\mathrm{I}(\bot)$ equivalent to the infinitely long policy $\mathrm{I}(\bot) \rightsquigarrow \mathrm{I}(\bot) \rightsquigarrow \mathrm{I}(\bot) \ldots$, where $\bot$ is the bottom element of the security lattice $\mathcal{L}$ and $\mathrm{I}(\bot)$ is the integrity part of $\bot$.

3.2 Intransitive Endorsement Security Policy

By using the construct $endorse(e, \mathrm{I}(\ell))$, untrusted code is provided with the limited ability to affect declassification. Our goal here is to control tightly where endorsing can occur rather than permit arbitrary endorsing. The partial order relation between levels $\mathrm{I}(\ell_y) \sqsubseteq_I \mathrm{I}(\ell_x)$ means that data x is less trustworthy than data y and its integrity

may be endorsed to $\mathrm{I}(\ell_y)$ using the expression $endorse(x, \mathrm{I}(\ell_y))$. Although $endorse(x, \mathrm{I}(\ell_y))$ violates the information flow ordering $\mathrm{I}(\ell_y) \sqsubseteq_I \mathrm{I}(\ell_x)$, it remains secure as long as the expression complies with the endorsing policy $\mathrm{I}(\ell_x) \rightsquigarrow \mathrm{I}(\ell_y)$ which is enforced on x locally.

From a pragmatic point of view, each data has its local intransitive endorsing policy and endorsement is governed by these local policies. The policy describes where information may flow relatively to the integrity level of the system. Furthermore, the endorsement constructs in the code can be thought of as the sole place where low-integrity data is given the ability to affect declassification. Such constructs describes where physically in the code low-integrity data may be endorsed.

In order to cope with endorsing, we need to modify the robust declassification policy. The modified policy can be used to enforce two interesting security properties: robust declassification (if the construct *endorse* is not used) and intransitive endorsement (even if it is). First, an expression satisfies robust declassification if endorsing does not occur in the expression. Second, endorsing obeys the endorsing policies of the form $p ::= I \rightsquigarrow p$.

Note that there exist some special cases where declassifying information is not affected by endorsing low-integrity data although endorsing expressions exist. For example, given a variable x annotated with endorsing policy $\mathrm{I}(\ell_x) \rightsquigarrow I_1 \rightsquigarrow I_2$, we use $endorse(endorse(x, I_1), I_2)$ indicating an endorsing sequence $\mathrm{I}(\ell_x) \rightsquigarrow I_1 \rightsquigarrow I_2$ to upgrade the integrity of a variable x to the integrity level I_1 first and then to the integrity level I_2 . But $endorse(x, I_3)$ indicating an endorsing sequence $\mathrm{I}(\ell_x) \rightsquigarrow I_3$ is not allowed because $\mathrm{I}(\ell_x) \rightsquigarrow I_3$ does not belong to the endorsing policy of x. Furthermore, endorsing the integrity of x to I_2 may only affect the decision to declassify whose integrity is at or below I_2 . To be more precise, we have the following definition:

Definition 3.1 (Intransitive Endorsement). Suppose that an attacker is at security level ℓ_a . Then for an expression e with free variable x which has type τ_x such that $\mathrm{I}(\ell_a) \sqsubseteq_I \mathrm{I}(label(\tau_x))$ and is annotated with an endorsing policy $\mathrm{I}(\ell_0) \rightsquigarrow \mathrm{I}(\ell_1) \rightsquigarrow \cdots \rightsquigarrow \mathrm{I}(\ell_k)$, if there exists an endorsing sequence $\mathrm{I}(label(\tau_x)) \rightsquigarrow \mathrm{I}(\ell'_1) \rightsquigarrow \cdots \rightsquigarrow \mathrm{I}(\ell'_j)$ for x in e, then

$$\forall i \in \{1, \ldots, j\}, M_1, M_2, v_1, v_2.$$
$$\big[(\neg valid(\mathrm{I}(label(\tau_x))) \vee (valid(\mathrm{I}(label(\tau_x)) \cdots \mathrm{I}(\ell'_i)) \wedge \mathrm{I}(\ell_a) \sqsubseteq_I \mathrm{I}(\ell'_i))) \wedge$$
$$(T_m(\langle [v_1 / x]e, M_1 \rangle) \approx_{\ell_a} T_m(\langle [v_1 / x]e, M_2 \rangle)) \big]$$
$$\Rightarrow T_m(\langle [v_2 / x]e, M_1 \rangle) \approx_{\ell_a} T_m(\langle [v_2 / x]e, M_2 \rangle),$$

and if no endorsing exists in C, then

$$\forall M_1, M_2, v_1, v_2.\ T_m(\langle [v_1 / x]e, M_1 \rangle) \approx_{\ell_a} T_m(\langle [v_1 / x]e, M_2 \rangle)$$
$$\Rightarrow T_m(\langle [v_2 / x]e, M_1 \rangle) \approx_{\ell_a} T_m(\langle [v_2 / x]e, M_2 \rangle).$$

For an endorsing sequence $\mathrm{I}(label(\tau_x)) \rightsquigarrow \mathrm{I}(\ell'_1) \rightsquigarrow \cdots \rightsquigarrow \mathrm{I}(\ell'_j)$, we use $valid(\mathrm{I}(label(\tau_x)) \cdots \mathrm{I}(\ell'_i))$ to check whether each item in the sequence is permitted by the endorsing policy. It will return true only if $\mathrm{I}(label(\tau_x))$,…, $\mathrm{I}(\ell'_i)$ are all permitted. For example, $valid(\mathrm{I}(label(\tau_x)))$ and $valid(\mathrm{I}(label(\tau_x))\mathrm{I}(\ell'_1))$ will return true only if

$\mathrm{I}(label(\tau_x)) = \mathrm{I}(\ell_0)$ and $\mathrm{I}(\ell'_1) = \mathrm{I}(\ell_1)$, while $valid(\mathrm{I}(label(\tau_x))\mathrm{I}(\ell'_1)\mathrm{I}(\ell'_2))$ will return false if $\mathrm{I}(\ell'_2) \neq \mathrm{I}(\ell_2)$. In the above definition, we consider two special cases where a command must satisfy *robust declassification* although endorsing expressions exist:

1. If an endorsing policy $\mathrm{I}(\ell_0) \rightsquigarrow \mathrm{I}(\ell_1) \rightsquigarrow \cdots \rightsquigarrow \mathrm{I}(\ell_k)$ is enforce on the variable x with type τ_x such that $\mathrm{I}(label(\tau_x)) \neq \mathrm{I}(\ell_0)$, $\neg valid(\mathrm{I}(label(\tau_x)))$ returns true, which means that x cannot be endorsed to any integrity level.
2. If the endorsing sequence $\mathrm{I}(label(\tau_x)) \rightsquigarrow \mathrm{I}(\ell'_1) \rightsquigarrow \cdots \rightsquigarrow \mathrm{I}(\ell'_i)$ is permitted by the endorsing policy, x with type τ_x at security level $(\mathrm{C}(label(\tau_x)), \mathrm{I}(label(\tau_x)))$ can be endorsed to $\mathrm{I}(\ell'_i)$. After this endorsement, x is at level $(\mathrm{C}(label(\tau_x)), \mathrm{I}(\ell'_i))$. But its integrity is still lower than attackers if $\mathrm{I}(\ell_a) \sqsubseteq_I \mathrm{I}(\ell'_i)$. It intuitively means that x remains untrusted and cannot affect the decision of declassification.

4 Type-Based Security Analysis

In this section, we first present a security-type system to illustrate how to enforce intransitive endorsement. The type system specifies the type rules of λ_{endorse}. Then we establish a type soundness theorem stating that all well-typed expressions satisfy intransitive endorsement. Typing rules for λ_{endorse} are presented in Figure 5.

The *security environment* Γ describes the security type of each variable. A collection of variables declared in Γ is the domain of Γ and the security types are the range of Γ. To introduce the concept of endorsing policies into the type system, we extend the security type such that it consists of a base type β annotated with a security level and an endorsing policy. For example, $\Gamma(x)=\beta_{(\ell_x, p_x)}$ means that variable x in Γ has the type $\beta_{(\ell_x, p_x)}$. The security lattice $\mathcal{L}$, security environment and endorsing policy lattice $\mathcal{L}_p$ together constitute a *security policy*, specifying that information flow from a variable x to a variable y is allowed only if $\ell_x \sqsubseteq \ell_y$ and $p_x \leq p_y$.

Typing for expressions has the form $\Gamma[\mathrm{pc}] \vdash e : \beta_{(\ell, p)}$ meaning that an expression e has security type $\beta_{(\ell, p)}$ and is well-typed under Γ and a *program counter pc*. We use $\Gamma[\mathrm{pc}] \vdash e$ as the abbreviation for it. This assertion is known as a typing judgment. The judgment $\Gamma[\mathrm{pc}] \vdash \Diamond$ means that Γ is a well-formed environment under the context *pc*. Program counters range over security levels and help track information flow due to control flow. So they do not contain endorsing policies because local endorsing policies are only used in endorsing expressions to indicate where the integrity of data can be upgraded.

Our type system uses the constructs of *declassify* and *endorse* to change the security level of information without affecting the execution of the command. The security policy definition is with respect to the attacker with a confidentiality level $\mathrm{C}(\ell_a)$ and an integrity level $\mathrm{I}(\ell_a)$. H_I is the set of the integrity levels that are higher than $\mathrm{I}(\ell_a)$. In declassify construct, $\mathrm{I}(\ell), \mathrm{I}(\mathrm{pc}) \in H_I$ means that the data which may be declassified and the security context *pc* must be in higher integrity level than the given attacker. This typing rule is proposed by Myers [8].

In endorsing construct, an expression e at integrity level $\mathrm{I}(\ell)$ to $\mathrm{I}(\ell')$ is allowed due to its local endorsing policy $\mathrm{I}(\ell) \rightsquigarrow \mathrm{I}(\ell') \rightsquigarrow p$. After endorsement, the endorsing policy $\mathrm{I}(\ell') \rightsquigarrow p$ is enforced on e and the confidentiality of e is not affected. The designated typing rule allows us to tightly restrict where endorsing can occur.

$$\Gamma\,[\mathrm{pc}]\vdash n:\mathbf{Nat}_{(\ell,\,p)} \quad \Gamma\,[\mathrm{pc}]\vdash \mathit{unit}:\mathbf{Unit}_{(\ell,\,p)} \quad \Gamma\,[\mathrm{pc}]\vdash \mathit{true}:\mathbf{Bool}_{(\ell,\,p)}$$

$$\Gamma\,[\mathrm{pc}]\vdash \mathit{false}:\mathbf{Bool}_{(\ell,\,p)} \quad \Gamma\,[\mathrm{pc}]\vdash m^{\tau}:(\mathbf{ref}\,\tau)_{(\ell,\,p)} \quad \frac{\Gamma', x:\tau, \Gamma''\,[\mathrm{pc}]\vdash \Diamond}{\Gamma', x:\tau, \Gamma''\,[\mathrm{pc}]\vdash x:\tau}$$

$$\frac{\Gamma\,[\mathrm{pc}]\vdash e_1:(\mathbf{ref}\ \beta_{(\ell,\,p)})_{(\ell',\,p')} \quad \Gamma\,[\mathrm{pc}]\vdash e_2:\beta_{(\ell,\,p)} \quad \mathrm{pc}\sqcup\ell'\sqsubseteq\ell \quad p'\le p}{\Gamma\,[\mathrm{pc}]\vdash e_1:=e_2:\mathbf{Unit}_{(\ell',\,p')}}$$

$$\frac{\Gamma, x:\tau, [\mathrm{pc}]\vdash e:\tau'}{\Gamma\,[\mathrm{pc}]\vdash \lambda x{:}\tau.\ e:(\tau\to\tau')_{(\ell,\,p)}} \qquad \frac{\Gamma\,[\mathrm{pc}]\vdash \lambda x{:}\tau.\ e_1:(\tau\to\beta_{(\ell,\,p)})_{(\ell',\,p')} \quad \Gamma\,[\mathrm{pc}]\vdash e_2:\tau}{\Gamma\,[\mathrm{pc}]\vdash e_1e_2:\beta_{(\ell\sqcup\ell',\,p\sqcup p')}}$$

$$\frac{\Gamma\,[\mathrm{pc}]\vdash e_1:\mathbf{Unit}_{(\ell,\,p)} \quad \Gamma\,[\mathrm{pc}]\vdash e_2:\tau}{\Gamma\,[\mathrm{pc}]\vdash e_1;e_2:\tau} \qquad \frac{\Gamma\,[\mathrm{pc}]\vdash e:\mathbf{Bool}_{(\ell,\,p)}\ \Gamma\,[\mathrm{pc}\sqcup\ell]\vdash e_1\ \Gamma\,[\mathrm{pc}\sqcup\ell]\vdash e_2}{\Gamma\,[\mathrm{pc}]\vdash \mathrm{if}\ e\ \mathrm{then}\ e_1\ \mathrm{else}\ e_2}$$

$$\frac{\Gamma\,[\mathrm{pc}]\vdash e:(\mathbf{ref}\,\beta_{(\ell,\,p)})_{(\ell',\,p')}}{\Gamma\,[\mathrm{pc}]\vdash !e:\beta_{(\ell\sqcup\ell',\,p\sqcup p')}} \qquad \frac{\Gamma\,[\mathrm{pc}]\vdash e:\beta_{(\ell,\,p)} \quad \mathrm{I}(\ell),\mathrm{I}(\mathrm{pc})\in H_I}{\Gamma\,[\mathrm{pc}]\vdash \mathit{declassify}(e,\mathrm{C}(\ell')):\beta_{((\mathrm{C}(\ell'),\,\mathrm{I}(\ell)),\,p)}}$$

$$\frac{\Gamma\,[\mathrm{pc}]\vdash e:\beta_{(\ell,\,p)} \quad \mathrm{pc}\sqsubseteq\ell}{\Gamma\,[\mathrm{pc}]\vdash \mathrm{ref}^{\beta(\ell,\,p)}e:(\mathbf{ref}\,\beta_{(\ell,\,p)})_{(\ell',\,p')}} \qquad \frac{\Gamma\,[\mathrm{pc}]\vdash e:\beta_{(\ell,\ \mathrm{I}(\ell)\rightsquigarrow\mathrm{I}(\ell')\rightsquigarrow\,p)}}{\Gamma\,[\mathrm{pc}]\vdash \mathit{endorse}(e,\mathrm{I}(\ell')):\beta_{((\mathrm{C}(\ell),\,\mathrm{I}(\ell')),\,\mathrm{I}(\ell')\rightsquigarrow\,p)}}$$

Fig. 5. Typing rules

Three security properties defined in section 2 and 3 —*noninterference*, *robust declassification*, *intransitive endorsement*—can be enforced by our type system. If a well-typed expression does not have occurrence of declassification and endorsement, it satisfies noninterference. Robust declassification is also satisfied if no endorsement occurs. We formalize these as the two following theorems:

Theorem 4.1. If $\Gamma\,[\mathrm{pc}]\vdash e$ and no declassification and endorsement occur in e, then e satisfies noninterference.

Theorem 4.2. If $\Gamma\,[\mathrm{pc}]\vdash e$ and no endorsement occurs in e, then e satisfies robust declassification.

Moreover, we concern about the case in which endorsement is used. The type rule of intransitive-endorse allows the programmer to tightly restrict where endorsement can occur. The type soundness result guarantees that all well-typed expressions satisfy intransitive endorsement.

Theorem 4.3. If $\Gamma\,[\mathrm{pc}]\vdash e$, then e satisfies intransitive endorsement.

The proof of these three theorems is sketched in the appendix.

In paper [12], Sabelfeld and Sands propose four principles for declassification policies. We now discuss the relation between our property and declassification principles.

The first principle is *semantic consistency*, which states that the security of a program is invariant under semantics-preserving transformations of declassification-free subprograms. Our novel security policy satisfies this principle. Because the policy is

built on top of a notion of the attacker's view which is defined by low-level indistinguishability of traces up to high-stuttering, a modification of a declassification-free fragment of a program in a semantics preserving way does not make a difference from the security definition's point of view.

The second principle is *conservativity*, which states that security for programs with no declassification is equivalent to noninterference. This principle holds according to Theorem 4.1. A well-typed expression satisfies noninterference not only if no declassification operations occur in a program, but also if no endorsement operations occur.

The third principle is *monotonicity of release*, which states that adding further declassifications to a secure program cannot render it insecure. This principle fails because the definition of intransitive endorsement assumes that the attacker can observe the fact that an endorsement operation is being performed, regardless of its content. Consider the program $h' := 0$; if $h > 0$ then $l := h'$ else $l := 0$ which is secure. On the other hand, adding a declassification annotation and an endorsement one, as in:

$$\text{decalssify}(h' := 0, \text{L})\,;\text{if } h > 0 \text{ then endorse}(l := h', \text{H}) \text{ else } l := 0;$$

renders the program insecure, because an attacker observing the presence or absence of an endorsement action can learn whether h is greater than 0 or not.

The fourth principle is *non-occlusion*, which states that the presence of a declassification operation cannot mask other covert information leaks. This principle is supported by intransitive endorsement. Although the attacker may affect declassification, it must obey endorsing policy enforced on data locally. Furthermore, endorsing policy annotations in code may not mask other unrelated information leaks.

5 An Example

This section we give an example of a program to illustrate some important features in the language. Consider a purchase service selling electronic journals. The contents of journals have to be kept secret from the client until he has paid enough money for it. We show the procedure which dynamically declassifies information depending on the money paid by the client as follows, assuming the addition of greater-than-or-equal-to test >= for integers, and the use of *let* $x{:}\tau{=}e'$ *in* e as a syntactic sugar for $(\lambda x{:}\tau.\, e)e'$.

```
λ paid: Nat((L,L),L⇝H) . λ journal.contents: Nat((H,H),φ) .
λ journal.price: Nat((L,H),φ) .
    Let x: Nat((L,H),φ)=endorse(paid,H) in
       If x>=journal.price then
           declassify(journal.contents,L) else unit
```

For simplicity of presentation, we only consider two security levels—H and L. The price of journal is at level (L,H), a pair consisting of the low confidentiality part and the high integrity part, respectively, with a null endorsing policy. The procedure tests whether the client has paid enough money to buy the journal. If so, the contents of journal are declassified to have label (L,H). The decision to perform the declassification is based in part on the variable *paid*. If the integrity level of *paid* is not endorsed to H, the *declassify* expression is rejected by the type system because the security context is at low integrity level.

Thus, it is necessary to use the endorsing expression to give *paid* the ability to affect declassification. The value of *paid* comes from a client who may be an attacker. So *paid* is initially labeled with low integrity level. Endorsing *paid* to H is permitted only if the local endorsing policy of *paid* declared by the programmer permits. The above program can be proved to be well-typed and satisfy the *intransitive endorsement* property. From this point of view, the endorsement mechanism is not misused to lead to undesired declassification.

6 Related Work

Information release (or declassification) is the key challenge for language-based information flow security and much recent and ongoing work concerns policies for declassification. The recent work [12] by Sabelfeld and Sands contains an exhaustive survey on the literature regarding the subject of declassification. They have classified mechanisms for declassification along several dimensions: "Who" is authorized to perform declassification, "Where" declassification is allowed to take place, "What" information is released, and "When" declassification should be allowed to happen.

As mentioned in Section 1, popular approaches to policies along the *who* and *where* dimensions are respectively based on *robust declassification* [7, 8] and *intransitive noninterference* [9, 10, 11].

A natural starting point along the *what* dimension is the delimited release proposed by Sabelfeld and Myers [13]. This policy restricts the use of declassification so that cannot be exploited by laundering attacks. The idea is that declassification is acceptable provided that the program does not modify data if that could influence the value of declassified expressions.

Most recently, Mantel and Reinhard [16] suggested three definitions ($WHAT_1$, $WHAT_2$, and *WHERE*) for controlling the dimensions *what* and *where*. But there definitions consider the dimensions in separation. In comparison to such definitions, Askarov and Sabelfeld [17] proposed *localized delimited release* which combines the *what* and *where* dimensions. Their approach captures the location of release by instructing the semantics with the set of released expressions and extending the definition of *delimited release* with this information.

Chong and Myers introduced *noninterference* "*until*" [14] that expresses the sequence of levels through which a value can be declassified, provided some conditions are satisfied. This approach controls *when* information can be released.

7 Conclusion

In this paper, we have presented intransitive endorsement, a security characterization that combines the *who* and *where* dimensions of information release. This combination is reassured by the semantic consistency, conservativity, and non-occlusion principles of declassification [12]. For controlling the *who* dimension, our security property restricts that declassification can only occur in the high integrity context. For controlling *where*, our property guarantees that endorsing the integrity of data is controlled by the local endorsing policy of each data. As a result, we are able to upgrade the integrity of untrusted code in a controlled way to grant the code an ability to affect information release.

References

1. Heintze, N., Riecke, J.G.: The SLam Calculus: Programming with Secrecy and Integrity. In: Proc. ACM Symp. on Principles of Programming Languages, pp. 365–377. ACM Press, New York (1998)
2. Biba, K.J.: Integrity Considerations for Secure Computer Systems. Technical Report ESD-TR-76-372, USAF Electronic Systems Division, Bedford, MA (1977)
3. Zdancewic, S., Myers, A.C.: Secure Information Flow via Linear Continuations. Higher Order and Symbolic Computation, 15(2/3) (2002)
4. Sabelfeld, A., Myers, A.C.: Language-Based Information-Flow Security. IEEE Journal on Selected Areas in Communications 21(1), 5–19 (2003)
5. Giacobazzi, R., Mastroeni, I.: Abstract Noninterference: Parameterizing Noninterference by Abstract Interpretation. In: Proc. ACM Symp. on Principles of Programming Languages, pp. 186–197. ACM Press, New York (2004)
6. Giacobazzi, R., Mastroeni, I.: Adjoining Declassification and Attack Models by Abstract Interpretation. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 295–310. Springer, Heidelberg (2005)
7. Zdancewic, S.: A Type System for Robust Declassification. In: Proc. the 19th Conference on the Mathematical Foundations of Programming Semantics. Electronic Notes in Theoretical Computer Science (2003)
8. Myers, A.C., Sabelfeld, A., Zdancewic, S.: Enforcing Robust Declassification. In: Proc. IEEE Computer Security Foundations Workshop, pp. 172–186. IEEE Computer Society Press, Los Alamitos (2004)
9. Rushby, J.M.: Noninterference, Transitivity, and Channel-control Security Policies. Technical Report CSL-92-02, SRI International (1992)
10. Roscoe, A.W., Goldsmith, M.H.: What is intransitive noninterference? In: Proc. IEEE Computer Security Foundations Workshop, pp. 228–238. IEEE Computer Society Press, Los Alamitos (1999)
11. Mantel, H., Sands, D.: Controlled Downgrading Based on Intransitive Noninterference. In: Chin, W.-N. (ed.) APLAS 2004. LNCS, vol. 3302, pp. 129–145. Springer, Heidelberg (2004)
12. Sabelfeld, A., Sands, D.: Dimensions and Principles of Declassification. In: Proc. IEEE Computer Security Foundations Workshop, pp. 255–269. IEEE Computer Society Press, Los Alamitos (2005)
13. Sabelfeld, A., Myers, A.C.: A Model for Delimited Information Release. In: Futatsugi, K., Mizoguchi, F., Yonezaki, N. (eds.) ISSS 2003. LNCS, vol. 3233, pp. 174–191. Springer, Heidelberg (2004)
14. Chong, S., Myers, A.C.: Security Policies for Downgrading. In: Proc. 11th ACM Conference on Computer and Communications Security, pp. 198–209. ACM Press, New York (2004)
15. Askarov, A., Sabelfeld, A.: Gradual Release: Unifying Declassification, Encryption and Key Release Policies. In: Proc. IEEE Symp. on Security and Privacy, IEEE Computer Society Press, Los Alamitos (2007)
16. Mantel, H., Reinhard, A.: Controlling the What and Where of Declassification in Language-based Security. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 141–156. Springer, Heidelberg (2007)
17. Askarov, A., Sabelfeld, A.: Localized Delimited Release: Combining the What and Where Dimensions of Information Release. In: Proc. the 2007 workshop on Programming languages and analysis for security (2007)

Appendix

Proof of Theorem 4.1. We sketch a proof by induction on the typing derivation for an expression e. With the exception of the expression $e_1 := e_2$, the theorem is proved with a straightforward induction on the evaluation of other expressions, because our type system is similar to those previously proposed [1, 3], except for the addition of rules for declassifying and endorsing expressions. The induction on $e_1 := e_2$ is not standard, because it contains $p \leq p'$ in its preconditions. So we only consider this case.

Suppose that for an expression $e_1 := e_2$ and two memories M_1 and M_2 we have $M_1 =_{\ell_a} M_2$ for some security level ℓ_a. By the typing rule of $e_1 := e_2$, e_1 denotes a typed memory location m^{τ} with the type $(\text{ref } \beta_{(\ell, p)})_{(\ell', p')}$ and e_2 has the type $\beta_{(\ell, p)}$. By induction on the derivation of $e_1 := e_2$, $\langle e_1 := e_2, M_1\rangle \rightarrow \langle m_1^{\tau} := v, M_1\rangle \rightarrow \langle unit, M_1[m_1^{\tau} \mapsto v]\rangle$ and $\langle e_1 := e_2, M_2\rangle \rightarrow \langle m_2^{\tau} := v', M_2\rangle \rightarrow \langle unit, M_2[m_2^{\tau} \mapsto v']\rangle$. To prove the theorem, we need to show $M_1[m_1^{\tau} \mapsto v] =_{\ell_a} M_2[m_2^{\tau} \mapsto v']$. In case $\ell \not\sqsubseteq \ell_a$, an attacker who is below or at ℓ_a can not observe values of memory locations whose security levels are above ℓ_a. Although the values of e_1 and e_2 under M_1 and M_2 may be different, the attacker cannot detect this difference. In case $\ell \sqsubseteq \ell_a$, which implies $\ell' \sqsubseteq \ell \sqsubseteq \ell_a$, we obtain that $m_1^{\tau} = m_2^{\tau}$ and $v = v'$ due to $M_1 =_{\ell_a} M_2$. Note that the endorsing policies of e_1 and e_2 have no effect on the computation, because no declassification and endorsement occur in the expression $e_1 := e_2$. Hence, we can prove that $M_1[m_1^{\tau} \mapsto v] =_{\ell_a} M_2[m_2^{\tau} \mapsto v']$.

Proof of Theorem 4.2. Paper [8] has translated the robust declassification property into the language-based setting. Their approach is based on a simple sequential language while we introduce robust declassification based on a call-by-value λ-calculus. But it is not a fundamental limitation of our presentation. Paper [8] has proved a well-typed command in the sequential language satisfies robust declassification, and it is also easy to prove the theorem based on λ_{endorse}.

Proof of Theorem 4.3. Suppose there is an attacker at security level ℓ_a. In general, an attacker may modify data whose integrity is at or below $\mathrm{I}(\ell_a)$. Here we only consider the case in which endorsing occurs in the expression e. If no endorsement occurs, e satisfies robust declassification by Theorem 4.2. When endorsement is used, suppose e contains exactly n declassify expressions $declassify(e_1, \mathrm{C}(\ell_1)), ..., declassify(e_n, \mathrm{C}(\ell_n))$, and m endorse expressions $endorse(e'_1, \mathrm{I}(\ell'_1)), ..., endorse(e'_m, \mathrm{I}(\ell'_m))$. It is either the case that the endorse expression $endorse(e'_i, \mathrm{I}(\ell'_i))$ $(\forall i \in m)$ has no impact on the decision to declassify if $\mathrm{I}(\ell a) \sqsubseteq \mathrm{I}(\ell' i)$ or if the endorsement is not permitted by endorsing policies when endorsement occurs, or the case that the integrity of e'_i is upgraded to $\mathrm{I}(\ell' i)$ by $endorse(e'_i, \mathrm{I}(\ell'_i))$ and $\mathrm{I}(\ell' i) \sqsubseteq \mathrm{I}(\ell a)$. In the former case, the integrity of e'_i will not be changed if the endorsement is not permitted, or the data of e'_i still cannot affect the decision to declassify whose integrity is at or above $\mathrm{I}(\ell a)$ if $\mathrm{I}(\ell a) \sqsubseteq \mathrm{I}(\ell' i)$. It seems as the case in which no endorsing occurs. Hence, we conclude that e satisfies intransitive endorsement. In the latter case, although the decision to declassify can be affected by the data of e'_i, it is permitted by the specified endorsing function $endorse(e'_i, \mathrm{I}(\ell'_i))$ with the local endorsing policy of e'_i. So information release through $declassify(e_j, \mathrm{C}(\ell_j))$ $(\forall j \in n)$ is secure.

Improving the Time Complexity of Matsui's Linear Cryptanalysis

B. Collard, F.-X. Standaert*, and Jean-Jacques Quisquater

UCL Crypto Group, Université Catholique de Louvain

Abstract. This paper reports on an improvement of Matsui's linear cryptanalysis that reduces the complexity of an attack with *algorithm 2*, by taking advantage of the Fast Fourier Transform. Using this improvement, the time complexity decreases from $O(2^k * 2^k)$ to $O(k * 2^k)$, where k is the number of bits in the keyguess. This improvement is very generic and can be applied against a broad variety of ciphers including SPN and Feistel schemes. In certain (practically meaningful) contexts, it also involves a reduction of the attacks data complexity (which is usually the limiting factor in the linear cryptanalysis of block ciphers). For illustration, the method is applied against the AES candidate Serpent and the speed-up is given for exemplary attacks.

Keywords: block ciphers, linear cryptanalysis, Fast Fourier Transform.

1 Introduction

The linear cryptanalysis [1] is one of the most powerful attacks against modern block ciphers in which an adversary exploits a linear approximation of the type:

$$P[\chi_P] \oplus C[\chi_C] = K[\chi_K] \tag{1}$$

In this expression, P, C and K respectively denote the plaintext, ciphertext and the secret key while $A[\chi]$ stands for $A_{a_1} \oplus A_{a_2} \oplus ... \oplus A_{a_n}$,with $A_{a_1}, ..., A_{a_n}$ representing particular bits of A in positions $a_1, ..., a_n$ (χ is usually denoted as a mask). In practice, linear approximations of block ciphers can be obtained by the concatenation of one-round approximations and such concatenations (also called characteristics) are mainly interesting if they maximize the deviation (or bias) $\epsilon = p - \frac{1}{2}$ (where p is the probability of a given linear approximation).

In its original paper, Matsui described two methods for exploiting the linear approximations of a block cipher, respectively denoted as ,,. .,,. and ,,. .,,. . In the first one, given an r-round linear approximation with sufficient bias, the algorithm simply counts the number of times the left side of Equation 1 is equal to zero for N pairs (plaintext, ciphertext). If $T > N/2$, then it assumes either $K[\chi_K] = 0$ if $\epsilon > 0$ or $K[\chi_K] = 1$ if $\epsilon < 0$ so that the experimental value $(T - N/2)/N$ matches the theoretical bias. If $T > N/2$, an opposite reasoning holds. For the attack to be successful, it is shown in [1] that the number of available (plaintext, ciphertext)-pairs must be proportional to $\frac{1}{\epsilon^2}$.

* Postdoctoral researcher of the Belgian Fund for Scientific Research (FNRS).

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 77–88, 2007.

In the second method , an r-1-round characteristic is used and a partial decryption of the last round is performed by guessing the key bits involved in the approximation. As a consequence, all the guessed key bits can be recovered rather than the parity $K[\chi_K]$ which yields much more efficient attacks in practice. Moreover, as it uses a r-1-round characteristic instead of a r-round one for , it has a smaller data complexity. However, this improved efficiency has its counterpart in a higher computational complexity, due to the use of possibly large key guesses (involving a large number of key bits).

In this paper, we consequently introduce a general method for improving the time complexity of a linear cryptanalysis attack using . Although the limiting factor for linear cryptanalysis attacks is usually the data complexity, such an improvement is relevant and can be motivated both by practical and theoretical reasons, as the following scenarios underline.

- In the evaluation of linear cryptanalysis attacks against modern block ciphers, the attacker usually does a tradeoff between the bias of the approximation and the size of the keyguess. For example, when targeting 10 rounds of Serpent in [6], the authors found a 9-round approximation with bias 2^{-52} but with 92 bits of keyguess (23 active Sboxes). As this leads to an attack with time complexity $O(2^{184})$, they had to choose an approximation with a smaller bias (namely 2^{-58}) but only 44 bits of keyguess. Using our improvement, we can take advantage of the 2^{-52} bias with a time complexity of about $2^{98.5}$, thus reducing the data complexity from 2^{118} to 2^{106}.
- Since most recent ciphers (the AES candidates) have strong diffusion properties, the number of active S-boxes in linear cryptanalysis attacks against their reduced-round versions is usually too high for the time complexity of these attacks to be tractable. The improvement proposed in this paper can consequently be used to perform actual cryptanalytic experiments against these reduced-round ciphers. Therefore, we expect that it will lead to a better understanding of certain open issues, about the exploitation of multiple approximations in linear cryptanalysis.

Independently of these theoretical expectations, we believe that the proposed improvement of the time complexity, from $O(2^k * 2^k)$ to $O(k * 2^k)$ (where k is the number of bits in the keyguess) is meaningful in itself. We note finally that the idea of taking advantage of the Fast Fourier Transform to speed-up the computations in cryptanalysis is not new. For example [3] and [4] describe FFT-based techniques to improve correlation attacks against stream ciphers. However, we are not aware of any publication mentioning explicitly the applicability of the FFT to the linear cryptanalysis of block ciphers.

The rest of the paper is structured as follows. Section 2 describes a generic framework for the analysis of Matsui's linear cryptanalysis using . Section 3 details our improved key guess strategy and Section 4 applies our technique to improve some previous cryptanalytic results against the AES candidate Serpent. Finally, our conclusions are in Section 5.

2 General Framework for *Algorithm 2*

Suppose a linear approximation on r rounds with bias ϵ, requiring $N \approx O(1/\epsilon^2)$ known plaintext-ciphertext pairs for a successful attack. Moreover, this approximation has q active S-boxes in the last round and k bits to guess. In the original [illegible] proposed by Matsui, a partial decryption of the last round is performed for every ciphertext by guessing the key bits involved in the approximation. The parity of the approximation for the plaintext and the partially decrypted ciphertext is then evaluated and a counter corresponding to the guess is incremented if the relation holds, decremented otherwise. The key candidate with the highest counter in absolute value is finally assumed to be the correct key. As a partial decryption is proceeded for every ciphertext and every keyguess, the time complexity of this algorithm is in $O(N \cdot 2^k)$ partial decryptions.

However, as we only consider a limited number of bits (those in the active S-boxes) during the partial decryption of the ciphertexts, the same work is done many times. Indeed, the number of texts required to mount an attack is typically largely superior to the size of the keyguess (. $N \gg 2^k$). On the basis of this observation, Matsui proposed in [2] an improvement which considerably reduces the time complexity of an attack. Although it was first applied to the DES, this improvement is valid in the general case. The modified algorithm can be divided in 2 phases (according to the framework proposed in [7] sec. 2.1):

Distillation phase (for each generated ciphertext)**:**

- Initialize an array of 2^k counters.
- For each generated ciphertext, extract the k-bit value corresponding to the active S-boxes and evaluate the parity of the plaintext subset defined by the approximation. Increment or decrement the counter corresponding to the extracted k-bit value according to the parity.

Analysis phase (once all the ciphertext have been generated)**:**

- For each k-bit ciphertext and k-bit subkey, partially decrypt the k-bit ciphertext under the k-bit subkey and evaluate the parity of the output subset (as defined by the linear approximation). Keep this value in a table of size $2^k \cdot 2^k$.
- For each k-bit subkey, evaluate its experimental bias by checking, for each k-bit ciphertext, the parity of the approximation and the value of the corresponding counter. Then output the subkey which has maximal bias.

During the distillation phase, we construct a table that indexes for each k-bit ciphertext, the difference between the frequency of its apparition leading to a null input parity and the frequency leading to a non-null input parity. This information is sufficient to evaluate the bias of the approximation for each key during the analysis phase. This process can be done "on the fly", while the plaintexts are being encrypted. As only simple operations like bit extractions and incrementations are performed during this phase, its complexity is generally assumed to be negligible compared to the one of the encryption process.

During the analysis phase, the actual bias for each subkey candidate is evaluated. In order to avoid multiple evaluations of the same operation, a table is constructed which indexes, for each k-bit ciphertext and each subkey candidate, the parity of the output subset obtained after the partial decryption of the ciphertext XORed with the subkey. For a given subkey candidate, its bias can then be evaluated by summing, for each k-bit ciphertext, the corresponding counter, taking the parity of the approximation for the given ciphertext and subkey into account (this parity is given by the sign of the counter and the correct index in the precomputed table). In this way, the table is accessed 2^k times for each possible subkey, leading to a total time complexity of $O(2^k \cdot 2^k)$, compared to the $O(N \cdot 2^k)$ operations for a naive implementation of [illegible]. Importantly, this complexity depends only on the number of subkey candidates and not on the number of texts used. Note finally that the table can be computed row by row in order to save memory space.

3 Improving the Framework

In this section, we present a simple but powerful modification of the above algorithm that allows us to significantly decrease the time complexity of an attack. As the modification concerns only the analysis phase, the distillation phase remains unchanged and so does the data complexity.

3.1 Rewriting the Algorithm

The table defined during the analysis phase can be seen as a large matrix $\boldsymbol{C}$ of size $2^k \cdot 2^k$ defined by the following function:

$$\boldsymbol{C}(i,j) = parity(\mathrm{S}^{-1}(i \oplus j)) \qquad (0 \leq i,j \leq 2^k - 1) \tag{2}$$

where $\mathrm{S}^{-1}(l)$ represents the inverse of the last layer of S-boxes for the k-bit digit l and $parity()$ is a function mapping any k-bit subset to ± 1 according to its parity ($+1$ if the parity of the subset is zero, -1 otherwise). With such a definition, the bias ϵ_i corresponding to a particular keyguess i is given by the equation:

$$\epsilon_i = \sum_{j=0}^{2^k-1} parity(\mathrm{S}^{-1}(i \oplus j)) \cdot \boldsymbol{x}(j) = \sum_{j=0}^{2^k-1} \boldsymbol{C}(i,j) \cdot \boldsymbol{x}(j) = \boldsymbol{C}(i,:) \cdot \boldsymbol{x} \tag{3}$$

where $\boldsymbol{x}$ is the vector of counters such as defined in the distillation phase. This equation evaluates the bias of the linear approximation for a particular k-bit subkey candidate i. Consequently, the vector $\boldsymbol{\epsilon}$ of the experimental bias for every subkey candidates can be computed by the matrix-vector product:

$$\boldsymbol{\epsilon} = \boldsymbol{C} \cdot \boldsymbol{x} \tag{4}$$

At this point, the complexity for the evaluation of the experimental biases is still in $O(2^k \cdot 2^k)$ as it implies a matrix-vector product with size 2^k.

3.2 Analysis of the New Algorithm

We underline the fact that the matrix $\boldsymbol{C}$ has a very particular structure. Taking this structure into account will allow us to significantly reduce the number of operations required to evaluate the vector $\boldsymbol{\epsilon}$ of the bias.

First, as $\boldsymbol{C} = f(i \oplus j)$ for a known function f, every rows or column of $\boldsymbol{C}$ defines the complete matrix (in particular, $\boldsymbol{C}$ is symmetric). For example,

$$\boldsymbol{C}(i,j) = f(i \oplus j) = f((i \oplus j) \oplus 0) = \boldsymbol{C}(i \oplus j, 0) \tag{5}$$

Let us introduce the following definitions ([10]):

Definition 1 (circulant). ff

Definition 2 (block circulant). ff

Definition 3 (level circulant).

- 1 ff
- 2 ff
- 3 ff
  2
- q ff
  q − 1

Proposition 1. $\boldsymbol{C}(i,j) = f(i \oplus j)$ $\boldsymbol{C}$ $\underbrace{(2,2,...,2)}_{k \quad times}$

Demonstration 1. fi $\boldsymbol{M}$ $(2^k * 2^k)$ $\boldsymbol{M}(i,j) = i \oplus j (0 \leq i,j \leq 2^k - 1)$ $\boldsymbol{M}$ $(2^{k-1} * 2^{k-1})$

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{M_{11}} & \boldsymbol{M_{12}} \\ \boldsymbol{M_{21}} & \boldsymbol{M_{22}} \end{pmatrix}$$

$(0 \leq a,b \leq 2^{k-1} - 1)$

- $\boldsymbol{M_{11}}(a,b) = \boldsymbol{M}(a,b) = a \oplus b$
- $\boldsymbol{M_{12}}(a,b) = \boldsymbol{M}(a, b + 2^{k-1}) = a \oplus b \oplus 2^{k-1}$
- $\boldsymbol{M_{21}}(a,b) = \boldsymbol{M}(a + 2^{k-1}, b) = a \oplus 2^{k-1} \oplus b$
- $\boldsymbol{M_{22}}(a,b) = \boldsymbol{M}(a + 2^{k-1}, b + 2^{k-1}) = a \oplus 2^{k-1} \oplus b \oplus 2^{k-1}$

$a + 2^{k-1}$ $a \oplus 2^{k-1}$ $0 \leq a \leq 2^{k-1} - 1$ $\boldsymbol{M_{11}} = \boldsymbol{M_{22}}$ $\boldsymbol{M_{12}} = \boldsymbol{M_{21}}$ $\boldsymbol{M}$ $\boldsymbol{M_{12}} = \boldsymbol{M_{11}} \oplus 2^{k-1}$ $\boldsymbol{M_{11}}$ $\boldsymbol{M} = \boldsymbol{M_{11}}$ $k = k - 1$ $f(\boldsymbol{M})$ $f()$

3.3 Fast Algorithm

We now describe how the properties given above can be used to speed up the linear cryptanalysis. We exploit the following result ([10] for a proof):

Theorem 1. C $F = F_m \otimes F_n \otimes F_o \otimes ... \otimes F_r$

$$C = F^* diag(\lambda) F, \tag{6}$$

λ C $\otimes$ F_n $n * n$ fi

$$F_n(i,j) = \frac{1}{\sqrt{n}} w^{i \cdot j} \quad (0 \leq i, j \leq n-1), \tag{7}$$

$$w = e^{\frac{2\pi\sqrt{-1}}{n}} \tag{8}$$

The matrix F is the k-dimensional Discrete Fourier Transform matrix. Therefore, the multidimensional Fast Fourier Transform allows us to quickly compute the matrix-vector product with F or F^*. Using the FFT, the complexity of this product decrease from $O(n^2)$ to $O(n\ log_2(n))$ [9].

Proposition 2. λ C

$$\lambda = FC(:,1)\sqrt{mno...r}, \tag{9}$$

$C(:,1)$ fi C

Demonstration 2. □

$$C = F^* diag(\lambda) F \tag{10}$$

F $FF^* = I$

$$FC = diag(\lambda) F \tag{11}$$

fi

$$(FC)(:,1) = (diag(\lambda)F)(:,1) = \lambda \circ F(:,1) \tag{12}$$

□ fi F

$$F(:,1) = \frac{1}{\sqrt{mno...r}}(1,1,1...1)^T \tag{13}$$

$$\lambda = (FC)(:,1)\sqrt{mno...r} = FC(:,1)\sqrt{mno...r} \tag{14}$$

Hence, the matrix-vector product $\boldsymbol{\epsilon} = \boldsymbol{C}\boldsymbol{x}$ is equivalent to $\boldsymbol{\epsilon} = \boldsymbol{F}^* diag(\boldsymbol{\lambda})\boldsymbol{F}\boldsymbol{x}$. The eigenvalues of $\boldsymbol{C}$ can be computed using the formula: $\boldsymbol{\lambda} = \boldsymbol{F}\boldsymbol{C}(:,1)\sqrt{2^k}$. Therefore, the matrix-vector product can be computed using the three following matrix-vector products: $\boldsymbol{y} = \boldsymbol{F}\boldsymbol{x}$, $\boldsymbol{z} = \boldsymbol{F}\boldsymbol{C}(:,1)\sqrt{2^k}$ and $\boldsymbol{\epsilon} = \boldsymbol{F}^*(\boldsymbol{z} \circ \boldsymbol{y})$ (where $\circ$ is the Hadamard product). As each of these three products involves the matrix $\boldsymbol{F}$, the complete computation can be made by applying only three k-dimension FFTs of size 2^k, leading to a complexity of $3 \cdot 2^k \cdot log_2(2^k) = 3 \cdot k \cdot 2^k$. The Matlab implementation code for this improved strategy is given below (see Algorithm 1). As a typical numerical example, for a $2^{20} * 2^{20}$ matrix $\boldsymbol{C}$, the matrix-vector product is computed in less than 5 seconds on a Pentium D 3.20GHz.

Algorithme 1. Matlab code

```
function b=product(C,x)

% compute the product Cx by the mean of the fft
% C is a level k circulant matrix of type (2,2,2,...,2).
% C is completely specified by its first column
% x is an unspecified vector

k= log2(size(C,1));

% compute F*x:
x= reshape(x,2*ones(k));
prod1= fftn(x);
prod1= reshape(prod1,2^k,1);

% compute the eigenvalues of C
c= reshape(C(:,1), 2*ones(k));
eig= fftn(c);
eig= reshape(eig,2^k,1);

% compute eig*F*x
prod2= eig.*prod1;

% compute b=F'*eig*F*x
b= reshape(prod2,2*ones(k));
b= ifftn(b);
b= reshape(b,2^k,1);

return b;
```

3.4 Implication for Multiple Linear Approximations

In context of multiple linear approximations ([11], [7]), more speed-up may be achievable. Every input mask defines its own vector of counter $\boldsymbol{x}$, while every output mask defines a different matrix $\boldsymbol{C}$. Consequently, the use of multiple

approximations with the same active S-boxes but different input masks (as it is usually the case) requires to compute the eigenvalues only once, as the matrix $\boldsymbol{C}$ remains the same. Thus, the time complexity of linear cryptanalysis with n approximations is reduced to the computation of $2n+1$ FFTs instead of $3n$ FFTs. This involves an additional reduction of up to 33% for the time complexity.

3.5 Extension to Key Additions *Modulo* 2^k

In certain ciphers, the mixing with the key material is done using a modular addition instead of a XOR with the subkey (practical examples include [12], [13]). A similar approach as in the previous sections can be applied to reduce the time complexity of such systems.

Definition 4 (left-circulant). ff

Proposition 3. $\boldsymbol{C_{left}}(i,j) = f(i+j \bmod 2^k)$ $(0 \leq i, j \leq 2^k - 1)$ $\boldsymbol{C_{left}}$

Demonstration 3. λ $\boldsymbol{C_{left}}(a+\lambda, b-\lambda) = f(a+\lambda+b-\lambda \bmod 2^k) = f(a+b \bmod 2^k)$ 2^k $\bmod 2^k$

We can easily convert a left-circulant matrix $\boldsymbol{C_{left}}$ to a circulant matrix $\boldsymbol{C}$ with the same first row thanks to a particular matrix of permutation $\boldsymbol{\Gamma}$:

$$\boldsymbol{C_{left}} = \boldsymbol{\Gamma C}, \tag{15}$$

where:

$$\boldsymbol{\Gamma} = \begin{pmatrix} 1\,0\,0 \ldots 0\,0 \\ 0\,0\,0 \ldots 0\,1 \\ 0\,0\,0 \ldots 1\,0 \\ \vdots\,\vdots\,\vdots \quad\quad \vdots\,\vdots \\ 0\,1\,0 \ldots 0\,0 \end{pmatrix}$$

This requires the application of 2^k row permutations and it is useful for the following result that we require in our investigations (see [10] pp.72-73):

Theorem 2. $\boldsymbol{C}$ $\boldsymbol{F}$ 2^k

$$\boldsymbol{C} = \boldsymbol{F}^* diag(\boldsymbol{\lambda}) \boldsymbol{F}, \tag{16}$$

$\boldsymbol{\lambda}$ $\boldsymbol{C}$

Again, we can easily compute the matrix-vector product between $\boldsymbol{C}$ and $\boldsymbol{x}$ using 3 (one-dimensional) FFTs. The algorithm is roughly the same as described above,

except that the multi-dimensional FFTs are replaced by one-dimensional FFTs and that we must perform a last permutation $\boldsymbol{\Gamma}$ in the end (in order to switch from a right- to a left-circulant matrix). Finally, for a matrix $\boldsymbol{C_{left-toeplitz}}$:

$$\boldsymbol{C_{left-toeplitz}}(i,j) = f(i+j \ mod \ 2^m);\ (0 \leq i,j \leq 2^k-1),\ m \neq k, \qquad (17)$$

with the left-Toeplitz structure, it can be shown that the matrix-vector product can be computed by embedding the $2^k * 2^k$ matrix in a $2^{k+1} * 2^{k+1}$ matrix with left-circulant structure, leading to a complexity of $O(2*(k+1)*2^k)$.

4 Practical Improvements

The improved algorithms described above can be straightforwardly applied to improve previous cryptanalytic results. For illustration purposes, we applied them to the AES candidate Serpent [5]. Using the FFT to compute the linear approximation biases for the subkey candidates allows speeding-up the attack of Biham , , as summarized in Table 1. In Table 2, we additionally report on the improved linear and multiple linear cryptanalysis of Serpent, described in [8].

Table 1. Previous and improved attacks on Reduced-rounds Serpent

Rounds	Type of attack	complexity		
		data	time	memory
10	Lin.Cryptanalysis [6]	2^{118}KP	$2^{88} \rightarrow 2^{51}$	2^{44}
	Lin.Cryptanalysis [6]	2^{116}KP	$2^{96} \rightarrow 2^{55.2}$	2^{48}
	Lin.Cryptanalysis [6]	2^{106}KP	$2^{184} \rightarrow 2^{100.1}$	2^{92}
11	Lin.Cryptanalysis [6]	2^{118}KP	$2^{214} \rightarrow 2^{148.7}$	$2^{88} \rightarrow 2^{140}$
KP - Known Plaintexts Complexity is measured in number of arithmetic operation. Memory is mesured in Bytes.				

We note that, as previously mentioned, certain improvements are particularly relevant with respect to the experimental testing of the attacks. For example, targeting 7 rounds of Serpent with a multiple linear cryptanalysis attack appears as a reasonable target thanks to time complexity reduction. Using dedicated hardware like , , [14], it could even be possible to attack up to 8-round Serpent. The reduced time complexity also allows considering the exploitation of better biased linear approximations with larger key guesses and therefore to reduce the data complexity of the best reported attacks. For example, Table 1 includes the scenario described in the introduction of this paper: moving from a time complexity of 2^{184} to a time complexity of 2^{100} allows to reduce the attack data complexity from 2^{118} to 2^{106}. This example clearly emphasizes the practical impact of our result on the overall complexity of linear cryptanalysis attacks.

Additionally to the results presented in [8], Table 2 includes an attack against 11-round Serpent. We use a 9-round linear approximation starting and ending

Table 2. Additional improved attacks on Reduced-rounds Serpent (see [8])

Rounds	Type of attack	complexity		
		data	time	memory
7	Lin.cryptanalysis	2^{52}KP	$2^{40} \rightarrow 2^{25.9}$	2^{20}
	Mult.Lin.Cryptanalysis(8 appr.)	2^{47}KP	$2^{43} \rightarrow 2^{28.4}$	2^{23}
8	Lin.cryptanalysis	2^{62}KP	$2^{56} \rightarrow 2^{34.4}$	2^{28}
	Mult.Lin.Cryptanalysis(8 appr.)	2^{57}KP	$2^{59} \rightarrow 2^{36.9}$	2^{31}
	Mult.Lin.Cryptanalysis(104 appr.)	2^{55}KP	$2^{62.7} \rightarrow 2^{40.5}$	$2^{34.7}$
9	Lin.cryptanalysis	2^{80}KP	$2^{88} \rightarrow 2^{51}$	2^{44}
	Mult.Lin.Cryptanalysis(128 appr.)	2^{71}KP	$2^{95} \rightarrow 2^{57.5}$	2^{51}
	Mult.Lin.Cryptanalysis(3712 appr.)	2^{68}KP	$2^{99.9} \rightarrow 2^{62.3}$	$2^{55.9}$
10	Lin.cryptanalysis ($\epsilon = 2^{-55}$)	2^{112}KP	$2^{88} \rightarrow 2^{51}$	2^{44}
	Mult.Lin.Cryptanalysis(2048 appr.)	2^{99}KP	$2^{99} \rightarrow 2^{61.5}$	2^{55}
	Lin.cryptanalysis ($\epsilon = 2^{-59}$)	2^{120}KP	$2^{64} \rightarrow 2^{38.6}$	2^{32}
	Mult.Lin.Cryptanalysis(2048 appr.)	2^{107}KP	$2^{75} \rightarrow 2^{49}$	2^{43}
11	Lin.cryptanalysis ($\epsilon = 2^{-58}$)	2^{118}KP	$2^{178} \rightarrow 2^{116.3}$	2^{108}

with S-box 9. This approximation was generated similarly to the ones presented in [8]. It has a bias of 2^{-58} and a total of 27 active S-boxes (15 in the first round and 12 in the last round). The attack follows the same principle as the ones presented so far, except that we must also perform a partial encryption in the beginning of the cipher. We first define an array $\boldsymbol{x}$ of 2^{108} counters in the following way: for each plaintext-ciphertext pair, we extract the 108-bit value corresponding to the active S-boxes and we increment the corresponding counter. Then we define a matrix $\boldsymbol{C}$ of size $2^{108} * 2^{108}$ as:

$$\boldsymbol{C(i,j)} = parity(S(i_{1:60} \oplus j_{1:60}) || S^{-1}(i_{61:108} \oplus j_{61:108})) \tag{18}$$

That is to say, $\boldsymbol{C(i,j)}$ is the parity of the linear approximation after partial en/decryption of the 108-bit text . with 108-bit subkey . . As seen previously, the experimental bias for any keyguess is given by the matrix-vector product $\boldsymbol{C} \cdot \boldsymbol{x}$. Thanks to the level circulant structure of $\boldsymbol{C}$, the time complexity is equal to $3 \cdot 108 \cdot 2^{108} = 2^{116}$. Without this trick, the time complexity would have been $2^{88} + 2^{60} \cdot (2^{118} + 2^{88}) = 2^{178}$ (see [6] for the details). As a comparison, the best-reported attack on 11-round Serpent uses a combination of linear and differential cryptanalysis techniques [15]. It has a data complexity of $2^{125.3}$ choosen plaintexts, $2^{139.2}$ encryptions and 2^{60} bytes of memory.

5 Conclusion and Further Works

In this paper, we presented an improvement of Matsui's linear cryptanalysis that reduces the time complexity of an attack using . . . . from $O(2^k * 2^k)$ to $O(k * 2^k)$ partial decryptions, where k is the number of bits in the keyguess. Moreover, in the case of multiple linear cryptanalysis, additional speed-ups can be reached. This improvement is very generic and can be applied against a

broad variety of ciphers including SPN and Feistel schemes. For illustration purposes, we applied the method to the block cipher Serpent and exhibited the reduced complexities of some (state-of-the-art) exemplary attacks. As a scope for further research, the exploitation of the improved time complexity of attacks using multiple linear approximations should allow performing actual experiments and therefore evaluate the validity of certain assumptions detailed in [7].

Acknowledgements

The authors thank Pr. Paul Van Dooren from UCL\INMA for his helpful advices. We also thank anonymous reviewers from SAC and ICISC for their comments.

References

1. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
2. Matsui, M.: The First Experimental Cryptanalysis of the Data Encryption Standard. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 1–11. Springer, Heidelberg (1994)
3. Chose, P., Joux, A., Mitton, M.: Fast Correlation Attacks: An Algorithmic Point of View. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 209–221. Springer, Heidelberg (2002)
4. Lu, Y., Meier, W., Vaudenay, S.: The Conditional Correlation Attack: A Practical Attack on Bluetooth Encryption. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 97–117. Springer, Heidelberg (2005)
5. Anderson, R., Biham, E., Knudsen, L.: Serpent: A Proposal for the Advanced Encryption Standard. In: The proceedings of the First Advanced Encryption Standard (AES) Conference, Ventura, CA (1998)
6. Biham, E., Dunkelman, O., Keller, N.: Linear Cryptanalysis of Reduced Round Serpent. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 16–27. Springer, Heidelberg (2002)
7. Biryukov, A., De Cannière, C., Quisquater, M.: On Multiple Linear Approximations. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 1–22. Springer, Heidelberg (2004)
8. Collard, B., Standaert, F.-X., Quisquater, J.-J.: Improved and Multiple Linear Cryptanalysis of Reduced Round Serpent. In: The proceedings of InsCrypt (to appear, 2007)
9. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. 19, 297–301 (1965)
10. Davis, P.J.: Circulant Matrices, pp. 176–191. Wiley-Interscience, Chichester (1979)
11. Kaliski, B.S., Robshaw, M.J.B.: Linear Cryptanalysis using Multiple Approximations. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 26–39. Springer, Heidelberg (1994)
12. Lai, X., Massey, J.L.: A Proposal for a New Block Encryption Standard. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 389–404. Springer, Heidelberg (1991)

13. Wheeler, D.J., Needham, R.M.: TEA, a Tiny Encryption Algorithm. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 363–366. Springer, Heidelberg (1995)
14. Kumar, S., Paar, C., Pelzl, J., Pfeiffer, G., Schimmler, M.: Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, Springer, Heidelberg (2006)
15. Biham, E., Dunkelman, O., Keller, N.: Differential-linear Cryptanalysis of Serpent. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 9–21. Springer, Heidelberg (2003)

On Large Distributions for Linear Cryptanalysis

Alexander Maximov

LACS, University of Luxembourg
L-2311 Avenue Pasteur, 102a, Luxembourg
movax@mail.ru

Abstract. Calculating the distribution of certain functions during the linear cryptanalysis of stream ciphers is a frequently encountered problem. Let a function N (or a noise variable) be expressed via k mutually independent and uniformly distributed n-bit random variables $X_1, X_2, \ldots, X_k$. The possibility to construct its distribution depends on the form of the expression N, and sometimes it becomes a bottleneck of the cryptanalysis.

In this paper we propose several new techniques to construct such distributions and widen the class of functions for which its distribution can efficiently be calculated.

Keywords: Linear cryptanalysis, complexity reduction, approximations, large distributions, pseudo-linear functions.

1 Introduction

Linear cryptanalysis is an important type of analysis of cryptographic primitives. It is, for example, the best known attack on DES [1,2]. Many recently proposed stream ciphers, such as Scream [3], SNOW 2.0 [4], A5/1 [5], E_0 [6], RC4 [7], SOBER t16/t32 [8,9], and others, are shown to be weak agains linear cryptanalysis [10,11,12,13,14,15,16]. In this kind of analysis, nonlinear blocks are substituted by a linear transformation. Due to such an approximation a new variable, called , is added to the result of the linear transformation. When the cipher becomes linear, a certain equation connecting words of the plaintext, the ciphertext and the secret key [1] can be derived. These equations are usually time invariant, one side of which is a known value – a sample value Z_t at time t, and on the other side of the equation is a linear combination N of noise variables. This allows to collect samples Z_t, the distribution of which converges to the distribution of N. A sufficient number of samples can allow to recover the complete or a part of the secret key.

The efficiency of such attacks usually depends on the possibility to construct the distribution of the noise. This particular problem often becomes a bottleneck in various linear cryptanalyses, since the construction of the distribution for a complex function can be computationally infeasible for nowadays resources. To

[1] In stream ciphers it is assumed that *the keystream* is available, which is usually a word based addition of the plaintext and the ciphertext.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 89–101, 2007.

be more specific, assume that the operation $X \boxplus \alpha Y$ is replaced by $X \oplus Y$, where $\boxplus$ is the addition modulo 2^n and α is some element of the field $\mathbb{F}_{2^n}$. The effectiveness of the approximation is given by the probability $\Pr\{(X \boxplus \alpha Y) \oplus (X \oplus Y) = \gamma\}$. However, its distribution is hard to compute. When $n = 32$, a straightforward solution requires 2^{64} of operations, whereas the algorithm proposed in this paper needs only 2^{37} of time.

Similar problems were studied in several previous papers, see, e.g. [17,18,19,20]. A recent paper [21], however, introduces the most general class called [illegible] (PLFs), for which their distributions can be calculated. These functions are useful not only for calculating the bias, but also for calculating correlations, differences, collisions, and other important pieces during cryptanalysis. However, PLFs do not cover all cases that one can meet in cryptanalysis, and this motivates us to search new supplementary solutions and techniques to widen this class of functions and to make a more complicated cryptanalysis possible.

Let $X_1, X_2, \ldots, X_k$ be k n-bit mutually independent and uniformly distributed random variables, and N be a function on them. In this paper we propose a set of algorithms that widen the class of PLFs and allow to deal with much more complicated noise functions in order to improve various results in cryptanalysis. We also discuss several aspects in relation to binary and multidimentional cryptanalysis. We show that an n bits analysis is always more powerful than a binary analysis. However, when n is large its reduction can in principal be done in two ways. In the first solution one can consider a truncated version of the noise variable $L \cdot N$, where L is some masking matrix. Another solution is combining binary approximations and consider them jointly. Assiciated with each of these solutions we give complexities and computation techniques to deal with these cases.

This paper is organized as follows. The paper starts with preliminaries and basic definitions in Section 1.1. The relations between a binary and a multidimensional approximations are discussed in Section 2. The new algorithms are given in Section 3. Finally, we conclude in Section 4.

1.1 Notation and Preliminaries

Let an n bit random variable be denoted by a capital letter X, and its probability space by $\mathcal{X}$. Binary representation of an n bit integer number X is $\overline{x_{n-1}, \ldots, x_1, x_0}$. An integer $X[a:b]$ is $\overline{x_b, \ldots, x_{a+1}, x_a}$. By P_X and $P_{f(X)}$ the distribution tables of X and a function $f(X)$ are denoted, respectively. The probability $\Pr\{X = x\}$ will also be referred as $P_X(x)$, for simplicity. The bias of P_X is defined as

$$\epsilon = \mathrm{bias}(P_X) = \sum_{x \in \mathcal{X}} \left| P_X(x) - \frac{1}{|\mathcal{X}|} \right|, \tag{1}$$

whose range is $0 \le \epsilon \le 2$. For a binary random variable X we have $P_X(0) = \frac{1}{2}(1-\epsilon)$ and $P_X(1) = \frac{1}{2}(1+\epsilon)$ (or vice versa). The uniform distribution is when $\epsilon = 0$.

Bitwise XOR and arithmetical addition modulo 2^n are denoted by $\oplus$ and $\boxplus$, respectively. Binary operators applied to two vectors A and B will usually mean component-wise procedure, whereas $< A, B >$ is a scalar product of two vectors, or two numbers in their binary representation. The n bit [illegible] of a vector $X = (x_0, x_1, \ldots, x_{2^n-1})$ at a point w is

$$\mathrm{FHT}_n(X)_w = \sum_{i=0}^{2^n-1} x_i \cdot (-1)^{<i,w>}, \quad \forall w \in \mathbb{Z}_{2^n}, \tag{2}$$

and its inverse is $\mathrm{FHT}_n(X)_w^{-1} = 2^{-n} \cdot \mathrm{FHT}_n(X)_w$. We will also refer to the class of [illegible] introduced in [21]. These are functions of a particular form for which there exist efficient algorithms to calculate their distributions. In brief, in PLFs the operations $\boxplus, \boxminus$, all Boolean operations, constants and brackets are allowed.

2 Binary vs. Multidimensional Approximations

A binary approximation is a common approximation in the linear cryptanalysis. Usually one has to find such a relation between bits of the keystream that their sum would have maximum bias. The bias can be even larger if one consider two or more binary approximations jointly.

Nowadays, most of designs are word (n-bit) oriented, and the search of a linear relation ends up with a relation in words over some field. Let us have such a linear expression which allows to sample from the keystream. These samples will follow the distribution that is also called a [illegible] P_N, where N is now a random noise variable. Usually, the expression for the noise N is an n-bit function on k n-bit mutually independent and uniformly distributed random variables. When one searches for a binary relation, in many cases it means that the samples are taken from the binary noise variable $L_1 \cdot N$, where L_1 is a [illegible], a non-zero vector of n bits that determines a certain binary approximation.

2.1 Relation Between Approximations

Instead of observing one bit of information from the keystream one could consider a multiple-bits approximation $L_1 \cdot N$, where L_1 is a collection of l_1 orthogonal binary approximations. The matrix L_1 is of size $l_1 \times n$, and it must be of full rank, i.e., there is no row that is linearly expressed via the other ones. Let us have another multiple-bits approximation matrix L_2 that is orthogonal to L_1.

Lemma 1. [illegible] L_1 [illegible] L_2 [illegible] l_1 [illegible] l_2 [illegible] [illegible] $P_{L_1 \cdot N}$ [illegible] $P_{L_2 \cdot N}$ [illegible] [illegible] [illegible] $(L_1\ L_2)$ [illegible] [illegible] $\max\{\epsilon_1, \epsilon_2\}$ [illegible] $\epsilon_1 = \mathrm{bias}(P_{L_1 \cdot N})$ [illegible] $\epsilon_2 = \mathrm{bias}(P_{L_2 \cdot N})$

... $N'_1 = L_1 \cdot N$ $N'_2 = L_2 \cdot N$ ... $N' = (N'_1 \ N'_2)$ ...

$$\epsilon = \mathrm{bias}(P_{N'}) = \sum_{x_1 \in \mathbb{Z}_2^{l_1}} \sum_{x_2 \in \mathbb{Z}_2^{l_2}} \left| \Pr\{N' = (x_1, x_2)\} - \frac{1}{2^{l_1+l_2}} \right|$$

$$\geq \sum_{x_1 \in \mathbb{Z}_2^{l_1}} \left| \sum_{x_2 \in \mathbb{Z}_2^{l_2}} \left(\Pr\{N'_1 = x_1\} \cdot \Pr\{N'_2 = x_2 | N'_1 = x_1\} - \frac{1}{2^{l_1+l_2}} \right) \right| \quad (3)$$

$$= \sum_{x_1 \in \mathbb{Z}_2^{l_1}} \left| \Pr\{N'_1 = x_1\} - \frac{1}{2^{l_1}} \right| = \epsilon_1.$$

... $\epsilon \geq \epsilon_2$ ... □

The direct consequence of the Lemma is that n-bit analysis of a cipher is never worse than a reduced case, i.e.,

$$\mathrm{bias}(P_{L \cdot N}) \leq \mathrm{bias}(P_N), \quad (4)$$

for any l-bit ($1 \leq l \leq n$) approximation L.

Assume one makes an approximation of a cipher and a word n bit oriented expression of the noise is derived. For various cryptanalysis purposes one could be interested in the bias of the noise, or even its complete distribution. Unfortunately, in many cases these targets are not possible to reach, due to a high computation complexity. The alternative is to study a truncated version $L \cdot N$ of the noise variable, where L is some $l < n$ bits approximation.

Lemma 2. ... L ... N ... $\mathrm{FHT}(P_N)$ ... □

The proof of the Lemma follows directly from the definition of FHT. An l bits approximation L can be constructed as the collection of l separate mutually orthogonal best binary approximations.

There are algorithms for probing the FHT of a function in certain points. In a general case, the question which point gives the maximum absolute value is still an open question. However, for specific cases (for example, when the function is a PLF), this open question is solvable by using the technique of bitwise analysis.

This was one way of a multidimensional analysis, and another way is when the structure of the joint distribution has a specific form, and one such form is studied in the following sub section.

2.2 Multidimensional Noise as a Collection of Binary Noises

Let us have a linear combination L, the way of sampling from the keystream. These samples are biased and, for example, come from a filter generator, or

from an NLFSR, or from another structure. In this section we study a multidimensional analysis when two or more consequtive binary samples are considered jointly.

Let $(x_1, x_2, \ldots, x_{2^l-1})$ be $2^l - 1$ mutually independent binary variables with the corresponding biases $(\epsilon_1, \epsilon_2, \ldots, \epsilon_{2^l-1})$, and l noise variables were constructed as follows.

$$n_i = \bigoplus_{\forall k:k[i:i]=1} x_k, \quad \forall i = 0, 1, \ldots, l-1. \tag{5}$$

We are interested to construct the distribution of the following l-bit noise variable $N = (n_0, \ldots, n_{l-1})$.

$$l \begin{cases} n_0 = x_1 \oplus & x_3 \oplus & x_5 \oplus & x_7 \oplus & \ldots \\ n_1 = & x_2 \oplus x_3 \oplus & & x_6 \oplus x_7 \oplus & \ldots \\ n_2 = & & x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus & & \ldots \\ n_3 = & & & x_8 \oplus & \ldots \\ \vdots & & & & \end{cases} \tag{6}$$

The distribution of such a variable can be constructed via a set of convolutions in time $O(l2^{2l})$. However, for large l the computation could be time consuming. We propose a more efficient way that has the complexity $O(l2^l)$, and the following theorem can be stated.

Theorem 1. $n_1, \ldots, n_l$ l 2^l-1 $x_1, \ldots, x_{2^l-1}$ (5) $\epsilon_1, \epsilon_2, \ldots$ P_N l $N = (n_1, \ldots, n_l)$

$$P_N = \mathrm{FHT}^{-1}\left[\sqrt{\exp\left(\sum E - \mathrm{FHT}(E)\right)}\right], \tag{7}$$

E

$$E = (0, \log\epsilon_1, \log\epsilon_2, \ldots, \log\epsilon_{2^l-1}). \tag{8}$$

$\exp, \sqrt{\cdot}$ $\sum E$ E

$\mathrm{FHT}(P_N)_t = \prod_{i=1}^{2^l-1} \epsilon_i^{<i,t>}$, $< i, t >$

$$\begin{aligned} \log\mathrm{FHT}(P_N)_t &= \sum_{i=1}^{2^l-1} \log\epsilon_i \cdot < i, t > = \sum_{i=1}^{2^l-1} \log\epsilon_i \cdot \frac{1}{2}\left(1 - (-1)^{\sum_{q=0}^{l-1} i[q:q]\cdot t[q:q]}\right) \\ &= \frac{1}{2}\left(\sum_{i=1}^{2^l-1} \log\epsilon_i - \sum_{i=1}^{2^l-1} \log\epsilon_i \cdot (-1)^{\sum_{q=0}^{l-1} i[q:q]\cdot t[q:q]}\right) = \frac{1}{2}\left(\sum E - \mathrm{FHT}(E)\right), \end{aligned} \tag{9}$$

□

The application of the Theorem above can be demonstrated by the following example.

Let the samples from the keystream be collected such that the noise variable n_t at any time t is

$$n_t = z_t + z_{t+1} + z_{t+5} + z_{t+6} + z_{t+7}. \tag{10}$$

Let also the bias of a single term be $\mathrm{bias}(z_t) = 0.5$, i.e., this is the bias of a single $\mathtt{AND}(x, y)$ term. The total bias of each n_t is 2^{-5}. Let us consider (n_t, n_{t+1}, n_{t+2}) jointly. Then we have

Shares	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7
Expression	.	$z_t + z_{t+5}$	.	$z_{t+1} + z_{t+6}$	$z_{t+3} + z_{t+9}$	.	$z_{t+2} + z_{t+8}$	z_t
$\log_2 E =$	0	-2	0	-2	-2	0	-2	-1

(11)

Application of Theorem 1 allows to construct P_N, the bias of which appears to be $2^{-3.9956}$, which is larger than in the case of a binary approximation. □

3 New Techniques to Compute Large Distributions

Let some (perhaps, a noise) function F be a function on k n-bit mutually independent and uniformly distributed variables $X_1, X_2, \ldots, X_k$. In this section we present several new techniques for calculating large distributions for a wide class of functions.

3.1 Splitting into Sub Distributions Through Events

Assume the function F is not a PLF, and it is not obvious how to compute its distribution. For example, consider the function $F = (X_1 \boxplus \pi(X_2)) \oplus X_1 \oplus X_2$, where X_1 and X_2 are two n-bit inputs, and π is a bits permutation matrix of size $n \times n$. If n is large (say, $n = 32$), the distribution of P_F cannot be computed in a classical way (it would require 2^{64} operations), and one needs to apply specific technique. Below we propose a general technique for solving this and many other similar bottlenecks in linear cryptanalysis.

Method of Events. In the the input space is to be separated into smaller mutually independent subspaces. The function F can, therefore, be evaluated in a separate way, one-by-one applied to the separate parts of the input space. Although the arguments themselves can be split into the subspaces, the evaluation of the function F will likely to have internal intermediate connections between these subspaces, and these sorts of relations can be expressed in terms of events. For example, when we perform arithmetical summation $A \boxplus B$, typical events are carry bit values that connect neighbouring bits.

Let us split n bits of the input variables $X_1, \ldots, X_k$ into b blocks of the corresponding sizes $n_1, n_2, \ldots, n_b$, as shown in Figure 1. The application of the

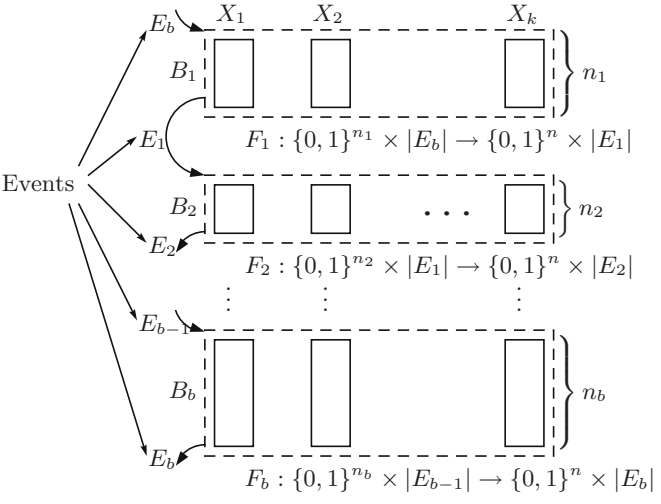


Fig. 1. Illustration to the method of events

function F to some i^{th} block will, in general, produce an intermediate n-bit result, **and** [illegible] from the i^{th} set E_i can happen, which is the influence of this i^{th} block to the next one(s). From another hand, the evaluation of the i^{th} block itself also depends on the event which might happen during the evaluation of the previous block(s) (usually it is the $(i-1)^{\text{th}}$ block)[2]. In a deterministic system the size of each event space $|E_i|$ is finite. Therefore, every i^{th} block can be described as a function F_i, which has an [illegible], an [illegible], and k [illegible] n_i [illegible], as follows.

$$F_i : \{0,1\}^{n_i} \times |E_{i-1}| \to \{0,1\}^n \times |E_i|. \tag{12}$$

Splitting the evaluation of the function F into b sub functions, as well as the choice of events, additionally depends on the choice of a [illegible] operation. It can be either over $\oplus$, or $\boxplus$. The calculation of the complete distribution P_F is combined in accordance. For example, when the linking operator is $\oplus$ the total distribution of the function F is calculated via P_{F_i}s as

$$P_F = \sum_{\substack{e_1 \in E_1 \\ \cdots \\ e_b \in E_b}} \bigoplus_{i=1}^{b} P_{F_i|(e_{i-1},e_i)}, \tag{13}$$

where $P_{F_i|(e_{i-1},e_i)}$ is an n-bit distribution of the function F_i conditioned on the input event e_{i-1} and the output event e_i. The operator $\oplus$ over the distribution

[2] Here and later, for simplicity, the indices of the blocks $1, 2, \ldots, b$ will be considered in a ring, i.e., the $j = (i-1)^{\text{th}}$ block for $i = 1$ is actually $j = b$.

tables means a [illegible] of b selected subdistributions, and it can be done via FHT for the time $O(bn2^n)$ (see [21]). The $\sum$ is a usual arithmetical point-to-point sum of the resulting distribution tables. At the end of the calculations, the distribution of the function F is received.

An important note is that the distributions P_{F_i} should be [illegible], i.e., the sum of the probabilities must be such that

$$\sum_{x \in \mathcal{X}} P_{F_i|(e_{i-1},e_i)}(x) = \frac{\#\text{ of combinations in } B_i \text{ s.t. } e_i \text{ happens}}{2^{kn}}, \tag{14}$$

which means that the sum of the probabilities of this not normalized distribution P_{F_i} is $2^{(n_i-n)k}$, instead of 1.

Complexity. The technique described above requires time

$$O(bn2^n \cdot \prod_{i=1}^{b} |E_i| + C \cdot \sum_{i=1}^{b} |E_{i-1}|), \tag{15}$$

where C is the average time required to construct one sub distribution P_{F_i}. The first sum of the complexity includes b fast Hadamard transforms and their combining procedure, and the second sum is the construction time of all sub distributions.

The trivial way to construct one sub distribution $P_{F_i|(e_{i-1},e_i)}$ is to try all possible combinations of the input variables in the block B_i, and check that the condition e_i is satisfied. This would take time $O(2^{kn_i})$. For the same time a class of sub distributions $\{e_i \in E_i : P_{F_i|(e_{i-1},e_i)}\}$ can be constructed. However, in some cases it can be done faster, if there are certain properties of the function F that can be used. This may evidently reduce the coefficient C. Unfortunately, another coefficient $\prod_{i=1}^{b} |E_i|$ can be large, making the overall time complexity high. This problem can be overcomed by the [illegible], which will be introduced in the next sub section.

Applicability. So far we have not yet discussed any possible way of a proper separation of the computations. We have to admit that this is a challenge, and no rules can be given. However, later we will give an example of this technique showing how in principal it can be done.

3.2 Time Complexity Reduction: Method of Gluing of the Events

Method of Gluing. In general, we have a set of input blocks $B_1, \ldots, B_b$, and a set of events $E_1, \ldots, E_e$. In brief, the method of events is a "divide-and-conquer" technique where one first computes sub distributions separately, and then combine them in a special way, via the events. The relation between Bs and Es can in general be represented as a connection graph. The blocks Bs represent vertices in the graph. Two vertices B_i and B_j are connected by a directed edge E_k, if E_k is the outcome event for B_i, and an income event for B_j. As an example,

in the previous subsection the blocks $B_1, \ldots, B_b$ were connected via the events $E_1, \ldots, E_b$, and the corresponding connection graph is shown in Figure 2a.

The combining process given in (13) requires to test the complete cartesian product of the events $|E_1 \times \ldots \times E_b|$, and it can be time consuming. The trick of the reduction of this complexity is in "gluing" the neighbouring blocks together one by one. For example, "gluing" of B_1 and B_2 into B_{1-2} can be done as follows.

1. for all $e_b \in E_b$ and $e_2 \in E_2$
2. $\quad P_{F_{1-2}|(e_b,e_2)} = \sum_{e_1 \in E_1} P_{F_1|(e_b,e_1)} \oplus P_{F_2|(e_1,e_2)}$

Complexity. The time complexity of one gluing is $O(C \cdot |E_b||E_1||E_2| \cdot n2^n)$. Applying this principle to the equation (13) one-by-one, as shown in Figure 2, the reduction of the time complexity can be significant. The coefficient $O(\prod_{i=1}^{b} |E_i|)$ is now decreased till $O(|E_b| \sum_{i=1}^{b-1} |E_i||E_{i+1}|)$, and the overall complexity is reduced to

$$O\left(bn2^n|E_b| \sum_{i=1}^{b-1} |E_i||E_{i+1}| + C \cdot \sum_{i=1}^{b} |E_{i-1}|\right). \tag{16}$$

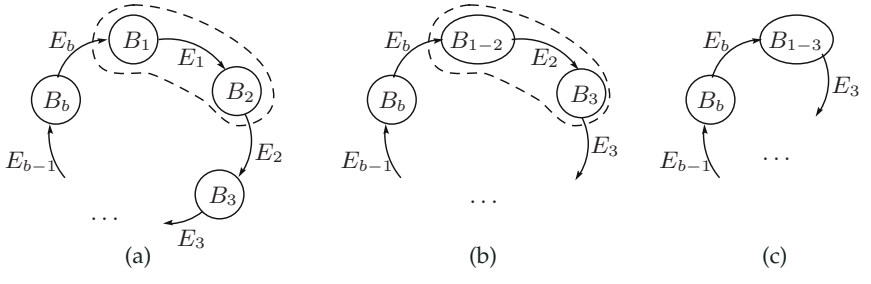


Fig. 2. Gluing method for a simple connection graph of the events

Complicated Cases. Presented techniques are not limited to a simple chain of blocks connected by events as shown in Figure 2. The connection graph can in general have a complicated case, such as in Figure 3. In the graph there are two cliques of sizes 2 and 3. Any merging of blocks in the same clique of size q would require to consider at least q events as a cartesian product. For example, merging B_4 and B_5 into B_{4-5} would require to consider all input and output events connected to the vertices B_4 and B_5, which are E_3, E_4, E_5. In another order of gluing, for example, starting with B_3 and B_5, an additional event E_1 is required to be considered, and the time complexity would increase. To find an optimal order to merge the blocks one can use standard algorithms from computer science.

Let us consider one example from linear cryptanalysis, in order to demonstrate the details of the new technique.

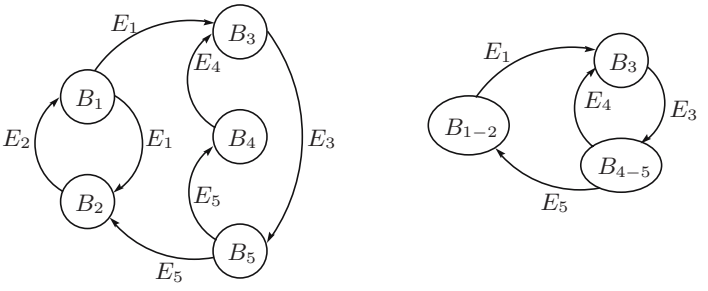


Fig. 3. A more complicated case

Let $n = 32, k = 3$, and the function (noise variable) N is the following, taken from the real cryptanalysis practice.

$$N = (A \boxplus B \boxplus C) \oplus A \oplus M_\alpha B \oplus S(C), \tag{17}$$

where M_α is a matrix representing multiplication by α in a field $\mathbb{F}_{2^{32}}$, and $S(X)$ is an S-box: $\mathbb{F}_{2^8}^4 \to \mathbb{F}_{2^8}^4$, defined as

$$S(X) = S_R(X[31..24]) \,||\, S_R(X[23..16]) \,||\, S_R(X[15..8]) \,||\, S_R(X[7..0]),$$

where $S_R(\cdot)$ is the Rijndael bytewise S-box. We are interested in the distribution P_N of the noise variable.

The solution can be done in the following way. One convenient division of the 32 bits into separate blocks is the bytewise division. Let us for simplicity represent the 32-bit word A in a bytewise form as $A = (a_0, a_1, a_2, a_3)$, and similar for B and C. The function N can be rewritten as

$$N = \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \boxplus \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} \boxplus \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}}_{\begin{array}{l} a_0 \boxplus b_0 \boxplus c_0 \\ a_1 \boxplus b_1 \boxplus c_1 \\ a_2 \boxplus b_2 \boxplus c_2 \\ a_3 \boxplus b_3 \boxplus c_3 \end{array} \begin{array}{l} \leftarrow E_1 \\ \leftarrow E_2 \\ \leftarrow E_3 \end{array}} \oplus \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \oplus M_\alpha \cdot \left(\begin{pmatrix} b_0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ b_1 \\ 0 \\ 0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 0 \\ b_2 \\ 0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 0 \\ 0 \\ b_3 \end{pmatrix} \right) \oplus \begin{pmatrix} S_R(c_0) \\ S_R(c_1) \\ S_R(c_2) \\ S_R(c_3) \end{pmatrix} \tag{18}$$

Three events E_1, E_2, E_3 are the carry values due to the arithmetical addition of the arguments. Since the carry value can be one of $\{0, 1, 2\}$, the size of each event space is 3. The nonlinear function of N can now be separated into the sum (XOR) of four independent sub functions $N = N_0 \oplus N_1 \oplus N_2 \oplus N_3$, which are connected together via the events.

The construction of one sub distribution P_{N_i}, $i = 0, 1, 2, 3$, takes 2^{24} of time, and the total time to construct all sub distributions is $10 \cdot 2^{24}$. One convolution requires 2^{37} of time. If the gluing technique is used, then the number of convolutions needed to be done is 7 (to create B_{0-1} and B_{2-3} requires 3 convolutions for each operation, plus one for the final convolution). Thus, the total time complexity to construct this distribution of the noise variable is $10 \cdot 2^{24} + 7 \cdot 2^{37} \approx 2^{39.8}$. □

The example above demonstrates how the new technique can be applied in linear cryptanalysis to receive the distribution of the noise in a feasible time. Note that in the classical way of the calculations would require time 2^{96}.

3.3 Supplementary Techniques: Substitution and Prediction

In this section two additional minor techniques are given in brief. The expression of the function F can be simplified via the [illegible]. In some cases it could also convert an unsolvable problem to a solvable one.

[illegible] is that a random variable X_i can be substituted by another random variable $Y_i = \mu_i(X_i)$, where μ_i is [illegible] one-to-one mapping function $\mu_i : \mathcal{X} \to \mathcal{X}$. Since X_i is uniformly distributed, the new variable Y_i is uniformly distributed as well. Therefore, such a substitution does not harm the final distribution of the function P_F, rather the form of the function can be changed significantly.

[illegible] is that a similar substitution can be applied to the whole expression F. Instead of searching for P_F one could search for $P_{\mu(F)}$ for some mapping μ. The biases of P_F and $P_{\mu(F)}$ are the same, since μ is just a permutation of the distribution table P_F.

As an example, consider the following expression of the noise variable

$$N = \beta(X \boxplus \alpha Y) \oplus X \oplus Y \quad \in \mathbb{F}_{2^{32}}, \tag{19}$$

where α, β are elements of some finite field $\mathbb{F}_{2^{32}}$. The distribution of N is not possible to calculate in this form. However, if we make a substitution $Y = \alpha^{-1}Z$ and multiply everything by β^{-1}, the situation is much better:

$$\beta^{-1}N = (X \boxplus Z) \oplus \beta^{-1}X \oplus (\beta\alpha)^{-1}Z. \tag{20}$$

For this type of expression its distribution can be calculated by the method of events.

[illegible] is that an operation over two random variables of sizes n_1 and n_2 (let $n_1 \leq n_2$) bits requires time around $O(f(n_2))$, where f is some complexity function. The technique of prediction is possible when $n_2 - n_1$ bits of the result can be [illegible] in advance. For example, if $X = (x \quad 0)$ and $Y = (y_1 \quad y_2)$ are (2n)-bit variables, their convolution can efficiently be done in time $O(n2^n)$, instead of $O(2n2^{2n})$. This reduction of the complexity is possible since the half of the bits of the result can be predicted.

4 Conclusions

In this paper we have shown an advantage of a multidimensional analysis against binary analysis, and several new techniques to calculate distribution of various complicated functions were presented. We have applied these algorithms to attack on SNOW 2.0. [4]. These new techniques allowed us to compute the complete distribution of the noise variable (see Example 1) in a week on the usual PC with the processor Intel Core2 6700 at 2.66GHz with 4Gb of memory and 400Gb of HDD. The resulting bias appeared to be around 2^{-85}, which is the best known result so far. We skip the details since it does not differ very significantly from the previously best known bias. In fact, these techniques can be very useful in other further cryptanalysis as well.

References

1. Schneier, B.: Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd edn. John Wiley&Sons, New York (1996)
2. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
3. Halevi, S., Coppersmith, D., Jutla, C.S.: Scream: A software-efficient stream cipher. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 195–209. Springer, Heidelberg (2002)
4. Ekdahl, P., Johansson, T.: A new version of the stream cipher SNOW. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 47–61. Springer, Heidelberg (2003)
5. Briceno, M., Goldberg, I., Wagner, D.: A pedagogical implementation of A5/1 (1999) (accessed August 18, 2003), available at http://jya.com/a51-pi.htm
6. SIG Bluetooth. Bluetooth specification (2003) (accessed August 18, 2003), available at http://www.bluetooth.com
7. Smart, N.: Cryptography: An Introduction. McGraw-Hill, New York (2003)
8. Hawkes, P., Rose, G.G.: Primitive specification and supporting documentation for SOBER-t16 submission to NESSIE. In: Proceedings of First Open NESSIE Workshop (2000) (accessed October 5, 2003) available at http://www.cryptonessie.org
9. Hawkes, P., Rose, G.G.: Primitive specification and supporting documentation for SOBER-t32 submission to NESSIE. In: Proceedings of First Open NESSIE Workshop (2000) (accessed October 5, 2003) available at http://www.cryptonessie.org
10. Watanabe, D., Biryukov, A., De Canniere, C.: A distinguishing attack of SNOW 2.0 with linear masking method. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 222–233. Springer, Heidelberg (2004)
11. Johansson, T., Maximov, A.: A linear distinguishing attack on Scream. In: Information Symposium in Information Theory—ISIT 2003, p. 164. IEEE Computer Society Press, Los Alamitos (2003)
12. Biham, E., Dunkelman, O.: Cryptanalysis of the A5/1 GSM stream cipher. In: Roy, B., Okamoto, E. (eds.) INDOCRYPT 2000. LNCS, vol. 1977, pp. 43–51. Springer, Heidelberg (2000)
13. Maximov, A., Johansson, T., Babbage, S.: An improved correlation attack on A5/1. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 1–18. Springer, Heidelberg (2004)

14. Fluhrer, S.R., McGrew, D.A.: Statistical analysis of the alleged RC4 keystream generator. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 19–30. Springer, Heidelberg (2001)
15. Lu, Y., Vaudenay, S.: Cryptanalysis of bluetooth keystream generator two-level e0. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, Springer, Heidelberg (2004)
16. Ekdahl, P., Johansson, T.: Distinguishing attacks on SOBER-t16 and SOBER-t32. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 210–224. Springer, Heidelberg (2002)
17. Lipmaa, H., Moriai, S.: Efficient algorithms for computing differential properties of addition. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 336–350. Springer, Heidelberg (2002)
18. Lipmaa, H., Wallén, J., Dumas, P.: On the additive differential probability of exclusive-or. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 317–331. Springer, Heidelberg (2004)
19. Maximov, A.: On linear approximation of modulo sum. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 483–484. Springer, Heidelberg (2004)
20. Lipmaa, H.: On differential properties of pseudo-Hadamard transform and related mappings. In: Menezes, A.J., Sarkar, P. (eds.) INDOCRYPT 2002. LNCS, vol. 2551, pp. 48–61. Springer, Heidelberg (2002)
21. Maximov, A., Johansson, T.: Fast computation of large distributions and its cryptographic applications. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 313–332. Springer, Heidelberg (2005)

Passive Attacks on a Class of Authentication Protocols for RFID

Basel Alomair, Loukas Lazos, and Radha Poovendran

Network Security Lab.
Electrical Engineering Department
University of Washington
{alomair,llazos,rp3}@u.washington.edu

Abstract. Mutual authentication mechanisms can be used in RFID systems to preserve the confidentiality of the RFID tags. Hiding the unique IDs of the tags is critical to prevent unauthorized tag tracking. In this paper, we analyze two mutual authentication protocols called M^{2}AP and EMAP, recently proposed by Peris-Lopez et. al. We show that a *passive* adversary eavesdropping on the open wireless medium, can extract the unique ID of the RFID tag by collecting an expected $O(\log_2 L)$ challenge-response exchange messages between the tag and the reader, where L is the length of the tag's unique ID. To date, previously known attacks on M^{2}AP and EMAP require the active probing of each tag. Furthermore, attacks on M^{2}AP require $O(L)$ active queries to be sent to the tag by a rogue reader, as opposed to $O(\log_2 L)$.

Keywords: RFID, authentication, privacy, passive attack.

1 Introduction

Radio Frequency Identification (RFID) systems enable the unique identification of an item with an embedded RFID tag. Making use of radio frequency based technology, RFID tags can be scanned in a non-line-of-sight manner and can be batch processed [19]. Hence, RFID systems facilitate a variety of applications such as, supply chain management, inventory tracking, building access control, and smart home appliances.

RFID systems consist of three main components: RFID tags, RFID readers, and a database. To obtain the ID from an RFID tag, the reader requests for the tag's ID. The tag responds with a quantity that can be uniquely associated with its ID. The reader looks up the tag's ID in the database to obtain related information such as a detailed description of the product carrying the RFID tag. Unlike bar codes, RFID tags are associated with a unique identifier that can be linked to the individual product, not only to the product type. Since every tag carries a unique ID, tracking individual tags is feasible via a relatively low-cost RFID reader, thus compromising the privacy of the tag and eventually of the owner of the product [16]. As an example, scanning parked cars with an RFID reader can reveal which one has more valuables inside. Furthermore, an individual can be tracked simply by tracking the ID of any RFID tag he/she carries.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 102–115, 2007.

To prevent tracking of the tag by its unique ID, the tag must respond only to queries originated by authorized parties (readers). Furthermore, readers must have a mechanism to verify that any response to their queries comes only from the valid tags. These properties can be guaranteed if the reader and the tag can mutually authenticate one another.

While the problem of mutual authentication has well known solutions for computationally capable devices [18], it becomes particularly challenging in RFID systems due to the stringent hardware constraints of RFID tags. At present, a typical low-cost RFID tag has few thousands gates, in which only few hundreds of them can be dedicated to security [12]. Public Key Encryption (PKI) is beyond the computational power of the RFID tags due to the required exponentiations [12]. Even the symmetric encryption algorithms, like AES, typically require, on the order of, 20,000-30,000 gates [12][1], while cryptographic hash functions, such as SHA-1, are also too costly to be used in low-cost RFID tags [12]. In [1,2], Peris-Lopez et. al. have recently proposed two extremely lightweight protocols, called M^{2}AP and EMAP, where tags were assumed to have minimal computational power able to perform only bitwise XOR ($\oplus$), AND ($\wedge$), OR ($\vee$), and modulo addition operations. The basic idea behind M^{2}AP and EMAP is to use a temporary [illegible] (IDP) to hide the tag ID when communicating with a reader. The tag responds to the reader queries with an IDP, that can be linked to the tag's unique ID only by authorized readers.

Our contributions. In this paper, we analyze two lightweight mutual authentication protocols [1,2], called M^{2}AP and EMAP. We show how an adversary eavesdropping on the wireless channel can breach the confidentiality of the communication by extracting the tag's unique ID. Our attack model does not require the ability to modify the contents of transmitted messages, nor does it require the ability to actively probe tags; simple bitwise operations are sufficient to extract the unique ID of the tag. We provide probabilistic analysis of our attacks on both protocols and show that the problem of extracting the tag's ID can be mapped to a set cover problem. Our mapping shows that the number of protocol runs needed to extract the tag's unique ID is [illegible] in the length of the ID. Our attacks are [illegible] and require eavesdropping of only $O(\log_2 L)$ protocol runs, as opposed to the [illegible] attacks presented in [3,4].

The rest of the paper is organized as follows. In Section 2, we state our assumptions. In Section 3, we describe the M^{2}AP and EMAP mutual authentication protocols. In Section 4, we describe attacks against the M^{2}AP and EMAP, and provide probabilistic analysis of our passive attacks against them. In Section 5, we present related work. We present our conclusions Section 6.

2 Adversarial Model

We assume a passive adversary able to eavesdrop on messages exchanged between legitimate RFID tag-reader pairs. We also assume that the adversary can store

[1] In [20], however, Feldhofer et al. described an AES implementations for RFID which requires about 3600 gates.

the messages it observes. Although a passive adversary is close to the weakest adversary one can have, our adversary, however, is a rather weak adversary as it only requires the ability to perform simple bit-wise operations and modulo additions. We do not consider an active adversary able to probe tags as in [3,4].

3 Description of the M^{2}AP and EMAP Protocols

3.1 The M^{2}AP Mutual Authentication Protocol

In the M^{2}AP protocol [1], each tag stores three quantities: the tag's secret unique ID, an IDP, and a secret key K=$K_1 \parallel K_2 \parallel K_3 \parallel K_4$, where $\parallel$ denotes the concatenation operation. For each tag, the IDP and secret key K are stored in the database. The tag's unique ID is static while the IDP and the key K are updated after every successful mutual authentication run. As a mutual authentication run, or protocol run, we define the execution of the following steps that lead to the mutual authentication of the reader-tag pair and the update of the IDP and K.

STEP 1: –Initially, the reader sends a '*hello*' message to the tag which responds with its current IDP. Using the IDP, the reader retrieves the key K from the database.

STEP 2: –After receiving the IDP, and retrieving K, the reader generates two fresh random numbers (nonces), n_1 and n_2, and forwards the following three messages, A, B, and C in the clear, to the tag:

$$A = IDP \oplus K_1 \oplus n_1,\ B = (IDP \wedge K_2) \vee n_1,\ C = IDP + K_3 + n_2. \quad (1)$$

Upon receiving A, B, and C, the tag extracts the nonce n_1 from A as $n_1 = A \oplus IDP \oplus K_1$, and authenticates the reader by checking that $B = (IDP \wedge K_2) \vee n_1$. If authentication of B fails, the tag does not respond to the reader.

STEP 3: –After the reader has been authenticated, the tag extracts the nonce n_2 from message C as $n_2 = C - IDP - K_3$, and generates two messages, D and E, as follows:

$$D = (IDP \vee K_4) \wedge n_2, \quad E = (IDP + ID) \oplus n_1. \quad (2)$$

The reader authenticates the tag, by checking that $D = (IDP \vee K_4) \wedge n_2$.

STEP 4: –The reader extracts the tag's unique ID from the message E as $ID = (E \oplus n_1) - IDP$.

STEP 5: –The reader and the tag update the IDP and K as follows:

$$IDP^{(n+1)} = (IDP^{(n)} + (n_2 \oplus n_1)) \oplus ID,$$

$$K_1^{(n+1)} = K_1^{(n)} \oplus n_2 \oplus (K_3^{(n)} + ID),\ K_2^{(n+1)} = K_2^{(n)} \oplus n_2 \oplus (K_4^{(n)} + ID),$$

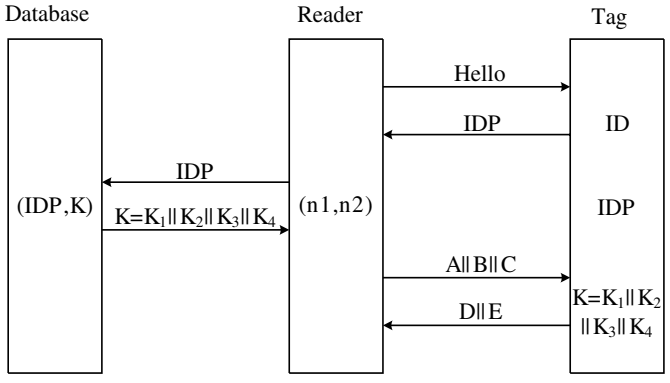


Fig. 1. The M^{2}AP and EMAP protocols. The reader interrogates the tag which responds with its current IDP. Using the tag's IDP, the reader looks up the database for the corresponding key K. The reader combines two random numbers (n_1,n_2), with the IDP and K to generate $A \parallel B \parallel C$ and authenticate itself to the tag. The tag responds with $D \parallel E$, where the tag's unique ID is embedded in E.

$$K_3^{(n+1)} = (K_3^{(n)} \oplus n_1) + (K_1^{(n)} \oplus ID),\ K_4^{(n+1)} = (K_4^{(n)} \oplus n_1) + (K_2^{(n)} \oplus ID).$$

The updated IDP and K are mutually stored for the next protocol run.

3.2 The EMAP Mutual Authentication Protocol

EMAP follows the same five steps described in M^{2}AP, with the only difference being the way that messages B, C, D, and E are generated and how the IDP and K are updated. In EMAP, the messages A, B, C, D, and E are generated as follows:

$$A = IDP \oplus K_1 \oplus n_1,\ B = (IDP \vee K_2) \oplus n_1,\ C = IDP \oplus K_3 \oplus n_2, \tag{3}$$

$$D = (IDP \wedge K_4) \oplus n_2,\ E = (IDP \wedge n_1 \vee n_2) \oplus ID \bigoplus_{i=1}^{4} K_i. \tag{4}$$

The updating of the IDP and K works as follows:

$$IDP^{(n+1)} = IDP^{(n)} \oplus n_2 \oplus K_1, \tag{5}$$
$$K_1^{(n+1)} = K_1^{(n)} \oplus n_2 \oplus (ID(1:48) \parallel F_p(K_4^{(n)}) \parallel F_p(K_3^{(n)})), \tag{6}$$
$$K_2^{(n+1)} = K_2^{(n)} \oplus n_2 \oplus (F_p(K_1^{(n)}) \parallel F_p(K_4^{(n)}) \parallel ID(49:96)), \tag{7}$$
$$K_3^{(n+1)} = K_3^{(n)} \oplus n_1 \oplus (ID(1:48) \parallel F_p(K_4^{(n)}) \parallel F_p(K_2^{(n)})), \tag{8}$$
$$K_4^{(n+1)} = K_4^{(n)} \oplus n_1 \oplus (F_p(K_3^{(n)}) \parallel F_p(K_1^{(n)}) \parallel ID(49:96)), \tag{9}$$

where $F_p(x)$ is the 24-bit sequence representing the parity of every 4 bit block of x.

4 Passive Attacks Against M²AP and EMAP

4.1 Passive Attack Against M²AP

In this section, we show how an adversary can extract the tag's unique ID by observing, on average, a [illegible] (in the length of ID) number of mutual authentication runs between the tag and the reader. To obtain the ID, the adversary needs to observe only the IDP, B, and E message exchange in each protocol run. For clarity, we first illustrate the attack via an example. Then, we show that the problem of extracting the tag's ID can be mapped to a set covering problem. Based on our mapping, we provide a probabilistic analysis of our attack.

Example: For clarity, we only show the values of the messages observed by the adversary that are relevant to the attack. Assume that the unique ID of an RFID tag is six bit long and is '001100'. Initially, the reader broadcasts a [illegible] message to announce its presence. The tag challenges the reader by sending its current $IDP^{(1)} = 101100$. The reader looks up the database to find the corresponding secret key $K^{(1)}$, generates two random numbers $(n_1^{(1)},n_2^{(1)})$, and challenges the tag with $A^{(1)} \parallel B^{(1)} \parallel C^{(1)}$, where $B^{(1)} = 011000$. After the reader is authenticated, the tag responds with $D^{(1)} \parallel E^{(1)}$, where $E^{(1)} = 101000$. Notice that, from message B in equation (1), if $(IDP)_i = 0$, then $(n_1)_i = (B)_i$, where the i subscript denotes the i^{th} bit of IDP, n_1, and B, respectively. Thus, the adversary can compute $n_1^{(1)} = *1**00$, where '$*$' represents an unknown bit. Therefore, $E^{(1)} \oplus n_1^{(1)} = *1**00$, and substituting into (2) we get:

$$ID = (E^{(1)} \oplus n_1^{(1)}) - IDP^{(1)} = *1**00 + 010100, \tag{10}$$

From the first protocol run, the adversary identifies the two least significant bits of the tag's ID as '00' using (10).

In the next protocol run, the adversary gathers the quantities $IDP^{(2)} = 010001$, $B^{(2)} = 111001$, and $E^{(2)} = 100100$. Using B in equation (1), the adversary computes $n_1^{(2)} = 1*100*$; hence, $E^{(2)} \oplus n_1^{(2)} = 0*110*$, and a second equation for the tag's ID can be constructed:

$$ID = (E^{(2)} \oplus n_1^{(2)}) - IDP^{(2)} = 0*110*+101111. \tag{11}$$

Substituting the two least significant bits '00' in (11), the adversary can compute $ID = **1100$. By substituting '$**1100$' back in (10), the adversary identifies the fifth bit of the ID as '0', and substituting the five known bits of the ID back in (11), the adversary identifies the tag's unique ID as '001100'. Figure 2 presents the protocol exchanges for the two instances of mutual authentication between the tag and the reader in our example.

Let $x^{(n)} = E^{(n)} \oplus n_1^{(n)}$ denote the first term of the right hand side in (10) and let $(m)_i$ denote the i^{th} bit of message m. Note from our example that $(x^{(n)})_i$ is known if $(IDP^{(n)})_i = 0$. The set of equations, similar to (10) and (11), constructed by the adversary by observing protocol runs, can be solved

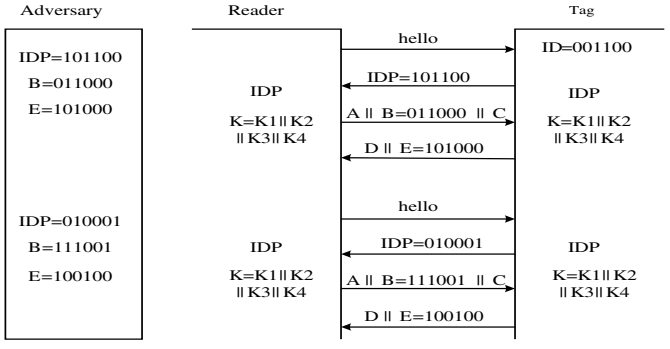


Fig. 2. Two mutual-authentication message exchanges between a reader and a tag. To the left is the information collected by an adversary that lead to disclosing the tag's unique ID

for the ID if $(x^{(n)})_i$ is known for one n. This condition is equivalent to $(IDP^{(n)})_i = 0$ for one n. That is, each position of the observed IDP's has a zero in one IDP. In turn, this observation can be to a variant of the set cover problem expressed in the following lemma.

Lemma 1. $S = \{1, 2, ..., L\}$ L ID $S_0^{(n)} = \{i \mid the\ i^{th}\ bit\ of\ the\ n^{th}\ IDP\ is\ 0\}$ ID $\cup_n S_0^{(n)} = S$ $S_0^{(n)}$ S

Let $\cup_n S_0^{(n)} = S$. Based on (2), we have

$$(ID) = ((E^{(n)}) \oplus (n_1^{(n)})) - (IDP^{(n)}) = x^{(n)} - (IDP^{(n)}). \tag{12}$$

For each bit of the ID we can write

$$(ID)_i = (x^{(n)})_i - (IDP^{(n)})_i + C_i^{(n)} \quad (mod\ 2), \tag{13}$$

where

$$C_i^{(n)} = f((x^{(n)})_{i-1}, (IDP^{(n)})_{i-1}, C_{i-1}^{(n)}) \tag{14}$$

denotes the carry from the modulo 2 addition in (12). To compute the carry C_i, we must know $(ID)_{i-1}$ from the LSB up to the $(i-1)^{th}$ bit. For the LSB, $C_1 = 0$ and hence, $(ID)_1$ can be extracted from any $S^{(n)}$ with $1 \in S^{(n)}$. Once $(ID)_1$ has been extracted, using (13), $(x^{(n)})_1$ can be extracted for all n. Therefore, using (14), C_2 can be extracted for all n. Now, equation (13), can be used to solve for $(ID)_2$ from any $S^{(n)}$ with $2 \in S^{(n)}$. Since for all $i \in [1 : L]$, there exists an $S^{(n)}$ with $i \in S^{(n)}$, one can recursively solve for the tag's ID from the least significant to the most significant bits.

Given that the bit values of the IDP are drawn from a probability distribution, we can compute the average number of protocol runs required to recover the unique ID using the following lemma.

Table 1. Mapping the ID recovery problem to a set cover problem

ID **recovery problem**	$\leftrightarrow$	**Set covering problem**
$S = \{1, 2, ..., L\}$	$\leftrightarrow$	Entire set S
$IDP^{(n)}$ containing k zeros	$\leftrightarrow$	Subset of S with cardinality k
Observing protocol runs with IDP's having at least one zero in every position	$\leftrightarrow$	Finding a set of subsets of S that covers S

Lemma 2. *p ... IDP ... L ... IDP ... ID ... m ...*

$$\Pr(\textit{disclosing the ID after } m \textit{ messages}) = (1 - p^m)^L. \tag{15}$$

... $\epsilon \in (0, 1)$ ... ID ... $1 - \epsilon$ ...

$$m = \lceil \frac{\ln(1 - \exp^{\frac{\ln(1-\epsilon)}{L}})}{\ln p} \rceil. \tag{16}$$

... The proof of lemma 2 is provided in the appendix.

From lemma 2, we observe that the probability of extracting the ID of the tag is a monotonically increasing function of the number of observed protocol runs m, converging to 1. Lemma 2 also specifies how many protocol runs one must eavesdrop, before the entire ID can be extracted, with a desired probability. In Figure 3(a), we show the probability of extracting the ID of the tag as a function of the number of protocol runs observed, for different values of L and for $p = \frac{1}{2}$. We now compute the average number of protocol runs an adversary needs to eavesdrops to extract the tag's ID.

Lemma 3. *p ... IDP ... L ... IDP ... m ... ID ... m ...*

$$E[m] = \sum_{k=1}^{L} \binom{L}{k} \frac{(-1)^{k+1}}{1 - p^k}. \tag{17}$$

... The proof of lemma 3 is provided in the appendix.

For $L = 96$ and $p = \frac{1}{2}$ as specified in [1], the expected number of protocol runs needed to extract the tag's unique ID is 7.9252. Figure 3 (b) shows the analytically derived relation between the expected number of protocol runs needed to extract the tag's ID and the length of the ID. We observe that $E[m]$ grows linearly with the logarithm of the ID length L.

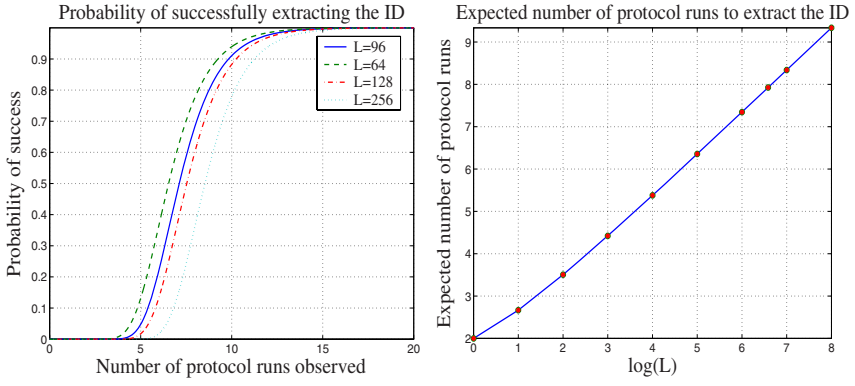


Fig. 3. (a) The probability of extracting the ID of the tag as a function of the number of protocol runs observed, for varying ID lengths, (b) The expected number of messages needed to extract the ID as a function of the length of the ID in a logarithmic scale

4.2 Passive Attack Against EMAP

In EMAP [2], a particular emphasis is put on the properties of message E in equation (4), due to the fact that the tag's unique ID is extracted via E [2]. However, the ID is also used in equations (6) - (9) for the key updating. Therefore, an adversary can extract the tag's ID using equations (6) - (9), without breaking message E. Let S_0 denote the set of indexes where the bits of the IDP are 0 and S_1 denote the set of indexes where the bits of the IDP are 1, that is,

$$S_0 = \{i \mid the\ i^{th}\ bit\ of\ IDP\ is\ 0\}, \tag{18}$$

$$S_1 = \{i \mid the\ i^{th}\ bit\ of\ IDP\ is\ 1\}. \tag{19}$$

The attack against EMAP consists of the following steps:

Step 1: From message B in equation (3), we have $(IDP)_i \vee (K_2)_i = 1,\ \forall i \in S_1$, regardless of the values of $(K_2)_i$. Therefore, $(n_1)_i = \overline{(B)_i},\ \forall i \in S_1$, where $\overline{b}$ denotes the complement of the bit b.

Step 2: Message A in equation (3) has two unknowns, namely, the secret key K_1 and the nonce n_1. The partial information about n_1 obtained from Step 1 can be substituted in (3) to extract bits of K_1,

$$(K_1)_i = (A)_i \oplus (n_1)_i \oplus (IDP)_i, \quad \forall\ i \in S_1. \tag{20}$$

Step 3: From the message D in equation (4), we have $(IDP)_i \wedge (K_4)_i = 0,\ \forall i \in S_0$ regardless of the values $(K_4)_i$. Therefore, $(n_2)_i = (D)_i,\ \forall i \in S_0$.

Step 4: Equation (5) has two unknowns, the secret key K_1 and the nonce n_2. The partial information about n_2 obtained in Step 3 can be substituted in (5) to extract bits of K_1 as follows:

$$(K_1)_i = (IDP^{(n+1)})_i \oplus (IDP^{(n)})_i \oplus (n_2)_i, \quad \forall\, i \in S_0. \tag{21}$$

Up to this point, the adversary knows $(K_1)_i,\ \forall i \in S_1$ from (20), and $(K_1)_i,\ \forall i \in S_0$ from (21). Hence, the secret key K_1 has been fully extracted.

Step 5: By substituting K_1 into (3) and (5), the adversary obtains n_1 and n_2 as shown below:

$$n_1 = A \oplus IDP^{(n)} \oplus K_1^{(n)},\ n_2 = IDP^{(n+1)} \oplus IDP^{(n)} \oplus K_1^{(n)}. \tag{22}$$

Step 6: By eavesdropping the next protocol run, the adversary can extract the updated value $K_1^{(n+1)}$ as described in Steps 1-4. Substituting the values of $K_1^{(n)}$ and $K_1^{(n+1)}$ in (6), the first half of the tag's unique ID is revealed:

$$(ID)_i = (K_1^{(n+1)})_i \oplus (K_1^{(n)})_i \oplus n_2, \quad \forall i \in [1 : \frac{L}{2}]. \tag{23}$$

Step 7: From messages B and D in equations (3) and (4) respectively, $(K_2)_i = (B)_i \oplus (n_1)_i,\ \forall i \in S_0$ and $(K_4)_i = (D)_i \oplus (n_2)_i,\ \forall i \in S_1$. Therefore, in every protocol run, the bits of K_2 corresponding to the zero bits of IDP are known, and the bits of K_4 corresponding to the one bits of IDP are known. Thus, for $i = \frac{L}{2} + 1 : L$, if $(IDP^{(n)})_i = (IDP^{(n+1)})_i = 0$ then $(K_2^{(n)})_i$ and $(K_2^{(n+1)})_i$ are known and, hence, using equation (7),

$$(ID)_i = (K_2^{(n)})_i \oplus (K_2^{(n+1)})_i \oplus (n_2)_i. \tag{24}$$

Likewise, if $(IDP^{(n)})_i = (IDP^{(n+1)})_i = 1$ then $(K_4^{(n)})_i$ and $(K_4^{(n+1)})_i$ are known, and using equation (9),

$$(ID)_i = (K_4^{(n)})_i \oplus (K_4^{(n+1)})_i \oplus (n_1)_i. \tag{25}$$

Using (24) and (25), the second half bits of the ID are extracted if two consecutive IDP's have the same bit value in that position. Hence, the adversary can solve for each bit in the second half of the ID depending on the value of the IDP's in its position. Lemma 4 analyzes the performance of our passive attack against EMAP.

Lemma 4. [illegible] p [illegible] IDP [illegible] L [illegible] IDP [illegible] ID [illegible] m [illegible]

$$Pr(\text{[illegible]}) = \begin{cases} 0, & m < 2 \\ (1 - (2p - 2p^2)^m)^{\frac{L}{2}}, & m \geq 2 \end{cases} \tag{26}$$

[illegible] $\epsilon \in (0,1)$ [illegible] ID [illegible] $1 - \epsilon$ [illegible]

$$m = \lceil 1 + \frac{\ln(1 - \exp^{\frac{2\ln(1-\epsilon)}{L}})}{\ln(2p - 2p^2)} \rceil. \tag{27}$$

ID.

$$E[m] = 1 + \sum_{k=1}^{\frac{L}{2}} \binom{\frac{L}{2}}{k} \frac{(-1)^{k+1}}{1-(2p-2p^2)^k}. \quad (28)$$

In our attack, no information about the ID can be extracted by just eavesdropping the first protocol run. However, by eavesdropping two consecutive protocol runs, the adversary is guaranteed to recover the first half of the ID (Steps 1-6). For the second half of the ID, bits are recovered probabilistically by solving the update equation of K_2 or K_4 (Step 7). When two consecutive IDP's have the same value in one bit position, the adversary can solve for the bit of the ID at that position. That is, for the i^{th} bit of ID, the adversary will successfully solve for its value if $(IDP^{(n)})_i = (IDP^{(n+1)})_i$, both are 0 or 1 which occur with probabilities $(1-p)^2$ and p^2, respectively. This means that the probability of successfully solving for each bit of the second half of the ID is given by $(1-p)^2+p^2$. Also note that, to extract all bits of the second half of the ID, a match between two consecutive IDPs has to occur, $\forall i \in [\frac{L}{2}+1 : L]$. Hence, the problem of extracting all second half ID bits, can be mapped to the same set covering problem expressed in Lemma 1, with a different success probability, and ID length equal to $\frac{L}{2}$. Following the same analysis as in Lemma 2 and 3, we can compute the quantities in (26), (27), and (28) by substituting the success probability for M^{2}AP '$(1-p)$', with $(1-p)^2+p^2$. Note that at least two protocol runs are needed to extract useful information and, hence, 1 is added to the expressions in (27), and (28).

In Figure 4(a),(b), we show the histogram of the probability of extracting the tag ID after eavesdropping exactly m protocol runs for M^{2}AP and EMAP, respectively, for $L = 96$ and $p = 0.5$. Note that the average number of protocol runs required to extract the tag ID is 7.9273 and 7.9223 for M^{2}AP and EMAP respectively, while the theoretical result obtained by Lemma 2 shows that $E[m] = 7.9252$.

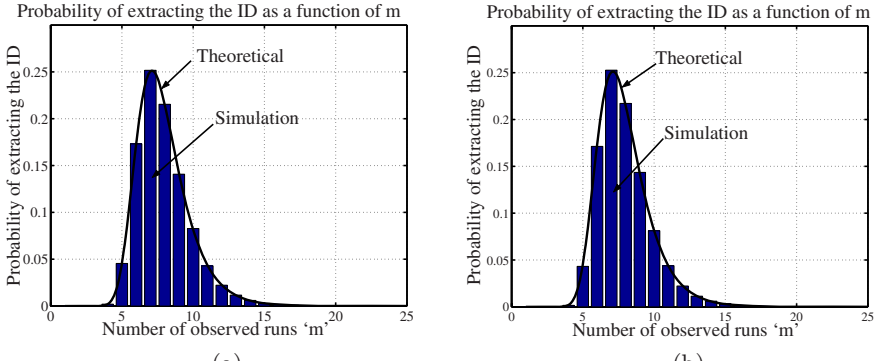


Fig. 4. Probability of extracting the tag ID after eavesdropping exactly m protocol runs when $L = 96$ and $p = 0.5$ for (a) the M^{2}AP protocol, (b) the EMAP protocol

5 Related Work

The problem of mutual authentication in RFID systems has been studied under different constraints [1,2,6,9,10,12,14]. Juels and Pappu have suggested the use of a public key cryptosystem to solve the problem of consumer privacy in RFID banknotes [9]. Avoine, however, described possible limitations on the security of the protocol in [11]. Feldhofer et.al. proposed the use of AES symmetric key cipher to achieve mutual authentication between tags and readers for tags able to perform AES Encryption/Decryption [10]. Weis et.al. described privacy and security risks for RFID systems and proposed solutions based on one way hash functions [12]. Juels proposed the use of a pool of pseudonyms for each tag to protect the privacy of the tag's ID [8].

Vajda and Buttyan proposed lightweight cryptographic primitives for tag authentication based on simple bitwise operations [14]. In [6], Juels and Weis proposed HB$^+$, a lightweight authentication protocol based on the human-to-computer authentication protocol designed by Hopper and Blum [5]. The security proof of the HB$^+$ against active attacks was based on the Learning Parity with Noise (LPN) problem. Gilbert et.al., however, showed a linear time active attack on the HB$^+$ protocol [7].

In [1,2], Peris-Lopez et.al. proposed M^{2}AP and EMAP, mutual authentication protocols that we analyze in this paper. In [3], Li and Wang describe active attacks against M^{2}AP that require $O(L)$ interactions between the adversary and the tag to extract its ID. Our attack requires passive observation of an average of $O(\log_2 L)$ protocol runs to extract the tag's ID. Li and Deng described active attacks against EMAP in [4]. Their attack relies on active probing of the tag via rogue reader.

6 Conclusion

In this paper, we addressed the problem of mutual authentication in RFID systems. We analyzed M^{2}AP and EMAP, two lightweight mutual authentication protocols and showed how a passive adversary can extract the tag's unique ID by observing, on average, a logarithmic (in the length of the ID) number of protocol runs. We provided a probabilistic analysis of our attacks by mapping the problem of extracting the tag's ID to a set covering problem.

References

1. Peris-Lopez, P., Hernandez-Castro, J., Estevez-Tapiador, J.M., Ribagorda, A.: M2AP: A Minimalist Mutual-Authentication Protocol for Low-cost RFID Tags. In: Ma, J., Jin, H., Yang, L.T., Tsai, J.J.-P. (eds.) UIC 2006. LNCS, vol. 4159, Springer, Heidelberg (2006)
2. Peris-Lopez, P., Hernandez-Castro, J., Estevez-Tapiador, J.M., Ribagorda, A.: EMAP: An Efficient Mutual Authentication Protocol for Low-cost RFID Tags. In: OTM Federated Conferences and Workshop: IS Workshop. LNCS, Springer, Heidelberg (2006)

3. Li, T., Wang, G.: Security Analysis of Two Ultra-Lightweight RFID Authentication Protocols. In: IFIP SEC (2007)
4. Li, T., Deng, R.H.: Vulnerability Analysis of EMAP - An Efficient RFID Mutual Authentication Protocol. In: AReS 2007: Second International Conference on Availability, Reliability and Security (2007)
5. Hopper, N.J., Blum, M.: Secure Human Identification Protocols. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, Springer, Heidelberg (2001)
6. Juels, A., Weis, S.: Authenticating Pervasive Devices with Human Protocols. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, Springer, Heidelberg (2005)
7. Gilbert, H., Robshaw, M., Sibert, H.: An Active Attack Against HB^+ – A provably Secure Lightweight Authentication Protocol Protocol (2005)
8. Juels, A.: Minimalist Cryptography for Low-Cost RFID Tags. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, Springer, Heidelberg (2005)
9. Juels, A., Pappu, R.: Squealing Euros: Privacy Protection in RFID-Enabled Banknotes. In: Wright, R.N. (ed.) FC 2003. LNCS, vol. 2742, Springer, Heidelberg (2003)
10. Feldhofer, M., Aigner, M., Dominikus, S.: An Application of RFID Tags using Secure Symmetric Authentication. In: SecPerU 2005. International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing (2005)
11. Avoine, G.: Privacy Issues in RFID Banknote Protection Schemes. In: International Conference on Smart Card Research and Advanced Applications – CARDIS (2004)
12. Weis, S., Sarma, S., Rivest, R., Engels, D.: Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems. In: SPC 2003. International Conference on Security in Pervasive Computing (2003)
13. Ohkubo, M., Suzuki, K., Kinoshita, S.: Efficient Hash-Chain Based RFID Privacy Protection Scheme. In: Davies, N., Mynatt, E.D., Siio, I. (eds.) UbiComp 2004. LNCS, vol. 3205, Springer, Heidelberg (2004)
14. Vajda, I., Buttyán, L.: Lightweight Authentication Protocols for Low-Cost RFID Tags. In: Dey, A.K., Schmidt, A., McCarthy, J.F. (eds.) UbiComp 2003. LNCS, vol. 2864, Springer, Heidelberg (2003)
15. Defend, B., Fu, K., Juels, A.: Cryptanalysis of Two Lightweight RFID Authentication Schemes. In: International Workshop on Pervasive Computing and Communication Security – PerSec (2007)
16. Juels, A.: RFID Security and Privacy: A research Survey (2005)
17. Avoine, G.: Bibliography on Security and Privacy in RFID Systems http://lasecwww.epfl.ch/gavoine/rfid/
18. Buchmann, J.A.: Introduction to cryptography. Springer, Heidelberg (2004)
19. Garfinkel, S.L., Juels, A., Pappu, R.: RFID Privacy: An overview of Problems and Proposed Solutions. IEEE Security & Privacy (2005)
20. Feldhofer, M., Dominikus, S., Wolkerstorfer, J.: Strong Authentication for RFID Systems using the AES Algorithm. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, Springer, Heidelberg (2004)

Appendix

Let $(M^{(i)})_j$ denote the j^{th} bit of the i^{th} message. Define $S^{(m)} = \bigcap_{k=1}^{m} M^{(k)}$ to be the result of bitwise AND for the messages $M^{(1)}$ through $M^{(m)}$, and define the random variable X_i as follows:

$$X_i = \begin{cases} 1, & \sum_{j=1}^{L} (S^{(i)})_j \;=\; 0, \\ 0, & otherwise. \end{cases} \tag{29}$$

Then, if the probability of any bit of $M^{(i)}$ being 1 is equal to p, we get

$$\Pr((S^{(m)})_k = 1) = \Pr(\bigcap_{l=1}^{m}\{M_k^{(l)} = 1\}) = \prod_{l=1}^{m}\Pr\{M_k^{(l)} = 1\} = p^m, \tag{30}$$

$$Pr(S_k^{(m)} = 0) = 1 - Pr(S_k^{(m)} = 1) = 1 - p^m. \tag{31}$$

Proof of Lemma 2

From equation (31) and by independence of bits, we get:

$$\Pr(X_m = 1) = (1 - p^m)^L. \tag{32}$$

Therefore, for any $\epsilon > 0$, we have:

$$\begin{aligned}
\Pr(X_m = 1) &> 1 - \epsilon \\
&\Rightarrow (1 - p^m)^L > 1 - \epsilon \\
&\Rightarrow 1 - p^m > \exp^{\frac{\ln(1-\epsilon)}{L}} \\
&\Rightarrow p^m < 1 - \exp^{\frac{\ln(1-\epsilon)}{L}} \\
&\Rightarrow m > \frac{\ln(1 - \exp^{\frac{\ln(1-\epsilon)}{L}})}{\ln p}.
\end{aligned} \tag{33}$$

Proof of Lemma 3

Define the random variable Y to be the number of messages such that $X_m = 1$ for the first time, then Y can be written as $Y = \min_i\{X_i = 1\}$. Then, $\{Y = i\} \Leftrightarrow \{X_i = 1 \; and \; X_{i-1} = 0\}$ and, hence,

$$\begin{aligned}
\Pr(Y = i) &= \Pr(X_i = 1, X_{i-1} = 0) = \Pr(X_i = 1 \mid X_{i-1})\Pr(X_{i-1}) \\
&= \sum_{j=0}^{L-1}\Pr(X_i = 1 \mid \sum_{k=1}^{L} S_k^{(i-1)} = j)\Pr(\sum_{k=1}^{L} S_k^{(i-1)} = j),
\end{aligned} \tag{34}$$

where in equation (34) we sum over all possible number of zeros in $S^{(i-1)}$. But,

$$\Pr(X_i = 1 \mid \sum_{k=1}^{L} S_k^{(i-1)} = j) = (1 - p)^{L-j} \tag{35}$$

because for all bits where $S_k^{(i-1)} = 1$, $M_k^{(i)} = 0$ to satisfy $X_i = 1$. Equation (34) can be shown as follows: From (30), the probability of a single bit in $S^{(i-1)}$ to be equal to 1 is p^{i-1}. Therefore, by independence, the probability of having $L - j$ ones in $S^{(i-1)}$ is given by:

$$\Pr(\sum_{k=1}^{L} S_k^{(i-1)} = L - j) = (p^{i-1})^{L-j}. \tag{36}$$

Similarly, from (31), the probability of any single bit of $S^{(i-1)}$ to be equal to 0 is $1-p^{i-1}$ and, hence,

$$\Pr(S^{(i-1)} \; has \; j \; zeros) = (1-p^{i-1})^j. \tag{37}$$

Finally, there are $\binom{L}{j}$ different ways of choosing the positions of the j zeros. Thus, by combining (36) and (37), the probability of having exactly j zeros in $S^{(i-1)}$ can be written as:

$$\Pr(\sum_{k=1}^{L} S_k^{i-1} = j) = \binom{L}{j}(1-p^i)^j (p^i)^{L-j}. \tag{38}$$

From (35), (38), and $1-p^{i-1} = (1-p)\sum_{k=0}^{i-2} p^k$, it follows that,

$$\begin{aligned}
\Pr(Y=i) &= \sum_{j=0}^{L-1} (1-p)^{L-j} \; \binom{L}{j} \; (1-p^{i-1})^j \; (p^{i-1})^{L-j} \\
&= (1-p)^L \; \sum_{j=0}^{L-1} \binom{L}{j} \; (\sum_{k=0}^{i-2} p^k)^j \; (p^{i-1})^{L-j} \\
&= (1-p)^L \; [(\sum_{k=0}^{i-1} p^k)^L \; - \; (\sum_{k=0}^{i-2} p^k)^L] = (1-p^i)^L - (1-p^{i-1})^L.
\end{aligned}$$

Hence, the expected number of messages required is:

$$\begin{aligned}
E[Y] &= \sum_{i=1}^{\infty} i \; \Pr(Y=i) = \sum_{i=1}^{\infty} i \; [(1-p^i)^L - (1-p^{i-1})^L] \\
&= \sum_{i=1}^{\infty} i \; [\sum_{k=0}^{L} \binom{L}{k} (-1)^k (p^k)^i - \sum_{k=0}^{L} \binom{L}{k} (-1)^k (p^k)^{i-1}] \\
&= \sum_{i=1}^{\infty} i \; \sum_{k=0}^{L} (p^k - 1) \binom{L}{k} (-1)^k (p^k)^{i-1} = \sum_{k=1}^{L} (p^k - 1) \binom{L}{k} (-1)^k \sum_{i=1}^{\infty} i \; (p^k)^{i-1} \\
&= \sum_{k=1}^{L} (p^k - 1) \binom{L}{k} (-1)^k (\frac{1}{1-p^k})^2 = \sum_{k=1}^{L} \binom{L}{k} (-1)^{k+1} (\frac{1}{1-p^k})
\end{aligned}$$

and the lemma follows.

Side Channel Attacks on Irregularly Decimated Generators

Chuan-Wen Loe and Khoongming Khoo

DSO National Laboratories, 20 Science Park Drive, Singapore 118230
lchuanwe@dso.org.sg, kkhoongm@dso.org.sg

Abstract. We investigate three side channel attacks on ABSG, a variant of *irregularly decimated generators* (IDG). The three attacks are timing analysis, phase-shift fault analysis and bit-flipping fault analysis. We also modify the attacks to non side-channel cryptanalyses, but on the assumption that the key/IV mixing is not well designed. This paper hopes to provide more understanding on actual hardware implementations of IDG as cipher components. Finally, we combine our fault analysis on ABSG with linearization attack to cryptanalze DECIM, an ESTREAM candidate cipher. We manage to reduce the attack complexity from 2^{80} to $2^{42.5}$.

Keywords: Fault Attack, Timing Analysis, Irregularly Decimated Generators, ABSG, DECIM.

1 Introduction

Cryptanalysis traditionally has been the study of ciphers from a mathematical point of view. These attacks exploit the algorithmic nature of the cipher and compromise the security. Modern ciphers are designed to prevent these known attacks, making cryptanalysis difficult. However in recent years there is a new class of cryptanalysis that studies the implementation weakness of ciphers. These attacks study the physical information (time, power, etc) leaked by the product ciphers and compromise it. Even strong ciphers like AES and RSA are also vunerable to side channel attacks [14] [15].

In this paper, we investigate the side channel security of [illegible] (IDG). IDG is a bit generator that is irregularly clocked by a regularly clocked generator. Due to the irregular clocking, IDG generates a highly non-linear bitstream. Examples of IDGs are [illegible] (SG) [4], [illegible] (SSG) [13] and [illegible] (BSG) [8]. In this paper we focus on ABSG [9], a variant of BSG. We choose to study ABSG as it faciltates our analysis of DECIM later in the paper.

ABSG outputs bitstream at an irregular rate which average to 3 clock cycles per bit. Hence to prevent timing attack, ABSG outputs to a buffer first. The time taken to fill up the buffer varies depending on the key. Our first attack investigated timing attack on the buffer mechanism and show that different buffer size affects the security of the cipher.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 116–130, 2007.

The authors of [11] developed a phase-shift fault attack on clock controlled LFSR based stream ciphers. In our second attack, we adapt this attack and show that it can be extended on to ABSG. To perform the phase-shift attack, one must be able to inject faults into cipher and cause it to malfunction for a short period of time. When the cipher malfunctions, the cipher components will be desynchonized and the released output bitstream corresponds to shifts of the ABSG input bitstream. By analyzing the fault-induced output sequences, we are able to recover the ABSG input bitstream with reduced complexity.

In [11], the authors also developed a bit-flipping fault attack on clock controlled LFSR based stream ciphers. In our third attack, we extend this attack to ABSG. In a bit-flipping fault attack, faults are injected into the cipher to cause some ABSG input bits to be toggled. By analyzing the fault-induced output sequences, we can also recover the ABSG input bitstream with reduced complexity.

Side channel analysis are not always practical, especially if the attacker does not have access to the device. We include a section that extends the side channel attacks to non side-channel attacks. However it is only possible under the assumption that the cipher has some weak components.

Finally, we study the ESTREAM candidate cipher DECIM. It was designed by Berbain et. al. [3] and was accepted into Phase 3 of the ESTREAM project under profile 2 (hardware) [1]. Because the security of DECIM relies on the strength of ABSG, we can apply our earlier results to cryptanalyze it. We show that by combining the fault analyses of Section 4 and 5 with the linearization attack of [6], we can compromise the cipher. The attack can reduce the complexity from 2^{80} (based on exhaustive search of the 80-bit secret key) to $2^{42.5}$. This complexity can be improved to $2^{33.7}$ if we assume Coppersmith-Winograd's method for solving linear equations [5].

2 Outline of ABSG

[9] [10] showed the insecurity of BSG (earlier variant of ABSG) after it was introduced a year before. To improve the security of BSG, the authors of [9] tweaked BSG into 2 different variants, ABSG and MBSG. The security of ABSG and MBSG is similar and both require $2^{L/2}$ time and data complexity to recover the full key, where L is the linear complexity of the input stream, which can be the length of the secret key.

ABSG is a bit generator that searches for a pattern in the input bitstream and outputs a shorter bitstream. In some sense ABSG is like a compression function that convert substrings to a single bit based on a lookup table. A. Gouget et al. also showed that the rate of output is approximately 1 bit per 3 clocks cycles. The following table shows that substrings of the form 00, 101, 1001, ... are compressed to 0, while substrings of the form 11, 010, 0110, ... are compressed to 1.

Input Pattern	Output Bit
$\{00, 10^i1\}$	0
$\{11, 01^i0\}$	1

Let s = 01010011101001000111101 be the input bit sequence. The output of ABSG is 10111010.

$$\underbrace{010}_{1}\underbrace{1001}_{0}\underbrace{11}_{1}\underbrace{010}_{1}\underbrace{010}_{1}\underbrace{00}_{0}\underbrace{11}_{1}\underbrace{101}_{0}$$

3 Timing Analysis on ABSG

When ABSG takes in a bitstream, it waits for a delimiter bit before it outputs a bit. The time taken to search for the delimter for each output bit varies depending on the input bitstream. Hence, the ability to measure the lag time between each output bit will result in trival full key recovery. To prevent this attack, ABSG must output to a buffer. Only when the buffer of size B is filled, then the cipher will release the bits at a constant rate. While the bits are released from the buffer, output bits of ABSG is still injected into the buffer at a slower rate.

The size of the buffer is chosen such that the probability that the buffer will be empty before encryption is low. At the same time, designers want the buffer to be as small as possible to save resources and startup timing. [4].

It is clear that the latency to fill up the buffer is the only component that will vary with respect to the key. Hence it is natural to apply timing analysis on the buffer. The overview of the attack is to measure the total number of input bits T required to fill up the buffer of size B. The next step is to brute force all possible ways to arrange T input bits that will result in the B observed output bits. We assume the attacker has means to measure the latency accurately and hence be able to determine T.

We let each bit from the buffer (output) to be an urn and the input bits are balls. We know that each output bit must come from at least 2 input bits; therefore it implies that there must be at least 2 balls in each urn. Hence we need to arrange the additional $(T - 2B)$ balls into B urns.

If $T = L$, the time complexity to recover L ABSG input bits is the number of ways to arrange $(T - 2B)$ balls into B urns:

$$\binom{(T-2B)+B-1}{T-2B} = \binom{T-B-1}{T-2B} \tag{1}$$

If $T < L$, we need to guess the remaining $L - T$ bits. A trivial way is to brute force the remaining $L - T$ bits, increasing the complexity by 2^{L-T}. However we can improve the brute force complexity by making some intelligent guesses.

We guess that the $(B + 1)^{th}$ output bit is constructed from $L - T$ input bits. Hence if the $(B+1)^{th}$ bit is 1, we can be sure that the remaining $L-T$ input bits are 0111...10. If the guess is wrong, we make another guess that the $(B + 1)^{th}$

and $(B+2)^{th}$ output bits are constructed from $L-T$ input bits. So if the two bits are 10, the $L-T$ input bits can be 111000...01, 0101000...01 or any of the possible configuration. If the guess is still wrong, we procced to the 3 bits and so on until $(B+b')^{th}$ bits.

Since we know that each output bit after the B^{th} bit comes from at least 2 input bits, we can be sure that the maximum value for b' is $(L-T)/2$. We need is to recover the remaining $L-T$ bits with all the possible value of b. Hence total complexity is the number of ways to arrange $(L-T-2b)$ balls into b urns, where $1 \leq b \leq b' = (L-T)/2$:

$$\binom{T-B-1}{T-2B} \sum_{1 \leq b \leq (L-T)/2} \binom{L-T-b-1}{L-T-2b} \tag{2}$$

If $T >$L, The reasoning is similar to above. We ignore the arrangements of the bits after the L^{th} bit. So we first assume that first $(b=1)$ output bit from the buffer is contructed from L input bits (E.g. input is 0111...10 with 2 zeros and L-2 ones). We test if the guess is correct. If the guess is wrong we guess that the first two $(b=2)$ output bits from the buffer are contructed from L input bits (E.g. input is 1001011...10). We test all possible configurations for the input. If the guess is wrong, we procced to first three bits and so on.

Since we know that each output bit comes from at least 2 input bits, we can be sure that the maximum value for b is $L/2$. Moreover since $T > L$, all we need is to recover the first L bits. The complexity is the number of ways to arrange $(L-2b)$ balls into b urns, where $1 \leq b \leq L/2$. In addition if equation (1) has lower complexity despite $T > L$, we can just attack the cipher with the first attack. This is so since it is alright if we recover more than L bits. Hence overall complexity:

$$min\left(\binom{T-B-1}{T-2B}, \sum_{1 \leq b \leq L/2} \binom{L-b-1}{L-2b}\right) \tag{3}$$

T is not constant; it varies with respect to the initial state (key). However the compression rate of ABSG is approximately 3:1, this implies that $T \approx 3B$. The table below shows the complexity given that $L = 128$ and $T = 3B$.

The complexity decreases as buffer size approaches 40. In comparison, the attacker requires both time and data complexity of 2^{64} in the attack on ABSG in [9]. Thus our timing attack can be viewed as a trade-off of a 2^{23} increase in computational complexity in exchange for a savings of 2^{64} in memory.

This timing cryptanalysis complexity varies with respect to compression rate and buffer size. The attack can be improved if we are allowed to do timing analysis on multiple sessions. The attacker has to find the session with the smallest $T < 3B$. From our experiments, $T = 2.5B$ occurs approximately 0.31% of the time. Given that, the best time complexity is $2^{64.93}$ with buffer of size 50. To the best of our knowledge there is no literature that studies the security of the size of the buffer.

Table 1. Result for timing attack with $L = 128$, $T = 3B$

B = Buffer Size	n where Complexity = 2^n
30	80.24
32	80.03
34	79.82
36	79.61
38	79.41
40	79.21
42	79.47
43	81.46
44	83.44
45	85.42
>46	87

4 Phase Shift Fault Analysis on ABSG

Fault analysis on public key cryptosystem was introduced by Boneh and Lipton at Bellcore in 1996. It was later adapted to symmetric ciphers like DES by Biham and Shamir [2]. The idea is to create faults in the cryptographic device such that it will malfunction for a brief moment. After the short moment, the components in the device will be desynchonized from the correct operation. Together with the correct output, the faulty output yield important information about the initial state. In [7], the authors discovered that by stopping one of the LFSR of A5/1, attackers can easily compromise the device. Also in [11], the authors generalize the (Phase Shift) Attack and gave an example on the Shrinking Generator.

In this paper, we adapt the Phase Shift Attack in [11] and applied it on ABSG. In this attack, we clocked an additional cycle to the input stream and halt the ABSG. In this way, the faulted sequence will start 1 bit after the original sequence. The output will differ and yield more information. We have a simple algorithm about how to execute the attack.

Algorithm 1. S_0

$k > 0$ k $k = 2$

$N = 1$

N

$t = 0$

$t = t + 1$

$t < N$

S_N

$(b+k)$

S_0 S_1 S_N

fi b

$N = N + 1$

4.1 An Example

We begin the attack by looking at an example. We chose $k = 2$. For simplicity, we assumed we have multiple faults to begin with. This is a little different from Algorithm 1. Assume the input sequence as follows:

0101001001110111010 1...

Next, we look at the output sequence S_0 and its shifts S_i.

Correct sequence S_0: 1001011... (i)
1-bit shifted sequence S_1: 000110... (ii) (~~0~~10100100111011101011...)
2-bit shifted sequence S_2: 11110... (iii) (~~01~~0100100111011101011...)
3-bit shifted sequence S_3: 001011... (vi) (~~010~~100100111011101011...)

When the input sequence is shifted by 3 bits, we get back the original output sequence shifted by 1 bit. Hence we can be sure that the input sequence begins with 010, since the first output bit is one and is from 3 bits of input. There are 4 possible solutions to continue the contruction of the input sequence.

Guess	Remarks
01000...	Contradict output sequence (iii)
01001...	Contradict output sequence (iii)
01010...	No contradiction
01011...	Contradict output sequence (i)

Hence we left with only"01010" as a solution. From there, we can further expand the input sequence leaving with one solution "0101001...". "0101000..." contradicts (iii) and both "0101010..." and "0101011..." contradicts (ii).

4.2 Simulation Results

In a practical attack as stated in the algorithm, we can begin with 1 fault analysis sequence and start constructing the input sequence. Once we have an ambiguous decision, we store all the possible solutions. Once the number of possibilities exceeds the attacker's resources, attacker needs to produce more fault analyses sequence.

We want to point out that the complexity of this attack varies depending on the key. From our simulations, We can recover 256 bits of input stream from 0.008s to hours of waiting before we kill the process. The system configuration

is Intel Xeon CPU 3.06GHz, cache 512 KB with 2Gb RAM. Table 4 in the appendix (Section 9) summarises our simulations.

Table 4 tabulates the time required to recover input length of 128, 256 and 18529 bits. Hence, if the input generator has linear complexity of 128, the time required will be equal to the time taken to recover 128 consecutive input bits. Besides the time, the table also shows the success rate to find the unique solution. We regard samples that take longer than 10 minutes to find a unique solution as failures. Samples with more than one solution are considered as failures too, even when one of the solutions are correct.

5 Bit Flipping Fault Analysis on ABSG

In [11], Hoch and Shamir considered an input fault attack on a filter function generator where one or more bits of the LFSR are complemented. In this section, we generalize the input fault attack to the ABSG where one or more bits of the input sequence to the ABSG are complemented. This is achieved by a side channel attack where the adversary induces a chosen fault on the input bit stream of the ABSG.

Algorithm 2 is presented to recover the ABSG input sequence based on induced faults in the input sequence.

Algorithm 2. n $k < n$ k $m = n/2$

fi m n

$in_1, in_2, \ldots, in_r$

d 1 k

Δ $wt(\Delta) = d$

Δ

$in_1, in_2, \ldots, in_r$

$Output = ABSG(Input \oplus \Delta)$ $Output_i = ABSG(in_i \oplus \Delta)$

in_i in_i

fi

n fi

m

5.1 An Example

We demonstrate algorithm 2 with an example. Suppose the input sequence of an ABSG is

10101100110111001010

which produces the output 011101. We shall demonstrate how an attacker who can induce fault on the input sequence and observe the output can recover the ABSG input. We assume the attacker uses the parameters $n = 6$, $m = 3$ and $k = 2$.

First, the attacker can keep a table of all 6-bit input sequence to an ABSG and record down their 3-bit output. This is tabulated in Table 2.

Table 2. 3-bit output based on 6-bit input sequence to ABSG compression

Input Sequence	Output Sequence	Input Sequence	Output Sequence
000000	000	100000	$0**$
000001	001	100001	$0**$
000010	000	100010	$0**$
000011	001	100011	$0**$
⋮	⋮	⋮	⋮
011110	$1**$	111110	110
011111	$1**$	111111	111

In the table, * in the output means the bit can take the value 0 or 1. For example, the first five bits 10001 of the entry 100011 is compressed to 0, however we cannot deduce any information on the next two output bits from the sixth input bit 1.

For the input sequence 000001, it is an incomplete sequence and should have had the output 00. But because we will be treating it as part of a longer input sequence, therefore we write down its predicted 3-bit output which is 001.

The attacker observes the output 011101 and conclude from Table 2 that the first six bits of the input sequence is one of fourteen possibilities:

$$000100, 000101, 000110, 000111, 001101, 001111, 100000, 100001$$
$$100010, 100011, 100101, 100111, 101010, 101011$$

If we apply a fault to the first input bit, then the input sequence is 0010110... and the output sequence is 0000100. Based on this, we can eliminate some of our choices. Precisely, those whose first three output bits are different from 000 (as shown in Table 3).

Now we are left with seven choices

$$000100, 000110, 001101, 100000, 100010, 101010, 101011.$$

In a similar way, by performing fault attack on position $2, 3, 4, 5, 6$ of the input stream, we can narrow our possible candidates to 3 choices: 100000, 101010, 101011. By performing another fault attack which tweaks the first two bits,

Table 3. Elimination of Wrong Sequences by observing ABSG output where first Input bit is Faulted

Guess	First Bit Faulted	Output	Accept?	Guess	First Bit Faulted	Output	Accept?
000100	100100	00∗	Yes	100001	000001	001	No
000101	100101	01∗	No	100010	000010	000	Yes
000110	100110	00∗	Yes	100011	000011	001	No
000111	100111	01∗	No	100101	000101	01∗	No
001101	101101	00∗	Yes	100111	000111	01∗	No
001111	101111	01∗	No	101010	001010	00∗	Yes
100000	000000	000	Yes	101011	001011	00∗	Yes

the input sequence is 011011001101110010110 with output 1101111. Only the candidate 101011 among the above three choices give the correct first three bits of the output stream, thus we have found the correct first six bits of the input stream by performing seven faults on the input sequence.

To continue the attack, we will have to take into account the end part 011 of the first six input bits 101011. This is because 011 is not a complete sequence for compression in ABSG. Thus the next six bits we look at will be $011 * **$ (corresponding to position $4, 5, 6, 7, 8, 9$ of the input sequence) with 3-bit output 111 from position $2, 3, 4$ of the output 011101. By a similar method as described above, we deduce that these six bits should be 011001 by comparing the ABSG sequences which are faulted in the 1^{st}, 2^{nd},..., 6^{th} input bits. Therefore the first nine ABSG input bits are 101011001. We can continue the attack to obtain the whole ABSG input sequence by deducing each subsequent segments of 6 bits through the faulted ABSG sequences.

5.2 Simulation Result

Table 5 in the appendix (Section 9) shows a simulation of the bit-flipping fault attack. We tabulate the time required to recover input length of 128, 256 and 18529 bits. Similar to Table 4, it shows the success rate to find the unique solution. As before, we assume samples that takes longer than 10 minutes and samples with more than one solution as failures. As we can see from the results, the complexity is similar to the phase shift attack.

6 Non Side-Channel Attacks

Side channel attacks requires the attacker to have access to the cryptographic device. In the case where the attacker does not have this advantage, he can fake a "fault" on the device and gather additional information about the key. This is only possible if the design of the cryptographic device is flawed. The two subsections below will explain possible extension to the fault attacks based on the assumption that some components are weak.

6.1 Weak Key/IV Mixing Functions

This section briefly describes a possible non side-channel attack extended from the phase shift attack on stream ciphers based on IDG. It is possible based on the assumption that the mixing component for the secret key and initialization vector (IV) is weak.

In this weak key/IV mixing setup, we assume that the function $F(\cdot)$ which parses bitstream into the ABSG (or any IDG) is a pseudorandom number generator (PRNG). The seed for the PRNG $F(\cdot)$ is formed by $seed = Mix(K, IV)$ where K is the secret key and IV is the initial vector. This can be written as:

$$\begin{aligned} seed &= M(K, IV) \\ F(seed) &= (a_0, a_1, a_2, \ldots) \\ &= \text{ ABSG input bit stream.} \end{aligned}$$

We assume that we can find a set of IV's such that the ABSG input bit streams they produce are phase-shifts of each other. Because of this property, these bit streams are similar to "faulted" inputs in a phase-shift fault attack. We let K be the secret key and IV_0, IV_1, IV_2, ..., IV_n be the set of IVs. Then we can perform a phase-shift fault attack with n faults if the following equation holds.

$$\begin{aligned} &L^n(F(M(K, IV_n))) = \ldots = L^2(F(M(K, IV_2))) \\ &= L(F(M(K, IV_1))) = F(M(K, IV_0)). \end{aligned}$$

where L is the left-shift by one operator.

An example of when our scenario can apply is where $F(\cdot)$ is a Linear Feedback Shift Register (LFSR). And the mixing function $M(K, IV)$ allows the adversary to find a set of initial vectors $IV_0, IV_1, \ldots$ such that $M(K, IV_i)$ corresponds to the internal state of the LFSR (initialized by $M(K, IV_0)$) at iteration i.

6.2 Weak Keys for Input Fault Attack

As before, we assume that the seed for the function $F(\cdot)$ which parses bitstream into the ABSG (or any IDG) is formed by $seed = M(K, IV)$ where K is the secret key and IV is an initial vector. We simulate a chosen input fault attack with $InputFault = \Delta$ by a chosen IV attack. We want to find IV_1, IV_2 such that:

$$F(M(K, IV_1)) \oplus F(M(K, IV_2)) = \Delta.$$

As an example, we consider the case where the function $F(\cdot)$ is a linear feedback shift register $LFSR(\cdot)$ of length L bits based on a primitive feedback polynomial (for maximal period). To perform fault input attack, we only need to consider L bits of the LFSR output, i.e. $LFSR(seed) = L$-bit output of the linear feedback shift register $LFSR$ when it is initialized by $seed$.

In that case, $LFSR(\cdot)$ is a linear bijection.

Let the mixing function be an XOR of the secret key K and initial vector IV: $M(K, IV) = K \oplus IV$. We want:

$$LFSR(K \oplus IV_1) \oplus LFSR(K \oplus IV_2) = \Delta.$$

Because $LFSR(\cdot)$ is linear, this reduces to:

$$LFSR(K) \oplus LFSR(IV_1) \oplus LFSR(K) \oplus LFSR(IV_2) = \Delta.$$
$$\implies LFSR(IV_1) \oplus LFSR(IV_2) = \Delta.$$

Given IV_1, we can find IV_2 by linear algebra because $LFSR(\cdot)$ is a linear bijection.

7 Cryptanalysis on DECIM

DECIM is one of the candidates that has been selected for Phase 3 of the ESTREAM project [1]. The strength of the cipher relies on a ABSG component for its security. It is a stream cipher with an 80-bit key and a 64-bit IV. The key and IV are mixed and loaded into a 192-bit LFSR. A quadratic Boolean function taps 14 bits from the LFSR and output bitstreams into the ABSG component. After generating enough input bits, the output of the ABSG will fill up a buffer and release the bits at a constant rate. For more details, please refer to the full paper [3].

In our cryptanalysis of DECIM, we first assume the ABSG input bitstream can be found. As described above, the ABSG input is generated from a (quadratic) filter function generator. Thus we can form quadratic equations which involves:

$$\binom{192}{2} + \binom{192}{1} + 1 = 18529 \tag{4}$$

monomials in terms of the 192 unknown LFSR bits. By collecting 18529 ABSG input bits (filter function output), we can form this number of quadratic equations. Then by a linearization process where we replace each linear/quadratic monomial by a new variable, we can solve the linear system to recover the secret initial state of the LFSR, i.e. break the DECIM cipher. The complexity to solve the linear system by Gaussian elimination is $18529^3 \approx 2^{42.53}$. This complexity can be improved to $18529^{2.376} \approx 2^{33.69}$ if we apply the Coppersmith-Winograd method [5].

Thus we are left with the task of uncovering 18529 bits of the ABSG input. From the simulations documented in Tables 4 and 5, we showed that we can retrieve 18529 ABSG input bits within minutes, based on approximately $18529/3 \approx 6086$ DECIM keystream bits. Hence the overhead of the side channel attack is small.

The difficulty lies in the actual implementation of the attack. A possible way is to control the clocking of the ABSG. Once that is done, we can easily do a phase shift attack by halting the ABSG for a brief moment while the LFSR resume operation as normal. Bits generated from the LFSR will be discarded by the ABSG since it stops operation for that short moment, creating a "shift" in the input sequence. To implement the Bit Fliping Attack, one must be able to control the bits before it is parsed into the ABSG.

We have studied the key/IV mixing component of DECIM and have not found any weakness in them. The timing analysis of Section 3 is also not possible since the linear complexity $L = 18529$ is too huge, making the attack worse than brute force.

8 Conclusion

This paper shows some side channel weakness in using irregularly decimated generators as components in ciphers like DECIM. While the actual attack might face some practical issues [7], it can be used as a trapdoor in the cipher. This paper hopes to provide more design awareness for side channel attacks.

In summary, we learned that the size of the buffer affects the security of the system from timing attacks. Also ciphers must be resistant against the fault attacks as we shown before. Most importantly, key/IV setup must be complex enough to prevent weak IV pairs. From the simulations, we know that there are keys (input stream) that are very hard to recover from the fault attacks. The study of these strong keys will be just as interesting. Other future works are to formalize the complexity for the fault analyses and to improve the attack via correlation attack.

References

1. ESTREAM Phase 3 Candidates, can be found at `http://www.ecrypt.eu.org/stream/phase3list.html`
2. Biham, E., Shamir, A.: Research announcement: A New Cryptanalytic Attack on DES, found at `http://www.fit.vutbr.cz/cvrcek/cards/newdes.ps`
3. Berbain, C., Billet, O., Canteaut, A., Courtois, N., Debraize, B., Gilbert, H., Goubin, L., Gouget, A., Granboulan, L., Lauradoux, C., Mimnier, M., Pornin, T., Sibert, H.: DECIM, a new stream cipher for hardware applications, at `http://www.ecrypt.eu.org/stream/ciphers/decim/decim.pdf`
4. Coppersmith, D., Krawczyk, H., Mansour, Y.: The Shrinking Generator. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 22–39. Springer, Heidelberg (1994)
5. Coppersmith, D., Winograd, S.: Matrix Multiplication via Arithmetic Progressions. Journal of Symbolic Computations 9, 251–280 (1990)
6. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 392–407. Springer, Heidelberg (2000)
7. Gomulkiewicz, M., Kutylowski, M., Vierhaus, H.T., Wlaź, P.: Synchronization Fault Cryptanalysis for Breaking A5/1. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 415–427. Springer, Heidelberg (2005)
8. Gouget, A., Sibert, H.: The bit-search generator. In: The State of the Art of Stream Ciphers: Workshop Record, Brugge, Belgium, pp. 60–68 (October 2004)
9. Gouget, A., Sibert, H., Berbain, C., Courtois, N., Debraize, B., Mitchell, C.: Analysis of Bit-Search Generator and Sequence Compression Techniques. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 196–214. Springer, Heidelberg (2005)
10. Hell, M., Johansson, T.: Some Attacks on Bit-Search Generator. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 215–227. Springer, Heidelberg (2005)
11. Hoch, J.J., Shamir, A.: Fault Analysis of Stream Ciphers. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 240–253. Springer, Heidelberg (2004)
12. Kelsey, J., Schneier, B., Wagner, D., Hall, C.: Side Channel Cryptanalysis of Product Ciphers. In: Quisquater, J.-J., Deswarte, Y., Meadows, C., Gollmann, D. (eds.) ESORICS 1998. LNCS, vol. 1485, pp. 97–110. Springer, Heidelberg (1998)

13. Meier, W., Staffelbach, O.: The Self-Shrinking Generator. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 205–214. Springer, Heidelberg (1995)
14. Bernstein, D.: Cache-Timing Attacks on AES (2005), http://cr.yp.to./antiforgery/cachetiming-20050414.pdf
15. Kocher, P.C.: Timing attacks on implementation of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)

9 Appendix

Table 4. Simulation of the Phase-Shift Fault Attack on 10000 ABSG Samples with Input Length 128, 256 and 18529 Bits

Faults	128 (10000 samples)			256 (10000 samples)					18529 (100 samples)				
	Time		Success	Time				Success	Time				Success
	< 1s	< 1min		< 1s	< 1min	< 10min	> 10min		< 1min	< 5min	< 10min	> 10min	
2	9978	22	49.87%	9596	320	60	24	49.67%	61	3	0	36	54.00%
3	9985	15	74.76%	9842	116	24	18	75.41%	79	4	0	17	76.00%
4	9993	7	87.73%	9903	79	13	5	88.11%	87	2	0	11	87.00%
5	9992	8	93.50%	9963	26	7	4	94.04%	96	0	0	4	96.00%
6	10000	0	97.04%	9972	26	1	1	96.79%	99	0	0	1	99.00%
7	9999	1	98.45%	9992	7	0	1	98.70%	97	1	0	2	97.00%
8	10000	0	99.16%	9991	3	4	2	99.16%	100	0	0	0	100.00%
9	10000	0	99.64%	9997	1	1	1	99.53%	100	0	0	0	100.00%
10	10000	0	99.79%	10000	0	0	0	99.82%	100	0	0	0	100.00%

Table 5. Simulation of the Bit-Flipping Fault Attack on 10000 ABSG Samples with Input Length 128, 256 and 18529 Bits

Number of Faults	128 (10000 samples)			256 (10000 samples)					18529 (100 samples)				
	Time		Success	Time				Success	Time				Success
	< 1s	< 1min		< 1s	< 1min	< 10min	> 10min		< 1min	< 5min	< 10min	> 10min	
2	9970	30	44.97%	9648	277	50	25	46.98%	8	49	3	40	42.00%
3	9993	7	75.04%	9808	146	34	12	75.60%	76	1	0	23	71.00%
4	9994	6	87.49%	9921	63	13	3	88.07%	95	1	0	4	90.00%
5	9997	3	93.03%	9964	33	2	1	93.39%	82	12	0	6	93.00%
6	9998	2	96.86%	9983	12	3	2	96.82%	100	0	0	0	100.00%
7	10000	0	98.60%	9980	15	4	1	98.43%	100	0	0	0	100.00%
8	10000	0	99.20%	9993	7	0	0	99.07%	100	0	0	0	100.00%
9	10000	0	99.59%	9995	4	0	1	99.58%	100	0	0	0	100.00%
10	10000	0	99.76%	10000	0	0	0	99.85%	99	0	0	1	99.00%

Asynchronous Pseudo Physical Memory Snapshot and Forensics on Paravirtualized VMM Using Split Kernel Module

Ruo Ando, Youki Kadobayashi, and Youichi Shinoda

National Institute of Information and Communication Technology,
4-2-1 Nukui-Kitamachi, Koganei,
Tokyo 184-8795 Japan
ruo@nict.go.jp
http://www2.nict.go.jp/y/y212/index_en.html

Abstract. VMM (virtual machine monitor) provides the useful inspection and interposition of the guest OS. With proper modification of the guest OS and VMM, we can obtain incident-driven memory snapshot for malicious code forensics. In this paper we propose an asynchronous memory snapshot and forensics using split kernel module. Our split kernel module works for the virtualized interruption handling, which notifies the security incident on the guest OS. On frontend, we insert virtualized interruption into source code of MAC (mandatory access control) module and other security modules. Then, backend kernel module receives interruption as the asynchronous incident notification. In experiment, we take RAM snapshot of LKM-rootkit installation using system call extension. Frequently appeared strings are extracted in order to find the evidence memory blocks which was assigned for LKM-rootkit. Also, it is showed that asynchronous snapshot enables us to find the evidence of malicious software in memory snapshot by simple string analysis in linear time.

Keywords: Asynchronous snapshot, paravirtualized VMM, memory forensics, virtualized interruption, split kernel module.

1 Introduction

1.1 Virtual Machine Monitor

VMM (virtual machine monitor) is a thin layer of software between the physical hardware and the guest operating system. The rapid increase of CPU performance enables VMM to run several operating systems as virtual machine, sharing (multiplexing) CPU, memory and I/O devices in reasonable processing time. In [1], it is pointed that recent VMM is a successful implementation of microkernels. Under the guest OS, VMM runs directly on the hardware of a machine which means that VMM can provides the useful inspection and interposition of the guest OS.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 131–143, 2007.

1.2 HIDS, NIDS and VMM Based IDS

IDS (Intrusion detection system) is classified into HIDS (Host-based IDS) and NIDS (Network-based IDS). Previously, there are tradeoffs between NIDS and HIDS about visibility and attack residence. HIDS provides good visibility. However, HIDS has weaker isolation than NIDS. That is, once the operating system running HIDS has been compromised, attacker can stop HIDS. On the other hand, NIDS offer higher attack residence instead of the cost of visibility. In [6], it is pointed that VMM-based IDS can provide good visibility while maintaining secure isolation for the IDS. VMM-based IDS takes advantages in three points, isolation, inspection and interposition. Besides, the utilities of VMM makes it possible to take timely snapshot of the pseudo physical (virtualized) memory of the guest OS.

1.3 Memory Snapshot and Forensics Using VMM

VMM has access to all of VM's state. VMM enables us to access all events of CPU, memory and I/O devices of the guest (virtualized) OS. In previous operating system, it is not easy to take snapshot of the system itself. This is partly possible by using debuggers, but it is still not easy to capture all of system's state. It is important for us IDS designers that with proper modification of VMM and guest OS, automatic snapshot driven by security incidents is possible. In this paper we propose an asynchronous pseudo physical memory snapshot and forensics on paravirtualized VMM using split kernel module. Once split kernel module is implemented and registered correctly, the modification is simple. Timely memory snapshot is useful for detecting and inspecting malicious software on the guest OS.

Asynchronous memory snapshot enables us to extract evidence memory blocks by simple string analysis. By frequently appeared string analysis, we can know malware (malicious software) of which name is unknown and intruder's behavior from our memory snapshot. Previous malware detector is basically signature-based, which means it cannot detect the malware of which name is not matched (unknown). Attackers can easily avoid black-list based detection by changing or obfuscating the signature of their malware.

Our forensics is n-gram frequency analysis based. We can discover and sort the parts of memory snapshot according to frequently appeared strings. By doing this, we can detect which blocks was used by attacker (assigned for malware operation). Then, we can extract the further information of malware from evident blocks. Let us discuss the example:

```
72 65 5f 72 6f 6f 74 5f  66 69 6c 6c 64 69 72 20 re_root_filldir
6e 61 6d 65 20 2d 3e 20  74 65 73 74 c0 bc 5e 47 name -> test..^G
20 0a 4d 61 79 20 33 31  20 30 35 3a 31 39 3a 31  .May 31 05:19:1
32 20 6c 6f 63 61 6c 68  6f 73 74 20 6b 65 72 6e 2 localhost kern
65 6c 3a 20 61 64 6f 72  65 5f 72 6f 6f 74 5f 66 el: adore_root_f
69 6c 6c 64 69 72 20 6e  61 6d 65 20 2d 3e 20 6f illdir name -> o
75 74 6f c7 bc 5e 47 20  0a 4d 61 79 20 33 31 20 uto..^G .May 31
```

Table 1. Frequently-appeared strings in memory snapshot. Snapshot is taken on XEN kernel 2.6.18 with LKM-Rootkit "adore" installed.

name	grep hit	name	grep hit
kernel	4699	usr	1651
driver	4937	xen	7544
module	3476	adore	1780

This is the part of memory dump (snapshot), which is used by malware installation. This hexadecimal dump means that function (system call) "filldir" is changed to adore_root_filldir on May 31. Adore is the name of malware which is appeared frequently in memory dump. To extract such a information, memory snapshot needs to be obtained timely during the execution of some system calls on the guest OS. In previous signature-based system, if the word of "adore" is not in database, we cannot detect it. In proposed system, once the memory snapshot is taken timely, this part of memory dump is extracted automatically by simple n-gram analysis.

2 Proposed Method

Proposed method is divided into two operations, architecture modification and memory inspection. First, we append split kernel module to paravirtualized VMM. Second, we insert probe code into the points where exploitation could be occurred. For example, i-node permission of secure OS, gcc security extension and security library could be inserting points. Also, some important system calls (such as create_module) need to be modified. Third, we extract frequently appeared n-gram from RAM snapshot. In this process, we divided RAM snapshot into M blocks. Finally, we sort strings according to frequency and choose top N. We can find the evidence block which is used by malware as follows.

```
for i to N
       for j to M
          grep string[i] EVENT_DRIVEN_SNAPSHOT_BLOCKS[j]
       next
next
```

Then the evidence blocks introduced in section 1 are extracted automatically. Motif in Figure 1 means a string that appears in memory snapshot frequently. Table 1 shows example of motif on kernel memory. In Linux, kernel code is memory resident. So the string such as kernel, driver and module appears frequently. In this case of Table 1, snapshot was taken just when LKM-rootkit "adore" was installed. In this snapshot, string adore appears 1780 times. Even if attacker installs software of which name is unknown (new to signatures), the name of the malware appears as frequently as other frequent strings. Our approach is frequent string discovery based.

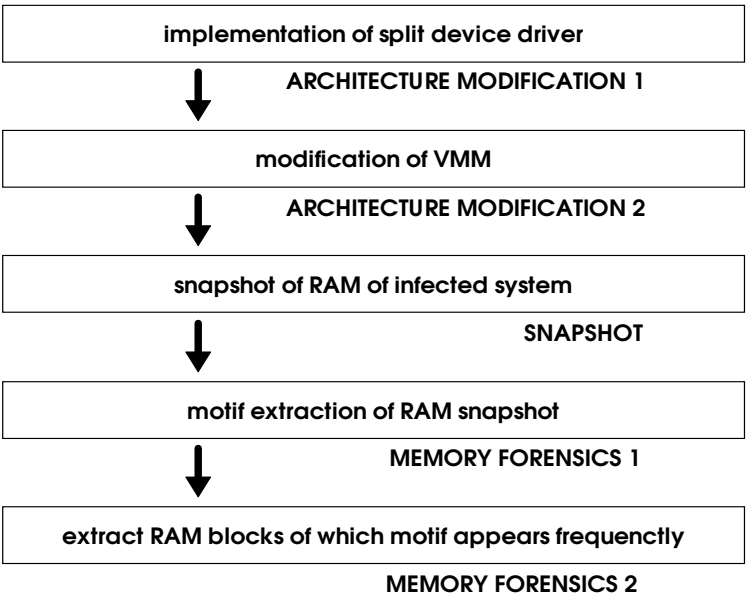


Fig. 1. Flow chart of proposed method. Proposed method is divided into three steps: architecture modification, snapshot and memory forensics.

3 Related Work

VMM is also called hypervisor. Hypervisor is not new technology, originated from IBM mainframe in 1960s. However, VMM now offers huge potential to generic servers about consolidation and secure isolation [2]. Classical aspects of secure kernel for VMM is discussed in [3][4]. Recently, XEN [5][6] and KVM (kernel virtual machine) [7] are going to mainline of Linux kernel.

To obtain system control, security module needs to be at lower-level of system. With the deployment of VMM under operating system, some security applications of VMM have been proposed. The design and deployment of IDS on VMM is discussed in [8]. They propose VMI (virtual machine introspection) applied for IDS policy management. Some of the LKM-rootkit is tested to validate their system. In [9][10], moving functionality of MAC to outside guest VM is discussed. However, in these papers, memory forensics to extract further information of malware is not discussed explicitly.

Concerning another advantage of VMM, VMM provides new concepts for verification and debugging. These are called as deterministic replay [11] or time-travel debugging [12] which is also useful for security. Security application of logging and replay using VMM are discussed in [13]. However, their paper is not specified for the detection of kernel mode malware such as LKM-rootkit.

Computer forensics is much concerned with rootkit detection and inspection. VMM based rootkit is discussed on [14]. In [15], they do not mention in detail how to detect the event of installation of VMR. The thrust of this paper is the specific modification of VMM architecture in order to enhance memory snapshot for malware forensics. Besides detection, proposed system enables us to extract further information of the exploitation of kernel based malware. Once the split kernel module is implemented and registered correctly, modification of guest is simple. Also, evidence memory block can be extracted by a simple string analysis.

4 Paravirtualized VMM of XEN

In this paper we implement the proposed system on virtual machine monitor XEN. Virtualization technology is divided into two categories: full-virtualization and para-virtualization. Para-virtualization needs kernel-modification in order to run guest OS on ring 0 at the same time. Full-virtualization need a new CPU mechanism such as Virtualization Technology of Intel(R) and AMD-V of AMD(R), which makes it possible to run VMM on specified mode. Para-virtualization is more lightweight particularly about I/O performance. Another advantage of para-virtualization is that we can insert original hypervisor call into source code of guest kernel. This enables us to construct new (specified for security) notification channel between hosted OS and VMM.

Figure 2 illustrates architecture of XEN. Domain U (guest OS) has the virtualized frontend driver, like proxy for I/O access for real device driver under Domain 0 (host OS). I/O request of the process of guest OS goes through the split device drivers (backend and frontend driver).

4.1 Event-Channel of XEN

Paravirtualized VMM has the asynchronous notification mechanism between guest OS and privileged OS. In XEN, hardware access request of domain U goes through frontend and backend driver using event-channel. Event channel is the virtualized interruption mechanism for activating interruption from guest domain (U) to host domain(0). In proposed system, we newly construct event channel for the notification of security event on guest OS.

4.2 Split Kernel Module

Figure 2 shows generic framework of split kernel module of virtual machine monitor. Guest OS (domain U) has frontend kernel module. Host OS (domain 0) has backend kernel module. These modules communicate with each other by virtualized interruption and shared memory. In XEN, virtualized interruption is called event channel. The mechanism of shared memory is called grant table. When the interruption port is activated by event-channel, string (information) in grant table is transferred between domains.

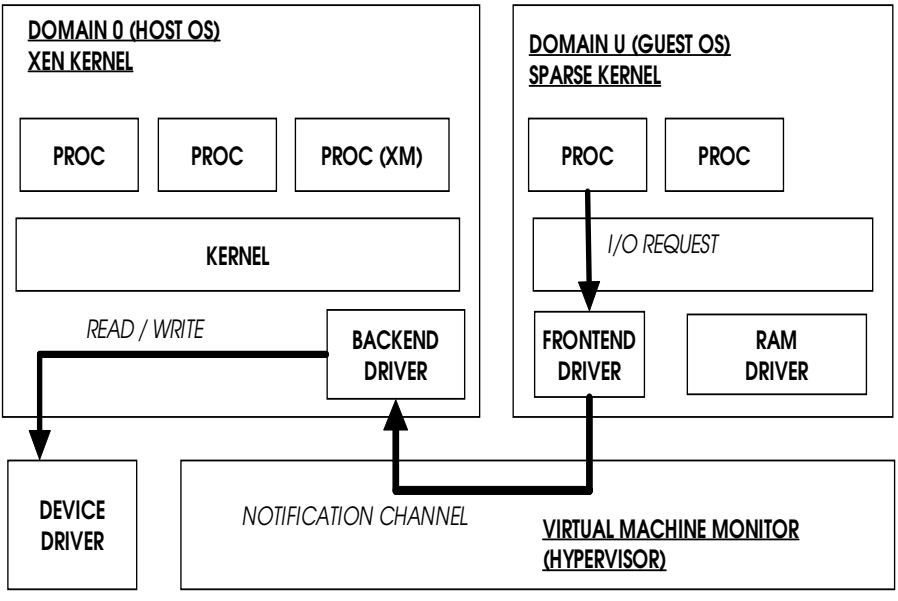


Fig. 2. Paravirtualized VMM. Guest kernel is modified to "sparse kernel" and split kernel module is added. IRQ goes through frontend and backend kernel module.

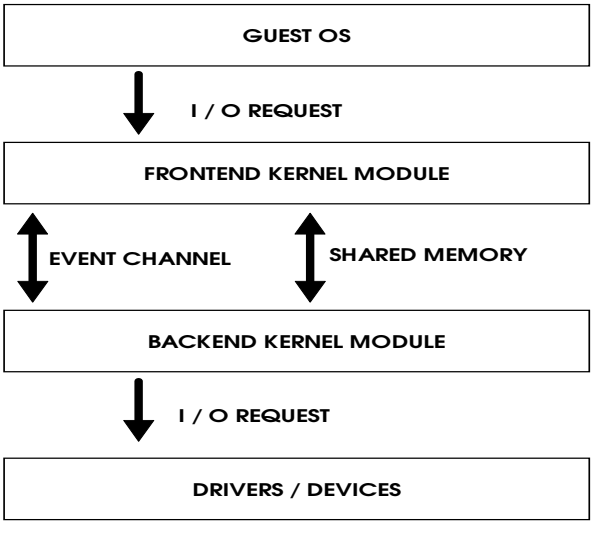


Fig. 3. Split kernel module consists of frontend and backend one. These two modules communicate by event-channel and shared memory. Event channel is the virtualized interruption. Shared memory mechanism is called grant table.

5 Asynchronous Notification Using Split Kernel Module

In proposal method, memory snapshot need to be taken timely for frequency analysis of string. Polling or sequential mechanism is not proper to cope with the security incidents. Particularly, on-line or asynchronous mechanism is necessary for inspecting malware installation and its behavior. In proposed system we construct the asynchronous notification channel using split kernel module (device driver). When security incidents (such as buffer overflow, malware installation and malicious resource access) has been occurred on the guest OS, these are notified to our frontend device driver as virtualized interruption. Then, the notification is transferred to backend device driver through our notification channel. Kernel module is suitable for coping with asynchronous security incidents.

In modified VMM, the detected event is translated as a hardware interrupt (IRQ). Let us show the list of the guest OS.

```
256: 1782  Dynamic-irq  timer0
257: 0     Dynamic-irq  resched0
258: 0     Dynamic-irq  callfunc0
259: 227   Dynamic-irq  xenbus
260: 187   Dynamic-irq  xencons
261: 891   Dynamic-irq  blkif
262: 0     Dynamic-irq  blkif
263: 0     Dynamic-irq  eth0
264: 0     Dynamic-irq  sec-notify
```

This is the list of dynamic (virtualized) IRQs of the guest OS. We have obtained this list by "cat /proc/interrupt". 261 is interrupt handler of block device. 262 is swap. 263 is network interface. 264 sec-notify is our frontend driver of split kernel module. We append 264 dynamic-irq sec-notify and by using this IRQ, the incident notification is transferred to VMM.

Once our driver is registered correctly, modification of guest OS is very simple. All we need to do is inserting this code into some points of guest OS.

```
int port=9;

   evtchn_op_t op;
   op.cmd          = EVTCHNOP_send,
   op.u.send.port = port;

(void)HYPERVISOR_event_channel_op(&op);
```

This code is activating event-channel and sending the interruption signal to our frontend drivers in port 9. In this case, port number 9 is assigned to Dynamic-IRQ 264. By inserting this code, probes of guest OS can inform the incident of VMM and the host OS. The incident notification signal goes through frontend driver, VMM, backend driver and finally reach VM manager process of host OS. The deployment of proposed system with split kernel module is completed in these steps.

```
(1) Implementing front/backend driver (split kernel module)
(2) Registering driver to Xenstore
(3) System call extension: inserting hypervisor call
    into system call (create_module)
(4) MAC extension: inserting hypervisor call
    into i-node permission checker
(5) Buffer overflow handling: inserting hypervisor call
    into GCC-extension or fault handler
```

Xenstore in (2) is the utility of XEN, device database of the guest OS. By step (3)(4) and (5), hypervisor call (the code of event channel) is inserted into probes. By doing this, secure OS and other protection module can communicate with our split device driver. We discuss the step (3) and (4) in the next section. These steps are completed only by inserting five lines of code above into each security module.

6 Enhancing Memory Snapshot for Malware Forensics

Compared with other operating systems, Linux has not paid much attention for debugging facilities inside kernel. Until Linux 2.6.16 (or later) is modified to be able to be run on VMM, kernel memory dump needs additional utility inside the guest OS. Fortunately in XEN, command "xm save" provides snapshot of memory. Memory snapshot is indispensable for malware forensics. In this section we discuss the way to take a timely snapshot of malicious behavior and analyze it.

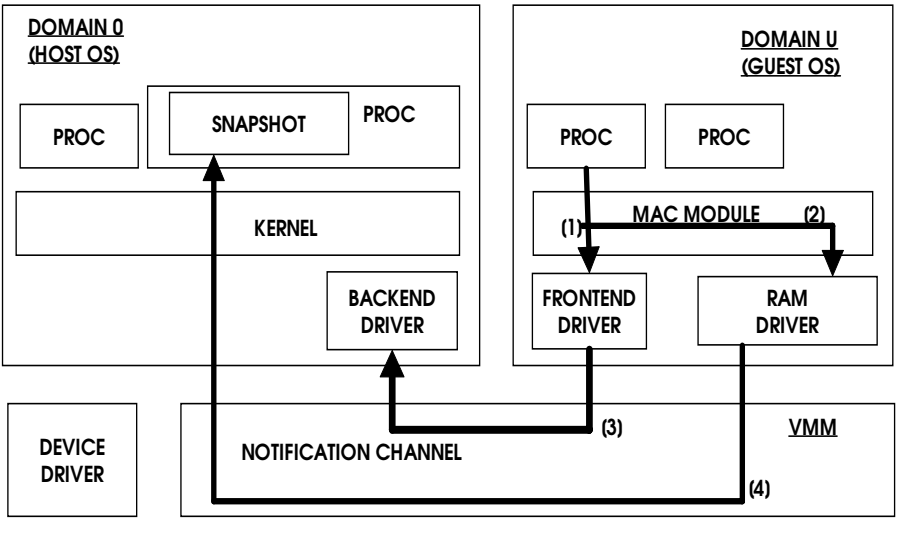


Fig. 4. Incident-driven memory snapshot

Figure 4 shows proposed snapshot system. Our method consists of four steps. In step [1] and [2], buffer overflow, LKM installation and illegal file access are detected the modification of exception handler, secure OS module and gcc-extension. In step [3], the notification of incidents goes from frontend driver to backend driver (interruption is activated). In step [4], domain management tools (XM) takes snapshot of RAM.

6.1 LKM-Rootkit

LKM (Loadable kernel module) is a kind of module which is plugged (embedded) dynamically into kernel. Unfortunately, this mechanism is also used for stealth rootkit implementation. LKM-rootkit changes core system of kernel such as system call table and proc file system. Therefore, once LKM-rootkit is installed, it is not easy to detect it by the application running on user space. VMM takes advantages in coping with this malware because VMM is deployed "below LKM-rootkit" and can hook illegal changes on the guest OS.

6.2 System Call Extension

We insert the code of event-channel into the source code of create_module. Create_module is the system call, which is invoked when new module is installed into kernel. By the modification of create_module, we can obtain memory snapshot just when LKM rootkit is installed. Also, we modified the system calls chown or lchown. After the installation of LKM rootkit, these system calls are used to hide file and process.

6.3 Behavior Detect Using MAC

Another extension of proposed system is inserting the of code of event-channel into LSM (Linux Security Module) code. In this paper we apply MAC extension for LIDS (Linux Intrusion Detection System) [15]. LIDS is security patch and admin tools for Linux kernel to achieve MAC framework. We insert hypervisor call EVTCHN_send into i-node permission routine of LIDS as follows:

```
static int
lids_inode_permission(struct inode *inode, int mask,
struct nameidata *nd)
```

When READ/WRITE request is hooked, we can get detailed information from inode.i_ino, inode.i_sb.s_dev and d.d_iname. By using MAC module, we can obtain memory snapshot when illegal file access (such as /proc and /tmp) is occurred on the guest OS.

7 Experimental Result

In experiment, we count frequency of 5-gram (5-characters strings) found in 128MB snapshot of RAM. We divide 128MB RAM into 1MB*128 blocks.

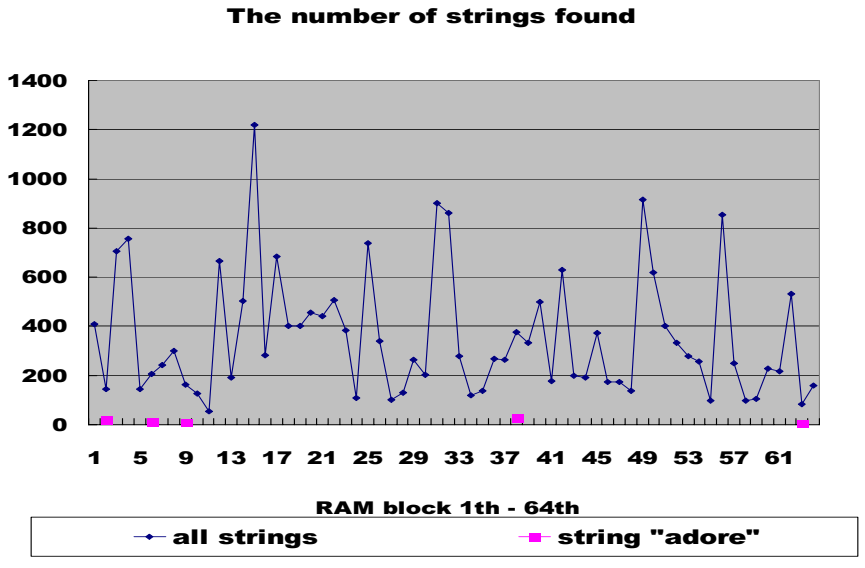


Fig. 5. The number of five characters strings (5-gram) found. The string adore is found in 14 blocks. Perhaps the blocks of which the frequency is relatively small including adore are important (34 and 38).

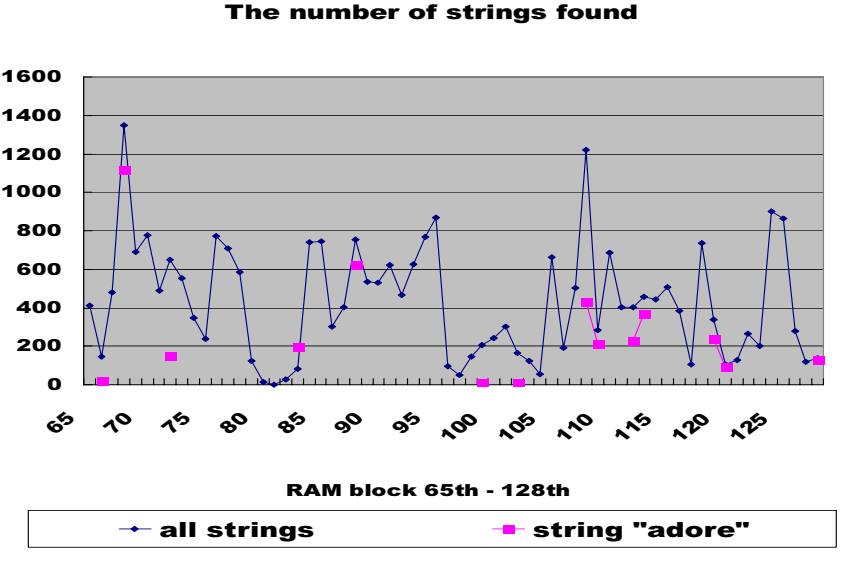


Fig. 6. The number of five characters strings (5-gram) found. The string adore is found in 13 blocks. The blocks of which the frequency is relatively small including adore are important (66 and 121).

Table 2. Numerical result of extracting the string adore from memory snapshot. The number of blocks where adore was found is 27. Proposed method is linear-algorithm based. Computing time is constant regardless of the number of strings found and grep hits.

the number of strings found	47618 strings
grep hits (1-64) with strings found	51908 blocks
grep hits (65-128) with strings found	36691 blocks
the number of blocks with "adore" found	27 blocks
grep time (1-64)	22.198s
grep time (65-128)	23.433s
grep time with grep -C N adore (1-64)	20.681s
grep time with grep -C N adore (65-128)	20.229s
grep hits (1-128) with strings insmod	67 hits

Figure 5 and 6 shows the number of times 5-gram found on 1-64th and 65-128th blocks. Also, the frequency of the string "adore" is plotted. The number of blocks where the string adore is found is 27. The total number of 5-gram appeared is 47618 with average 372.01 per one block. The total number of the string adore appeared is 7842 with average 245.06 per one block. We can conclude the frequency of the string adore is relatively high compared with other five strings.

Table 2 shows the number of strings found and grep time of 1-64th and 65-128th block. Totally, we can search (grep) all the strings found within 45 seconds. Also, we can search with the pipe option "grep -C N adore". With N 10, we can search all blocks for about 40 seconds. The string analysis is completed in linear time about the size of memory.

8 Conclusion

VMM provides the useful inspection and interposition of the guest OS for handling security incidents. With the proper modification of guest OS and VMM, we can obtain incident-driven RAM snapshot of guest OS for malware forensics. In this paper we propose the application of split kernel module for asynchronous memory snapshot and forensics. On frontend, we insert virtualized interruption code of event-channel (evtchn_send) for activating virtualized IRQ of frontend driver. Then, backend kernel module receives the interruption as the incident notification through event-channel port. Once split kernel module is implemented and registered correctly, modification of guest OS is simple. Only several lines of code need to be inserted into the source code of MAC, system call and other security modules. In experiment, we take RAM snapshot of LKM-rootkit installation using system call extension. Asynchronous (incident-driven) snapshot makes it possible to detect malware installation and malicious behavior by simple n-gram string analysis. Numerical result shows that we can find the evidence blocks of RAM within 40-45 seconds by grep command. Extraction is completed in linear time about the size of memory. Fur further work, full virtualization is promising topic for the extension of proposed system.

References

1. Hand, S., Warfield, A., Fraser, K., Kotsovinos, E., Magenheimer, D.: Are Virtual Machine Monitors Microkernels Done Right? In: Proceedings of the Tenth Workshop on Hot Topics in Operating Systems (HotOS-X) (June 2005)
2. Goth, G.: Virtualization: Old Technology Offers Huge New Potential. IEEE Distributed Systems Online 8(2) (2007)
3. Karger, P.A., Zurko, M.E., Bonin, D.W., Mason, A.H., Kahn, C.E.: A Retrospective on the VAX VMM Security Kernel. IEEE Trans. Software Eng. 17(11), 1147–1165 (1991)
4. Karger, P.A., Zurko, M.E., Bonin, D.W., Mason, A.H., Kahn, C.E.: A Retrospective on the VAX VMM Security Kernel. IEEE Trans. Software Eng. 17(11), 1147–1165 (1991)
5. XEN virtual machine monitor, http://www.cl.cam.ac.uk/Research/SRG/netos/xen/
6. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP 2003. Proceedings of the 19th Symposium on Operating System Principles, Bolton Landing, NY (October 2003)
7. KVM: Kernel-based virtualization driver, available at: http://kvm.qumranet.com/
8. Garfinkel, T., Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection. In: NDSS 2003. Proceedings of Network and Distributed System Security, pp. 191–206 (February 2003)
9. Quynh, N.A., Ando, R., Takefuji, Y.: Centralized Security Policy Support for Virtual Machine. In: LISA 2007. Proceedings of USENIX, 20th Large Installation System Administration Conference (December 2006)
10. Sailer, R., Jaeger, T., Valdez, E., Caceres, R., Perez, R., Berger, S., Griffin, J.L., van Doorn, L.: Building a MAC-Based Security Architecture for the Xen Opensource Hypervisor. In: Srikanthan, T., Xue, J., Chang, C.-H. (eds.) ACSAC 2005. LNCS, vol. 3740, Springer, Heidelberg (2005)
11. Xu, M., Malyugin, V., Sheldon, J., Venkitachalam, G., Weissman, B.: ReTrace: Collecting Execution Trace with Virtual Machine Deterministic Replay. In: MoBS 2007. Proceedings of Third Annual Workshop on Modeling, Benchmarking and Simulation (June 2007)
12. Bhansali, S., Chen, W.-K., De Jong, S., Edwards, A., Drinic, M.: Framework for Instruction-level Tracing and Analysis of Programs. In: VEE 2006. Proceedings of Second International Conference on Virtual Execution Environments (June 2006)
13. Dunlap, G.W., King, S.T., Cinar, S., Basrai, M., Chen, P.M.: ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In: OSDI 2002. Proceedings of the 2002 Symposium on Operating Systems Design and Implementation (December 2002)
14. King, S.T., Chen, P.M., Wang, Y.-M., Verbowski, C., Wang, H.J., Lorch, J.R.: SubVirt: Implementing malware with virtual machines. In: Proceedings of IEEE Symp. on Security and Privacy (the Oakland Conference) (May 2006)
15. LIDS: Linux Intrusion Detection System, available at http://www.lids.org/
16. Uhlig, R., Neiger, G., Rodgers, D., Santoni, A.L., Martins, F.C.M., Anderson, A.V., Bennett, S.M., Kagi, A., Leung, F.H., Smith, L.: Intel Virtualization Technology. IEEE Computer 38(5), 48–56 (2005)

Appendix: Modification for Full-Virtualization

Full virtualization is available in recent processors, for example, Intel(R) Virtualization Technology (Intel-VT) [16]. In previous x86 processors, ring 0 is assigned to VMM and guest OS at the same time. Therefore, guest OS need to be modified. Figure 7 shows the full-virtualization system where ring 0 is assigned to VMM in VMX root mode and ring 0 is assigned to guest OS in non-root mode. When the control switches from VMX non-root mode to VMX root mode, the context of CPU is stored to VMM. On this mechanism, we can inform the incident on guest OS of VMM by changing special registers. Instead of implementing the split device driver, we can insert code of changing registers (such as DR, TR and MSR) into probes. For example, in KVM, the function handle_dr of VMM could be one of the inserting point. In the case XEN, software interruption for the host OS needs to be generated in VMM.

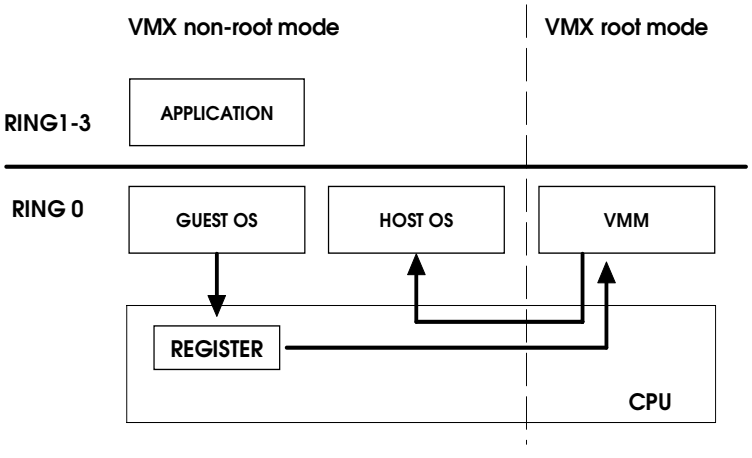


Fig. 7. Full virtualization. Ring 0 is available for both OS and VMM. With the proper modification of the register handling function of VMM, notification can be transferred to the host OS by changing special registers such as DR, TR and MSR.

Filesystem Activity Following a SSH Compromise: An Empirical Study of File Sequences

Jesus Molina[1], Xavier Chorin[2], and Michel Cukier[2]

[1] Department of Electrical and Computer Engineering
University of Maryland, College Park
chus@umd.edu
[2] Center for Risk and Reliability
Department of Mechanical Engineering
University of Maryland, College Park
{xchorin,mcukier}@umd.edu

Abstract. A common method used to detect intrusions is monitoring filesystem data. Once a computer is compromised, an attacker may alter files, add new files or delete existing ones. Attackers may target any part of the filesystem, including metadata along with files (e.g., permissions, ownerships and inodes). In this paper, we will describe an empirical study that focused on computer attack activity after a SSH compromise. Statistical data will be provided on the number of files targeted and the associated activity (e.g., read, write, delete, ownership and rights). We extend this analysis to include the sequence of files and activities targeted. We focused on the most frequent sequences of consecutive files and activities, then explored in greater detail the longer sequences using state machines. Finally, we developed a simple state machine representing three major observed attack activities (i.e., reconnaissance, malware download and password change) with the number of transitions and time for each transition. The analysis of individual and sequences of files and activities will help to better understand attack activity patterns resulting in more efficient intrusion detection.

Keywords: SSH compromises, filesystem data, host intrusion detection systems, intrusion detection systems evaluation.

1 Introduction

Intrusion detection systems (IDSs) are critical tools used to detect unauthorized access to a system. IDSs can also detect if a system's user has gained access to a functionality that s/he is not privileged to. Filesystem data is a popular source of data used by IDSs, which is the focus of this paper. IDSs also collect information from other sources like system calls [1], memory [2] and network packets [3].

Tools based on filesystem data include integrity verifiers [4, 5], malware detectors [6], and software sensors [7]. These tools are implemented in software [8], kernel modules [9], part of the hypervisor of a virtualized system [10] and hardware [11].

Filesystems contain large amounts of information. Acquiring and analyzing all of the data is often infeasible as it translates into severe performance penalties and

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 144–155, 2007.

unacceptably long processing times. Hence, finding the attackers' filesystem activity patterns is required in order to optimize the information collected.

To date, most of these tools focus on possible information on an attack that is included in a file, disregarding the sequence of filesystem activity after a compromise. We expected to find important information that can be used to detect compromises in filesystem activity patterns. We also address the issue of understanding the efficiency of the data collected. Or, in other words, how long it will take an attacker to perform detectable activity on a supervised file.

This paper describes an empirical study of filesystem activity after a security compromise. Quantitative data was collected on the use of files targeted by attackers while reading, writing, deleting and modifying the file metadata. State machines were created based on file usage, to identify patterns between filesystem access and activity associated with attacks (e.g., password changes and malware installations).

The paper is structured as follows. Section 2 contains a description of the experimental setup. In Section 3, the collection and analysis of filesystem activity is discussed. In Sections 4 and 5, we present statistics on the individual files and sequences, respectively. Section 6 provides a discussion on filesystem sequences and file efficiency for files strongly related to attack activity. Section 7 provides a summary of related work, and Section 8 contains conclusions.

2 Experimental Setup

To collect data on attacker activity, we used a set of four high interaction Linux honeypot computers. Refer to [12] for details regarding the testbed architecture. The experimental setup is described in more detail in [13].

The four honeypots ran on an identical Linux disk image: a slimmed-down installation of Fedora Core 3, updated with the latest patches as of October 10, 2006. Since the primary interaction with the system was via SSH, the installation was conducted in a text-mode environment (the X Windows System and associated graphical programs were not installed).

To monitor attacker activity, we used a modified OpenSSH sever to collect password attempts, syslog-ng to remotely log important system events including logins and password changes, strace [14] to record system calls made by incoming SSH connections and the Honeynet Project's Sebek tool [15]. We added a single line of code that used syslog to record attempted passwords to modify the OpenSSH source tree. The program was concealed as a system script in order to prevent attacks directed against strace.

Each honeypot had one privileged root account and five non-privileged user accounts. We selected five usernames: admin, mysql, oracle, sarah and louise based on the results from a study on the most commonly attempted usernames and passwords. For each username, we rotated through four passwords (i.e., 'username', 'username'123, password, and 123456). After a password modification, the honeypot was redeployed and the next password was used. Two honeypots were set up with strong root passwords. The other two honeypots had root accounts that rotated the four passwords: root, root123, password and 123456.

To enable a rapid turnaround, we used a pre-built disk image and automated scripts to manage the honeypot's deployment. We monitored the syslog messages from each honeypot on a 24 hour cycle to check for logins and password changes. The honeypot was redeployed after a password modification to prevent locking out other attackers. To maximize the attackers' activity on the filesystem following a password change, we waited at least one hour before putting the disk image back onto the honeypot, running the deployment script and continuing to monitor the live syslog data.

3 Data Analysis

Data was collected during the 24 day period from November 14 to December 8, 2006. During this time attackers from 229 unique IP addresses attempted to log into the honeypots 269,262 times (an average of 2,805 attempts per computer per day). According to the syslog data, of 269,262 attempted attack, 824 logged in successfully and 157 changed an account password. Results from an extensive analysis of the syslog data was reported in [13].

The data analyzed in this paper consisted of system call data that were collected with the strace tool [14]. Strace intercepts and records all system calls made by a running process. We launched strace against the sshd daemon, switching on the built-in functionality for strace to record the activity of all the children spawned. To discriminate between compromises, we developed a script to isolate each different compromise among the strace data. We defined a compromise as a successful login followed by a bash session and all its children. Before processing attack sessions further, all administrative activity required to transfer logs to a central database and to reimage the honeypots were removed.

Using the strace data, we found 743 attacks compared to 824 attacks found by the syslog data [13]. One reason these results differ is because of the difference in the definition of a compromise used in syslog data (i.e., a successful login) versus strace data (i.e., a bash session). Syslog data included SCP and SFTP connections and aborted logins that were not included in the strace data. Moreover, some attackers were able to compromise the strace logging capability. We verified the data collected by strace against data collected by Sebek [15]. In particular, we identified strace data collection disruptions by attackers to ensure that such events were rare. We found four sessions in which strace failed to properly monitor violations. A careful examination revealed that this occurred after the attacker issued a kill command to the ssh daemon, thus terminating the daemon and strace recording. We do not believe that this activity was performed because of the presence of strace, but rather because the attackers' goal was to launch a rogue ssh server in place of the existing one. Finally, the strace logging capability was enabled at the beginning of the hour following a reimaging. For instance if a honeypot was reimaged at 2:10, strace logging started at 3:00, meaning that successful connections in between were not logged.

For the remaining 743 attacks, we removed empty sessions. An empty session was defined as a bash session with no activity other than a login and logout. To identify these sessions, we determined the number of files read by sessions with no activity and the commands that were run during these sessions. Then we matched sessions ending with the same command or with fewer file reads. We verified that all sessions

with a greater number of files read contained some type of activity. To prevent errors caused by the data collection process, we verified that the sessions ended with a specific read activity that appeared in the login and logout. Using this procedure, a total of 421 empty sessions were found. The large number of empty sessions indicates that automatic tools are used to attempt dictionary attacks. After the tools achieve a successful login, they report the correct login and password to the attacker; no commands are executed.

The remaining analysis focused on the 320 non-empty sessions. The non-empty session were processed to find the files written, read, deleted or whose ownership or rights had changed. The scripts singled out all filesystem related system calls for each process contained in a session. We stored this information by process ID to simplify post-processing. Finally, the filesystem calls were processed to collect statistics for each session and individual statistics for each file.

4 Statistics on Filesystem Activity: Files

In this section, we present statistics related to the number of files that were read, written, deleted, or whose ownership or rights changed for the 320 non-empty sessions in the strace data. Only unique file activities were analyzed (in other words, duplicate file activities in the same session were discarded). For example, if a file was read several times within a session, the read activity for the file was counted once. However, if the same file was read and written to in the same session, the file was counted twice: once as read and once as written.

Table 1 contains the per session minimum, maximum, average and standard deviation of the number of files read, written, deleted or whose rights or ownership changed. All attacks included many (minimum of 20) different files reads as expected. However, surprisingly, some attacks completely lacked write or delete activity, but included rights or owner changes. As expected, the average number of files read was quite high (144.7) while the average number of files written was low (32.1). More surprising was the low average number of files deleted (6.6). Also interesting was the low average number of files whose rights changed (2.2) and the large number of files whose owner changed (17.5). The average number of files whose owner changed was significantly higher than the number of files deleted and the number whose rights changed and was equal to half the number of files written. More statistics on the number of files targeted can be found in [16].

We compiled the activity for each file that appeared at least once per session, disregarding the type of action that produced the activity. We removed file activity that was part of the normal login/logout process. We presented files with similar purposes together to identify attack patterns.

Table 1. Statistics on the Number of Files Targeted

	Read	Write	Delete	Rights	Owner
Minimum	20	0	0	0	0
Maximum	484	656	418	42	852
Average	144.7	32.1	6.6	2.2	17.5
St. Dev.	78.3	90.2	41.4	5.7	96.6

Reading files was the most common activity performed in the filesystem. However, the number of unique files read was small compared to other activities. This is partly because an important part of reading files is invoking libraries during execution and files providing system information. Most attackers performed common actions that led to a restricted set of libraries. For example, 165 sessions contained /usr/lib/libcrack.so.2 and 143 sessions contained /usr/libresolv.so.2, which are indications of password and network related activity, respectively. Malicious activity was evident in the number of times certain files were read that were related to hardware and software information. The read action /proc/cpuinfo appeared in 110 sessions and /proc/(PID)'/status in 97 sessions.

The most frequently written file was /etc/npasswd, which appeared in 161 sessions, showing that most attackers attempted password modifications. Also noteworthy was the corruption of related password files: /etc/shadow was corrupted in 14 sessions while creating accounts with blank passwords. Another common malicious activity was to install external programs: in a total of 57 sessions new files were created containing malware. Surprisingly, while the tools carried diverse names, after decompressing the malware, many shared common file names. For example, unix2.users appeared in 17 sessions and psybnc.pid appeared in another 17 sessions. The attackers' use of the later file showed an interesting property: all processes written to psybnc.pid appeared to be cloaked as harmless services (httpd, ssh, ntpd, init). However, the files used were not cloaked as system files, thus showing the filesystem data audit is a good vector to use to detect concealed malware posing as system services.

IDSs often monitor written activity on system files, especially binary files to detect rootkits and Trojan horses [17]. However, our results showed few attackers corrupted these files. Only two sessions replaced binary files. Attackers modified key system files (e.g., /etc/hosts.allow and /etc/services) in nine sessions.

Rights or ownership changes appeared most often as part of the malware installation. They also appeared on binary files as part of rootkit installations (two sessions) and as part of track deletions (two sessions).

Finally, the files that were deleted were those created by the attacker or in user logs, e.g., (bash_login), showing the cleanup performed by attackers. We found that most attackers neglected to perform cleanup, which was an unexpected finding. History files containing activity information were deleted 19 times via direct delete commands or targeted cleanup utilities. Leftover installation residues from malware were deleted in a total of 12 sessions.

5 Statistics on Filesystem Activity: File Sequences

In this section, we focus on sequences of files and activities. We analyzed sequences of consecutive files for the five activities considered (i.e., read, write, delete, ownership and rights). Then, we focused on longer sequences of targeted files and associated activity represented in state machines.

We define a state as a combination of a targeted file and the associated activity. For example, reading file /etc/nsswitch.conf, deleting file /etc/group.lock, or changing the rights of file /usr/lib/libsh/shsb are all defined as states. We define five states for the

activities associated with all the files created due to the installation and execution of malware ("read EXTERNAL", "write EXTERNAL", "delete EXTERNAL", "ownership EXTERNAL" and "rights EXTERNAL"). We define the END state as the state associated with the end of the session. Depending on the activity, we obtained different numbers of states. We counted 730 states for reading, 19 for writing, 129 for deleting, and only 3 and 1 for rights and ownership changes, respectively. We also observed different numbers of sequences of consecutive files. We observed 1954 sequences for reading, 20 for writing, 104 for deleting and only 3 and 1 for rights and ownership changes, respectively. The most frequently observed sequences (i.e., the top five if more than five sequences were observed) of consecutive states were (we provide in parenthesis the number of times this specific sequence was found):

- File reading:
  /proc/X/cmdline -> /proc/X/stat (14643)
  /proc/X/stat" -> /proc/X/cmdline (9696)
  /proc/X/status -> /proc/X/cmdline (5408)
  /proc/X/stat -> /proc/X/status (4944)
  /etc/selinux/config -> /proc/mounts (2687)
- File writing:
  EXTERNAL -> EXTERNAL (6543)
  bash_history -> END (238)
  EXTERNAL -> bash_history (9)
  EXTERNAL -> END (8)
  /etc/shadow- -> /etc/shadow+ (7)
- File deletion:
  EXTERNAL -> EXTERNAL (1592)
  /etc/group.lock -> /etc/gshadow.lock (6)
  /home/oracle/.Xauthority-n -> /home/oracle/.Xauthority-c (5)
  /home/oracle/.Xauthority-c -> /home/oracle/.Xauthority-l (5)
  /etc/passwd.lock -> /etc/shadow.lock (5)
- Ownership changes:
  EXTERNAL -> EXTERNAL (3)
- Right changes:
  EXTERNAL -> EXTERNAL (384)

With respect to file reading, the sequences corresponded to normal program behavior. We observed that for file writing, the most frequent sequences were associated with sequences from and to the EXTERNAL state, representing sequences of files written when malware was installed and executed. The other sequences were linked to session exits (i.e., the END state was the state associated with the end of the session). These results show that there is a high frequency of files that are written to as the last action in a session. The .bash_history files were not written to intentionally by attackers, but by bash, when the user exited the session. In many sessions (238), writing the .bash_history file was the last action that happened. For the remaining sessions (82), the last action depended on the way the attacker exited the session and if s/he disabled

bash_history logging. Hence a session that arrives to END without writing to bash might be an indication of malicious activity. With respect to file deletion, the most frequent sequences were from and to the EXTERNAL state representing sequences of files deleted when some attackers deleted the malware that they had previously installed. The other sequences are the result either of a password change or the addition of a user. As for right and ownership changes, the most frequent sequences are sequences from and to the EXTERNAL state representing sequences where files' rights and ownerships were changed when malware was installed and executed. In many cases we observed a file being written to followed by ownership changes before another file was written to and its ownership changed, etc. Such behavior is typically the result of a rootkit.

We now refine the analysis on the sequences of files targeted following an SSH compromise by considering longer sequences represented using state machines. For right and ownership changes, analyzing sequences of targeted files does not provide additional insight compared to the analysis of the individual files and associated activity described in Section 4. For the reading, writing and deletion attack activities, we created a state machine, where each state represents a file with the corresponding activity.

Over 1527 files contained some activity in the 320 non-empty sessions that led to a very large state machine. Therefore, we simplified the state machine by considering only transitions that occurred at least 150 times among the 320 sessions. In the simplified state machine (which is too large to be printed in this paper), we have 100 states instead of 1527 and 133 transitions instead of 3307. The state machine only has a few states with activity other than reading: write /etc/nshadow, write EXTERNAL, rights EXTERNAL, write bash_history and delete EXTERNAL. We observed several transitions occurred exactly 320 times, which is equal to the number of non-empty sessions. These files were involved in the login process following the SSH compromise. Certain files were clustered depending on the attacker activity. For example, we found a branch associated with downloading a file (the branch with the "read /usr/bin/wget" state). An excerpt of this branch is shown in Fig. 1a. We also observed a sequence that represented changing a password (the branch with the "read /usr/bin/passwd" state). Another branch (with state "read /usr/bin/w") can be linked to reconnaissance activity where the attacker was checking who was logged on (Fig. 1b contains an excerpt of that branch). Other reconnaissance activities can be spotted by following the branches that include /proc/stat, /proc/cputime, etc.

Fig. 2 indicates that the last step of a session often consists of writing a file (.bash_history). The last step can be initiated after a password change (write /etc/nshadow) or the installation and execution of malware (write EXTERNAL).

To make the state machine associated with the files deleted more readable, we only included transitions that were observed at least two times among the 320 non-empty sessions. The first sequence of states shown in Fig. 3 indicated a password change. The second sequence that consists of the delete EXTERNAL state, represents the deletion of malware installers that often contain multiple files. The third sequence is associated to the addition of a user.

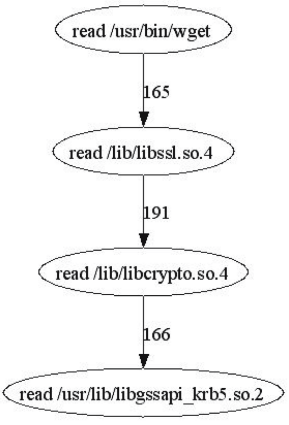


Fig. 1a. File Download Activity

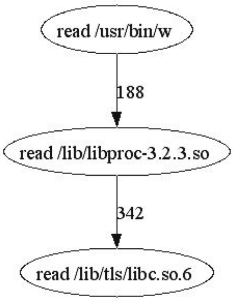


Fig. 1b. Reconnaissance Activity

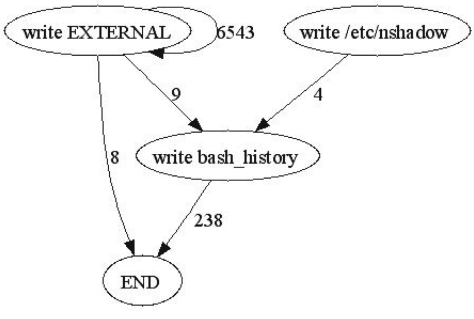


Fig. 2. State Machine for Files Written

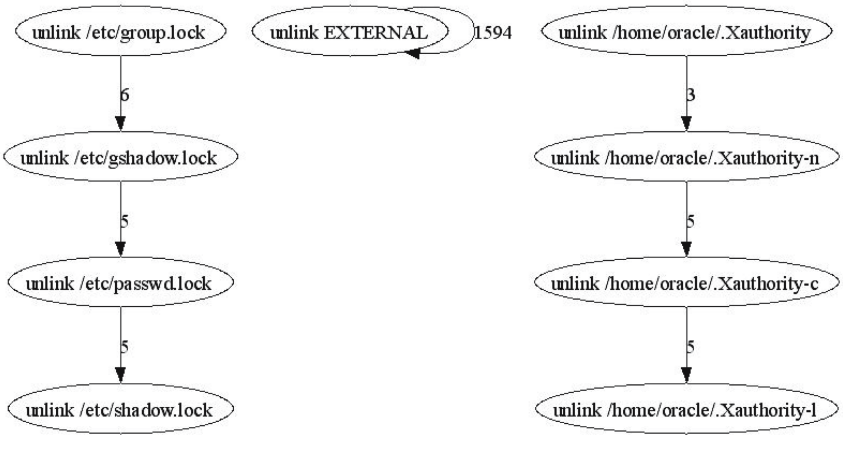


Fig. 3. State Machine for Files Deleted

6 Attack Sequences Related to Attacker Activity

In this section, we focus on the transition patterns related to a reduced set of states, where each state is associated to a typical attacker activity. The three states are: "read /proc/stat", associated with attacker reconnaissance, "write MALWARE", associated to downloading malware, and "write /etc/nshadow", associated with a password change. Fig. 4 and Fig. 5, respectively, represent the state machine with the number of transitions between states and the average time, in seconds, between states.

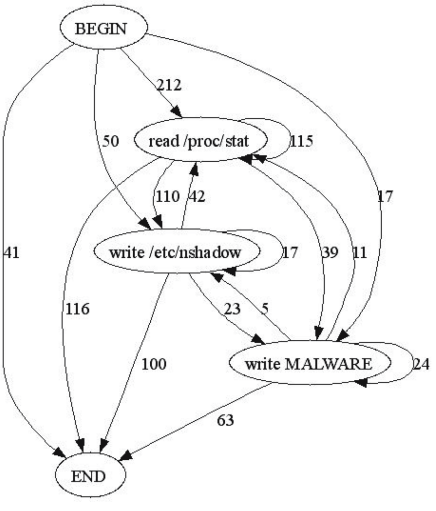


Fig. 4. Attacker Activity State Machine

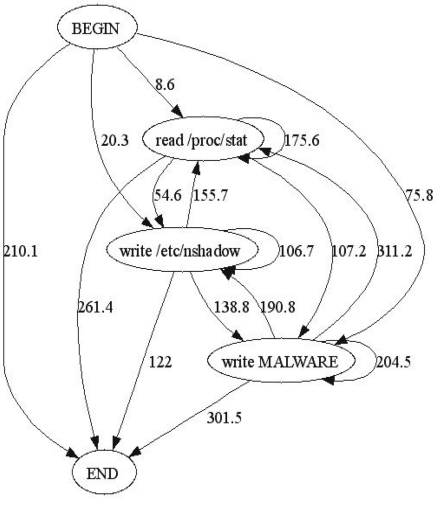


Fig. 5. Attacker Activity State Machine Average Time (in seconds)

In Fig. 4, we see that most attackers engaged in reconnaissance activities first (212 transitions to "read /proc/stat" from BEGIN). Modifying the password file is usually performed after learning about the system: the largest number of transitions comes from "read /proc/stat" (110). The most common action performed by an attacker before exiting is reconnaissance. This may be because attackers want to verify the final state of the machine or because the machine found could not satisfactorily perform malicious activities. Attackers rarely do not conduct any activity on the selected files; with just 41 transitions with only BEGIN to END. The "write MALWARE" state is most likely to be reached by the attacker after writing the password file or performing reconnaissance activities. This is natural, as the attacker wants to make sure the malware has been successfully installed and can be controlled at later time. Installing malware is usually the last file activity performed by the attacker, and only five and 11 transitions occurred from "write MALWARE" to "write etc/shadow" and to "read /proc/stat/", respectively.

A commonly asked question with intrusion detection systems is how promptly an attack can be detected when monitoring a specific file. In Fig. 5, we observed that most attackers modified the password file after checking the system first, which takes

on average one minute. Malware is often installed later in the session, consuming an average of 76 seconds if the malware installation occurs after the BEGIN state, increasing the time to an average of 139 seconds for malware written immediately after modifying the password. Installing malware took the longest time, and if it is the last action of the attacker, finalizing this activity will occur on average after 300 seconds have elapsed. This shows that filesystem IDSs based on malware matching will detect the attack later in the session, and most likely configuration files, including the password file will have been modified by the attacker already. An interesting result is how fast an attacker can modify the password file after the beginning of the session, taking only 20 seconds.

7 Related Work

Previous work on attacker behavior tended to focus on descriptions of high-level activity from an attacker-centric perspective. This was true for many publications appearing in the "Know Your Enemy" series [18] that focused on honeypot-related projects.

Alata and colleagues [19] performed an analysis of post-compromise attacker behavior. They observed that the most common first step in an attack was a password change. Most attackers also downloaded files (i.e. malicious programs), then tried to install and run executables. Similarly, Raynal and colleagues [20] performed an in-depth forensic analysis of post-compromise attacker behavior. They developed three general categories of attacker behavior: discovery, installation and usage.

Our objective was to detail the impact of the activity described in attacker centric studies from a defender-centric perspective, in this case the filesystem. Few such studies exist, all using other data sets. Killourhy and colleagues [21] explored data driven behavior to create a defense-centric taxonomy based on system call usage. They created an attacker-defender testbed with 25 attacks launched against a target machine vulnerable to all exploits. The behaviors of the 25 exploits were subdivided based on the trace of system calls. Barse and Jonsson [22] conducted a similar study to describe system log activity. They launched a set of attacks and described the manifestations of the attacks on different types of system log data. In a follow up [23], they described how to use the attack manifestations to automatically separate attacks from normal behavior on system logs.

A similar study to ours focused on non-malicious activity was presented in [24]. Sequences of filesystem accesses on a Linux system were classified as normal or anomalous, to create an anomaly-based IDS. The method was based on creating patterns in filesystem access based on a group of 5 normal users. The results displayed a false positive rate of 2%. This study could be complemented with known malicious patterns in order to decrease the false positive rate.

8 Conclusions

This paper describes an empirical study of filesystem activity after a security compromise. Quantitative data was collected on the use of files targeted by attackers

while reading, writing, deleting and modifying the file metadata. State machines were created based on file usage, to identify patterns between filesystem access and activity associated with attacks (e.g., password changes and malware installations).

The analysis of the files and associated activity pointed out three main attacker activities: 1) reconnaissance, 2) password changes, and 3) malware download and execution. When focusing on sequences of files and activities we observed, as expected, that most involved file reading. Besides file reading, all other most frequent activities involved the malware files. In the developed state machines, we also matched some branches with the main three attacker activities. We then built a simplified state machine based on these attacker activities to study the frequency of transitions between states and the average transition times. In most cases, reconnaissance is the first attacker activity, followed by changing the password. Malware download and execution was observed less frequently. The longest observed transition (between 200 and 300 seconds on average) is after having downloaded and executed malware (the installation of the malware seems to take itself on average 200 seconds). The information regarding the individual and sequences of files and activities and the frequency of transitions and the transition times, could be useful to improve the efficiency of IDSs.

Acknowledgments. The authors thank Daniel Ramsbrock and Robin Berthier for implementing the experiment that led to the data collected for this paper. This research has been supported in part by NSF CAREER award 0237493.

References

1. Warrender, C., Forrest, S., Pearlmutter, B.: Detecting Intrusions using System Calls: Alternative Data Models. In: 1999 IEEE Symposium on Security and Privacy, pp. 133–145. IEEE Computer Society Press, Los Alamitos (1999)
2. Petroni, N.L., Fraser, T., Molina, J., Arbaugh, W.A.: Copilot - a Coprocessor-based Kernel Runtime Integrity Monitor. In: 13th Conference on USENIX Security Symposium (2004)
3. Roesch, M.: Snort: Lightweight Intrusion Detection for Networks. In: 13th USENIX Conference on Systems Administration, pp. 229–238 (1999)
4. Kim, G.H., Spafford, E.H.: The Design and Implementation of Tripwire: A File System Integrity Checker. In: 2nd ACM Conference on Computer and Communications Security, pp. 18–29. ACM Press, New York (1994)
5. AIDE: Advanced Intrusion Detection Environment. http://www.cs.tut.fi/ rammer/aide.html
6. Miretskiy, Y., Das, A., Wright, C.P., Zadok, E.: Avfs: An On-Access Anti-Virus File System. In: 13th USENIX Security Symposium, pp. 73–88 (2004)
7. LIDS: Linux Intrusion detection system. http://www.lids.org
8. Wotring, B., Potter, B., Ranum, M.: Host Integrity Monitoring Using Osiris and Samhain. Syngress Publishing Inc. (2005)
9. Patil, S., Kashyap, A., Sivathanu, G., Zadok, E.: Fs: An In-Kernel Integrity Checker and Intrusion Detection File System. In: 18th USENIX Conference on System Administration, pp. 67–78 (2004)
10. Litty, L.: Hypervisor-based Intrusion Detection. Master's thesis, University of Toronto (2005)

11. Molina, J., Arbaugh, W.A.: Using Independent Auditors as Intrusion Detection Systems. In: 4th International Conference on Information and Communications Security, pp. 291–302 (2002)
12. Panjwani, S., Tan, S., Jarrin, K.M., Cukier, M.: An Experimental Evaluation to Determine if Port Scans are Precursors to an Attack. In: DSN 2005. International Conference on Dependable Systems and Networks, pp. 602–611 (2005)
13. Ramsbrock, D., Berthier, R., Cukier, M.: Profiling Attacker Behavior Following ssh Compromises. In: DSN 2007. International Conference on Dependable Systems and Networks, pp. 119–124 (2007)
14. Strace, http://sourceforge.net/projects/strace
15. Sebek, http://www.honeynet.org/tools/sebek
16. Molina, J., Gordon, J., Chorin, X., Cukier, M.: An Empirical Study of Filesystem Activity Following a SSH Compromise. In: ICICS 2007. 6th International Conference on Information, Communications and Signal Processing (to appear, 2007)
17. Brumley, D.: Invisible Intruders: Rootkits in Practice. In: login: Magazine, Intrusion Detection Special Issue (1999)
18. Honeynet, http://www.honeynet.org/papers/index.html
19. Alata, E., Nicomette, V., Kaaniche, M., Dacier, M., Herrb, M.: Lessons Learned from the Deployment of a High-interaction Honeypot. In: EDCC 2006. 6th European Dependable Computing Conference, pp. 39–46 (2006)
20. Raynal, F., Berthier, Y., Biondi, P., Kaminsky, D.: Honeypot Forensics, Part II: Analyzing the Compromised Host. IEEE Security & Privacy 2(5), 77–80 (2004)
21. Killourhy, K., Maxion, R., Tan, K.: A Defense-centric Taxonomy Based on Attack Manifestations. In: DSN 2004. International Conference on Dependable Systems and Networks, pp. 102–111 (2004)
22. Barse, E.L., Jonsson, E.: Extracting Attack Manifestations to Determine Log Data Requirements for Intrusion Detection. In: Yew, P.-C., Xue, J. (eds.) ACSAC 2004. LNCS, vol. 3189, pp. 158–167. Springer, Heidelberg (2004)
23. Larson, U., Lundin-Barse, E., Jonsson, E.: METAL: A Tool for Extracting Attack Manifestations. In: 2nd International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 85–102 (2005)
24. Stolfo, S.J., Hershkop, S., Bui, L.H., Ferster, R., Wang, K.: Anomaly Detection in Computer Security and an Application to File System Accesses. In: Foundations of Intelligent Systems, pp. 14–28 (2005)

A Secure Virtual Execution Environment for Untrusted Code

Yan Wen and Huaimin Wang

School of Computer, National University of Defense Technology,
Changsha, China, 410073
wenyan@nudt.edu.cn, whm_w@163.com

Abstract. This paper proposes a Secure Virtual Execution Environment called *Pollux* for untrusted code. Pollux achieves both the OS isolation and the functionality benefits provided by the isolated untrusted applications. It accomplishes the OS isolation by introducing a hosted virtual machine as the untrusted code container. The key feature of Pollux is its capability of reproducing the host execution environment, thus the behavior of isolated applications recurs as if they were running natively within the host OS. This characteristic is accomplished by the novel local-booted technology, which means the virtual machine boots not from a newly installed OS image but just from the preinstalled host OS. Thus, Pollux provides security against potential malicious code without negating the functionality benefits of benign programs. This paper focuses on the architecture of Pollux and outlines the implementation framework.

Keywords: Intrusion confinement, isolated execution, virtual execution environment, security, virtual machine.

1 Introduction

With the widespread deployment of firewalls and other similar network-based protection mechanisms, such as network intrusion detection systems and so forth, it has become more difficult for attackers to break into target computers through direct remote attack. However, even the best perimeter network-based solutions can be easily defeated by an attacker that can induce users to download and execute malicious programs. Under PC platform, users often execute downloaded freeware/shareware or mobile code. The risk of damage to the user's computer system due to untrusted code is high, yet a significant fraction of users seem to be willing to take this risk in order to benefit from the functionality offered by such code.

Some host-based mechanisms have been introduced to enhance the host security, i.e., access control, virus detection, and so on. But access control can be fooled by authorized but malicious users, masqueraders, and misfeasors. Although virus detection and similar technologies can be deployed to detect widely prevalent malicious codes, they are limited not only in theory but in practice, because in theory it is undecidable whether an arbitrary program contains a computer virus [1, 2], and in

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 156–167, 2007.

practice it is also very difficult to analyze the polymorphic or metamorphic virus code accurately.

Sandboxing is a more promising approach for defending against potential malicious code. In sandbox, the resource accesses required by untrusted code are restricted to ensure security. The main drawback of sandboxing based approaches is the difficulty of policy selection. Too often, sandboxing tools incorporate highly restrictive policies that preclude execution of most useful applications. The net result is that users end up choosing functionality over security.

As a supplementation, isolated execution is proposed to bound the damage caused by undetected or detected intrusions during their latencies without negating the functionality benefits of untrusted code. But under PC platform, current isolation approaches cannot achieve both the OS isolation and execution environment reproduction. The former is a prerequisite to make the system be immune to kernel-mode malicious code, and the latter is the key to reproduce the behavior of the untrusted code.

To address this problem, we propose a Secure Virtual Execution Environment (Pollux) for untrusted code. The untrusted code container of Pollux is a hosted system virtual machine (*Pollux* VM), so the OS isolation is guaranteed. With our local-booted technology, *Pollux VM* can boot from the underlying host OS instead of a newly installed OS image, viz. loads another instance of the host OS. In addition, no privileged operations will be restricted in this local-booted OS. Hence, the accurate behavior reproduction of untrusted code is assured while the host OS, acting as the trusted applications' container, is shielded from the effects of these untrusted code. Thereby, Pollux achieves both OS isolation and the behavior reproduction of untrusted applications, i.e., minimizing security risks without negating the functionality benefits provided by isolated programs.

The rest of the paper is organized as follows. In Section 2, we discuss the architecture of our approach and its advantages. Section 3 covers the implementation details of Pollux. And Section 4 provides an evaluation of the functionality as well as the Pollux performance. In Section 5, we review previous works on isolated execution technology. Last, we summary the main features of Pollux and outline the future work in Section 6.

2 The Architecture of Pollux

To accomplish both OS isolation and the functionality benefits of the untrusted code, the isolation mechanism provided by Pollux must meet three requirements as follows:

Requirement 1 ***OS Isolation*****:** The isolation mechanism should provide a virtual computer system which is isolated from the trusted applications. From the point of view of the trusted environment, this is a necessary condition for being resistant to the attacks from privileged malicious code.

Requirement 2 ***OS and Application Transparency*****:** No modification should be made to the conventional OS and applications for deploying this isolation mechanism. This requirement is composed of four sub-demands.

Requirement 2A. The source code of conventional OS and applications should not be patched for the isolation mechanism, because it is generally believed that there is no way to get the source code of the prevalent commercial applications and OSes of PC, such as Microsoft Windows etc.

Requirement 2B. Within the isolated untrusted environment, all the resource accesses and the privileged operations should not be restricted to ensure the functionality of isolated applications.

Requirement 2C. In trusted environment, the performance costs should be minimized.

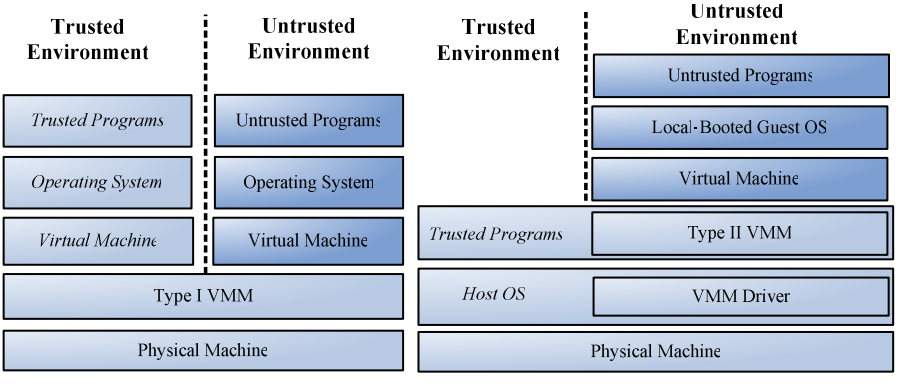
Requirement 2D. The pre-existing host OS should not be reinstalled for the isolation mechanism. Unlike mainframes that are configured and managed by experienced system administrators, desktop and workstation PC's are often pre-installed with a standard OS and managed by the end-user. In this environment, it is extremely important to allow the user to adopt isolation technology without losing the ability to continue using his existing OS and applications.

Requirement 3 ***Configurable Execution Environment Reproduction*****:** Since the behavior of an application is usually determined by the execution environment, especially the OS and the contents of file systems. So Pollux must reproduce the execution environment from the trusted environment. For this requirement, two aspects must been taken in account: security and performance.

Requirement 3A. This reproduction should not be implemented via duplicating the complete resource of trusted environment, for this causes so much deployment overhead that no PC user can afford it.

Requirement 3B. The resource to be reproduced to untrusted environment, especially the file system contents must be configurable for users. For enhancing the confidentiality, user should not reproduce the security-sensitive files.

To meet **Requirement 1**, the isolation mechanism must introduce the virtual machine monitor as the software layer to close off the trusted environment and untrusted ones. According to the definition of Goldberg [3], a virtual machine monitor (VMM) is software for a computer system that creates efficient, isolated



(a) Native Architecture based on Type I VMM (b) Hosted Architecture based on Type II VMM

Fig. 1. Alternatives of Pollux Architecture

programming environments that are "duplicates", which provide users with the appearance of direct access to the real machine environment. These duplicates are referred to as virtual machines, wherein a statistically dominant subset of the virtual processor's instructions execute on the host processor in native mode. There are two different types of VMMs that can create a virtual machine environment: Type I and Type II. Type I VMM runs on a bare machine. It is an OS with virtualization mechanisms. It performs the scheduling and allocation of the system's resources. A Type II VMM runs as an application. The OS that controls the real hardware of the machine is called the "host OS", which does not need or use any part of the virtualization environment. Every OS that is run in the Type II virtual machine is called a "guest OS." In a Type II VMM, the host OS provides resource allocation and a standard execution environment to each guest OS. Therefore, there are two types of architectures based on relevant VMM, as illustrated in Fig. 1.

Except for **Requirement 2C** and **Requirement 2D**, both of these architectures meet the aforementioned requirements all square. **Requirement 2C** focuses on the performance of the trusted environment. Within native architecture, shown in Fig. 1 (a), all OSes run above the virtual machine, thus every OS, including the one acting as the trusted environment, cannot but suffer the performance penalty introduced by VMM. But within hosted architecture, the trusted environment, i.e., the host OS, suffers no performance degradation introduced by virtualization. So in this regard, hosted architecture based on Type II VMM has the advantage of native architecture. Now considering **Requirement 2D** within native architecture, it would be unacceptable for a PC user to completely replace an existing OS with a VMM. In contrast, Type II VMM allows co-existing with a pre-existing host OS.

Besides, under PC Platform, pushing functionality from traditional OS down one layer in the software stack into the VMM, illustrated in Fig. 1 (a), has a prominent disadvantage for development: The diversity of PC's devices would remarkably increase the development complexity of Type I VMM. There is a large diversity of

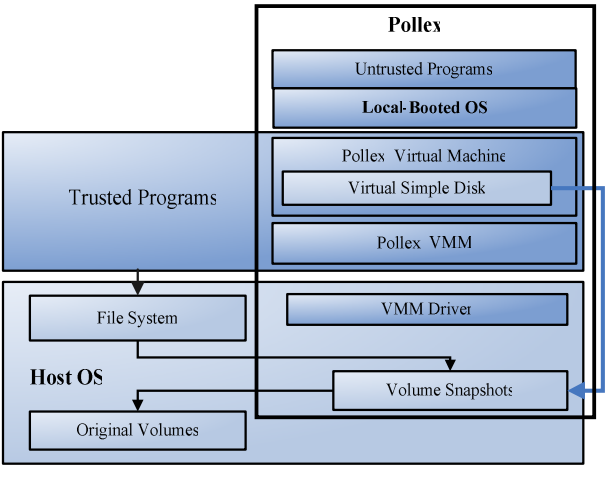


Fig. 2. The Architecture of Pollux

devices that may be found in PCs, which is a result of the PC's "open" architecture. In the implementation of Type I VMM, it would have to manage these devices. This will require a great programming effort to provide device drivers in the VMM for all supported PC devices. But Type II VMM can be simpler to implement by using well-known OS abstractions.

Of course, Type II VMM has its own disadvantages compared to Type I VMM. The most significant trade-off of it is potential I/O performance degradation within guest OS. But such I/O performance gap between them can be closed with the improvement of hardware virtualization technology [4, 5] and the virtualized I/O device optimization technologies [6].

In summary, considering that PC platform is the prime concern for Pollux, as well as the significant predominance of Type II VMM under PC platform, we select hosted architecture over native architecture. As illustrated in Fig. 2, the core components of Pollux are the *Pollux Virtual Machine Monitor (Pollux VMM)* and the *Local-Booted Pollux Virtual Machine (Pollux VM)*. The *Pollux VMM* runs above the host OS in the form of a Type II VMM and creates the local-booted virtual machines as the untrusted code container. The *Local-Booted OS*, wherein untrusted programs run, just boots in the *Pollux VM*. In another word, the *Local-Booted OS* just is a virtualized instance of the underling host OS. In these local-booted OSes, the behavior of untrusted programs is reproduced accurately while isolating their effects from the host OS, viz. the execution environment of the trusted applications.

To address the challenge of implementing a high-performance Type II VMM, especially for the non-virtualizable Intel x86 processors, we introduce an *Instruction Scan and Dynamic Translation (ISDT)* technology.

Another significant challenge to implement local-booting is how to reuse the system volume, wherein *Host OS* is preinstalled. While *Pollux VM* is running, the *Host OS* is also modifying the same system volume. However, the *Local-Booted OS* cannot be aware of these modifications and vice versa. So they will crash because of the content inconsistency between memory and disk.

Pollux resolves these conflicts by dint of the *Virtual Simple Disk* based on *Volume Snapshot*. *Volume Snapshot* shields the modification effects of *Host OS* from *Local-Booted OS* and vice versa. *Virtual Simple Disk* acts as the virtual storage device to export the *Volume Snapshot* to *Pollux VM*. Before exporting *Volume Snapshots*, the user can remove the files or folders he does not want to make visible inside Pollux. This is the way for Pollux to meet **Requirement 3B**.

Section 3 will outline how Pollux addresses these challenges.

3 Pollux Implementation

The hosted architecture of Pollux is shown in Fig. 2 and it is obvious OS-independent. But considering the prevalence of Windows and Intel processors under PC platform, Pollux has been firstly implemented in Windows with Intel x86 processors. Thus, the term of *OS* in Fig. 2 will be referred to *Windows* in the remainder of this paper.

3.1 Pollux VMM

The details of *Pollux VMM* implementation is beyond the scope of this paper. Instead, the framework of *Pollux VMM* is explained in this subsection. In a classically virtualizable architecture, all instructions that read or write privileged state can be made to trap when executed in an unprivileged context. But it's well-known that Intel x86 processor is not virtualizable [7]. To address this issue, we have come up with a set of unique techniques that we call *ISDT* (*Instruction Scan and Dynamic Translation*) technology, which is composed of two components: *Code Scanner* (*CS*) and *Code Patcher* (*CP*). Before executing any ring 0 code, *CS* scans it recursively to discover problematic instructions. *CP* then performs in-situ patching, i.e. replace the instruction with a jump to hypervisor memory where an integrated code generator has placed a more suitable implementation. We implement *CP* and *CS* based on the disassembler and translation engine of QEMU [8], a fast machine emulator using an original portable dynamic translator.

3.2 Virtual Simple Disk Based on Volume Snapshot

Just as its name implies, a *Volume Snapshot* is an identical copy of its original volume just at the time it was created via shielding the modification effects of *Host Windows* from *Pollux VM* and vice versa. In virtue of this characteristic, Pollux resolves the conflicts due to the writing accesses of *Host Windows* and *Pollux VM*. Furthermore, it can be mounted in *Host Windows* as a general volume to configure its contents to meet **Requirement 3B**.

To isolating the file system modification between *Host Windows* and *Local-Booted Windows*, *Volume Snapshot* adopts the Copy-on-Write (*COW*) mechanism, i.e., taking a copy of the sectors that will be modified on the original volume. In our approach, the place holding the copied data is called *Snapshots Area*, or *SArea* for short.

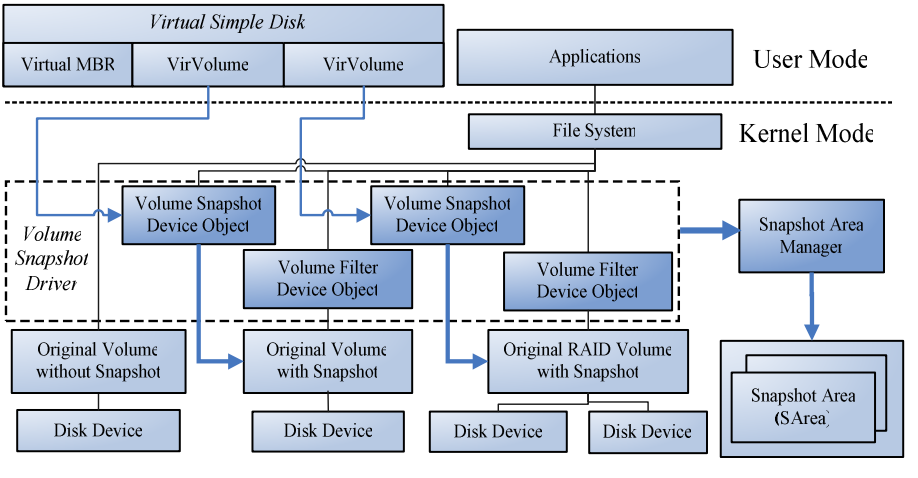


Fig. 3. Volume Snapshot Driver Architecture

Fig. 3 illustrates the architecture of the storage driver stack after installing the *Volume Snapshot Driver*. This driver creates two types of device objects, one is a *Volume Filter Device Object* located above the original volume to filter all the I/O Request Packages (IRP) sent to it and execute the *COW* operations, and the other is a *Volume Snapshot Device Object* which exports all general volume interfaces to provide a way to access *Volume Snapshots*. When a writing IRP comes down to the original volume, the filter device object will examine its parameters to check whether it will modify any of the sectors which have not been duplicated to *SArea*. If it will, the device object will hold back this IRP until the copy operation has been completed. And the contents written to snapshots will be also redirected to *SArea* via relevant volume device objects. Thus, the bidirectional isolation of file system modification is fulfilled. Consequently, the consistency of file systems in *Host Windows* and *Local-Booted Windows* is guaranteed.

The virtual storage device should be a physical disk but not a volume, so we introduce *Virtual Simple Disk* (*VSD*) to combine and expose the *Volume Snapshot*s to *Pollux VM*. As shown in Fig. 3, a *VSD* is composed of a set of *Volume Snapshot*s, a virtual primary partition table in MBR, and partition tables in extended partition. Within a *VSD*, a data structure called *VirVolume* is created to maintain the state parameters of its corresponding *Volume Snapshot*.

4 Evaluation

4.1 Evaluation of Functionality

Firstly, we evaluate the key function of Pollux, i.e., booting from the underling *Host Windows*. Fig. 4 is a screenshot of a running *Pollux VM* shown in the window with a caption of *Secure Virtual Execution Environment*. The resolution of *Local-Booted Windows* within *Pollux VM* is 1024x768 while the resolution of *Host Windows* is 1280x800, so the icon arrangement within its desktop differs from that of *Host Windows*. As shown in Fig. 5, the programs of *Explorer* and *MediaPlayer* are running in this *Local-Booted Windows*. Compared the file system volumes shown in the *Explorer* programs running within *Local-Booted Windows* and outside, we can find that only the volumes *C:* and *D:* are exported to it. This just brings forth the Pollux's capability of *Configurable Execution Environment Reproduction*: the resource to be reproduced to Pollux can be configurable for users.

Next, we test the basic functions of a system virtual machine, including instruction set and hardware virtualization. Instruction set virtualization is verified by the QEMU's *test-i386* tool, which we have ported to Windows. This tool tests all the x86 user-mode instructions, including SSE, MMX and VM86 instructions. The results show that the execution of all the instructions is equivalent with that in *Host Windows*.

Moreover, we ran PassMark on *Local-Booted Windows*. All the virtual hardware devices works perfectly well, including IDE disk, CD-ROM, network card, display adapter and so forth.

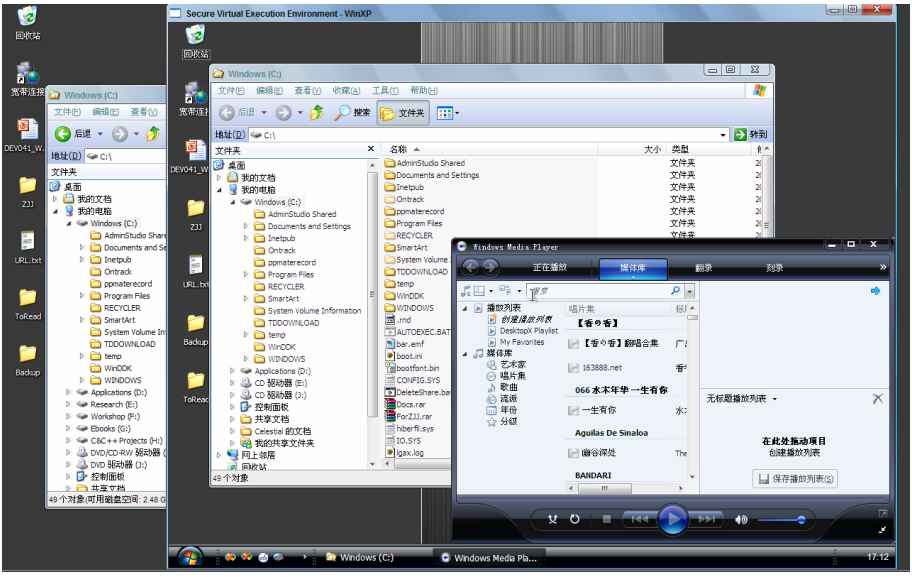


Fig. 4. Screenshot of a Running Pollux VM

4.2 Performance Evaluation

We have evaluated the performance for the two key components of Pollux: *Pollux VMM* and *Virtual Simple Disk* based on *Volume Snapshots*. The hardware host is a Samsung X30 notebook PC, containing a 1.60GHz Intel(R) Pentium(R) M processor and 1G bytes memory. The host OS is Windows XP SP2. We allocate 512M bytes memory for *Pollux VM* when testing its performance. And when performing evaluation in *Host Windows*, we only plug in 512M bytes memory. The tool we used to evaluate the performance of *Virtual Simple Disk* is the latest released version of *IoMeter* [9], which is a well-known open source I/O performance test tool.

We find that compute-intensive benchmarks run essentially at native speed on *Pollux VMM*. However, for the workloads including progressively more privileged operations (context switches, memory mapping, I/O, interrupts, system calls), *Pollux VMM* suffer more overheads.

Table 1. CPU Performance Results for Volume Snapshot Driver

	CPU Photo Worxx	CPU Queen	CPU Zlib	FPU Julia	FPU Mandel	FPU SinJulia	Mem Copy	Mem Latency	Mem Reading	Mem Writing
Host Windows	6751	1345	9989 KB/s	1244	588	2135	1980 MB/s	104.3 ns	2538 MB/s	2081 MB/s
Local-Booted Windows	6723	1326	9786 KB/s	1192	574	2105	1956 MB/s	104.9 ns	2490 MB/s	2039 MB/s
% of native	99.59	98.59	97.97	95.82	97.62	98.59	98.79	99.43	98.11	97.98

Pollux VMM. For a desktop-oriented workload, we ran Everest Ultimate 2006 both natively and in a *Local-Booted Windows*. Everest Ultimate is a synthetic suite of microbenchmarks intended to isolate various aspects of workstation performance. Since user-level computation is almost not taxing for VMMs, we expect *Local-Booted Windows* runs to score close to native. Table 1 confirms this expectation, showing a slowdown over native of 0.41-4.18%, with a 1.75% average slowdown for *Pollux VMM*.

Virtual Simple Disk based on Volume Snapshots. For the volume snapshot driver changes the I/O flow of original volumes, the reading/writing performance of these volumes will be depressed. To analyze the performance overhead clearly, we define a function named as *IoTime (Op, Target)*, *Op* refers to the type of I/O operation, including *Reading* and *Writing*, and *Target* indicates the volumes from where the program reads or writes. So *IoTime* defines the I/O time spent on reading or writing the *Target*. And there are three types of targets: *Original Volume* (*OV*), *Snapshot Area* (*SA*) and *Volume Snapshot* (*VS*). We define a constant value named *TestTime,* which is the time that snapshot driver spent on calculating the block number of the I/O request and testing whether this block is dirty. And we define *P (Block No)* as the probability that the block to write is dirty.

Thus, we can calculate the I/O time of the original volume with snapshots by the following formulas.

$$\text{IoTime}(Read, OV) = TestTime + \text{IoTime}(Read, OV) \tag{1}$$

$$\text{IoTime}(Write, OV) = TestTime + \text{P (Block No)x}(\text{IoTime}(Read, OV) + \text{IoTime}(Write, SA)) + \text{IoTime}(Write, OV) \tag{2}$$

And the following formulas presents the I/O time of the volume snapshot.

$$\text{IoTime}(Op, VS) = \begin{cases} \text{IoTime}(Op, SA), \text{ if the block is dirty} \\ \text{IoTime}(Op, OV), \text{ if the block is not dirty} \end{cases} \tag{3}$$

For the *TestTime* only comprises two arithmetic instructions and two memory access instructions, the former includes a *right shift* operation and a *bit test* operation. These two arithmetic instructions can all be completed in one instruction circle. Therefore *TestTime* can be ignored by compared with disk I/O operation time.

After examining the above formulas, we can get the conclusion that the overheads of reading original volume and reading/writing volume snapshot can be ignored. Indeed our test shows that the disk I/O of these three types incurs overheads typically less than 5%.

But for writing the original volume with snapshots, the second formula shows that the maximum overhead will be one writing operation added by one writing operation. However the COW algorithm incurs overhead to the same block only once, for one block has been performed COW; the subsequent writing requests can modify this block directly.

We also test the CPU performance of volume snapshot driver, and collect the results in Table 2. In this benchmark, the writing operations occupy 23% while the left is reading operations. And the I/O address in disk is uniform random number.

Table 2. Performance Comparison between host OS and Pollux VM

	Original Volume	Original Volume with Snapshot	Volume Snapshot
% CPU Utilization	2.785	4.184	4.891
%User Time	0.0333	0.0659	0.0012
%Privileged Time	2.755	4.146	4.910
% DPC Time	0.004	0.038	0.036
% Interrupt Time	2.173	2.605	3.660
Interrupt per Second	223.421	233.827	235.783

Results shows that CPU incurs overheads typically less than 3% compared with the native original volume without snapshots.

5 Related Work

Isolation Technology within Mono-OS. Isolated execution has previously been studied by researchers in the context of Java applets [10, 11]. Compared with general applications, such applets do not make much access to system resources. So the approach used by applets often relied on executing these untrusted applets on a "remote playground", i.e., an isolated computer. However, most of the desktop applications will usually require access to more resources such as the file system on the user's computer. To run such applications on a remote playground, the complete execution environment on the user's computer, especially the entire file system contents, should be duplicated to the remote playground.

Alcatraz [12] and its improved version [13], *Security Execution Environment (SEE)*, also have the capability of reproducing the behavior of applications, as if they were running natively on the underlying host OS. But this approach does not achieve OS isolation, so such protection mechanism can be bypassed by kernel-mode malicious code. And in SEE, a number of privileged operations, such as mounting file systems, and loading/unloading modules are not permitted.

Isolation Based on Virtual Machine. Covirt [14] proposes that most of applications may be run inside virtual machine instead of host machines. User-mode VMs have been proposed for the Linux OS [15]. All the above approaches suffer from the difficulty of environment reproduction.

Denali [16] is another virtual machine based approach that runs untrusted distributed server applications. Denali focuses on supporting lightweight VMs, relying on modifications to the virtual instruction set exposed to the guest OS and thus requiring modifications to the guest OS. In contrast, we are focusing on heavier weight VMs and make no OS modifications.

VMWare ESX Server provides an isolation approach for server platform with a similar objective to ours. XEN [17] and L4-based virtual machine [18] also implement isolated virtual execution environments. But all of these three environments are just located above computer hardware in form of Type I VMM. So

as discussed in section 2, they are not fit for PC platform because of their drawbacks caused by the native architecture of Type I VMM.

There are also some isolation mechanisms using Type II VMM, such as VMWare Workstation [6] and Parallels Workstation [19]. But they all cannot fulfill the computing environment reproduction. The COW/COW2 technology of QEMU [8], an open source emulator, can only isolate the modifications of guest OS from the file system in host OS. But it cannot shield the guest OS from the modifications made by host OS. Thus, the conflicts between the disks and file system content in guest OS trend to occur and the guest system will crash, so QEMU also failed to achieve the environment reproduction. Besides, its poor performance also prevents it from serving as an effective virtual execution environment. KVM [20], a Kernel-based Virtual Machine based on QEMU, significantly improves the performance. But it also cannot provide the capability of environment reproduction. In addition, it must modify the host OS and rely on the hardware virtualization technology, viz. Intel VT [21] and AMD-V [22].

6 Conclusions and Future Work

In this paper, we proposed an approach named Pollux for realizing a secure virtual execution environment and implemented its prototype. Pollux is versatile enough just like other virtual machines and isolation solutions. The most important benefit of Pollux is that it provides the capability of execution environment reproduction while accomplishing OS isolation. As mentioned in section 5, other isolation solutions based on virtual machine technology cannot provide such consistency between isolated environments and host OS.

To employ Pollux, it's no need to modify the existing OS or applications. And our functional evaluation illustrates the effectiveness of the approach. The evaluation shows that *Pollux VMM* comes very close to native performance, reaching 98.25% on average. And *Virtual Simple Disk* I/O incurs overheads typically less than 10%, and CPU incurs overheads less than 3%.

Thus, in virtue of OS isolation and execution environment reproduction, Pollux provides security against potential malicious code without negating the functionality benefits provided by benign programs.

We are currently improving the memory management mechanism of Pollux by sharing the memory between *Pollux VM* and host OS in order to reduce the memory overhead. Multiprocessor virtualization capability is also to be added to Pollux VM to support Multiprocessor-Specialized host OS version. In addition, we are integrating some intrusion detection mechanisms into Pollux. Finally, we will investigate the technology of committing changes within local-booted OS to host OS.

References

1. Cohen, F.: Computational Aspects of Computer Viruses. Computers & Security 8, 325–344 (1989)
2. Chess, D.M., White, S.R.: An Undetectable Computer Virus (2000)

3. Goldberg, R.P.: Architectural Principles for Virtual Computer Systems, Ph.D. Thesis. Harvard University, Cambridge, MA (1972)
4. Neiger, G., Santoni, A., Leung, F., Rodgers, D., Uhlig, R.: Intel® Virtualization Technology: Hardware Support for Efficient Processor Virtualization. Intel Technology Journal 10, 167–177 (2006)
5. Adams, K., Agesen, O.: A Comparison of Software and Hardware Techniques for x86 Virtualization. In: ASPLOS 2006. Proceedings of The 12th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 2–13 (2006)
6. Sugerman, J., Venkitachalam, G., Lim, B.-H.: Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In: Proceedings of the 2001 USENIX Annual Technical Conference, Boston, Massachusetts, USA (2001)
7. ScottRobin, J.: Analyzing the Intel Pentium's Capability to Support a Secure Virtual Machine Monitor, Master's Thesis. Naval Postgraduate School, Monterey, CA, p. 133 (1999)
8. Bellard, F.: QEMU, a Fast and Portable Dynamic Translator. In: USENIX Association Technical Conference (2005)
9. The IoMeter Project, http://iometer.sourceforge.net
10. Chiueh, T.-c., Sankaran, H., Neogi, A.: Spout: A Transparent Distributed Execution Engine for Java Applets. In: Proceedings of the 20th International Conference on Distributed Computing Systems, p. 394 (2000)
11. Malkhi, D., Reiter, M.K.: Secure Execution of Java Applets using A Remote Playground. IEEE Transactions on Software Engineering 26, 1197–1209 (2000)
12. Liang, Z., Venkatakrishnan, V.N., Sekar, R.: Isolated Program Execution: An Application Transparent Approach for Executing Untrusted Programs. In: Omondi, A.R., Sedukhin, S. (eds.) ACSAC 2003. LNCS, vol. 2823, Springer, Heidelberg (2003)
13. Sun, W., Liang, Z., Sekar, R., Venkatakrishnany, V.N.: One-way Isolation: An Effective Approach for Realizing Safe Execution Environments. In: NDSS 2005. ISOC Network and Distributed System Security (2005)
14. Chen, P.M., Noble, B.D.: When Virtual is Better Than Real. In: 8th Workshop on Hot Topics in Operating Systems (2001)
15. Dike, J.: A User-mode Port of the Linux Kernel. In: Proceedings of the 4th Annual Linux Showcase & Conference, Atlanta, Georgia, USA (2000)
16. Whitaker, A., Shaw, M., Gribble, S.D.: Denali: A Scalable Isolation Kernel. In: Proceedings of the Tenth ACM SIGOPS European Workshop, Saint-Emilion, France (2002)
17. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauery, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. In: SOSP 2003. Proceedings of the 19th ACM Symposium on Operating Systems Principles, pp. 164–177. ACM Press, New York (2003)
18. Biemueller, S., Dannowski, U.: L4-Based Real Virtual Machines - An API Proposal. In: Proceedings of the MIKES 2007: First International Workshop on MicroKernels for Embedded Systems, Sydney, Australia, pp. 36–42 (2007)
19. SWsoft: http://www.parallels.com/en/products/workstation
20. Qumranet: KVM: Kernel-based Virtualization Driver (2006)
21. Uhlig, R., Neiger, G., Rodgers, D., Santoni, A.L., Martins, F.C.M., Anderson, A.V., Bennett, S.M., Kägi, A., Leung, F.H., Smith, L.: Intel Virtualization Technology. IEEE Computer 38 (2005)
22. AMD: AMD64 Vrtualization Codenamed "pacifica" Technology: Secure Virtual Machine Architecture Reference Manual (May 2005)

Liveness Detection of Fingerprint Based on Band-Selective Fourier Spectrum

Changlong Jin[1], Hakil Kim[1], and Stephen Elliott[2]

[1] Biometrics Engineering Research Center,
School of Information and Communication Engineering, INHA University, Korea
{cljin,hikim}@vision.inha.ac.kr,
[2] Biometric Standards Performance and Assurance Laboratory, Purdue University
elliott@purdue.edu

Abstract. This paper proposes a novel method for fingerprint liveness detection based on band-selective Fourier spectrum. The 2D spectrum of a fingerprint image reflects the distribution and strength in spatial frequencies of ridge lines. The ridge-valley texture of the fingerprint produces a ring pattern around the center in the Fourier spectral image and a harmonic ring pattern in the subsequent ring. Both live and fake fingerprints produce these rings, but with different amplitudes in different spatial frequency bands. Typically, live fingerprints show stronger Fourier spectrum in the ring patterns than the fake. The proposed method classifies the live and the fake fingerprints by analyzing the band-selective Fourier spectral energies in the two ring patterns. The experimental results demonstrate this approach to be a promising technique for making fingerprint recognition systems more robust against fake-finger-based spoofing vulnerabilities.

Keywords: Fingerprint, Liveness detection, Band-selective, Fourier Spectrum, Ridge-valley texture.

1 Introduction

Traditional secure tools, for example, keys and passwords, have disclosed their weaknesses such as theft and oblivion. On the contrary, biometric systems utilize the information of the human body and as such, have a higher perception of security and convenience. However, recent studies [1, 2] show that biometric systems are subject to various threats. Within automated fingerprint recognition systems, it is possible to spoof a variety of fingerprint sensors using fake fingers made of silicone or gelatin (i.e., gummy finger attack). The challenge to detect whether the biometric sample presented is from a live finger or not is becoming an active research issue.

The focus of this paper is to propose a novel methodology for detecting fingerprint liveness. Silicone, gelatin, and rubber are some of the materials that can be used to make fake fingers. Many approaches for differentiating between live and fake fingers are proposed on the basis of physiological characteristics such as blood pressure, temperature, odor, hardness and perspiration.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 168–179, 2007.

Two major approaches to liveness detection can be implemented. The first is the hardware-based approach including 1) blood pressure detection [3], 2) temperature detection [4], 3) skin electric resistance or electric capacity detection [4], and 4) odor detection [5]. These methods take advantage of these explicit physiological features, but are expensive and bulky. The other is the software-based approach, which is more complicated but does not require additional accessories as in the hardware-based approaches. Existing fingerprint sensors can easily use this software-based approach by just modifying the software. There are three types of methods:

1) Perspiration-based method [6, 7, 8, 9, 10]
Only a live finger can perspire and the perspiration can be detected by image analysis. This method is the most valuable and has been studied by many researchers. This method is susceptible to a number of factors including sensitivity to the pressure of the finger, the environment, user, and time interval.

2) Skin deformation-based method [11, 12]
This method is based on the difference of hardness (or elasticity). The difference of hardness will produce different deformations when pressing and rotating a finger on a sensor. Liveness can be detected by comparing these distortions. The key point of this method is the difference of the material hardness. Thus, the method performs poorly when the hardness of fake material is similar to live skin, and users need some training process.

3) Image quality-based method [13]
In fact, it is difficult to make a fake fingerprint image having the same or better image quality than that of live. In general, the quality of the fake fingerprint image is not good as live fingerprint image. Moon et. al.[13] detected the liveness of a fingerprint by calculating the standard deviation of the fingerprint image using the wavelet transform. The advantage of this method is that it is fast and convenient to use. Although Moon's work is only conceptual, it contributes an important hint that we can detect the liveness by checking the image quality.

This paper proposes a classification of live and fake fingerprints based on band-selective Fourier Spectral energy for fingerprint liveness detection. Fingerprint is interleaving texture of ridge-valley. When transforming the fingerprint into spectral domain, it will produce mainly two ring patterns (refer to Fig. 1). The radius of the first ring pattern reflects the fingerprint image ridge distance from $d = N/r$[14], where d, N, r refer to in turn the ridge distance in spatial domain, the fingerprint image width in spatial domain, and the radius of the first ring pattern. In other words, the strength of the amplitude (so-called spectral energy) reflects the strength of the ridge-valley texture.

When we make fake fingers of Gelatin and Silicone[2], it is hard to copy the live fingerprint exactly. With different material and different shape of fake fingers, there will be difference between live fingerprint and fake fingerprint images even if the fake fingerprints look very similar with the live ones. Therefore, liveness detection can be done by analyzing the difference of the spectral energies.

This paper consists of four sections: the proposed method is described in detail in Section 2, experimental results are reported in section 3, and section 4 draws some conclusions and future works.

2 Proposed Method

In this paper, we propose a novel method to detect the fake finger based on band-selective Fourier spectral energy. Fingerprint is a texture with the interleaving of ridge and valley. When the fingerprint is transformed into the spectral domain, there will be an annulus region, so called "inner ring" in this paper. The inter-ridge distances in 500 DPI fingerprint range from 7 to 10 pixels[14, 16], and then the corresponding radius of the inner ring in the spectral image varies from 28 to 46 [14] (Sensor A: 280x320). There is difference on the clearness of ridge-valley texture between the live and the fake fingerprint, which makes the difference on the strength of the inner ring. Therefore, the accumulated energy in the inner ring can be a feature in liveness detection. In this paper, the energy of the interval changing from 25 pixels to 59 pixels was calculated.

The harmonic of the inner ring is produced from 60 to 100 pixels, which is called "outer ring." The energy of this outer ring also depends on the ridge-valley texture and different for the live and fake fingerprint. The accumulated value is then used as the second feature in this method.

There are differences on overall spectral energy between live and fake spectral images. These differences are made by the size of the foreground of fingerprint image, the distribution of histogram, and the performance of the sensors. Therefore, the overall energy between 1 to 100 pixels is used as the third feature in this method. Figure 1 depicts the computation of spectral energies in the inner and the outer rings.

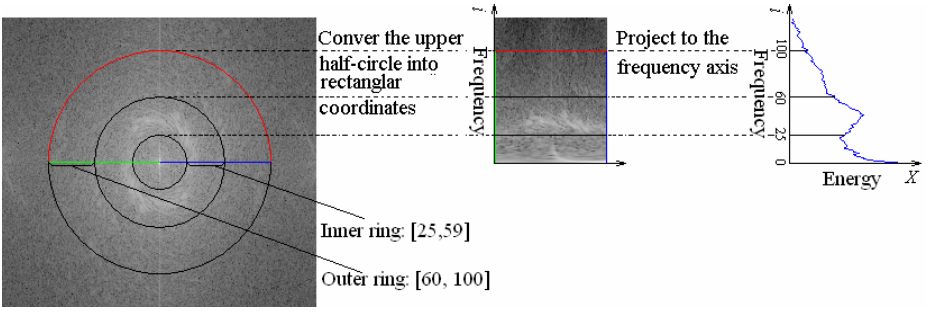


Fig. 1. The spectral image of fingerprint, the two ring patterns, and the computation of the Band-Selective energies

Figures 2 describes the procedure for computing the energy in these three intervals. At first, the fingerprint image is converted into the spatial frequency domain using Fast Fourier Transform. In order to avoid a big value contrast, logarithm operations are applied to the transformed image, and then the result image is normalized. Subsequently, the upper half-circle of the spectral image is converted into a rectangular coordinate using the homogeneous rubber sheet model presented by Daugman [15] and then projected to the frequency axis. Finally, these energies are accumulated on the three intervals: 25~59, 60~100, 1~100 using equations 1, 2 and 3.

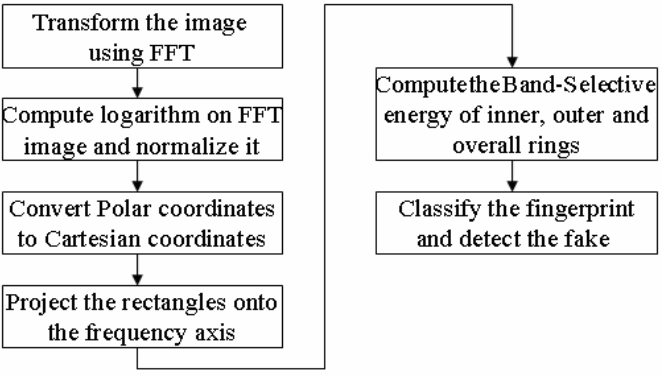


Fig. 2. Diagram of the proposed algorithm

$$E_{inner} = \sum_{i=25}^{59} \left| X_i \right|^2 \tag{1}$$

$$E_{outer} = \sum_{i=60}^{100} \left| X_i \right|^2 \tag{2}$$

$$E_{overall} = \sum_{i=1}^{100} \left| X_i \right|^2 \tag{3}$$

where i is the index of the frequency and $\left| X_i \right|^2$ refers to the energy at the spatial frequency i.

The feature vectors of the live fingerprint and the fake fingerprint based on these three energies have a cluster property in 3D feature space. Figure 3 shows the distribution of the overall data and figure 4 shows the distribution of all lives and all fake fingerprints of one finger. Because the fake fingerprints are copied from the molds (refer to the subsection 3.1) and the molds are copied from the live fingers, some detailed information of live finger was lost. In addition, the characteristics of fake material are different with those of live fingers. Therefore, it is hard to copy the live finger exactly by fake materials, consequently, there can be slight differences between the ridge-valley texture of fake and live.

When fingerprints transform into spectral domain, the energies of the ring patterns are different. Figure 3 shows that the center of fake energy cluster is lower than the center of live, namely the qualities of ridge-valley texture of fake fingerprint are lower than that of live fingerprint. Although two category clusters of overall samples overlapped for the most part, the energy vectors for an individual fingerprint are more separable between live and fake (as shown in figure 4). In this study, the energy vectors for live fingerprints are assumed to be normally distributed, and an elliptic classification boundary is designed to classify the live samples against gelatin and silicone fingerprints.

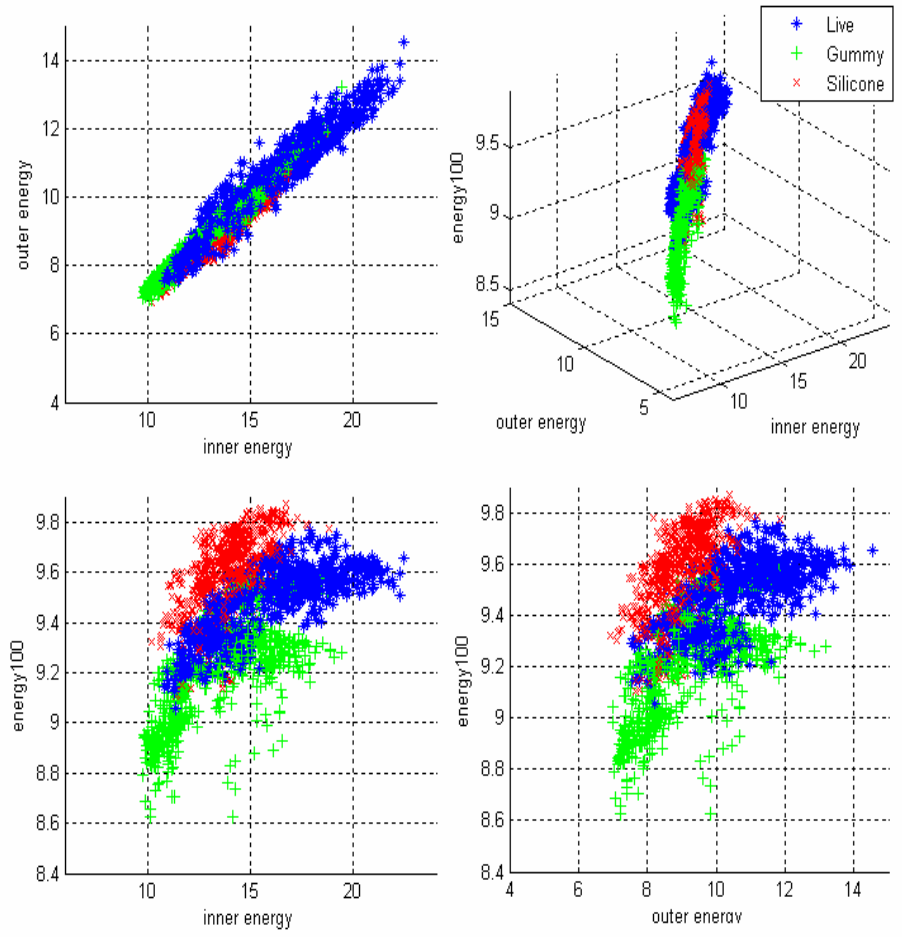


Fig. 3. Distributions of the overall data in 3 dimensions

In order to facilitate the classification boundary, the coordinates are rotated and moved to the center of the live samples. At first, the mean vector and covariance matrix are computed using equations 4 and 5, and then the eigenvalues and eigenvectors of the covariance matrix are computed. The coordinates are then moved by subtracting the mean from the live samples, and rotated by the eigenvectors. This process is depicted in figure 4.

$$\hat{\mu} = \frac{1}{N}\sum_{j=1}^{N} S_j \tag{4}$$

$$\hat{\Sigma} = \frac{1}{N-1}\sum_{j=1}^{N} (S_j - \hat{\mu})(S_j - \hat{\mu})^T \tag{5}$$

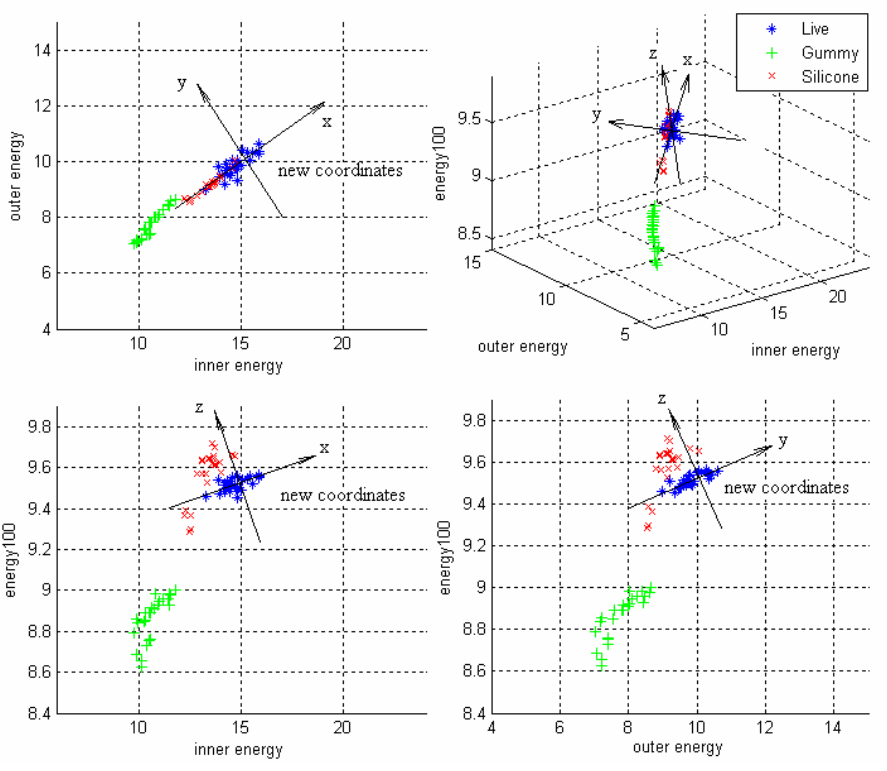


Fig. 4. Distributions of all live and all fake fingerprint of one finger

where S represents a sample vector consisting of the three band-selective energies, and N is the number of training samples.

In the new coordinates, the parameters *a, b, c* of the standard elliptic classification boundary are evaluated using equations 6, 7 and 8. Since the training samples are in a subset of the live samples, there are some errors. Therefore, a constant k is multiplied to the three parameters to adjust the elliptic classification boundary as shown in equation 9.

$$a = \left(\frac{1}{N-1} \sum_{i=1}^{N} (S_{x_i} - \overline{S_x})^2 \right)^{1/2} \tag{6}$$

$$b = \left(\frac{1}{N-1} \sum_{i=1}^{N} (S_{y_i} - \overline{S_y})^2 \right)^{1/2} \tag{7}$$

$$c = \left(\frac{1}{N-1} \sum_{i=1}^{N} (S_{z_i} - \overline{S_z})^2 \right)^{1/2} \tag{8}$$

$$\frac{x^2}{k \cdot a^2} + \frac{y^2}{k \cdot b^2} + \frac{z^2}{k \cdot c^2} = 1 \tag{9}$$

where $\overline{S}_x = \frac{1}{N}\sum_{i=1}^{N} S_{x_i}$, $\overline{S}_y = \frac{1}{N}\sum_{i=1}^{N} S_{y_i}$, $\overline{S}_z = \frac{1}{N}\sum_{i=1}^{N} S_{z_i}$, and S_x, S_y, S_z are the inner energy, outer energy and the overall energy of the sample S, respectively, a, b, c are the radii of the elliptic classification boundary, k is a constant of the radius, x, y, z correspond to inner energy, outer energy and overall energy respectively. (Refer to fig 4)

3 Experimental Results

3.1 Database

In order to evaluate the performance of the proposed approach, a database of fingerprints was collected from 30 subjects using three different commercial optical fingerprint scanners (Sensor A, Sensor B, Sensor C), without activating the sensor liveness detection function. Five impressions at different pressures (300g; 600g; 900g; 1200g; 1500g) for each finger were collected.

To generate fake fingerprint images[2], finger molds which are made by dental impression materials were created from thirty subjects and Gelatin and Silicone were used to form the fake fingertip. Then, fake fingerprint images are captured for each fake fingertip in the same manner as live fingertips. Total 6,750 images were collected {6,750 images = 90 fingers (30 lives, 30 Gelatin, 30 Silicone) × 5 impressions × 5 pressures × 3 sensors}.

3.2 Results

The experiments in this work are divided into two phases: One is the identification, and the other is the verification. In identification, the Support Vector Machine (SVM) was utilized as a classifier, where 2 of 5 samples were used as the training data and the rest as the testing data. The classification results are shown in table 1.

Table 1. Classification errors for Live/Fake Identification

Error rate (%) / Sensor	Live (450)	Fake(Gelatin, Silicone) (900)	Overall (1350)
Sensor A	22	11	15
Sensor B	14	6	9
Sensor C	34	18	23
Average error rate (%)	23	12	16

Experimental results show that the error rates of live are large than that of fake, but it depends on the classification boundary among the clusters in Fig. 3. The error rates of Sensor C are big than those of the other two sensors. This implies that Sensor C reproduces more detailed ridge-valley texture of fingerprint than the other two sensors, which becomes the weakness in liveness detection.

In verification phase, 5 live samples selected randomly out of 25 samples were used as the training data to evaluate the elliptic classifier, and the rest of live samples, 25 Gummy samples and 25 Silicone samples were used for testing, and all the test samples should be converted to the new coordinates described in equations 6 through 9. Because the training samples were selected randomly, so there should be an error for the elliptic classification boundary. In order to remove the randomness, the experiments were repeated 30 times by selecting different training samples and find the average error rate for every constant k. Figure 5, 6 and 7 show these results.

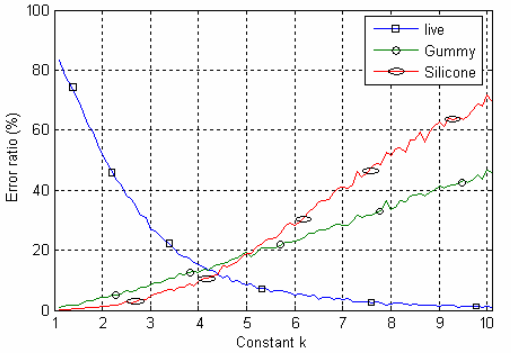


Fig. 5. Error rate of Sensor A

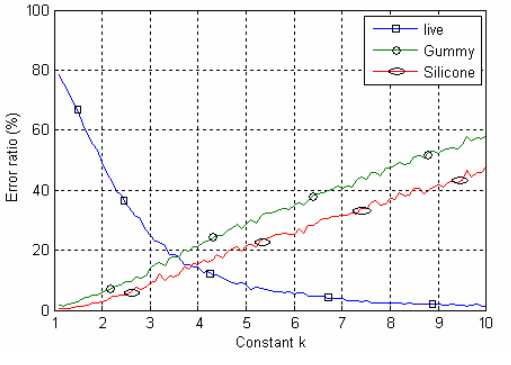


Fig. 6. Error rate of Sensor B

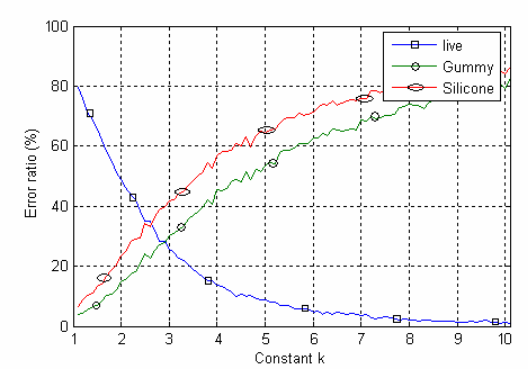


Fig. 7. Error rate of Sensor C

Three different commercial optical fingerprint scanners were utilized in this paper. Although these are all optical sensors, they come from different companies, and have different configuration of optical systems such as direction of the sensor prism, the number of lens, and the position and wavelength of the light source. Therefore, the optical characteristics of each sensor are different. This made the difference of fingerprint images scanned by each sensor. The cluster distance of Sensor A between fake and live are large than that of Sensor B and C, and the cluster center of Sensor C is the most shortest. Namely, the elliptic classifiers' radius of Sensor A can be larger than that of Sensor B and C, and the radii of Sensor C is the most smallest.

The constant k will influence the size of the classification boundary. As the increase of the constant k, the classifier boundary will be enlarged, so the error rate of live will decrease and the error rate of fake will increase. Experimental results show that the optimized constant k is different for each sensor. The best k lies in between 4 and 4.7 for Sensor A, between 3.2 and 4 for Sensor B, and between 2.4 and 3 for Sensor C.

Table 2. Performance compares for proposed method with other previous methods

Method	Error rate (%)	Precondition
Perspiration-based methods[2,3]	Approximately 10	Need to capture image pairs
Skin deformation-based method[11]	Approximately 16	Hardness hypothesis. Need a fingerprint scanner capable of capturing and delivering frames at proper rate.
Proposed method	Approximately 16 (Identification phases), 21 (Verification phases)	Only need one image

Table 2 compares the performance of the proposed method with other previous methods. Perspiration-based method needs to capture image pairs in two or more second. The performance of this approach is susceptible to a number of factors. The skin deformation-based method assumes that the hardness of fake fingers is higher than the live fingers, but the hardness depends on the adopted material and its shape (or thickness), and this approach also need a fingerprint scanner which can acquire the fingerprint at high rate. Although the error rates of the proposed method are higher than other previous methods, it only needs one image to calculate the spectral energy, and no other preconditions are needed.

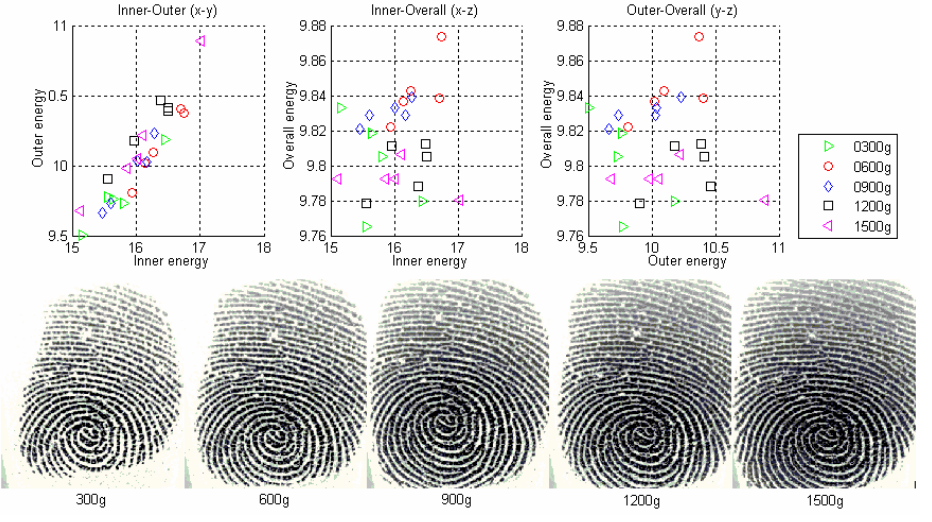


Fig. 8. Influences of the pressures (Silicone images scanned by Sensor A)

As mentioned in section 3.1, in order to study the influence of pressure, this database was collected in five pressure levels, and figure 8 shows the influences. The top row shows the energy distributions of 25 samples of one finger. The bottom row shows five images for each pressure levels. The figure reveals that the pressure has insignificant influence on the proposed energy measures.

4 Conclusions and Future Work

This paper introduces a fake finger detection approach based on band-selective Fourier spectrum. It is hard to make fake finger same as live finger exactly, and the difference of the ridge-valley texture produces different energy between ring pattern of fake and live. After transforming the fingerprint image into spatial frequency domain, the inner energy, the outer energy and the overall energy from the spectral image are in turn calculated and considered as a 3D feature vector. Subsequently, a 3D elliptic classification boundary is established for each live finger. The liveness detection is done by estimating the Euclidean distance from the elliptic classification

center to the presented samples. If the distance of presented sample is large than the boundary, then it is a fake finger, otherwise it is a live finger. Though the error rates are a little higher than the other algorithms, this method only needs to capture one fingerprint image in a short time and users need do nothing more than presenting the fingertip to the sensor. And this method can be applied to practical applications.

Future work will be mainly dedicated to the analysis of fingerprint image texture, with tests on more data and more sensors.

Acknowledgment. This work was supported by the Korea Science and Engineering Foundation (KOSEF) through Biometrics Engineering Research Center (BERC).

References

1. Uludag, U., Jain, A.K.: Attacks on Biometric Systems: A Case Study in Fingerprints. In: Proc. SPIE-EI 2004, January 18-22, 2004, San Jose, CA, pp. 622–633 (2004)
2. Matsumoto, T., Matsumoto, H., Yamada, K., Hoshino, S.: Impact of Artificial Gummy Fingers on Fingerprint Systems. In: Proceedings of SPIE, Optical Security and Counterfeit Deterrence Techniques IV, vol. 4677 (2002)
3. Drahanský, M., Nötzel, R., Funk, W.: Liveness Detection based on Fine Movements of the Fingertip Surface. In: 2006 IEEE Information Assurance Workshop, June 21-23, 2006, pp. 42–47 (2006)
4. van der Putte, T., Keuning, J.: Biometrical fingerprint recognition: don't get your fingers burned. In: Proceedings of IFIP TC8/WG8.8 Fourth Working Conference on Smart Card Research and Advanced Applications, pp. 289–303. Kluwer Academic Publishers, Dordrecht (2000)
5. Baldisserra, D., Franco, A., Maio, D., Maltoni, D.: Fake Fingerprint Detection by Odor Analysis. In: ICBA 2006. Proceedings International Conference on Biometric Authentication, Hong Kong (January 2006)
6. Derakhshani, R., Schuckers, S.A.C., Hornak, L.A., O'Gorman, L.: Determination of vitality from a non-invasive biomedical measurement for use in fingerprint scanners. Pattern Recognition 36, 383–396
7. Parthasaradhi, S.T.V., Derakhshani, R., Hornak, L.A., Schuckers, S.A.C.: Time-Series Detection of Perspiration as a Liveness Test in fingerprint Devices. IEEE Trans. on Systems, Man, and Cybernetics - Part C 35(3) (August 2005)
8. Tan, B., Schuckers, S.: Liveness Detection using an Intensity Based Approach in Fingerprint Scanners. In: Proceedings of Biometrics Symposium, Arlington, VA (September 2005)
9. Tan, B., Schuckers, S.: Liveness Detection for Fingerprint Scanners Based on the Statistics of Wavelet Signal Processing. In: Computer Vision and Pattern Recognition Workshop, 17-22 June 2006, p. 26 (2006)
10. Abhyankar, A., Schuckers, S.: Empirical Mode Decomposition Liveness Check in Fingerprint Time Series Captures. In: Computer Vision and Pattern Recognition Workshop, 17-22 June 2006, p. 28 (2006)
11. Antonelli, A., Cappelli, R., Maio, D., Maltoni, D.: Fake Finger Detection by Skin Distortion Analysis. IEEE Transactions on Information Forensics and Security 1(3), 360–373 (2006)
12. Chen, Y., Jain, A., Dass, S.: Fingerprint Deformation for Spoof Detection. In: Proceedings of Biometrics Symposium, Arlington, VA, pp. 27–28 (September 2005)

13. Moon, Y.S., Chen, J.S., Chan, K.C., So, K., Woo, K.C.: Wavelet based fingerprint liveness detection. Electronics Letters 41(20), 1112–1113 (2005)
14. Zhan, X., Yin, Y., Sun, Z., Chen, Y.: A method based on continuous spectrum analysis and artificial immune network optimization algorithm for fingerprint image ridge distance estimation. In: Computer and Information Technology, 2005. The Fifth International Conference on, 21-23 September 2005, pp. 728–733 (2005)
15. Daugman, J.: How Iris Recognition Works. In: Proceedings of 2002 International Conference on Image Processing, vol. 1 (2002)
16. Kovács-Vajna, Z.M., Rovatti, R., Frazzoni, M.: Fingerprint ridge distance computation methodologies. Pattern Recognition 33(1), 69–80 (2000)

Improving Upon the TET Mode of Operation

Palash Sarkar

Applied Statistics Unit
Indian Statistical Institute
203, B.T. Road, Kolkata
India 700108
palash@isical.ac.in

Abstract. Naor and Reingold had proposed the construction of a strong pseudo-random permutation (SPRP) by using a layer of ECB encryption between two layers of invertible block-wise universal hash functions. At Crypto 2007, Halevi presented constructions of invertible block-wise universal hash functions and a new mode of operation (called TET) based on them. In this paper, we present a new mode of operation called HEH using the Naor-Reingold approach. This is built using a new construction of invertible block-wise universal hash function. The new construction improves over Halevi's construction by removing restrictions on the hashing key. This in turn, leads to HEH improving over TET by allowing more efficient encryption and decryption of variable length messages as well as supporting better key agility. For the important application of disk encryption, we present a variant called HEHfp which has better key agility than TET.

Keywords: modes of operations, tweakable encryption, strong pseudo-random permutation, disk encryption.

1 Introduction

A block cipher is a fundamental primitive in cryptography. The formal model of a block cipher is that of a pseudo-random permutation (PRP) or a strong PRP (SPRP) [8]. By itself, a block cipher can encrypt fixed length strings. A mode of operation extends the domain of a block cipher to longer and variable length strings.

A variable input length SPRP can be considered to be a mode of operation of a block cipher. The notion of tweakable block cipher was introduced by Liskov-Rivest-Wagner [7]. This notion was extended to variable input length tweakable SPRP by Halevi-Rogaway [5]. Earlier, a method for constructing SPRPs was given by Naor-Reingold. An important application of tweakable SPRP is that of disk encryption as has been pointed out in [5]. Currently, the literature contains several constructions of tweakable SPRPs. These constructions can be classified into three main groups.

The first type of construction consists of using a layer of electronic codebook (ECB) encryption between two invertible block-wise universal hashing layers.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 180–192, 2007.

This method was introduced by Naor-Reingold [11,10]. Recently, there has been an interest in this type of constructions and the proposals PEP [2] and TET [4] are of this type. The second type consists of using a counter mode of encryption between two layers of universal hash function computation. This idea was introduced in XCB [9] and later constructions are HCTR [13] and HCH [1]. The third type of construction is to use a mixing layer between two layers of encryption. This technique was introduced by Halevi-Rogaway [5] and the constructions CMC [5], EME [6] and EME* [3] are of this type.

TET is a very recent construction which follows the Naor-Reingold hash-ECB-hash approach. For fixed length messages, TET has good performance. It, however, has two drawbacks. First, it is not suited for variable length messages and second, the key agility of TET is not good, in the sense that a lot of computation needs to be done for every key change. The drawback of TET for variable length messages has been mentioned in [4, Page 423] itself: "Hence, TET is not very appealing as a variable-input-length mode". Key agility was not considered in [4] at all.

Our Contributions: The purpose of the current work is to propose a new construction of tweakable SPRP following the Naor-Reingold approach. We call this HEH (for Hash-ECB-Hash). The new construction removes the above mentioned drawbacks of TET, while retaining its performance. HEH is well suited for the special application of disk encryption.

As mentioned earlier, the Naor-Reingold approach is to use a layer of ECB encryption between two layers of invertible block-wise universal hash functions. TET uses this approach. The main novelty in TET is to design an invertible block-wise universal hash function. It is shown that both the hash function and its inverse are block-wise universal. The universal hash function defined in [4] has a drawback which in turn leads to the earlier mentioned drawbacks of TET. When m blocks are to be hashed, the hashing key τ has to satisfy the condition that $\sigma = 1 + \tau + \cdots + \tau^m \neq 0$ and τ/σ has to be computed for computing the inverse of the hash function.

In this paper, we design a new invertible block-wise universal hash function. But, in our case, the inverse is not block-wise invertible. Importantly, this does not matter in the design of SPRP. It is sufficient to place the ECB layer between the hash function and its inverse. In fact, this has already been done by Naor-Reingold [10].

An important advantage of the new hash function over the one in [4] is that there are no restrictions on the hashing key. It is this feature which ultimately allows HEH to improve over TET.

2 Invertible Block-Wise Universal Hash Function

Let $\mathbb{F}$ be a finite field. Additions and multiplications are done over this field.

The notion of block-wise universal hash function is defined for a keyed family of functions. Fix a positive integer m. Let $\mathcal{F} : \mathcal{K} \times \mathbb{F}^m \rightarrow \mathbb{F}^m$ be a keyed

family of functions where $\mathcal{K}$ is the key space. The family $\mathcal{F}$ is said to be ϵ-block-wise universal if for every $\mathbf{x}, \mathbf{x}' \in \mathbb{F}^m$, $1 \leq i, i' \leq m$ with $(\mathbf{x}, i) \neq (\mathbf{x}', i')$; $\Pr_K[Y_i = Y'_{i'}] \leq \epsilon$, where $(Y_1, \ldots, Y_m) = \mathcal{F}_K(\mathbf{x})$ and $(Y'_1, \ldots, Y'_1) = \mathcal{F}_K(\mathbf{x}')$. We are interested in invertible block-wise universal hash functions, i.e., $\mathcal{F}_K()$ should be invertible for each $K \in \mathcal{K}$.

2.1 Block-Wise Polynomial Evaluation [4]

In this section, we describe the constructions given in [4]. For $\tau \in \mathbb{F}$ and a positive integer m, let A_τ be the following matrix.

$$A_\tau = \begin{bmatrix} \tau & \tau^2 & & \tau^m \\ \tau & \tau^2 & & \tau^m \\ & & \ddots & \\ \tau & \tau^2 & & \tau^m \end{bmatrix}$$

Define $M_\tau = A_\tau + I$ and let $\sigma = 1 + \tau + \tau^2 + \cdots + \tau^m$. The matrix M_τ is invertible if and only if $\sigma \neq 0$ and then $M_\tau^{-1} = I - (A_\tau/\sigma)$. Let $\mathbf{x} = (X_1, \ldots, X_m)$. The map $\mathbf{x} \mapsto M_\tau \mathbf{x}^T$ is the following:

$$(X_1, \ldots, X_m) \mapsto (X_1 + R, \ldots, X_m + R) \tag{1}$$

where $R = \sum_{i=1}^{m} X_i \tau^i$.

Let $\beta \in \mathbb{F}$ and α be a fixed primitive element of $\mathbb{F}$. Further, define $\mathbf{b} = (\beta, \alpha\beta, \ldots, \alpha^{m-1}\beta)$. Two functions (and their inverses) from $\mathbb{F}^m$ to $\mathbb{F}^m$ are defined in the following manner.

$$\begin{array}{lll} \mathrm{BPE}_{\tau,\beta}(\mathbf{x}) = M_\tau \mathbf{x}^T + \mathbf{b} & \text{and} & \mathrm{BPE}^{-1}_{\tau,\beta}(\mathbf{x}) = M_\tau^{-1}(\mathbf{x} - \mathbf{b})^T \\ \widetilde{\mathrm{BPE}}_{\tau,\beta}(\mathbf{x}) = M_\tau (\mathbf{x} - \mathbf{b})^T & \text{and} & \widetilde{\mathrm{BPE}}^{-1}_{\tau,\beta}(\mathbf{x}) = M_\tau^{-1} \mathbf{x}^T + \mathbf{b} \end{array} \tag{2}$$

The matrix-vector product $M_\tau \mathbf{x}$ and $M_\tau^{-1}\mathbf{x}$ can be computed as efficiently as polynomial evaluation. Using a suitable representation for $\mathbb{F}$ ensure that it is very efficient to multiply by the primitive element α. Thus, the cost of evaluating BPE is essentially the cost of polynomial evaluation. Using Horner's rule, computing BPE requires m multiplications over $\mathbb{F}$. If τ is fixed, then a pre-computed table can be used to speed up the polynomial computation [12].

For a fixed value of m and random and independent choices of τ (subject to the fact that $\sigma \neq 0$) and β from $\mathbb{F}$, it has been shown in [4], that the functions defined by (2) are block-wise universal.

Note: It has been remarked that the same proof also holds when m is allowed to vary. We note that this is incorrect. To see this consider the two distinct messages $\mathbf{x}_1 = (0, 0)$ and $\mathbf{x}_2 = (0, 0, 0)$. Then $\mathrm{BPE}_{\tau,\beta}(\mathbf{x}_1) = (\beta, \alpha\beta)$ and $\mathrm{BPE}_{\tau,\beta}(\mathbf{x}_2) = (\beta, \alpha\beta, \alpha^2\beta)$. The first two components of $\mathrm{BPE}_{\tau,\beta}(\mathbf{x}_1)$ and $\mathrm{BPE}_{\tau,\beta}(\mathbf{x}_2)$ are equal which violates the block-wise universality condition.

Drawbacks: The key τ has to be chosen such that $\sigma = \sum_{i=0}^{m} \tau^i$ is non-zero. This means that τ cannot be an arbitrary element of $\mathbb{F}$. The probability that $\sigma = 0$ for a randomly chosen τ is small, so this may not be a major problem in practice. On the other hand, it has been suggested in [4], that one can choose τ to be a random primitive element of $GF(2^n)$. This approach has practical difficulties. Often, the entity providing the new value of the key will not have access to the internal implementation of the algorithm. Without such access, in particular, without knowing the primitive polynomial realizing the field $GF(2^n)$, it is not possible to determine whether a particular element is a primitive element of the concrete realization of the field. Further, determination of primitive element requires substantial computation and the knowledge of the prime factors of 2^n-1. Thus, the idea of using τ to be a random primitive element of $GF(2^n)$ is quite impractical in practice.

More importantly, if the hash function needs to be evaluated for different values of m, then computing the inverse of BPE and $\widetilde{\text{BPE}}$ will require the computation of σ and τ/σ. This requires $(m-1)$ multiplications and one inverse over $\mathbb{F}$.

2.2 A New Construction

Fix a positive integer m and a primitive element α of $\mathbb{F}$. Let τ and β be independent and random elements of $\mathbb{F}$. Define $\mathbf{e} = (\alpha\beta, \alpha^2\beta, \ldots, \alpha^{m-1}\beta, \beta)$. As mentioned earlier, for a proper choice of the primitive element α, multiplication by α is very fast and the cost is negligible compared to a general multiplication over $\mathbb{F}$. We define the map $\Psi_{\tau,\beta} : \mathbb{F}^m \to \mathbb{F}^m$ in the following manner.

$$\Psi_{\tau,\beta}(X_1, \ldots, X_m) = (X_1 + Y, \ldots, X_{m-1} + Y, Y) + \mathbf{e} \tag{3}$$

where $Y = \sum_{i=1}^{m} X_i \tau^{m-i}$.

Invertibility is easily seen as follows. Let $(Y_1, \ldots, Y_m) = \Psi_{\tau,\beta}(X_1, \ldots, X_m)$. Set $(U_1, \ldots, U_m) = (Y_1, \ldots, Y_m) - \mathbf{e}$. Then $X_i = U_i - U_m$ for $1 \le i \le m-1$ and $X_m = U_m - \tau(\sum_{i=1}^{m-1} X_i \tau^{m-1-i})$. Using Horner's rule, computing either $\Psi_{\tau,\beta}$ or its inverse $\Psi_{\tau,\beta}^{-1}$ requires $(m-1)$ multiplications.

Examples: For $m = 4$, we provide the outputs of $\text{BPE}_{\tau,\beta}$ and $\Psi_{\tau,\beta}$ to illustrate the difference between the two functions.

$$\begin{aligned}
\text{BPE}_{\tau,\beta}(X_1, X_2, X_3, X_4) = (&X_1 + X_1\tau + X_2\tau^2 + X_3\tau^3 + X_4\tau^4 + \beta,\\
&X_2 + X_1\tau + X_2\tau^2 + X_3\tau^3 + X_4\tau^4 + \alpha\beta,\\
&X_3 + X_1\tau + X_2\tau^2 + X_3\tau^3 + X_4\tau^4 + \alpha^2\beta,\\
&X_4 + X_1\tau + X_2\tau^2 + X_3\tau^3 + X_4\tau^4 + \alpha^3\beta)\\
\Psi_{\tau,\beta}(X_1, X_2, X_3, X_4) = (&X_1\tau^3 + X_2\tau^2 + X_3\tau + X_4 + X_1 + \alpha\beta,\\
&X_1\tau^3 + X_2\tau^2 + X_3\tau + X_4 + X_2 + \alpha^2\beta,\\
&X_1\tau^3 + X_2\tau^2 + X_3\tau + X_4 + X_3 + \alpha^3\beta,\\
&X_1\tau^3 + X_2\tau^2 + X_3\tau + X_4 + \beta)
\end{aligned}$$

The order of evaluation in BPE and Ψ are in reverse order. This difference is, however, not significant. One can define BPE to evaluate in the reverse order (i.e., $X_1\tau^4 + X_2\tau^3 + X_3\tau^2 + X_4\tau$) as has indeed been done in [4] while defining TET. The significant differences between the two maps are in the degrees of the polynomials (in τ) and the treatment of the last component.

The following result establishes the block-wise universality of Ψ and its proof is based on the standard argument over roots of polynomials.

Theorem 1. [illegible] m [illegible] τ [illegible] β [illegible] $\mathbb{F}$ [illegible] $(Y_1, \ldots, Y_m) = \Psi_{\tau,\beta}(X_1, \ldots, X_m)$ $(Y'_1, \ldots, Y'_m) = \Psi_{\tau,\beta}(X'_1, \ldots, X'_m)$ $1 \leq i, i' \leq m$ [illegible] $((X_1, \ldots, X_m), i) \neq ((X'_1, \ldots, X'_m), i')$

[illegible] $i \neq i'$ [illegible] $\Pr_{\tau,\beta}[Y_i = Y'_{i'}] = \frac{1}{|\mathbb{F}|}$
[illegible] $i = i'$ [illegible] $\Pr_{\tau,\beta}[Y_i = Y'_i] \leq \frac{m-1}{|\mathbb{F}|}$

[illegible] [illegible] $\Psi_{\tau,\beta}$ [illegible] $\left(\frac{m-1}{|\mathbb{F}|}\right)$ [illegible]

Proof: The two cases are proved separately.

[illegible] $i \neq i'$ Without loss of generality, we assume that $1 \leq i < i' \leq m$. First suppose $i' < m$. From (3), $Y_i - Y'_{i'} = (\alpha^i - \alpha^{i'})\beta + R$, where R is a quantity which depends on τ and not on β. Since α is a primitive element of $\mathbb{F}$, we have $\alpha^i \neq \alpha^{i'}$ (for $m \leq \mathbb{F}^* - 1$). The event $Y_i = Y'_{i'}$ translates into the event $\beta = (\alpha^{i'} - \alpha^i)^{-1}R$. Since β is a random element of $\mathbb{F}$ and is independent of the right hand side, the probability that this happens is $1/|\mathbb{F}|$. If $i' = m$, then $Y_i - Y'_{i'} = (\alpha^i - 1)\beta + R$ and a similar argument holds.

[illegible] $i = i'$ In this case, we necessarily have $(X_1, \ldots, X_m) \neq (X'_1, \ldots, X'_m)$. First suppose $i < m$. Then $Y_i - Y'_i = (X_i + Y) - (X'_i + Y')$ where $Y = \sum_{i=1}^{m} X_i\tau^{m-i}$ and $Y' = \sum_{i=1}^{m} X'_i\tau^{m-i}$. We have

$$X_i + Y = X_1\tau^{m-1} + X_2\tau^{m-2} + \cdots + X_{m-1}\tau + (X_m + X_i).$$

Consider the map $(X_1, \ldots, X_m) \mapsto (V_1, \ldots, V_m) = (X_1, \ldots, X_{m-1}, X_m + X_i)$. This map is a bijection and so

$$(X_1, \ldots, X_m) \neq (X'_1, \ldots, X'_m) \text{ implies } (V_1, \ldots, V_m) \neq (V'_1, \ldots, V'_m).$$

As a consequence $(V_1 - V'_1, \ldots, V_m - V'_m) \neq (0, \ldots, 0)$. Now,

$$\begin{aligned} Y_i - Y'_i &= (X_i + Y) - (X'_i + Y') \\ &= (V_1 - V'_1)\tau^{m-1} + \cdots + (V_{m-1} - V'_{m-1})\tau + (V_m - V'_m). \end{aligned}$$

The last expression is a non-zero polynomial in τ and is zero if and only if τ is a root of this polynomial. Since τ is a random element of $\mathbb{F}$ and a polynomial of degree $(m-1)$ has at most $(m-1)$ distinct roots, we have $\Pr[Y_i = Y'_i] \leq (m-1)/|\mathbb{F}|$.

If $i = m$, then a similar argument holds. □

Variable m: Theorem 1 holds for a fixed value of m. If m is allowed to vary, then the result does not hold. This can be seen as in the case of BPE by considering the two distinct messages $(0,0,0)$ and $(0,0)$.

$\Psi^{-1}_{\tau,\beta}$ is not block-wise universal. This is seen by considering

1. $\Psi^{-1}_{\tau,\beta}(\alpha\beta, \alpha^2\beta, \alpha^3\beta, \beta) = (0,0,0,0)$ and
2. $\Psi^{-1}_{\tau,\beta}(A\tau^3+A+\alpha\beta, A\tau^3+\alpha^2\beta, A\tau^3+\alpha^3\beta, A\tau^3+\beta) = (A,0,0,0)$ for a non-zero A.

The last three components are equal, violating the block-wise universal property. Thus, we have an example of a function which is invertible and block-wise universal but its inverse is not block-wise universal. We have the following extension of Theorem 1.

Theorem 2. $m > 1$ τ, β $\mathbb{F}$ $\mathbf{x}_i = (X_{i,1}, \ldots, X_{i,m})$ $1 \le i \le q$ $\mathbb{F}^m$ $1 \le i \le q$ $\mathbf{y}_i = \Psi_{\tau,\beta}(\mathbf{x}_i)$ $\mathbf{y}_i = (Y_{i,1}, \ldots, Y_{i,m})$ τ, β $(i_1, j_1) \neq (i_2, j_2)$ Y_{i_1,j_1} Y_{i_2,j_2} $\frac{2(qm)^2}{|\mathbb{F}|}$

Proof: Suppose $(i_1, j_1) \neq (i_2, j_2)$. There are two cases. First suppose that $j_1 = j_2 = j$. There are $m \times \binom{q}{2}$ such pairs. For any such pair, the probability that $Y_{i_1,j} = Y_{i_2,j}$ is at most $(m-1)/2^n$ (from Theorem 1(2)). Thus, the total probability of collisions among such pairs is at most $m \times \binom{q}{2} \times (m-1)/|\mathbb{F}| \le (m^2q^2)/|\mathbb{F}|$. On the other hand, if $j_1 \neq j_2$, then the probability of $Y_{i_1,j_1} = Y_{i_2,j_2}$ is at most $1/|\mathbb{F}|$ (from Theorem 1(1)). There are $\binom{qm}{2} - m \times \binom{q}{2} \le (qm)^2$ such pairs and the probability of a collision among such pairs is at most $(qm)^2/|\mathbb{F}|$. Thus, the total probability of a collision among the Ys is at most $2(qm)^2/|\mathbb{F}|$. □

XOR Block-Wise Universal. The family $\mathcal{F}$ is said to be "XOR block-wise universal" if for every fixed δ and every $\mathbf{x}, \mathbf{x}' \in \mathbb{F}^m$, $1 \le i, i' \le m$ with $(\mathbf{x}, i) \neq (\mathbf{x}', i')$; $\Pr_K[Y_i - Y'_{i'} = \delta] \le \epsilon$, where $(Y_1, \ldots, Y_m) = \mathcal{F}_K(\mathbf{x})$ and $(Y'_1, \ldots, Y'_1) = \mathcal{F}_K(\mathbf{x}')$. The constructions in [4] are proved to be XOR block-wise universal, which the new construction is not. On the other hand, it is easy to modify the new construction to yield a XOR block-wise universal hash function.

As before, fix a positive integer m and a primitive element α of $\mathbb{F}$. Let τ and β be independent and random elements of $\mathbb{F}$. We define the map $\Psi^{\mathrm{XOR}}_{\tau,\beta} : \mathbb{F}^m \to \mathbb{F}^m$ in the following manner.

$$\Psi^{\mathrm{XOR}}_{\tau,\beta}(X_1, \ldots, X_m) = (X_1 + Y, \ldots, X_{m-1} + Y, Y) + \mathbf{e} \tag{4}$$

where $Y = \sum_{i=1}^{m} X_i \tau^{m+1-i}$.

The only difference is in the definition of Y, which is now a polynomial of degree m as compared to a polynomial of degree $(m-1)$ in the case of Ψ. The proof that Ψ^{XOR} is XOR block-wise universal is similar to the proof of Theorem 1. Also, it is not difficult to see that Ψ^{XOR} is invertible if $\tau \neq 0$. Computing the

inverse of Ψ^{XOR} requires the inverse of τ. As a XOR block-wise universal hash function, Ψ^{XOR} improves upon the constructions in [4] in the following way. Ψ^{XOR} requires that $\tau \neq 0$ and the inverse of τ, while the constructions in [4] require to compute $\sigma = 1 + \tau + \cdots + \tau^m$ and the inverse of σ (if $\sigma \neq 0$). This difference can be significant in practice. In this paper, we do not work with Ψ^{XOR}; instead we work only with Ψ, which does not require the inverse of τ for computing ψ^{-1}.

3 The HEH Construction

For the description of the tweakable SPRP, we will consider the finite field $\mathbb{F}$ to be $GF(2^n)$ and use the operator $\oplus$ to denote addition over this field. The field $GF(2^n)$ is realized using a primitive polynomial $\rho(x)$ of degree n and the primitive element α is simply taken to be x. Multiplication by x modulo $\rho(x)$ can be done very efficiently. As is standard, the elements of $GF(2^n)$ can be interchangeably considered to be either as polynomials over $GF(2)$ of degree at most $n-1$ or as n-bit strings. For $0 \leq i \leq 2^n - 1$, by $\mathsf{bin}_n(i)$ we denote the n-bit binary representation of i.

The basic structure of the HEH construction is shown in Figure 1. Pseudo-codes are given in Figure 2. In this construction, there are one block cipher key K and three hashing keys τ, β_1 and β_2. By suitably defining the hashing keys, it is possible to obtain several variants of the basic construction. This is shown in Figure 3.

In HEH, the number of blocks m can vary; n-bit tweaks (associated data) are supported and only a single block cipher key is used. The hashing key τ depends on the tweak T. Hence, it is not possible to speed up multiplication by τ using a pre-computed table. If such pre-computation is desired, then it is easy to modify HEH, to obtain a variant supporting pre-computation. Instead of setting $\tau = \gamma$, simply choose τ to be a random element of $GF(2^n)$. We call this variant HEHp.

An important special application of tweakable SPRP is that of disk encryption. In this application, the number of blocks m is fixed and the tweak is the sector address. Consequently, it is sufficient to take the tweak to be an n-bit string. Since m is fixed, it is possible to eliminate one block cipher call while deriving the hashing keys. Also, the hashing key τ is chosen to be a random element of $GF(2^n)$ so that pre-computation can be utilized. We call this variant HEHfp. In this variant, the hashing key is τ and the block cipher key is K.

3.1 Other Issues

We briefly consider several other issues in the design of a possible tweakable SPRP.

Arbitrary length messages. HEH and its variants defined so far can only handle messages which are multiples of block length n. Actually, the inner layer of ECB mode is not particularly suited for handling partial blocks. This is better tackled using a counter mode of encryption, as for example in HCH [1]. On the

Algorithm $\mathbf{E}_{K,\tau,\beta_1,\beta_2}(P_1,\ldots,P_m)$
1. $(PP_1,\ldots,PP_m) = \Psi_{\tau,\beta_1}(P_1,\ldots,P_m)$;
2. $(CC_1,\ldots,CC_m) = \mathrm{ECB}_K(PP_1,\ldots,PP_m)$;
3. $(C_1,\ldots,C_m) = \Psi^{-1}_{\tau,\beta_2}(CC_1,\ldots,CC_m)$.

Algorithm $\mathbf{D}_{K,\tau,\beta_1,\beta_2}(C_1,\ldots,C_m)$
1. $(CC_1,\ldots,CC_m) = \Psi_{\tau,\beta_2}(C_1,\ldots,C_m)$;
2. $(PP_1,\ldots,PP_m) = \mathrm{ECB}^{-1}_K(CC_1,\ldots,CC_m)$;
3. $(P_1,\ldots,P_m) = \Psi^{-1}_{\tau,\beta_1}(PP_1,\ldots,PP_m)$.

Fig. 1. Encryption and decryption using HEH. The block cipher key is K; and the hash key is (τ,β_1,β_2). See Figure 3 for details of how the hashing keys are derived in HEH and its variants. $\mathrm{ECB}_K(X_1,\ldots,X_m)$ returns $(E_K(X_1),\ldots,E_K(X_m))$ and $\mathrm{ECB}^{-1}_K(Y_1,\ldots,Y_m)$ returns $(E^{-1}_K(Y_1),\ldots,E^{-1}_K(Y_m))$.

Algorithm $\mathbf{E}_{K,\tau,\beta_1,\beta_2}(P_1,\ldots,P_m)$	**Algorithm** $\mathbf{D}_{K,\tau,\beta_1,\beta_2}(C_1,\ldots,C_m)$
1. $U = P_1$;	1. $U = C_1$;
2. for $i = 2$ to m do $U = U\tau \oplus P_i$;	2. for $i = 2$ to m do $U = U\tau \oplus C_i$;
3. $Q = \beta_1$;	3. $Q = \beta_2$;
4. for $i = 1$ to $m-1$ do	4. for $i = 1$ to $m-1$ do
5. $\quad Q = xQ$; $PP_i = P_i \oplus U \oplus Q$;	5. $\quad Q = xQ$; $CC_i = C_i \oplus U \oplus Q$;
6. $\quad CC_i = E_K(PP_i)$;	6. $\quad PP_i = E^{-1}_K(CC_i)$;
7. end do;	7. end do;
8. $PP_m = U \oplus \beta_1$; $CC_m = E_K(PP_m)$;	8. $CC_m = U \oplus \beta_1$; $PP_m = E_K(CC_m)$;
9. $V = CC_m \oplus \beta_2$; $Q = \alpha\beta_2$;	9. $V = PP_m \oplus \beta_1$; $Q = \alpha\beta_1$;
10. $C_1 = CC_1 \oplus Q \oplus V$; $W = C_1$;	10. $P_1 = PP_1 \oplus Q \oplus V$; $W = P_1$;
11. for $i = 2$ to $m-1$ do	11. for $i = 2$ to $m-1$ do
12. $\quad Q = \alpha Q$; $C_i = CC_i \oplus Q \oplus V$;	12. $\quad Q = \alpha Q$; $P_i = PP_i \oplus Q \oplus V$;
13. $\quad W = W\tau \oplus C_i$;	13. $\quad W = W\tau \oplus P_i$;
14. end do;	14. end do;
15. $C_m = V \oplus W\tau$;	15. $P_m = V \oplus W\tau$;
end.	**end.**

Fig. 2. Detailed pseudo-code of encryption and decryption using HEH

HEH	HEHp	HEHfp
1. $\gamma = E_K(T)$; 2. $\beta_1 = E_K(\gamma \oplus \mathsf{bin}_n(m))$; 3. $\beta_2 = x\beta_1$; 4. $\tau = \gamma$.	1. $\gamma = E_K(T)$; 2. $\beta_1 = E_K(\gamma \oplus \mathsf{bin}_n(m))$; 3. $\beta_2 = x\beta_1$; 4. choose τ randomly from $GF(2^n)$.	1. $\beta_1 = E_K(T)$; 2. $\beta_2 = x\beta_1$; 3. choose τ randomly from $GF(2^n)$.

Fig. 3. HEH and its variants obtained by suitably defining the hashing keys τ, β_1 and β_2. T is an n-bit tweak and m is the number of blocks. K is a randomly chosen block cipher key and in HEHp and HEHfp, τ is a randomly chosen n-bit string.

other hand, it is possible to define variants of HEH which can handle messages of any length greater than or equal to n. Further details will be provided in the expanded version of the paper.

Arbitrary length tweaks. For applications which require arbitrary length tweaks, one can use a pseudo-random function (PRF) with a separate and independent key to produce an n-bit tweak which can then be used in HEH. The hashing key τ is chosen independently of the PRF key, which allows for the hashing and the processing of tweak to proceed in parallel. This approach has been used in TET and if desired, a similar approach can also be used with HEH.

4 Discussion and Comparison

HEH uses the hash-ECB-hash approach introduced by Naor-Reingold. An earlier construction using the same approach is TET. (We do not consider PEP, since it is slower than TET.) The difference between the two is mainly in the definition of the universal hash functions. BPE (used in TET) is invertible XOR block-wise universal while Ψ (used in HEH) is only invertible block-wise universal. (The variant Ψ^{XOR} is XOR block-wise universal.) The efficiencies of computing both BPE and Ψ as well as their inverses are similar. One difference between the two functions is that the inverse of BPE is XOR block-wise universal while the inverse of Ψ is not block-wise universal (neither is the inverse of Ψ^{XOR} XOR block-wise universal). Importantly however, it is possible to construct a tweakable SPRP without requiring the inverse to be block-wise universal.

From the viewpoint of performance the significant difference between BPE and Ψ is the following. BPE [illegible] τ [illegible] $\sigma = \sum_{i=0}^{m} \tau^i$ [illegible] τ [illegible] Ψ Below we discuss the implications of this difference and how as a result HEH improves upon TET.

Variable Length Messages. For TET, since m varies, for each message, σ and σ^{-1}, has to be computed. Computing σ requires $(m-1)$ multiplications over $GF(2^n)$. This together with the requirement of computing an inverse makes TET unsuitable for variable length messages. Such computation is not required at all for HEH.

Key Agility. If m is fixed, then for TET, σ and σ^{-1} can be pre-computed. The hash key τ is obtained in TET by applying a PRF having key K_1 to a fixed input. Now, suppose the key K_1 is changed. Then τ also changes and hence σ and σ^{-1} also need to be re-computed. Thus, for TET, key change is computationally expensive. For HEH, key change does not require any field operation.

Computing a Field Inverse. TET requires computation of a field inverse, either in the online phase, or during a key change. For hardware only implementation, this means that an inversion circuit has to be implemented. HEH does not require any field inversion.

4.1 Comparison to Other Construction

There are three efficient constructions using the hash-counter-hash approach – XCB, HCTR and HCH. Among these constructions, HCH and XCB have quadratic security bounds and HCTR has a cubic bound. HEH has efficiency similar to these constructions; a quadratic security bound; and also has similar key agility. Thus, HEH shows that it is possible to use the hash-ECB-hash approach to obtain a construction which is as good as hash-counter-hash approach.

The comparison to the encrypt-mix-encrypt approach is based on the relative efficiency of a block cipher call and a $GF(2^n)$ multiplication. If one block cipher call takes more time than two multiplications, then the hash-encrypt-hash approach is faster, otherwise the encrypt-mix-encrypt approach is faster. A related issue is whether a pre-computed table can be used to speed up multiplication by the hashing key τ. For the single key HEH, this is not possible. On the other hand, for the simple variant HEHp, this is possible and it is also possible for HEHfp.

A detailed comparison among the different tweakable SPRPs is given in [1]. In Table 1, we provide a comparison among some of the more important features of four previous constructions with HEH. The four constructions are CMC, EME* (of the encrypt-mix-encrypt type), HCH (of the hash-Ctr-hash type) and TET (of the hash-ECB-hash type). These four constructions are representative of the currently best known constructions of each type.

For variable number of blocks (Table 1), TET requires some extra block cipher invocations and multiplications; basically the term multiplied by $\imath$. The value of $\imath$ itself depends on the number of blocks and the PRF key. Also, TET requires a $GF(2^n)$ inversion. These computations are required to obtain a hash key τ, $\sigma = 1 + \tau + \cdots + \tau^m \neq 0$ and to invert σ (if $\sigma \neq 0$). Thus, the restriction on the hashing key directly reflects on the performance of TET. We note that the

Table 1. Comparison of tweakable SPRPs where the number of blocks m can vary and n-bit tweaks are used. For TET, $\imath$ is a value (at least 1) which depends on m (and K). [BC]: block cipher invocation; [M]: $GF(2^n)$ multiplication; [I]: $GF(2^n)$ inversion; [BCK]: block cipher key; [AK]: auxilliary n-bit key material.

Mode	type	comp. cost	keys	passes	enc. layers
CMC [5]	enc-mix-enc	$(2m+1)$[BC]	1[BCK]	2	2
EME* [3]	enc-mix-enc	$(2m+\frac{m}{n}+1)$[BC]	1[BCK]+2[AK]	2	2
HCH [1]	hash-Ctr-hash	$(m+3)$[BC] $+2(m-1)$[M]	1[BCK]	2	1
TET [4]	hash-ECB-hash	$\imath((m-1)$[M]+2[BC]) $+(m+2)$[BC] $+2m$[M]+1[I]	2[BCK]	3	1
HEH	hash-ECB-hash	$(m+2)$[BC] $+2(m-1)$[M]	1[BCK]	3	1

Table 2. Comparison of different tweakable SPRPs where the number of blocks m is fixed and n-bit tweaks are used. [BC]: number of block cipher invocations; [M]: $GF(2^n)$ multiplication; [BCK]: block cipher key; [AK]: auxiliary n-bit string (including polynomial hash keys).

Mode	CMC [5]	EME* [3]	HCHfp [1]	TET [4]	HEHfp
[BC]	$2m+1$	$2m+1+m/n$	$m+2$	$m+1$	$m+1$
[M]	–	–	$2(m-1)$	$2m$	$2(m-1)$
[BCK]	1	1	1	2	1
[AK]	–	2	1	3	1

Table 3. Efficiency of key change. For TET, the value of $\imath$ depends only on K (since the number of blocks m is fixed) and is at least 1. [I]: $GF(2^n)$ inversion.

Mode	CMC	EME*	HCHfp	TET	HEHfp
comp. cost	–	–	–	$\imath((m-1)$[M]$+2$[BC]$)+1$[BC]$+1$[I]	–
key sch.	1	1	1	2	1
mult. tab.	–	–	1	1	1

value of $\imath$ is at least 1, which makes the performance of TET clearly inferior to that of HCH or HEH.

When m is fixed (Table 2), as in the case for disk encryption, the value of τ and σ^{-1} can be pre-computed in TET. This makes the actual encryption and decryption in TET quite efficient. However, the problem of generating τ and σ^{-1} is still present whenever the key needs to be changed. That is, even though m is fixed, whenever the PRF key changes, the value of τ and σ^{-1} has to be computed afresh. This adversely affects the key agility of TET. HEHfp is the variant of HEH which is suited for fixed values of m and can utilize pre-computation. The efficiency of HEHfp for encryption and decryption is similar (actually slightly better) to that of TET and HCHfp. The improvement over TET is to offer better key agility. Currently, if one wishes to use the Naor-Reingold approach, then HEHfp is the construction of choice to implement disk encryption schemes.

5 Security Statement for HEH

The discussion and notation used in this section is based on earlier work [5]. Due to lack of space, we will be brief. Detailed proof will be given in the expanded version of the paper.

The query complexity σ_n of an adversary is defined to be the total number of n-bit blocks it provides in all its encryption and decryption queries. This includes the plaintext and ciphertext blocks as well as the n-bit tweak. By $\mathbf{Adv}(\sigma_n)$ (with suitable sub and super-scripts) we denote the maximum advantage of any adversary with query complexity σ_n. The notation $\mathbf{Adv}(\sigma_n, t)$ denotes the maximum advantage of any adversary with query complexity σ_n and running time t.

The notation HEH[E] denotes a tweakable enciphering scheme, where the block cipher E is used in the manner specified by HEH. HEH[Perm(n)] denotes a tweakable enciphering scheme obtained by plugging in a random permutation from Perm(n) into the structure of HEH. For an adversary attacking HEH[Perm(n)], we do not put any bound on the running time of the adversary, though we still put a bound on the query complexity σ_n. We consider an adversary's advantage in distinguishing a tweakable enciphering scheme $\mathbf{E}$ from an oracle which simply returns random bit strings. This advantage is defined in the following manner.

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{HEH[Perm}(n)]}(A) = \left|\Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\mathbf{E}_\pi, \mathbf{D}_\pi} \Rightarrow 1\right] - \Pr\left[A^{\$(.,.),\$(.,.)} \Rightarrow 1\right]\right|$$

where $\$(., M)$ returns random bits of length $|M|$.

Theorem 3. [illegible] n [illegible] σ_n [illegible] σ_n [illegible] n [illegible]

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{HEH[Perm}(n)]}(\sigma_n) \leq \frac{4\sigma_n^2}{2^n} \tag{5}$$

[illegible] HEH[illegible] HEHp [illegible] HEHfp

6 Conclusion

In this paper, we have proposed a new tweakable SPRP called HEH following the hash-ECB-hash approach introduced by Naor-Reingold [11]. This is done by designing a new invertible block-wise universal hash function. The new hash function improves over the invertible block-wise universal hash function defined in [4] by removing restrictions on the hashing key. This in turn results in HEH being able to remove the drawbacks of the tweakable SPRP called TET which was also proposed in [4] An important special application of tweakable SPRP is disk encryption. For this application, we suggest a variant called HEHfp. Currently, HEHfp is the best candidate for implementing a disk encryption scheme using the Naor-Reingold approach.

References

1. Chakraborty, D., Sarkar, P.: HCH: A new tweakable enciphering scheme using the hash-encrypt-hash approach. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 287–302. Springer, Heidelberg (2006), full version available at, http://eprint.iacr.org/2007/028
2. Chakraborty, D., Sarkar, P.: A new mode of encryption providing a tweakable strong pseudo-random permutation. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 293–309. Springer, Heidelberg (2006)

3. Halevi, S.: EME*: Extending EME to handle arbitrary-length messages with associated data. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 315–327. Springer, Heidelberg (2004)
4. Halevi, S.: Invertible universal hashing and the tet encryption mode. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 412–429. Springer, Heidelberg (2007)
5. Halevi, S., Rogaway, P.: A tweakable enciphering mode. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 482–499. Springer, Heidelberg (2003)
6. Halevi, S., Rogaway, P.: A parallelizable enciphering mode. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 292–304. Springer, Heidelberg (2004)
7. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 31–46. Springer, Heidelberg (2002)
8. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM J. Comput. 17(2), 373–386 (1988)
9. McGrew, D.A., Fluhrer, S.R.: The extended codebook (XCB) mode of operation. Cryptology ePrint Archive, Report 2004/278 (2004), http://eprint.iacr.org/
10. Naor, M., Reingold, O.: A pseudo-random encryption mode. Manuscript, available from www.wisdom.weizmann.ac.il/~naor
11. Naor, M., Reingold, O.: On the construction of pseudorandom permutations: Luby-Rackoff revisited. J. Cryptology 12(1), 29–66 (1999)
12. Shoup, V.: On fast and provably secure message authentication based on universal hashing. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 313–328. Springer, Heidelberg (1996)
13. Wang, P., Feng, D., Wu, W.: HCTR: A variable-input-length enciphering mode. In: Feng, D., Lin, D., Yung, M. (eds.) CISC 2005. LNCS, vol. 3822, pp. 175–188. Springer, Heidelberg (2005)

New Local Collisions for the SHA-2 Hash Family

Somitra Kumar Sanadhya and Palash Sarkar

Applied Statistics Unit,
Indian Statistical Institute,
203, B.T. Road, Kolkata,
India 700108
{somitra_r,palash}@isical.ac.in

Abstract. The starting point for collision attacks on practical hash functions is a local collision. In this paper, we make a systematic study of local collisions for the SHA-2 family. The possible linear approximations of the constituent Boolean functions are considered and certain impossible conditions for such approximations are identified. Based on appropriate approximations, we describe a general method for finding local collisions. Applying this method, we obtain several local collisions and compute the probabilities of the various differential paths. Previously, only one local collision due to Gilbert-Handschuh was known. We point out two impossible conditions in the GH local collision and provide an example of an impossible differential path for linearized SHA-2 using this local collision. Sixteen new local collisions are obtained none of which have any impossible conditions. The probabilities of these local collisions are a little less than the GH local collision. On the other hand, the absence of impossible conditions may make them more suitable for (reduced round) collision search attacks on the SHA-2 family.

1 Introduction

Study of collision search attacks on practical hash functions is a topic of intense interest in recent times. Some spectacular successes have been reported for concrete and widely used proposals such as MD5 [13] and SHA-1 [12,1]. Other less popular hash functions such as RIPEMD and HAVAL have also been successfully attacked.

Currently, the two commonly used hash functions are MD5 and SHA-1. In view of the attacks on these functions, there seems to be a tendency to move to the more complicated SHA-2 family. As a result, these hash functions will receive much more attention from the research community.

Usually, the first step in a collision search attack is to find a local collision. This is a collision for a fixed number of steps of the round function. Details about the message expansion are ignored. Further, all nonlinear components of the hash design are approximated by some suitable linear functions. Once a local collision is obtained, one attempts to find a collision for the full hash function by taking into account the message expansion and the nonlinear behaviour of the hash design. For example, Wang et al.'s attack on the SHA-1 hash function

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 193–205, 2007.

[12] uses the local collision obtained by Chabaud and Joux [2]. For details about this approach one may refer to [2].

Known Results for the SHA-2 Family: Gilbert and Handschuh (GH) [4] were the first to study local collisions in the SHA-2 family. They reported a 9-round local collision and estimated the probability of the differential path to be 2^{-66}. The message expansion of the SHA-256 was studied by Mendel et al [7], who reported reduced round (near) collisions. The work [7] remarked that the probability of the GH local collision is 2^{-39}. This value of the probability was also obtained in [5] when modular differences are considered. An earlier work [6] studied a very simplified variant of SHA-256. The encryption mode of SHA-256 is analyzed in [15] and is not relevant to collision search attacks.

Our Contributions: All previous works have considered only the GH local collision. In this paper, we revisit the problem of obtaining a local collision for the SHA-2 family of functions. Local collisions are found by forming linear approximations of the Boolean functions f_{IF} and f_{MAJ} involved in round function of SHA-2. We make a systematic analysis of the linear approximations of the two Boolean functions. The differential analysis shows that certain kinds of linear approximations give rise to impossible conditions. Given any linear approximations for f_{IF} and f_{MAJ}, we describe a step-by-step method for finding a 9-step local collision for the corresponding linearized round function. This method has been applied on all feasible linear approximations. Two of the cases have been described in details. We also show how to extend the presented local collisions into 17 and 18 step collisions for SHA-2.

The GH local collision was obtained by approximating both f_{IF} and f_{MAJ} by 0. We show that both the approximations have one impossible condition each and this can lead to an impossible differential path. Note that the differential path is impossible for the linearized version of the hash function. It is not impossible for the actual design. An example is provided of an 12-step impossible differential path for the GH local collision. This path is impossible due to the impossible condition on the approximation of f_{IF} by 0. Mendel et al [7] circumvent the impossible conditions of the Boolean functions by using carry propagation in addition. However, this puts extra conditions on message bits reducing the freedom and thereby reducing the probability of the attack. We hope that the new local collisions will help carry out longer round attacks on SHA-2 family.

There are four linear approximations each of f_{MAJ} and f_{IF} which do not have any impossible conditions. These give rise to a total of 16 different linear approximations without any impossible conditions. We develop all these approximations to obtain 16 new local collisions without any impossible conditions. Also, we describe four other local collisions which have one impossible condition for f_{MAJ} and none for f_{IF}.

Probabilities of all the local collisions are computed. For the GH local collision we obtain a probability of 2^{-42}. The previous estimate by GH was 2^{-66}. The probabilities of the other local collisions are found to be between 2^{-45} to 2^{-54}. In [5], the probability of the GH local collision was computed to be 2^{-39} using

modular differences and in [7] it was remarked (without providing details) that this can be higher than 2^{-39} even with XOR differences. We note that whatever be the method for computing probability estimates, the relative probabilities of the different local collisions will probably remain the same. Further, even though the probabilities of the new local collisions are lower than the GH local collision, the absence of impossible conditions may offset this disadvantage when they are used to find actual (reduced round) collisions for the SHA-2 family.

2 SHA-2 Family of Hash Functions

The round function of the SHA-2 family operates on eight 32-bit registers denoted by (a, b, c, d, e, f, g, h) and updates them in each step. In this article, we analyze only the round function. For the complete description of the SHA-2 family see [10]. The 8 registers are updated according to the following equations (all additions are modulo 2^{32}):

$$\left.\begin{aligned} a_i &= \Sigma_0(a_{i-1}) + f_{MAJ}(a_{i-1}, b_{i-1}, c_{i-1}) + \Sigma_1(e_{i-1}) \\ &\quad + f_{IF}(e_{i-1}, f_{i-1}, g_{i-1}) + h_{i-1} + K_i + W_i \\ b_i &= a_{i-1} \\ c_i &= b_{i-1} \\ d_i &= c_{i-1} \\ e_i &= d_{i-1} + \Sigma_1(e_{i-1}) + f_{IF}(e_{i-1}, f_{i-1}, g_{i-1}) \\ &\quad + h_{i-1} + K_i + W_i \\ f_i &= e_{i-1} \\ g_i &= f_{i-1} \\ h_i &= g_{i-1} \end{aligned}\right\} \quad (1)$$

The f_{IF} and the f_{MAJ} are three variable Boolean functions defined as:

$$\begin{aligned} f_{IF}(x, y, z) &= (x \wedge y) \oplus (\neg x \wedge z) \\ f_{MAJ}(x, y, z) &= (x \wedge y) \oplus (y \wedge z) \oplus (z \wedge x) \end{aligned}$$

The functions Σ_0 and Σ_1 are defined differently for SHA-256 and SHA-512. For SHA-256, these functions are defined as:

$$\begin{aligned} \Sigma_0(x) &= ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \\ \Sigma_1(x) &= ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \end{aligned}$$

And for SHA-512, they are defined as:

$$\begin{aligned} \Sigma_0(x) &= ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x) \\ \Sigma_1(x) &= ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x) \end{aligned}$$

Our analysis treats Σ_0 and Σ_1 as operators, hence the discussion that follows holds for both SHA-256 and SHA-512 (In the following, we will interchangeably use $\Sigma_i(X)$ and $\Sigma_i X$). Since SHA-384 is just a truncated version of SHA-512, we refer to all the three hash functions as SHA-2 family.

3 Differential Properties of Boolean Functions

Let $f(x)$ be a Boolean function on n variables. By Δx we denote the XOR difference in the input of f, i.e., $\Delta x = x \oplus x'$ for two n-bit strings x and x'. The value of Δx can be any 2^n bit string. Given Δx, define $\Delta f = f(x \oplus \Delta x) \oplus f(x)$. The value of Δf is either 0 or 1 but is not uniquely determined by the value of Δx. Assuming that x is uniformly distributed over $\{0,1\}^n$, the value of Δf is 0 or 1 with certain probabilities.

There are two Boolean functions used in SHA-2, namely the f_{IF} and the f_{MAJ}, which are 3-input bit-wise 'If' and the 'Majority' functions respectively. The three inputs to the functions can have XOR differences of 0 or 1. Depending on their positions, the Boolean functions propagate the differences or absorb them. The differential properties are shown in Table 1. The first 3 columns in this table are the input differences to the Boolean functions, whose output differences are listed in next 2 columns. An entry of 0 (resp. 1) in a Boolean function column means that Δf is 0 (resp. 1) with probability 1. An entry $(0,1)$ denotes that Δf is 0 with probability half. We will use this table to compute the probabilities of the differential paths that we show later. Note that the differential properties of Boolean function f_{IF} and f_{MAJ} are also considered in [8] but our presentation is different.

Impossible Conditions: Suppose we approximate $f(x)$ by a linear function $l(x)$. Note that Δx fixes the value of Δl with probability one. Now suppose that for some Δx, the value of Δf is also determined with probability one and that $\Delta f \neq \Delta l$ for this value of Δx. Then the particular value of Δx for which this occurs is said to be an **impossible condition** for the approximation of f by l. The complete list of impossible conditions which arise when f_{IF} and f_{MAJ} are approximated by different linear functions is given in Table 2.

The probability that $f_{IF}(a,b,c) = 0$ is $1/2$ and the probability that $f_{IF}(a,b,c) = c$ (or b) is $3/4$. This suggests that approximating f_{IF} by c (or b) should be better than approximating f_{IF} by 0. From Table 1, the probability that

Table 1. Differential properties of f_{IF} and f_{MAJ}. A single 1 (0) in the last 2 columns means that this value holds with probability 1. The entry (0,1) implies that both the values are possible with probability $\frac{1}{2}$ each.

Δa	Δb	Δc	$\Delta f_{IF}(a,b,c)$	$\Delta f_{MAJ}(a,b,c)$
0	0	0	0	0
0	0	1	(0,1)	(0,1)
0	1	0	(0,1)	(0,1)
0	1	1	1	(0,1)
1	0	0	(0,1)	(0,1)
1	0	1	(0,1)	(0,1)
1	1	0	(0,1)	(0,1)
1	1	1	(0,1)	1

Table 2. Impossible conditions for the different linear approximations of $f_{IF}(a,b,c)$ and $f_{MAJ}(a,b,c)$. The entries in the table provide the values of $(\Delta a, \Delta b, \Delta c)$ which are the impossible conditions for the corresponding approximation.

	0	a	b	c	$a \oplus b$	$a \oplus c$	$b \oplus c$	$a \oplus b \oplus c$
f_{IF}	$(0,1,1)$	$(0,1,1)$	none	none	none	none	$(0,1,1)$	$(0,1,1)$
f_{MAJ}	$(1,1,1)$	none	none	none	$(1,1,1)$	$(1,1,1)$	$(1,1,1)$	none

$\Delta f_{IF} = \Delta c$ is 5/8, where as the probability for $\Delta f_{IF} = 0$ is still 1/2. Thus, on an average, the approximation of f_{IF} by c should be better than that by 0 even for a differential analysis.

Remark: It has been mentioned in [7, Page 130, Lines 4–5] that several approximations for f_{IF} and f_{MAJ} are possible and all of these hold with probability 0.5. Table 1 and the discussion above shows that this is not the case. Specifically, the approximation c (or b) is better than the approximation 0 for $f_{IF}(a,b,c)$.

4 Linear Approximation of SHA-2 Round Function

Local collisions are usually found for the linearized version of the hash function concerned [2,11]. Once it is found for the simple case, the probability for this local collision to hold for the actual hash function is computed. We proceed along similar lines and approximate all additions in SHA-2 by bit-wise XOR. There are many possibilities for the linear approximations of f_{IF} and f_{MAJ} functions. A general form of expressing these approximations is the following

$$\left.\begin{aligned} f_{MAJ}(a,b,c) &= x_1 a \oplus x_2 b \oplus x_3 c \\ f_{IF}(e,f,g) &= y_1 e \oplus y_2 f \oplus y_3 g \end{aligned}\right\} \tag{2}$$

where (x_1, x_2, x_3) and (y_1, y_2, y_3) are 3-bit strings. Thus, the linear approximations are completely specified by these two strings. Let $\Delta\text{reg}_i = (\Delta a_i, \Delta b_i, \Delta c_i, \Delta d_i, \Delta e_i, \Delta f_i, \Delta g_i, \Delta h_i)$. Then the linearized version of the SHA-2 round function can be expressed by an equation of the form

$$(\Delta\text{reg}_i)^t = A(\Delta\text{reg}_{i-1}, \Delta W_i)^t \tag{3}$$

where $()^t$ denotes transpose and A is a suitable matrix which is constructed depending upon the particular linear approximation being used. The form of A in terms of (x_1, x_2, x_3) and (y_1, y_2, y_3) is given by (4).

$$A = \begin{bmatrix} p_1 & x_2 & x_3 & 0 & p_2 & y_2 & y_3 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & p_2 & y_2 & y_3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\text{where } p_1 = (x_1 \oplus \Sigma_0) \text{ and } p_2 = (y_1 \oplus \Sigma_1). \tag{4}$$

The simplest is to approximate both f_{MAJ} and f_{IF} by the constant function 0 (i.e., $(x_1, x_2, x_3) = 0$ and $(y_1, y_2, y_3) = 0$) as has been done by GH [4]. These approximations, however, give rise to two impossible conditions as has been discussed in Section 3. There are four linear approximations of f_{IF} which do not have any impossible conditions. In Table 3 we consider the situation where f_{MAJ} is approximated by zero and f_{IF} is approximated by zero and the four other linear functions which do not have impossible conditions. From Table 2, we find that there are 16 possible combinations of linear approximations of f_{MAJ} and f_{IF} which do not have any impossible conditions. These are listed in Table 4.

5 Technique for Finding Local Collisions

We describe the method for finding a local collision spanning k steps. For the local collision to exist, the difference of registers at the start and at the end must be zero. Besides, the first and the last message differences must not be zero, to make it exactly a k-step collision.

The basic idea is to iterate the linear system in the forward direction; equate the register values to 0 after k steps and then solve the resulting equations. The forward iteration is done in the following manner.

1. $\Delta\text{reg}_0 = (0, 0, 0, 0, 0, 0, 0, 0)$.
2. For $i = 1$ to k do
3. $\quad (\Delta\text{reg}_i)^t = A(\Delta\text{reg}_{i-1}, \Delta W_i)^t$;
4. end do.

The procedure provides Δreg_k in terms of $\Delta W_1, \ldots, \Delta W_k$. We now have to set $\Delta\text{reg}_k = 0$ and solve for $\Delta W_1, \ldots, \Delta W_k$. Since the expressions for Δreg_k are quite complicated, there does not seem to be any general method for solving these equations. On the other hand, the equations do have a pattern, which we have exploited to obtain solutions. We explain our method for $k = 9$ for Case B of Table 3. Similar methods have been applied to the other two cases. All our computations have been carried out using the symbolic computation package Mathematica [14].

Table 3. Linear approximations for $f_{MAJ}(a, b, c)$, $f_{IF}(e, f, g)$ and the corresponding $(x_1, x_2, x_3, y_1, y_2, y_3)$. Case A has been considered by Gilbert-Handschuh. It has one impossible condition each for both f_{MAJ} and f_{IF}. Cases B to E have one impossible condition for f_{MAJ} and none for f_{IF}.

Case	$f_{MAJ}(a, b, c)$	$f_{If}(e, f, g)$	(x_1, x_2, x_3)	(y_1, y_2, y_3)
A	0	0	(0,0,0)	(0,0,0)
B	0	g_{i-1}	(0,0,0)	(0,0,1)
C	0	f_{i-1}	(0,0,0)	(0,1,0)
D	0	$e_{i-1} \oplus g_{i-1}$	(0,0,0)	(1,0,1)
E	0	$e_{i-1} \oplus f_{i-1}$	(0,0,0)	(1,1,0)

5.1 Case B of Table 3

The actual values of Δreg_9 in this case is given in Section A in [9]. Below we show how to solve for $\Delta W_1, \ldots, \Delta W_9$ under the condition $\Delta\text{reg}_9 = 0$.

Step 1: The expression for Δh_9 is of the form

$$\Delta h_9 = \Delta W_6 \oplus \Sigma_1(\Delta W_5) \oplus \Sigma_1^2(\Delta W_4) \oplus \Delta W_3 \oplus \Sigma_1^3(\Delta W_3) \oplus \Sigma_1^4(\Delta W_2) \oplus \Sigma_0(\Delta W_1) \oplus \Sigma_1^2(\Delta W_1) \oplus \Sigma_1^5(\Delta W_1).$$

Setting $\Delta h_9 = 0$ provides

$$\Delta W_6 = \Sigma_1(\Delta W_5) \oplus \Sigma_1^2(\Delta W_4) \oplus \Delta W_3 \oplus \Sigma_1^3(\Delta W_3) \oplus \Sigma_1^4(\Delta W_2) \oplus \Sigma_0(\Delta W_1) \oplus \Sigma_1^2(\Delta W_1) \oplus \Sigma_1^5(\Delta W_1). \quad (5)$$

Step 2: Eliminating ΔW_6 from $(\Delta a_9, \ldots, \Delta g_9)$ using (5), we obtain

$$\Delta g_9 = \Delta W_7 \oplus \Delta W_4 \oplus \Sigma_1(\Delta W_3) \oplus \Sigma_0(\Delta W_2) \oplus \Sigma_1^2(\Delta W_2) \oplus \Delta W_1 \oplus \Sigma_0^2(\Delta W_1) \oplus \Sigma_0(\Sigma_1(\Delta W_1)) \oplus \Sigma_1^3(\Delta W_1).$$

Setting $\Delta g_9 = 0$ provides

$$\Delta W_7 = W_4 \oplus \Sigma_1(\Delta W_3) \oplus \Sigma_0(\Delta W_2) \oplus \Sigma_1^2(\Delta W_2) \oplus \Delta W_1 \oplus \Sigma_0^2(\Delta W_1) \oplus \Sigma_0(\Sigma_1(\Delta W_1)) \oplus \Sigma_1^3(\Delta W_1). \quad (6)$$

Table 4. Linear approximations for $f_{MAJ}(a,b,c)$ and $f_{IF}(e,f,g)$ and corresponding $(x_1,x_2,x_3,y_1,y_2,y_3)$. These approximations do not have any impossible conditions for either f_{MAJ} or f_{IF}.

Case	$f_{MAJ}(a,b,c)$	$f_{IF}(e,f,g)$	(x_1,x_2,x_3)	(y_1,y_2,y_3)
1	a	f	(1,0,0)	(0,1,0)
2	a	g	(1,0,0)	(0,0,1)
3	a	$e \oplus f$	(1,0,0)	(1,1,0)
4	a	$e \oplus g$	(1,0,0)	(1,0,1)
5	b	f	(0,1,0)	(0,1,0)
6	b	g	(0,1,0)	(0,0,1)
7	b	$e \oplus f$	(0,1,0)	(1,1,0)
8	b	$e \oplus g$	(0,1,0)	(1,0,1)
9	c	f	(0,0,1)	(0,1,0)
10	c	g	(0,0,1)	(0,0,1)
11	c	$e \oplus f$	(0,0,1)	(1,1,0)
12	c	$e \oplus g$	(0,0,1)	(1,0,1)
13	$a \oplus b \oplus c$	f	(1,1,1)	(0,1,0)
14	$a \oplus b \oplus c$	g	(1,1,1)	(0,0,1)
15	$a \oplus b \oplus c$	$e \oplus f$	(1,1,1)	(1,1,0)
16	$a \oplus b \oplus c$	$e \oplus g$	(1,1,1)	(1,0,1)

Step 3: Eliminating ΔW_7 from $(\Delta a_9, \ldots, \Delta f_9)$ using (6), we obtain

$$\begin{aligned}\Delta f_9 = {} & \Delta W_8 \oplus \Delta W_5 \oplus \Sigma_1(\Delta W_4) \oplus \Sigma_0(\Delta W_3) \oplus \Sigma_1^2(\Delta W_3) \oplus \Delta W_2 \oplus \Sigma_0^2(\Delta W_2) \\ & \oplus \Sigma_0(\Sigma_1(\Delta W_2)) \oplus \Sigma_1^3(\Delta W_2) \oplus \Sigma_0^3(\Delta W_1) \oplus \Sigma_0^2(\Sigma_1(\Delta W_1)) \\ & \oplus \Sigma_0(\Sigma_1^2(\Delta W_1)) \oplus \Sigma_1^4(\Delta W_1).\end{aligned}$$

Setting $\Delta f_9 = 0$ provides

$$\begin{aligned}\Delta W_8 = {} & \Delta W_5 \oplus \Sigma_1(\Delta W_4) \oplus \Sigma_0(\Delta W_3) \oplus \Sigma_1^2(\Delta W_3) \oplus W_2 \oplus \Sigma_0^2(\Delta W_2) \\ & \oplus \Sigma_0(\Sigma_1(\Delta W_2)) \oplus \Sigma_1^3(\Delta W_2) \oplus \Sigma_0^3(\Delta W_1) \oplus \Sigma_0^2(\Sigma_1(\Delta W_1)) \\ & \oplus \Sigma_0(\Sigma_1^2(\Delta W_1)) \oplus \Sigma_1^4(\Delta W_1). \qquad (7)\end{aligned}$$

Step 4: Eliminating ΔW_8 in $(\Delta a_9, \ldots, \Delta e_9)$ using (7) we obtain

$$\begin{aligned}\Delta e_9 = {} & \Delta W_9 \oplus \Sigma_0(\Delta W_4) \oplus \Sigma_0^2(\Delta W_3) \oplus \Sigma_0(\Sigma_1(\Delta W_3)) \oplus \Sigma_0^3(\Delta W_2) \oplus \Sigma_0^2(\Sigma_1(\Delta W_2)) \\ & \oplus \Sigma_0(\Sigma_1^2(\Delta W_2)) \oplus \Delta W_9 \oplus \Sigma_0(\Delta W_1) \oplus \Sigma_0^4(\Delta W_1) \oplus \Sigma_0^3(\Sigma_1(\Delta W_1)) \\ & \oplus \Sigma_0^2(\Sigma_1^2(\Delta W_1)) \oplus \Sigma_0(\Sigma_1^3(\Delta W_1)).\end{aligned}$$

Setting $\Delta e_9 = 0$ provides

$$\begin{aligned}\Delta W_9 = {} & \Sigma_0(\Delta W_4) \oplus \Sigma_0^2(\Delta W_3) \oplus \Sigma_0(\Sigma_1(\Delta W_3)) \oplus \Sigma_0^3(\Delta W_2) \oplus \Sigma_0^2(\Sigma_1(\Delta W_2)) \\ & \oplus \Sigma_0(\Sigma_1^2(\Delta W_2)) \oplus \Delta W_1 \oplus \Sigma_0(\Delta W_1) \oplus \Sigma_0^4(\Delta W_1) \oplus \Sigma_0^3(\Sigma_1(\Delta W_1)) \\ & \oplus \Sigma_0^2(\Sigma_1^2(\Delta W_1)) \oplus \Sigma_0(\Sigma_1^3(\Delta W_1)). \qquad (8)\end{aligned}$$

Step 5: Eliminating ΔW_9 in $(\Delta a_9, \ldots, \Delta d_9)$ using (7) we obtain

$$\begin{aligned}\Delta d_9 = {} & \Sigma_0(\Delta W_5) \oplus \Sigma_0^2(\Delta W_4) \oplus \Sigma_0(\Sigma_1(\Delta W_4)) \oplus \Sigma_0^3(\Delta W_3) \oplus \Sigma_0^2(\Sigma_1(\Delta W_3)) \oplus \\ & \Sigma_0(\Sigma_1^2(\Delta W_3)) \oplus \Delta W_2 \oplus \Sigma_0(\Delta W_2) \oplus \Sigma_0^4(\Delta W_2) \oplus \Sigma_0^3(\Sigma_1(\Delta W_2)) \oplus \\ & \Sigma_0^2(\Sigma_1^2(\Delta W_2)) \oplus \Sigma_0(\Sigma_1^3(\Delta W_2)) \oplus \Sigma_0^2(\Delta W_1) \oplus \Sigma_0^5(\Delta W_1) \oplus \Sigma_1(\Delta W_1) \oplus \\ & \Sigma_0^4(\Sigma_1(\Delta W_1)) \oplus \Sigma_0^3(\Sigma_1^2(\Delta W_1)) \oplus \Sigma_0^2(\Sigma_1^3(\Delta W_1)) \oplus \Sigma_0(\Sigma_1^4(\Delta W_1)).\end{aligned}$$

Now the situation is different from the previous 4 steps. In the expression for Δd_9 we do not have any ΔW_i whose "coefficient" is 1. Only ΔW_5 occurs once with a "coefficient" of Σ_0. We solve for ΔW_5 in the following manner. Set

$$\Delta W_2 = \Sigma_0(x) \oplus \Sigma_1(\Delta W_1) \qquad (9)$$

where x is a variable to be determined later. With this substitution, we have $\Delta d_9 = \Sigma_0(\Delta W_5 \oplus X)$, for some expression X which we provide shortly. Now setting $\Delta d_9 = 0$, provides one solution to be $\Delta W_5 = X$, where the value of X is given by the right side of the following expression.

$$\begin{aligned}\Delta W_5 = {} & (1 \oplus \Sigma_0 \oplus \Sigma_0^4\Sigma_0^3\Sigma_1 \oplus \Sigma_0^2\Sigma_1^2 \oplus \Sigma_0\Sigma_1^3)(x) \oplus \Sigma_0(\Delta W_4) \oplus \Sigma_1(\Delta W_4) \oplus \\ & \Sigma_0^2(\Delta W_3) \oplus \Sigma_0(\Sigma_1(\Delta W_3)) \oplus \Sigma_1^2(\Delta W_3) \oplus \Sigma_0(\Delta W_1) \oplus \Sigma_0^4(\Delta W_1) \\ & \oplus \Sigma_1(\Delta W_1). \qquad (10)\end{aligned}$$

Step 6: Eliminating ΔW_5 in $(\Delta a_9, \Delta b_9, \Delta c_9)$ using (10) we obtain

$$\Delta c_9 = \Sigma_0^2(x) \oplus \Sigma_0(\Sigma_1(x)) \oplus \Delta W_3 \oplus \Sigma_0^2(\Delta W_1).$$

Setting $\Delta c_9 = 0$ provides

$$\Delta W_3 = \Sigma_0^2(x) \oplus \Sigma_0(\Sigma_1(x)) \oplus \Sigma_0^2(\Delta W_1). \tag{11}$$

Step 7: Eliminating ΔW_3 in $(\Delta a_9, \Delta b_9)$ using (11) we obtain

$$\Delta b_9 = \Sigma_0^2(\Sigma_1(x)) \oplus \Delta W_4 \oplus \Delta W_1 \oplus \Sigma_0^2(\Sigma_1(\Delta W_1)).$$

Setting $\Delta b_9 = 0$, provides

$$\Delta W_4 = \Sigma_0^2(\Sigma_1(x)) \oplus \Delta W_1 \oplus \Sigma_0^2(\Sigma_1(\Delta W_1)). \tag{12}$$

Step 8: Eliminating ΔW_4 from Δa_9 using (12), we obtain

$$\Delta a_9 = x \oplus \Delta W_1.$$

Setting $\Delta a_9 = 0$ provides

$$\Delta W_1 = x. \tag{13}$$

Equations (5), (6), (7), (8), (9), (10), (11), (12), and (13) form a solution to the problem of finding a local collision for the linearized round function. In this form, the equations are not easy to handle. But, if we start the process of back substitution, i.e., use $\Delta W_1 = x$ in (12) and then use the values of ΔW_1 and ΔW_4 in (11) and so on, then the solution is substantially simplified and we finally obtain

$$(\Delta W_1, \ldots, \Delta W_9) = \\ (x, \Sigma_0(x) \oplus \Sigma_1(x), \Sigma_0(\Sigma_1(x)), x, \Sigma_0(x) \oplus x, \Sigma_0(x) \oplus \Sigma_1(x), 0, x, x).$$

The technique described above does not work always. There are cases when we cannot solve the equations in the manner described earlier. A slightly modified method is used for such cases. Refer to [9] for complete description.

6 Differential Path

The values of the XOR differences of the registers at each step constitute a differential path. For a local collision, the initial and final XOR differences should be zero.

Probability of Differential Path: A differential path holds with probability one for the linearized version of the round function. However, when we move to the actual round function, then it holds with lesser probability which in some cases may even be zero. If the differential path holds with probability zero for the actual round function, then we call it to be an impossible differential path.

Such impossible differential paths arise due to the impossible conditions in the approximations of the constituent functions by linear functions. Later we will show examples of such differential paths including one obtained from the Gilbert-Handschuh local collision.

We next discuss how to compute the probability for a differential path. This computation is based on the following two points.

1. If a and b differ in one bit position, then $a+c$ and $b+c$ also differ in one bit position with probability one if the differing bit is the most significant bit, else with probability half. (This was also mentioned in [4].) We also assume that if a and b differ in k different bit positions none of which is the most significant bit, then $a+c$ and $b+c$ differ in these k positions with probability $1/2^k$.
2. Table 1 is used to determine the differential probabilities for the approximations of f_{IF} and f_{MAJ}.

Since the XOR and additive differences coincide for the most significant bit, to achieve higher probability, it is advantageous to ensure that many bits in the differential path are MSBs. The initial message difference in the beginning of the local collision is chosen to be 2^{31} based on this observation. For complete details of probability calculations see [9].

7 Reduced Round Collisions for the SHA-2 Family

In this section, we show that it is possible to combine the presented local collisions for getting upto 18 step reduced round collisions for the SHA-2 family. We specify the first step in SHA-2 by Step 0.

First of all, note that all the local collisions discussed in the present work span 9 steps and the message expansion of SHA-2 does not play any role in first 16 steps. Therefore if a local collision spans from Step i to Step $(i+9)$, and if we take $W_0 = W_1 \ldots = W_{i-1} = W_{i+10} = W_{i+11} = \ldots W_{16} = 0$, we get a collision for first 16 steps of SHA-2. All the 16 local collisions described in this work can be used to generate 16 step collisions for the SHA-2 family in this manner.

The 16 step collisions described above are not very ineresting since we have completely by-passed the issue of message expansion in obtaining them. Now we tackle the first step of message expansion. Message expansion rule for W_{16} is given by :

$$W_{16} = \sigma_1(W_{14}) + W_9 + \sigma_0(W_1) + W_0 \tag{14}$$

A local collision which starts at Step 2 will end at Step 10. The differential path for such a local collision will have $\Delta W_0 = \Delta W_1 = \Delta W_{14} = 0$ (If we choose the differentials of all the message words outside the span of the local collision to be zero). The local collisions described by Cases 1, 3, 5, 7, 10, 12, 14 and 16 of Table 5 are such that ΔW_9 will be zero for them (Refer the differential paths for the local collisions in [9]). Thus message expansion yields that $\Delta W_{16} = 0$. Hence we have many 17 step collisions for SHA-2 using a single local collision.

In the same manner it can be shown that starting a single local collision described by Cases 2, 4, 5, 7, 13, 14, 15 or 16 of Table 5 at Step 3, we can get a 17-step collision for SHA-2. Similarly, starting a single local collision described by Cases 5, 7, 14 or 16 of Table 5 will yield $\Delta W_{16} = \Delta W_{17} = 0$. Thus we can get 4 different 18 step collisions for SHA-2.

Previously only one 18 step collsion for SHA-256 has been reported in the literature [7]. In the present work we have shown that it is possible to obtain many 17 and 18 step collisions for both SHA-256 and SHA-512. We have explained the procedure for generating such collisions without actually exhibiting them. We hope to exhibit these collisions in a forthcoming paper.

To go beyond 18 steps will require combining several local collisions and it is currently being investigated. For example in [7] a 19 Step 1 bit near collision for SHA-256 is reported which is obtained by using 23 GH local collisions.

8 Results

The detailed differential paths for the cases of Table 3 and Table 4 are available in [9].

Further analysis of the presented local collisions reveals that all approximations of f_{MAJ} by the same linear function have the same differential path. The weight of the differential path increases with the increase in the number of variables in the linear approximation of f_{MAJ}. A summary of various features of the different local collisions are given in Table 5.

Remark: The probability of the differential path of Case A was estimated by GH to be 2^{-66}. Our calculations show this to be 2^{-42}. In [5] this probability was computed to be 2^{-39} when using modular differences (as opposed to XOR

Table 5. Summary of the different properties of the local collisions. Wt(DP) provides the weight of the differential path; Wt(MD) provides the weight of the message difference; Pr. provides the probability of the differential path; and NIC provides the number of impossible conditions. The cases are from Table 3 and 4. Case A is the GH local collision, rest are new local collisions.

Case	A	B	C	D	E
Wt(DP)	24	24	24	24	24
Wt(MD)	24	29	29	34	34
Pr.	$\frac{1}{2^{42}}$	$\frac{1}{2^{45}}$	$\frac{1}{2^{45}}$	$\frac{1}{2^{48}}$	$\frac{1}{2^{48}}$
NIC	2	1	1	1	1

Case	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Wt(DP)	28	28	28	28	28	28	28	28	28	28	28	28	36	36	36	36
Wt(MD)	35	33	35	35	29	35	35	37	35	31	37	35	37	37	43	41
Pr.	$\frac{1}{2^{48}}$	$\frac{1}{2^{48}}$	$\frac{1}{2^{51}}$	$\frac{1}{2^{51}}$	$\frac{1}{2^{49}}$	$\frac{1}{2^{49}}$	$\frac{1}{2^{52}}$	$\frac{1}{2^{52}}$	$\frac{1}{2^{48}}$	$\frac{1}{2^{48}}$	$\frac{1}{2^{51}}$	$\frac{1}{2^{51}}$	$\frac{1}{2^{54}}$	$\frac{1}{2^{54}}$	$\frac{1}{2^{57}}$	$\frac{1}{2^{57}}$
NIC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

differences considered here). Mendel et al [7] remarked (without providing details) that the probability can be higher than 2^{-39} even when considering XOR differences. We think that the relative probabilities of the different local collisions will remain the same irrespective of which method is applied to compute the probabilities.

The GH local collision (Case A) has the highest probability. It is, however, not necessarily the best possible local collision. This is due to the fact that it has two impossible conditions and may result in an impossible differential path. We illustrate this point using the impossible condition for f_{IF}. In [9] we show a 12-step impossible differential path for the GH local collision. This is obtained by interleaving two GH local collisions with the second one starting at the fourth step of the first one. In terms of the Chabaud-Joux [2] type disturbance vector, the 12-step differential path is given by the vector 1001. At Step 6 of this local collision, we have $\Delta e_6 = 0$, $\Delta f_6 = x \oplus \Sigma_0(x)$ and $\Delta g_6 = x$. This shows that whatever be the value of x, there will be one bit position where the differential input to f_{IF} is $(0, 1, 1)$. From Table 1 we have Δf_{IF} to be 1 with probability 1, where as the approximation of f_{IF} by $l = 0$ will have $\Delta l = 0$. This shows that although the differential path is valid for the linearized version with f_{IF} approximated by $l = 0$, it fails for the actual round function.

As mentioned earlier, the issue of impossible differential paths was also observed in [7]. They developed techniques for circumventing such impossible paths in their collision search attacks on reduced round SHA-2. On the other hand, if we use a local collision such as Case 1, then there are no impossible conditions. Consequently, no circumvention techniques will be required in collision search attacks. The probability of this local collision is a little lower than the GH local collision, but this may be offset by absence of impossible conditions. Further work on this topic can settle this point.

References

1. Biham, E., Chen, R., Joux, A., Carribault, P., Lemuet, C., Jalby, W.: Collisions of SHA-0 and reduced SHA-1. In: Cramer [3], pp. 36–57
2. Chabaud, F., Joux, A.: Differential collisions in SHA-0. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 56–71. Springer, Heidelberg (1998)
3. Cramer, R.J.F. (ed.): EUROCRYPT 2005. LNCS, vol. 3494, pp. 22–26. Springer, Heidelberg (2005)
4. Gilbert, H., Handschuh, H.: Security analysis of SHA-256 and sisters. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 175–193. Springer, Heidelberg (2004)
5. Hawkes, P., Paddon, M., Rose, G.G.: On corrective patterns for the SHA-2 family. Cryptology ePrint Archive, Report 2004/207 (August 2004), http://eprint.iacr.org/2004/207
6. Matusiewicz, K., Pieprzyk, J., Pramstaller, N., Rechberger, C., Rijmen, V.: Analysis of simplified variants of SHA-256. In: Wolf, C., Lucks, S., Yau, P.-W. (eds.) WEWoRC, GI. LNI, vol. 74, pp. 123–134 (2005)
7. Mendel, F., Pramstaller, N., Rechberger, C., Rijmen, V.: Analysis of step-reduced SHA-256. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 126–143. Springer, Heidelberg (2006)

8. Rijmen, V., Oswald, E.: Update on SHA-1. In: Menezes, A.J. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 58–71. Springer, Heidelberg (2005)
9. Sanadhya, S.K., Sarkar, P.: New local collisions for the SHA-2 hash family. Cryptology ePrint Archive, Report 2007/352 (September 2007), http://eprint.iacr.org/2007/352
10. Secure Hash Standard. Federal Information Processing Standard Publication 180-2. U.S. Department of Commerce, National Institute of Standards and Technology(NIST) (2002), available at: http://csrc.nist.gov/encryption/tkhash.html
11. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: Cramer [3], pp. 1–18
12. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
13. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer [3], pp. 19–35
14. Wolfram, S.: The Mathematica Book. Wolfram Media, 5th edn. (2003), http://www.wolfram.com
15. Yoshida, H., Biryukov, A.: Analysis of a SHA-256 variant. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 245–260. Springer, Heidelberg (2006)

Multi-collision Attack on the Compression Functions of MD4 and 3-Pass HAVAL

Hongbo Yu[1] and Xiaoyun Wang[2,⋆]

[1] Tsinghua University, Beijing 100084, China
yhb@mail.sdu.edu.cn
[2] Tsinghua University and Shandong University, China
xiaoyunwang@tsinghua.edu.cn, xywang@sdu.edu.cn

Abstract. In this paper, we present a new type of multi-collision attack on the compression functions of both MD4 and 3-Pass HAVAL. Different from Joux's multi-collision attack, our method focuses on the multi-collision of the compression function. For MD4, we utilize two different feasible collision differential paths to find a 4-collision with about 2^{21} MD4 computations. For 3-Pass HAVAL, we can find a 4-collision with complexity about 2^{30} and a 8-near-collision with complexity 2^9.

Keywords: Hash function, multi-collision, multi-near-collision, differential path, sufficient condition.

1 Introduction

Recently, cryptanalysis on hash functions has become a hot topic within the cryptographic community. Most existing hash functions have succumbed to the modular differential attack announced two years ago [10,12,13,11,14,15].

For an ideal secure hash function with n-bit output, the complexity to find a pairwise collision is about $O(2^{n/2})$ computations, and to find a $k(multi)$-collision needs about $O(2^{n(k-1)/k})$ computations. Here a k-collision consists of k different messages which are compressed to the same hash value. At Crypto'04[5], using the flaw of the iterated structure of the hash functions, Joux proposed a method to construct 2^t-collisions based on the pairwise collisions. Joux showed that for any iterated hash function it is relatively easy to find a 2^t-collision and it only costs t times as much as that of finding an ordinary pairwise collision. Based on the result, Joux proved that the concatenation of several hash functions does not increase their security. In 2005[4], Nandi and Stinson extended Joux's technique to handle iterated hash functions in which each message block is used at most twice. In FSE 2006[6], Hoch and Shamir considered the general case and proved that even if allowing each iterated hash function to scan the input multiple times in an arbitrary expanded order, their concatenation is not stronger than a single function.

⋆ Supported by 973 Project(No.2007CB807902) and the National Natural Science Foundation of China(NSFC Grant No.90604036).

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 206–226, 2007.

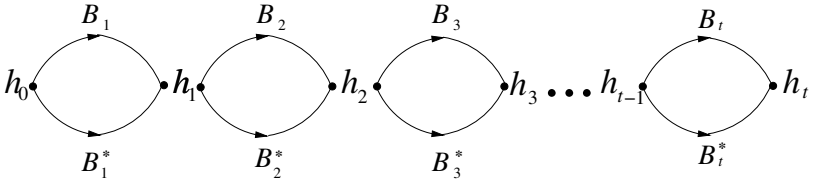


Fig. 1. Joux's 2^t-collisions construction. The 2^t messages are of the form $(b_1, b_2, ..., b_t)$ where b_i is one of the two blocks B_i and B_i^*.

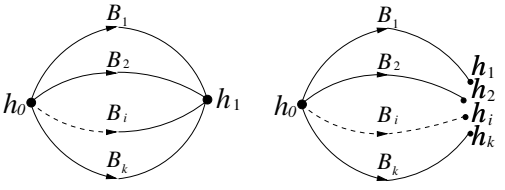


Fig. 2. Our k-collisions construction. The left denotes the multi-collision, and the right is the multi-near-collision. Each of B_i is a one-block message.

Motivated by Shamir's talk [6] and Joux's method, we provide a new attack to build the multi-collisions and multi-near-collisions for hash functions directly instead of combining multi-collisions from the pairwise collisions. The multi-near-collisions is a generalization for the pairwise near collisions presented by Biham and Chen [1]. The difference between two types of multi-collisions are shown in Fig. 1 and Fig. 2 respectively.

The paper is organized as follows. In section 2, we give a brief description of MD4 and 3-Pass HAVAL compression functions. In section 3, we recall the modular differential attack on hash functions, which is used as a fundamental tool to find multi-collisions. In section 4, we propose our new multi-collision and multi-near-collision attack. The details for finding 4-collisions on MD4 and 8-near-collisions on 3-Pass HAVAL are introduced in section 5 and 6 respectively. Finally we conclude the paper in section 7.

2 Description of MD4 and 3-Pass HAVAL

In this paper, we study multi-collisions and multi-near-collision for MD4 and 3-Pass HAVAL compression functions, so we only give a brief description of their compression functions.

2.1 MD4 Compression Function

The MD4 compression function takes a 128-bit chaining value and a 512-bit message block as the input values, processes 48 step operations and outputs a 128 -bit chaining values as hash value. For one 512-bit block $M = (m_0, m_1, ..., m_{15})$, the compression process is as follows:

1. Let (aa, bb, cc, dd) be the input 128-bit chaining variable.

$$a \longleftarrow aa,\ b \longleftarrow bb,\ c \longleftarrow cc,\ d \longleftarrow dd$$

2. Perform the following 48 steps (comprising three rounds):
   For i=0, 1, 2
   For j=0, 1, 2, 3

$$\begin{aligned} a &:= (a + \phi_i(b, c, d) + w_{i,4j} + k_i) \lll s_{i,4j} \\ d &:= (d + \phi_i(a, b, c) + w_{i,4j+1} + k_i) \lll s_{i,4j+1} \\ c &:= (c + \phi_i(d, a, b) + w_{i,4j+2} + k_i) \lll s_{i,4j+2} \\ b &:= (b + \phi_i(c, d, a) + w_{i,4j+3} + k_i) \lll s_{i,4j+3} \end{aligned}$$

   $s_{i,4j+t}$ $(t = 0,\ 1,\ 2,\ 3)$ are step-dependent constants. $w_{i,4j+t}$ is a message word and k_i is a fixed constant for every round. Symbol $\lll s$ represents the circular shift s bit positions to the left. And symbol + denotes addition modulo 2^{32}. The details of the message order and shift positions can be discovered in [7]. The round functions ϕ_0, ϕ_1 and ϕ_2 are defined as:

$$\begin{aligned} \phi_0(x, y, z) &= (x \wedge y) \vee (\neg x \wedge z) \\ \phi_1(x, y, z) &= (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) \\ \phi_2(x, y, z) &= x \oplus y \oplus z \end{aligned}$$

3. Add a, b, c and d respectively to the input chaining value.

$$\begin{aligned} aa &:= a + aa \\ bb &:= b + bb \\ cc &:= c + cc \\ dd &:= d + dd \end{aligned}$$

4. $H(M) := aa \| bb \| cc \| dd$. Here, $\|$ denotes the word concatenation.

In our paper, the search for differential paths is heavily dependant upon the properties of the round functions ϕ_1 and ϕ_2. These have been summarized in [12,17,9] and are listed again for convenience in Table 3 of our Appendix.

2.2 3-Pass HAVAL Compression Function

The 3-Pass HAVAL compression function takes a 256-bit chaining value and a 1024-bit message block $M = (m_0, m_1, ..., m_{31})$ as input values and outputs a 256-bit chaining value. The compression process is described as follows:

1. Let $(aa, bb, cc, dd, ee, ff, gg, hh)$ be the 256-bit input chaining values. Initialize chaining variables $(a,\ b,\ c,\ d,\ e,\ f,\ g,\ h)$ as $(aa, bb, cc, dd, ee, ff, gg, hh)$.
2. Perform the following 96 steps:
   For i=0,1,2
   For $j = 0$ to 31
   $p := f_{i+1}(g, f, e, d, c, b, a)$

$r := (p \gg 7) + (h \gg 11) + m_{ord(i,j)} + k_{i,j}$
$(a, b, c, d, e, f, g) = (r, a, b, c, d, e, f, g)$

The operation in each step employs a constant $k_{j,i}$(See ref.[16]). Symbol $\gg s$ represents the circular shift s bit positions to the right. The order of the messages words in each pass can be found in [16]. The round functions f_1, f_2 and f_3 are defined as follows:

$$f_1(g, f, e, d, c, b, a) = cd \oplus ag \oplus bf \oplus ce \oplus e$$
$$f_2(g, f, e, d, c, b, a) = adf \oplus bcf \oplus ef \oplus ef \oplus ac \oplus df \oplus bd \oplus bc \oplus fg \oplus g$$
$$f_3(g, f, e, d, c, b, a) = def \oplus cf \oplus be \oplus dg \oplus ad \oplus a$$

3. Add a, b, c, d, e, f, g, h respectively to the input value.
   $aa := a + aa$, $bb := b + bb$, ..., $hh := h + hh$
4. $H(M) := hh \| gg \| ff \| ee \| dd \| cc \| bb \| aa$.

Some main properties of the round function f_1 that are used to find differential paths are listed in Table 5 of Appendix.

3 Modular Differential Attack on Hash Functions

In this section, we use the MD4 as an example to outline the modular differential attack developed by Wang et al [12,13].

3.1 Selecting a Message Difference

The first step of the modular differential attack is to select an appropriate message difference which determines the success probability of the attack. The choice of the message differences determined by the purpose of the attack. For example, if we want to find a differential path for collisions, we can select the message differences as in paper [12]. These message differences not only result in a 5-step local collision in the third round, but also can decide a potential internal collision located in the first round and the previous steps of the second round. Correctly selecting the message differences is a key step in producing a possible collision path with high probability. If we want to apply the second-preimage attack or to recover the keys of the MACs based on MD4, we can select the message differences which lead to a differential path [17] with minimal sufficient conditions in total.

3.2 Searching the Differential Path

It is really difficult to find differential paths for some hash functions. We can utilize the properties of the round functions to produce some wanted bit differences and cancel the unwanted non-zero bit differences or message bit differences. Another important technique is to introduce the bit carries which can produce the above wanted bit differences. The differential paths can be found by the manual method [12,13,14,15] or by the computer-aided way[9,2].

3.3 Determining the Chaining Variable Conditions

In the process of searching for differential paths, the chaining variable conditions can be determined. A feasible differential path implies that all the chaining variable conditions deduced from the path don't contradict each other. It means that if a message M satisfies all the chaining variable conditions, M and $M+\Delta M$ (ΔM is a fixed message difference) must collide. So these conditions are called the sufficient conditions.

3.4 Message Modification

Once the collision differential path and the corresponding sufficient conditions are determined, the remaining task is how to find a message M so that M satisfies all the chaining variable conditions. Usually for a random message M, M and $M + \Delta M$ cannot cause collisions because of the large amount of sufficient conditions. If a condition is inconsistent with that of the sufficient conditions, we call it a wrong condition. According to the chaining variable conditions distribution, we can adopt different message modification techniques to force the modified message M to satisfy more sufficient conditions. For the conditions in the first round, we can implement the basic message modification technique to correct the wrong conditions. And for the conditions in the second round, the advanced message modification can be applied to correct part of wrong conditions. Usually, more conditions in the second round can be corrected by employing more complex and precise advanced message modifications. The detail of the techniques can be seen in [12,13,17].

4 New Multi-collision and Multi-near-Collision Attack

In this section, we describe our multi-collision and multi-near-collision attack on hash functions. Given two different collision differential paths for a hash function, if two sets of their sufficient conditions do not contradict each other, we will show how to utilize two collision paths to produce 4-collisions.

Provided that the first collision differential path P_1 corresponds to the message difference ΔM_1, $(M, M + \Delta M_1)$ is a collision under the sufficient conditions C_1. The second collision differential path P_2 corresponding to the message difference ΔM_2, $(M, M + \Delta M_2)$ is a collision under the sufficient conditions C_2.

If there are no contradictory conditions between C_1 and C_2, we set up $C{=}C_1 \cup C_2$. It is clear that, if M satisfies all conditions in C, $(M, M + \Delta M_1)$ is a collision that obeys the differential path P_1, $(M, M + \Delta M_2)$ is also a collision simultaneously which obeys the second collision path P_2. So, $(M, M + \Delta M_1, M + \Delta M_2)$ comprise the inputs to a 3-collision.

Furthermore, assume that the message $M + \Delta M_1$ satisfies all the conditions C_2, and then $(M + \Delta M_1, M + \Delta M_1 + \Delta M_2)$ is a collision corresponding to the path P_2. On the other hand, when the message $M + \Delta M_2$ satisfies all the conditions C_1, $(M + \Delta M_2, M + \Delta M_2 + \Delta M_1)$ is a collision corresponding to the

path P_1. This is an interesting phenomenon. For each case, $(M, M+\Delta M_1, M+\Delta M_2, M+\Delta M_1+\Delta M_2)$ consists of a 4-collision. For any two collision differential paths, even if the two sets of sufficient conditions have no contrary conditions, the message $M+\Delta M_1+\Delta M_2$ may not collide with other three messages because of a few subtle conditions on the intersecting bits between the two paths.

We generalize the above multi-collisions to k-collisions generated by t multi-collision differential paths where $k \leq 2^t$. Firstly, select t message differences $\Delta M_i, i = 0, 1, ..t-1$. For each ΔM_i, search a feasible collision differential path and deduce its corresponding condition set C_i. If there are some contradictory conditions among $C_i, i = 0, 1, ..t-1$, adjust some differential paths so that no contrary condition occurs. Finally find a message M which satisfies all conditions $C = \cup C_i$. Then the messages $(M, M+\Delta M_0, M+\Delta M_1, ...M+\Delta M_{t-1}))$ allow a t-collision. Perfectly, if the t differential paths are completely independent (i.e. there is no intersecting bits among $C_i, i = 0, 1, ..t-1$), all the messages in $\{M+\Delta M \mid \Delta M$ is a linear expression of $\Delta M_i, i = 0, 1, ..t-1\}$ collide with M, and produce a 2^t-collision.

Similarly, we can find the multi-near-collisions by the above method.

5 Finding 4-Collisions on MD4

In this section, we construct a 4-collision for the MD4 compression function which is illustrated in Fig.3.

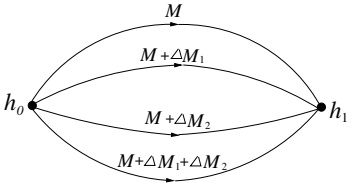


Fig. 3. The 4-collisions construction on MD4 compression function. Each of M_i is a one-block message.

We select the first message difference

$$\Delta M_1 = (0, 0, 0, 0, 2^{25}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),$$

and decide a collision differential path and its sufficient conditions (See Table 6). In fact, this collision differential path is a special instance of the 64 differential paths in [17]. The path has the least conditions among all the known MD4 collision paths.

The collision differential path in paper [12] is selected as the second differential path which is the most efficient path in finding the pairwise collisions so far. The message difference is:

$$\Delta M_2 = (0, 2^{31}, -2^{28} + 2^{31}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2^{16}, 0, 0, 0).$$

We make a simplification for the original differential path in [12] and get the new path shown in Table 7. Combining the two sets of conditions in the column 3 of Table 6 and Table 7, we get the final conditions, which are shown in Table 8.

The remaining work involves finding a one-block message M that satisfies all the conditions in Table 8 such that (M, $M+\Delta M_1$, $M+\Delta M_2$,$M+\Delta M_1+\Delta M_2$) causes a 4-collision.

It is easy to correct all the conditions in the first 16 steps by the basic message modifications. There are 63 conditions in steps 17-48, and using the advanced message modification, all of the 44 conditions in steps 17-23 are forced to hold. The precise details for advanced message modifications are omitted, and the main techniques are shown in [12,13,17].

After the advanced message modifications, there are 19 conditions left. So the probability that the modified message M satisfies all the conditions in Table 8 is improved to 2^{-19}. Considering the cost of the message modification, the complexity to find a 4-collision of compression function is about 2^{21}. By computer searching, it is very easy to find 4-collisions, an example of which is given in Table 1.

Table 1. A 4-collision for MD4 compression function. The IV and messages are hexadecimal format.

IV	67452301 efcdab89 98badcfe 10325476
M	74c5f8d6 33fc9eaa 0fd0a9e2 e83340d1 246c716a c0d1931b ef06af4c e7a20583 898483db 0d9a1026 fd62bb0f bb29de31 886af4fa 5c772a7d 6ce0f4fb 6a9c8ce8
$M+\Delta M_1$	74c5f8d6 33fc9eaa 0fd0a9e2 e83340d1 266c716a c0d1931b ef06af4c e7a20583 898483db 0d9a1026 fd62bb0f bb29de31 886af4fa 5c772a7d 6ce0f4fb 6a9c8ce8
$M+\Delta M_2$	74c5f8d6 b3fc9eaa 7fd0a9e2 e83340d1 246c716a c0d1931b ef06af4c e7a20583 898483db 0d9a1026 fd62bb0f bb29de31 8869f4fa 5c772a7d 6ce0f4fb 6a9c8ce8
$M+\Delta M_1$ $+\Delta M_2$	74c5f8d6 b3fc9eaa 7fd0a9e2 e83340d1 266c716a c0d1931b ef06af4c e7a20583 898483db 0d9a1026 fd62bb0f bb29de31 8869f4fa 5c772a7d 6ce0f4fb 6a9c8ce8
Common value	cbe16ea1 2d600674 3b42a32d a1458b54

For any initial value, we can build 4^t-collisions for MD4 as Joux 2^t-collisions construction. Let h be a hash function, and H is its compression function. A denotes the attack algorithm for MD4 4-collisions introduced above. The 4^t-collisions construction is as follows (See Fig.4):

- Let h_0 be the initial value of h.
- For i from 0 to $t-1$ do:
  - Call A and find four different 512-bit messages $B_{i,0}$, $B_{i,1}$, $B_{i,2}$ and $B_{i,3}$ such that $H(h_{i-1}, B_{i,0})$=$H(h_{i-1}, B_{i,1})$=$H(h_{i-1}, B_{i,2})$=$H(h_{i-1}, B_{i,3})$.
  - Let $h_i = H(h_{i-1}, B_i)$.
- Output the 4^t messages of the form $(b_0, b_1, ..., b_{t-1})$ where b_i is one of the four messages $B_{i,j} (j = 0, 1, 2, 3)$, i=0,1,..t-1.

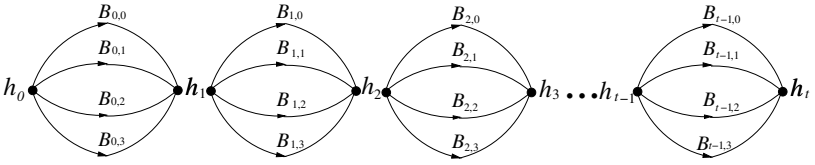


Fig. 4. Our 4^t-collisions construction. Each of $B_{i,j}$ is a one-block message.

6 Finding 4-Collisions on 3-Pass HAVAL

In order to construct 4-collisions for 3-Pass HAVAL compression function, two different differential paths are necessary. This includes the collision differential path of paper [11]. Its message difference written as $\Delta M_1 = (m_{1,i})_{0 \le i \le 31}$, where

$$m_{1,i} = \begin{cases} 2^{10}, & i = 0 \\ 2^{31}, & i = 11 \\ 2^{3}, & i = 18 \\ 0, & \text{others} \end{cases} \tag{1}$$

We have to find another differential path does not collide with the first one. Its message differences is written as $\Delta M_2 = (m_{2,i})_{0 \le i \le 31}$, where

$$m_{2,i} = \begin{cases} 2^{27}, & i = 20 \\ 0, & \text{others} \end{cases} \tag{2}$$

The two differential paths and their corresponding message differences are shown in Tables 9 and 10. Table 11 shows the sufficient conditions that guarantee these two differential paths. The conditions in the first 32 steps of Table 11 can be easily satisfied by the basic message modification. There remain 67 conditions to

Table 2. A 4-collision example for 3-Pass HAVAL compression function

IV	67452301 efcdab89 98badcfe 10325476
IV	ec4e6c89 082efa98 299f31d0 a4093822 03707344 13198a2e 85a308d3 243f6a88
M	73f4a38b d485fb7b 879c1aec cbbbc4a3 9b444b3e 94637936 cf763e2b 7c5eef85 8eea78dd bc1f5e15 7f993595 1ab5658f d541d6d4 059d4cc1 970c929f 0cc0a73a ba7d7af4 8f74ab90 f0b2178f 4c0f1deb 171de335 9c8af784 a2f045ba 078d7633 6e379233 86ce9498 de72bf85 7465522d 97a6ca66 c29c57c7 b5877b8f ac5407d1
$M + \Delta M_1$	73f4a78b d485fb7b 879c1aec cbbbc4a3 9b444b3e 94637936 cf763e2b 7c5eef85 8eea78dd bc1f5e15 7f993595 9ab5658f d541d6d4 059d4cc1 970c929f 0cc0a73a ba7d7af4 8f74ab90 f0b21797 4c0f1deb 171de335 9c8af784 a2f045ba 078d7633 6e379233 86ce9498 de72bf85 7465522d 97a6ca66 c29c57c7 b5877b8f ac5407d1
$M + \Delta M_2$	73f4a38b d485fb7b 879c1aec cbbbc4a3 9b444b3e 94637936 cf763e2b 7c5eef85 8eea78dd bc1f5e15 7f993595 1ab5658f d541d6d4 059d4cc1 970c929f 0cc0a73a ba7d7af4 8f74ab90 f0b2178f 4c0f1deb 1f1de335 9c8af784 a2f045ba 078d7633 6e379233 86ce9498 de72bf85 7465522d 97a6ca66 c29c57c7 b5877b8f ac5407d1
$M + \Delta M_1$ $+\Delta M_2$	73f4a78b d485fb7b 879c1aec cbbbc4a3 9b444b3e 94637936 cf763e2b 7c5eef85 8eea78dd bc1f5e15 7f993595 9ab5658f d541d6d4 059d4cc1 970c929f 0cc0a73a ba7d7af4 8f74ab90 f0b21797 4c0f1deb 1f1de335 9c8af784 a2f045ba 078d7633 6e379233 86ce9498 de72bf85 7465522d 97a6ca66 c29c57c7 b5877b8f ac5407d1
Common value	918b7e59 d3aa59c2 e3a3fd50 cc412921 6bac8ab6 5d418374 09f1f977 290cf7e4

Table 3. The 8(multi)-near-collisions attack example

IV_0	243f6a88	85a308d3	13198a2e	03707344	a4093822	299f31d0	082efa98	ec4e6c89
M_0	e6c99bc9	c99bc914	9bc914d8	c914d80b	14d80bf6	d80bf605	0bf605ef	f605ef83
	05ef831b	ef831b24	831b2486	1b24868f	24868fd3	868fd399	8fd39945	d399459d
	99459d9c	459d9cb7	9d9cb7ba	9cb7ba3f	b7ba3f31	ba3f31dd	3f31dd06	31dd067f
	dd067f7e	067f7ebb	7f7ebb63	7ebb63e8	bb63e8b9	63e8b986	e8b986dc	b986dc14
IV_1	0444e787	978e0d0a	c408c64f	74a629f6	ee1eb57d	fdb20640	6126dd36	4563c119
M	f09e3e1e	862e1e8a	3e1e8a1c	0e8a1c49	88ca9872	14490ac3	30523a85	12cbd516
	c3e51637	cd35b784	163503b0	a78401f9	e5539c0d	b37acef6	7a0e9574	cdf616d7
	f5d83836	16d7f41a	99f81c43	f61c4105	1c0105da	4105da86	05d88403	9a860533
	c6453355	c53355d0	3355d113	55d0d398	90d397dd	d3981d16	97dd1617	1d16175b
$M + \Delta M_1$	f09e3e1e	862e1e8a	3e1e8a1c	0e8a1c49	88ca9872	144912c3	30523a85	12cbd516
	c3e51637	cd35b784	163503b0	a78401f9	e5539c0d	b37acef6	7a0e9574	cdf616d7
	f5d83836	16d7f41a	99f81c43	f61c4105	1c0105da	4105da86	05d88403	9a860533
	c6453355	c53355d0	3355d113	55d0d398	90d397dd	d3981d16	97dd1617	1d16175b
$M + \Delta M_2$	f09e3e1e	862e1e8a	3e1e8a1c	0e8a1c49	88ca9872	14510ac3	30523a85	12cbd516
	c3e51637	cd35b784	163503b0	a78401f9	e5539c0d	b37acef6	7a0e9574	cdf616d7
	f5d83836	16d7f41a	99f81c43	f61c4105	1c0105da	4105da86	05d88403	9a860533
	c6453355	c53355d0	3355d113	55d0d398	90d397dd	d3981d16	97dd1617	1d16175b
$M + \Delta M_3$	f09e3e1e	862e1e8a	3e1e8a1c	0e8a1c49	88ca9872	1c490ac3	30523a85	12cbd516
	c3e51637	cd35b784	163503b0	a78401f9	e5539c0d	b37acef6	7a0e9574	cdf616d7
	f5d83836	16d7f41a	99f81c43	f61c4105	1c0105da	4105da86	05d88403	9a860533
	c6453355	c53355d0	3355d113	55d0d398	90d397dd	d3981d16	97dd1617	1d16175b
$M + \Delta M_1$ $+\Delta M_2$	f09e3e1e	862e1e8a	3e1e8a1c	0e8a1c49	88ca9872	145112c3	30523a85	12cbd516
	c3e51637	cd35b784	163503b0	a78401f9	e5539c0d	b37acef6	7a0e9574	cdf616d7
	f5d83836	16d7f41a	99f81c43	f61c4105	1c0105da	4105da86	05d88403	9a860533
	c6453355	c53355d0	3355d113	55d0d398	90d397dd	d3981d16	97dd1617	1d16175b
$M + \Delta M_1$ $+\Delta M_3$	f09e3e1e	862e1e8a	3e1e8a1c	0e8a1c49	88ca9872	1c4912c3	30523a85	12cbd516
	c3e51637	cd35b784	163503b0	a78401f9	e5539c0d	b37acef6	7a0e9574	cdf616d7
	f5d83836	16d7f41a	99f81c43	f61c4105	1c0105da	4105da86	05d88403	9a860533
	c6453355	c53355d0	3355d113	55d0d398	90d397dd	d3981d16	97dd1617	1d16175b
$M + \Delta M_2$ $+\Delta M_3$	f09e3e1e	862e1e8a	3e1e8a1c	0e8a1c49	88ca9872	1c510ac3	30523a85	12cbd516
	c3e51637	cd35b784	163503b0	a78401f9	e5539c0d	b37acef6	7a0e9574	cdf616d7
	f5d83836	16d7f41a	99f81c43	f61c4105	1c0105da	4105da86	05d88403	9a860533
	c6453355	c53355d0	3355d113	55d0d398	90d397dd	d3981d16	97dd1617	1d16175b
$M + \Delta M_1$ $+\Delta M_2$ $+\Delta M_3$	f09e3e1e	862e1e8a	3e1e8a1c	0e8a1c49	88ca9872	1c5112c3	30523a85	12cbd516
	c3e51637	cd35b784	163503b0	a78401f9	e5539c0d	b37acef6	7a0e9574	cdf616d7
	f5d83836	16d7f41a	99f81c43	f61c4105	1c0105da	4105da86	05d88403	9a860533
	c6453355	c53355d0	3355d113	55d0d398	90d397dd	d3981d16	97dd1617	1d16175b
Near-collision value	a : 10110000011011011010110110001011				e : 11110011110100101000011010101101			
	b : 1110?1111000?1011010?01000101001				f : 10010000111000001010000101101010			
	c : 00010011011011000100100011101101				g : 01100111101001000110110111110111			
	d : 11001101110000111101101000001111				h : 00001011010101101100111010011010			

satisfy from steps 33-96. Utilizing the advanced message modification technique, 39 out of them can be corrected and leaving 28 conditions unmodified. Considering the message modification , the complexity to find a message M so that satisfies all of the conditions is about 2^{30}. That is to say, it only costs 2^{30} 3-Pass HAVAL computations to find a 4-collisions $(M, M + \Delta M_1, M + \Delta M_2, M + \Delta M_1 + \Delta M_2)$. In fact, the complexity can be further reduced by more precise modification. One 4-collisions example is illustrated in Table 2.

7 Finding 8-Near-Collisions on 3-Pass HAVAL

In order to construct 8-near-collisions for 3-Pass HAVAL compression function, we need to find three feasible near-collision differential paths. One of them is shown in Table 12 where the message difference is selected as

$$\Delta M_1 = (\Delta m_i)_{0 \le i \le 31},\ \Delta m_i = 2^{11} \text{ when } i = 5, \text{ else } \Delta m_i = 0.$$

If a message M satisfies all 73 conditions in column 4 of Table 12, the value of $H(M)$ is almost the same as $H(M + \Delta M)$ except one bit.

In fact, a similar differential path can be constructed for any one of the 48 message differences $\Delta m_5 = \pm 2^j (0 \le j \le 31,\ j \ne 3, 4, 5, 10, 14, 15, 16, 31)$. We select two other differential paths corresponding to $\Delta m_5 = 2^{19}$ and 2^{27} respectively, and denote their message differences as ΔM_2 and ΔM_3. So we get three near-collision differential paths.

In order to implement the message modification, we sum up the three sets of sufficient conditions that are shown in Table 13. There are a total of 73×3 conditions, in which 210 conditions focus on the first round(steps 1-32). So using only the basic message modification, the probability to find a message which satisfies all the 219 conditions is improved to 2^{-9}.

That there are 6 conditions on $IV(a_0)$, which are not consistent with the original initial value IV_0 of HAVAL. In order to avoid this situation, an extra message block M_0 is needed so that the hash value $H(M_0)$ satisfies these 6 conditions. Under the new initial value $IV_1(H(M_0))$, we find 8 different messages M, $M + \Delta M_0$, $M + \Delta M_1$, $M + \Delta M_2$, $M + \Delta M_0 + \Delta M_1$, $M + \Delta M_0 + \Delta M_2$, $M + \Delta M_1 + \Delta M_2$ $M + \Delta M_0 + \Delta M_1 + \Delta M_2$. Their hash values are almost coincide in all but 3 bits. Our results are shown in Table 3. The hash values are expressed in binary format(the most left bit is MSB) to reveal the positions of the three differing bits. The 8-near-collision exactly traverses all of the 8 cases corresponding to these three positions.

8 Conclusion

We have presented a dedicated multi-collision attack on the compression functions of MD4 and 3-Pass HAVAL, and multi-near-collision attack on 3-Pass HAVAL. We believe that the multi-collisions technique poses further danger for practical applications of these hash functions.

Acknowledgments

We wish to thank Adi Shamir for his report presented in Tsinghua University which motivated us to start this research. Very special thanks to Matt Henricksen for giving hints to improve the exposition.

References

1. Biham, E., Chen, R.: Near-Collisions of SHA-0. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 290–305. Springer, Heidelberg (2004)
2. Canniére, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer, Heidelberg (2006)
3. Rompay, B., Biryukov, A., Preneel, B., Vandewalle, J.: Cryptanalysis of 3-Pass HAVAL. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 228–245. Springer, Heidelberg (2003)
4. Nandi, M., Stinson, D.R.: Multicollision Attacks on a Class of Hash Functions, IACR preprint archive (2005)
5. Joux, A.: Multicollisions in Iterated Hash Functions. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 306–316. Springer, Heidelberg (2004)
6. Hoch, J.J., Shamir, A.: Breaking the ICE - Finding Multicollisions in Iterated Concatenated and Expanded (ICE) Hash Functions. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 179–194. Springer, Heidelberg (2006)
7. Rivest, R.L.: The MD4 Message Digest Algorithm. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 303–311. Springer, Heidelberg (1991)
8. Rivest, R.L.: The MD5 message-digest algorithm, Request for Comments(RFC 1320), Internet Activities Board, Internet Privacy Task Force (1992)
9. Schlaffer, M., Oswald, E.: Searching for Differential Paths in MD4. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 242–261. Springer, Heidelberg (2006)
10. Wang, X.Y., Feng, F.D., Lai, X.J., Yu, H.B.: Collisions for Some Hash Functions MD4, MD5, HAVAL-128, RIPEMD, http:eprint.iacr.org/2004/264/199.pdf
11. Wang, X.Y., Feng, F.D., Yu, X.: An attack on HAVAL function HAVAL-128. Science in China Ser. F Information Sciences 48(5), 1–12 (2005)
12. Wang, X.Y., Lai, X.J., et al.: Cryptanalysis for Hash Functions MD4 and RIPEMD. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 1–18. Springer, Heidelberg (2005)
13. Wang, X.Y., Yu, H.B.: How to Break MD5 and Other Hash Functions. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
14. Wang, X.Y., Yu, H.B., Yin, Y.L.: Efficient Collision Search Attacks on SHA-0. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 1–16. Springer, Heidelberg (2005)
15. Wang, X.Y., Yin, Y.L., Yu, H.B.: Finding collisions on the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
16. Zheng, Y., Pieprzyk, J., Seberry, J.: HAVAL–A One-way Hashing Algorithm with Variable Length of Output. In: Advances in Cryptology, Auscrypt 1992 Proceedings, pp. 83–104 (1992)
17. Yu, H.B., Wang, G.L., Zhang, G.Y., Wang, X.Y.: The Second-Preimage Attack on MD4. In: Desmedt, Y.G., Wang, H., Mu, Y., Li, Y. (eds.) CANS 2005. LNCS, vol. 3810, pp. 1–12. Springer, Heidelberg (2005)

Appendix

Table 4. The property for the round function ϕ_1 and ϕ_2 of MD4. $\Delta x = 1$ denotes x changes from 0 to 1, $\Delta x = -1$ denotes x changes from 1 to 0.

Δx	Δy	Δz	$\Delta\phi_1 = 0$	$\Delta\phi_1 = 1$	$\Delta\phi_1 = -1$	$\Delta\phi_2 = 0$	$\Delta\phi_2 = 1$	$\Delta\phi_2 = -1$
0	0	0	1	–	–	1	–	–
0	0	1	$x = 1$	$x = 0$	–	$x = y$	$x \neq y$	–
0	0	-1	$x = 1$	–	$x = 0$	$x = y$	–	$x \neq y$
0	1	0	$x = 0$	$x = 1$	–	$x = z$	$x \neq z$	–
0	1	1	–	1	–	–	1	–
0	1	-1	–	$x = 1$	$x = 0$	1	–	–
0	-1	0	$x = 0$	–	$x = 1$	$x = z$	–	$x \neq z$
0	-1	1	–	$x = 0$	$x = 1$	1	–	–
0	-1	-1	–	–	1	–	–	1
1	0	0	$y = z$	$y = 1$, $z = 0$	$y = 0$, $z = 1$	$y = z$	$y \neq z$	–
1	0	1	$y = 0$	$y = 1$	–	–	1	–
1	0	-1	$y = 1$	–	$y = 0$	1	–	–
1	1	0	$z = 1$	$z = 0$	–	–	1	–
1	1	1	–	1	–	–	1	–
1	1	-1	1	–	–	–	1	–
1	-1	0	$z = 0$	–	$z = 1$	1	–	–
1	-1	1	1	–	–	–	1	–
1	-1	-1	–	–	1	–	–	1
-1	0	0	$y = z$	$y = 0$, $z = 1$	$y = 1$, $z = 0$	$y = z$	–	$y \neq z$
-1	0	1	$y = 1$	$y = 0$	–	1	–	–
-1	0	-1	$y = 0$	–	$y = 1$	–	–	1
-1	1	0	$z = 0$	$z = 1$	–	1	–	–
-1	1	1	–	1	–	–	1	–
-1	1	-1	1	–	–	–	–	1
-1	-1	0	$z = 1$	–	$z = 0$	–	–	1
-1	-1	1	1	–	–	–	–	1
-1	-1	-1	–	–	1	–	–	1

Table 5. Some properties for the first round function f_1 of the 3-Pass Haval

Δa	Δb	Δc	Δd	Δe	Δf	Δg	$\Delta f_1 = 0$	$\Delta f_1 = 1$	$\Delta f_1 = -1$
0	0	0	0	0	0	0	1	–	–
1	0	0	0	0	0	0	$g = 0$	$g = 1,\ cd + bf + ce + e = 0$	$g = 1,\ cd + bf + ce + e = 1$
-1	0	0	0	0	0	0	$g = 0$	$g = 1,\ cd + bf + ce + e = 1$	$g = 1,\ cd + bf + ce + e = 0$
0	1	0	0	0	0	0	$f = 0$	$f = 1,\ cd + ag + ce + e = 0$	$f = 1,\ cd + ag + ce + e = 1$
0	-1	0	0	0	0	0	$f = 0$	$f = 1, cd + ag + ce + e = 1$	$f = 1,\ cd + ag + ce + e = 0$
0	0	1	0	0	0	0	$e = d$	$e \neq d,\ ag + bf + e = 0$	$e \neq d,\ ag + bf + e = 1$
0	0	-1	0	0	0	0	$e = d$	$e \neq d,\ ag + bf + e = 1$	$e \neq d,\ ag + bf + e = 0$
0	0	0	1	0	0	0	$c = 0$	$c = 1,\ ag + bf + ce + e = 0$	$c = 1,\ ag + bf + ce + e = 1$
0	0	0	-1	0	0	0	$c = 0$	$c = 1,\ ag + bf + ce + e = 1$	$c = 1,\ ag + bf + ce + e = 0$
0	0	0	0	1	0	0	$c = 1$	$c = 0,\ cd + ag + bf = 0$	$c = 0,\ cd + ag + bf = 1$
0	0	0	0	-1	0	0	$c = 1$	$c = 0,\ cd + ag + bf = 1$	$c = 0,\ cd + ag + bf = 0$
0	0	0	0	0	1	0	$b = 0$	$b = 1,\ cd + ag + ce + e = 0$	$b = 1,\ cd + ag + ce + e = 1$
0	0	0	0	0	-1	0	$b = 0$	$b = 1,\ cd + ag + ce + e = 1$	$b = 1,\ cd + ag + ce + e = 0$
0	0	0	0	0	0	1	$a = 0$	$a = 1,\ cd + bf + ce + e = 0$	$a = 1,\ cd + bf + ce + e = 1$
0	0	0	0	0	0	-1	$a = 0$	$a = 1,\ cd + bf + ce + e = 1$	$a = 1,\ cd + bf + ce + e = 0$

Table 6. The collision differential path 1 of MD4

Step	Output for M	m_i	s_i	Δm_i	Output for M'	Sufficient conditions
1	a_1	m_0	3			
2	d_1	m_1	7			
3	c_1	m_2	11			
4	b_1	m_3	19			
5	a_2	m_4	3	2^{25}	$a_2[29]$	$a_{2,29} = 0$
6	d_2	m_5	7		d_2	$b_{1,29} = c_{1,29}$
7	c_2	m_6	11		c_2	$d_{2,29} = 0$
8	b_2	m_7	19		b_2	$c_{2,29} = 1$
9	a_3	m_8	3		$a_3[32]$	$a_{3,32} = 0$
10	d_3	m_9	7		d_3	$b_{2,32} = c_{2,32}$
11	c_3	m_{10}	11		$c_3[-11]$	$c_{3,11} = 1$, $d_{3,32} = 1$
12	b_3	m_{11}	19		b_3	$c_{3,32} = 1$, $d_{3,11} = a_{3,11}$
13	a_4	m_{12}	3		$a_4[3]$	$a_{4,3} = 0$, $b_{3,11} = 0$
14	d_4	m_{13}	7		d_4	$b_{3,3} = c_{3,3}$, $a_{4,11} = 1$
15	c_4	m_{14}	11		$c_4[-22]$	$c_{4,22} = 1$, $d_{4,3} = 0$
16	b_4	m_{15}	19		b_4	$d_{4,22} = a_{4,22}$, $c_{4,3} = 1$
17	a_5	m_0	3		$a_5[6]$	$a_{5,6} = 0$, $b_{4,22} = d_{4,22}$
18	d_5	m_4	5	2^{25}	$d_5[11, 31]$	$d_{5,11} = 0$, $d_{5,31} = 0$, $a_{5,22} = b_{4,22}$, $b_{4,6} = c_{4,6} + 1$
19	c_5	m_8	9		$c_5[-31]$	$c_{5,31} = 1$, $a_{5,11} = b_{4,11}$, $d_{5,6} = b_{4,6}$, $a_{5,31} = b_{4,31}$
20	b_5	m_{12}	13		b_5	$c_{5,6} = d_{5,6}$, $c_{5,11} = a_{5,11}$
21	a_6	m_1	3		$a_6[9]$	$a_{6,9} = 0$, $b_{5,11} = c_{5,11}$
22	d_6	m_5	5		$d_6[16]$	$d_{6,16} = 0$, $b_{5,9} = c_{5,9}$, $a_{6,31} = b_{5,31} + 1$
23	c_6	m_9	9		$c_6[8, -9]$	$c_{6,8} = 0$, $c_{6,9} = 1$, $d_{6,9} = b_{5,9}$, $a_{6,16} = b_{5,16}$
24	b_6	m_{13}	13		b_6	$d_{6,8} = a_{6,8}$, $c_{6,16} = a_{6,16}$
25	a_7	m_2	3		a_7	$b_{6,9} = d_{6,9} + 1$, $b_{6,8} = d_{6,8}$, $b_{6,16} = c_{6,16}$
26	d_7	m_6	5		$d_7[21]$	$a_{7,8} = b_{6,8}$, $a_{7,9} = b_{6,9}$, $d_{7,21} = 0$
27	c_7	m_{10}	9		$c_7[-17]$	$c_{7,17} = 1$, $a_{7,21} = b_{6,21}$
28	b_7	m_{14}	13		b_7	$d_{7,17} = a_{7,17}$, $c_{7,21} = a_{7,21}$
29	a_8	m_3	3		a_8	$b_{7,17} = d_{7,17}$, $b_{7,21} = c_{7,21}$
30	d_8	m_7	5		$d_8[26]$	$d_{8,26} = 0$, $a_{8,17} = b_{7,17}$
31	c_8	m_{11}	9		$c_8[-26]$	$c_{8,26} = 1$, $a_{8,26} = b_{7,26}$
32	b_8	m_{15}	13		b_8	
33	a_9	m_0	3		a_9	
34	d_9	m_8	9		d_9	$a_{9,26} = b_{8,26}$
35	c_9	m_4	11	2^{25}	c_9	
36	b_9	m_{12}	15		b_9	

Table 7. The collision differential path 2 of MD4

Step	Output for M	m_i	s_i	Δm_i	Output for M'	Sufficient conditions
1	a_1	m_0	3			
2	d_1	m_1	7	2^{31}	$d_1[7]$	$d_{1,7} = 0$
3	c_1	m_2	11	$-2^{28} + 2^{31}$	$c_1[-8, 11]$	$c_{1,8} = 1$, $c_{1,11} = 0$, $a_{1,7} = b_{0,7}$
4	b_1	m_3	19		$b_1[26]$	$b_{1,26} = 0$, $c_{1,7} = 1$, $d_{1,8} = a_{1,8}$, $d_{1,11} = a_{1,11}$
5	a_2	m_4	3		a_2	$c_{1,26} = d_{1,26}$, $b_{1,8} = 0$, $b_{1,11} = 0$, $b_{1,7} = 1$
6	d_2	m_5	7		$d_2[14]$	$d_{2,14} = 0$, $a_{2,26} = 0$, $a_{2,8} = 1$, $a_{2,11} = 1$
7	c_2	m_6	11		$c_2[-19, 22]$	$c_{2,19} = 1$, $c_{2,22} = 0$, $a_{2,14} = b_{1,14}$, $d_{2,26} = 1$
8	b_2	m_7	19		$b_2[13]$	$b_{2,13} = 0$, $c_{2,14} = 0$, $d_{2,19} = a_{2,19}$, $d_{2,22} = a_{2,22}$
9	a_3	m_8	3		$a_3[17]$	$a_{3,17} = 0$, $c_{2,13} = d_{2,13}$, $b_{2,14} = 0$, $b_{2,19} = 0$, $b_{2,22} = 0$
10	d_3	m_9	7		$d_3[20, -21, -22, 23]$	$d_{3,20} = 0$, $d_{3,21} = 1$, $d_{3,22} = 1$, $d_{3,23} = 0$, $b_{2,17} = c_{2,17}$, $a_{3,13} = 1$, $a_{3,19} = 1$, $a_{3,22} = 1$
11	c_3	m_{10}	11		$c_3[-30]$	$c_{3,30} = 1$, $a_{3,20} = b_{2,20}$, $a_{3,21} = b_{2,21}$, $a_{3,23} = b_{2,23}$, $d_{3,17} = 0$, $d_{3,13} = 1$
12	b_3	m_{11}	19		$b_3[32]$	$b_{3,32} = 0$, $d_{3,30} = a_{3,30}$, $c_{3,20} = 0$, $c_{3,21} = 0$, $c_{3,22} = 0$, $c_{3,23} = 0$, $c_{3,17} = 1$
13	a_4	m_{12}	3	-2^{16}	$a_4[23, 26]$	$a_{4,23} = 0$, $a_{4,26} = 0$, $b_{3,20} = 0$, $b_{3,21} = 1$, $b_{3,22} = 1$, $b_{3,23} = 0$, $c_{3,32} = d_{3,32}$, $b_{3,30} = 0$
14	d_4	m_{13}	7		$d_4[-27, -29, 30]$	$d_{4,27} = 1$, $d_{4,29} = 1$, $d_{4,30} = 0$, $b_{3,26} = c_{3,26}$, $a_{4,32} = 0$, $a_{4,30} = 1$
15	c_4	m_{14}	11			$a_{4,27} = b_{3,27}$, $a_{4,29} = b_{3,29}$, $d_{4,23} = 0$, $d_{4,26} = 0$, $d_{4,32} = 1$
16	b_4	m_{15}	19		$b_4[17, 19]$	$b_{4,17} = 0$, $b_{4,19} = 0$, $c_{4,27} = 0$, $c_{4,29} = 0$, $c_{4,30} = 1$, $c_{4,23} = 1$, $c_{4,26} = 1$
17	a_5	m_0	3		$a_5[-26, 27, -29, -32]$	$a_{5,26} = 1$, $a_{5,27} = 0$, $a_{5,29} = 1$, $a_{5,32} = 1$, $b_{4,27} = 1$, $b_{4,29} = 1$, $b_{4,30} = 1$, $c_{4,19} = d_{4,19}$, $c_{4,17} = d_{4,17}$
18	d_5	m_4	5		d_5	$b_{4,26} = c_{4,26}$, $b_{4,32} = c_{4,32}$, $a_{5,19} = c_{4,19}$, $a_{5,17} = c_{4,17}$
19	c_5	m_8	9		c_5	$d_{5,19} = a_{5,19}$, $d_{5,26} = b_{4,26}$, $d_{5,27} = b_{4,27}$, $d_{5,29} = b_{4,29}$, $d_{5,32} = b_{4,32}$, $d_{5,17} = a_{5,17}$
20	b_5	m_{12}	13	-2^{16}	$b_5[32]$	$b_{5,32} = 0$, $c_{5,26} = d_{5,26}$, $c_{5,27} = d_{5,27}$, $c_{5,29} = d_{5,29}$, $c_{5,32} = d_{5,32}$
21	a_6	m_1	3		$a_6[29, -32]$	$a_{6,29} = 0$, $a_{6,32} = 1$
22	d_6	m_5	5			$b_{5,29} = c_{5,29}$
23	c_6	m_9	9			$d_{6,29} = b_{5,29}$
24	b_6	m_{13}	13		b_6	$c_{6,29} = d_{6,29}$, $c_{6,32} = d_{6,32} + 1$
25	a_7	m_2	3	$-2^{28} + 2^{31}$	a_7	
36	b_9	m_{12}	15	-2^{16}	$b_9[-32]$	$b_{9,32} = 1$
37	a_{10}	m_2	3	$-2^{28} + 2^{31}$	$a_{10}[-32]$	$a_{10,32} = 1$
38	d_{10}	m_{10}	9			
39	c_{10}	m_6	11			
40	b_{10}	m_{14}	15			
41	a_{11}	m_1	3	2^{31}		

Table 8. A set of sufficient conditions for the 4-collision of MD4

Step	Output variable	Variable conditions
1	a_1	$a_{1,7} = b_{0,7}$
2	d_1	$d_{1,7} = 0$, $d_{1,8} = a_{1,8}$, $d_{1,11} = a_{1,11}$
3	c_1	$c_{1,7} = 1$, $c_{1,8} = 1$, $c_{1,11} = 0$, $c_{1,26} = d_{1,26}$
4	b_1	$b_{1,7} = 1$, $b_{1,8} = 0$, $b_{1,11} = 0$, $b_{1,26} = 0$, $b_{1,29} = c_{1,29}$
5	a_2	$a_{2,8} = 1$, $a_{2,11} = 1$, $a_{2,14} = b_{1,14}$, $a_{2,26} = 0$, $a_{2,29} = 0$
6	d_2	$d_{2,14} = 0$, $d_{2,19} = a_{2,19}$, $d_{2,22} = a_{2,22}d_{2,26} = 1$, $d_{2,29} = 0$
7	c_2	$c_{2,13} = d_{2,13}$, $c_{2,14} = 0$, $c_{2,19} = 1$, $c_{2,22} = 0$, $c_{2,29} = 1$, $c_{2,32} = 0$
8	b_2	$b_{2,13} = 0$, $b_{2,14} = 0$, $b_{2,17} = c_{2,17}$, $b_{2,19} = 0$, $b_{2,22} = 0$, $b_{2,32} = 0$
9	a_3	$a_{3,13} = 1$, $a_{3,17} = 0$, $a_{3,19} = 1$, $a_{3,20} = b_{2,20}$, $a_{3,21} = b_{2,21}$,$a_{3,22} = 1$, $a_{3,23} = b_{2,23}$, $a_{3,32} = 0$
10	d_3	$d_{3,11} = a_{3,11}$, $d_{3,13} = 1$, $d_{3,17} = 0$, $d_{3,20} = 0$, $d_{3,21} = 1$, $d_{3,22} = 1$, $d_{3,23} = 0$, $d_{3,30} = a_{3,30}$, $d_{3,32} = 1$
11	c_3	$c_{3,11} = 1$, $c_{3,17} = 1$, $c_{3,20} = 0$, $c_{3,21} = 0$, $c_{3,22} = 0$,$c_{3,23} = 0$, $c_{3,30} = 1$, $c_{3,32} = 1$
12	b_3	$b_{3,3} = c_{3,3}$, $b_{3,11} = 0$, $b_{3,20} = 0$, $b_{3,21} = 1$, $b_{3,22} = 1$, $b_{3,23} = 0$, $b_{3,26} = c_{3,26}$, $b_{3,30} = 0$, $b_{3,32} = 0$
13	a_4	$a_{4,3} = 0$, $a_{4,11} = 1$, $a_{4,23} = 0$, $a_{4,26} = 0$, $a_{4,27} = b_{3,27}$, $a_{4,29} = b_{3,29}$, $a_{4,30} = 1$, $a_{4,32} = 0$
14	d_4	$d_{4,3} = 0$, $d_{4,22} = a_{4,22}$, $d_{4,23} = 0$, $d_{4,26} = 0$, $d_{4,27} = 1$, $d_{4,29} = 1$, $d_{4,30} = 0$, $d_{4,32} = 1$
15	c_4	$c_{4,3} = 1$, $c_{4,17} = d_{4,17}$, $c_{4,19} = d_{4,19}$, $c_{4,22} = 1$, $c_{4,23} = 1$, $c_{4,26} = 1$, $c_{4,27} = 0$, $c_{4,29} = 0$, $c_{4,30} = 1$
16	b_4	$b_{4,6} = c_{4,6} + 1$, $b_{4,17} = 0$, $b_{4,19} = 0$, $b_{4,22} = d_{4,22}$, $b_{4,26} = 1$, $b_{4,27} = 1$, $b_{4,29} = 1$, $b_{4,30} = 1$, $b_{4,32} = c_{4,32}$
17	a_5	$a_{5,6} = 0$, $a_{5,11} = b_{4,11}$, $a_{5,17} = c_{4,17}$, $a_{5,19} = c_{4,19}$, $a_{5,22} = b_{4,22}$, $a_{5,26} = 1$, $a_{5,27} = 0$,$a_{5,29} = 1$, $a_{5,31} = b_{4,31}$, $a_{5,32} = 1$
18	d_5	$d_{5,6} = b_{4,6}$, $d_{5,11} = 0$, $d_{5,17} = a_{5,17}$, $d_{5,19} = a_{5,19}$, $d_{5,26} = b_{4,26}$, $d_{5,27} = b_{4,27}$, $d_{5,29} = b_{4,29}$, $d_{5,31} = 0$, $d_{5,32} = b_{4,32}$
19	c_5	$c_{5,6} = d_{5,6}$, $c_{5,11} = a_{5,11}$, $c_{5,26} = d_{5,26}$, $c_{5,27} = d_{5,27}$, $c_{5,29} = d_{5,29}$, $c_{5,31} = 1$, $c_{5,32} = d_{5,32}$
20	b_5	$b_{5,9} = c_{5,9}$, $b_{5,11} = c_{5,11}$, $b_{5,29} = c_{5,29}$, $b_{5,32} = 0$
21	a_6	$a_{6,9} = 0$, $a_{6,16} = b_{5,16}$, $a_{6,29} = 0$, $a_{6,31} = b_{5,31} + 1$, $a_{6,32} = 1$
22	d_6	$d_{6,8} = a_{6,8}$, $d_{6,9} = b_{5,9}$, $d_{6,16} = 0$, $d_{6,29} = b_{5,29}$
23	c_6	$c_{6,8} = 0$, $c_{6,9} = 1$, $c_{6,16} = a_{6,16}$, $c_{6,29} = d_{6,29}$, $c_{6,32} = d_{6,32} + 1$
24	b_6	$b_{6,8} = d_{6,8}$, $b_{6,9} = d_{6,9} + 1$, $b_{6,16} = c_{6,16}$
25	a_7	$a_{7,8} = b_{6,8}$, $a_{7,9} = b_{6,9}$, $a_{7,21} = b_{6,21}$
26	d_7	$d_{7,17} = a_{7,17}$, $d_{7,21} = 0$
27	c_7	$c_{7,17} = 1$, $c_{7,21} = a_{7,21}$
28	b_7	$b_{7,17} = d_{7,17}$, $b_{7,21} = c_{7,21}$
29	a_8	$a_{8,17} = b_{7,17}$, $a_{8,26} = b_{7,26}$
30	d_8	$d_{8,26} = 0$
31	c_8	$c_{8,26} = 1$
32	b_8	
33	a_9	$a_{9,26} = b_{8,26}$
34	d_9	
35	c_9	
36	b_9	$b_{9,32} = 1$
37	a_{10}	$a_{10,32} = 1$

Table 9. Collision differential path 1 of 3-Pass HAVAL for the 4-collision attack

Step	m'_{i-1}	Δm_i	Output for M'_1
1	m_0	2^{10}	a_1[11], a_0, b_0, c_0, d_0, e_0, f_0, g_0
2	m_1		a_2, a_1[11], a_0, b_0, c_0, d_0, e_0, f_0
3	m_2		a_3, a_2, a_1[11], a_0, b_0, c_0, d_0, e_0
4	m_3		a_4, a_3, a_2, a_1[11], a_0, b_0, c_0, d_0
5	m_4		a_5, a_4, a_3, a_2, a_1[11], a_0, b_0, c_0
6	m_5		a_6, a_5, a_4, a_3, a_2, a_1[11], a_0, b_0
7	m_6		a_7[-4,-5,-6,-7,8], a_6, a_5, a_4, a_3, a_2, a_1[11], a_0
8	m_7		a_8[-29,-30,-31,32], a_7[-4,-5,-6,-7,8], a_6, a_5, a_4, a_3, a_2, a_1[11]
9	m_8		a_9[-22,-23], a_8[-29,-30,-31,32], a_7[-4,-5,-6,-7,8], a_6, a_5, a_4, a_3, a_2
10	m_9		a_{10}, a_9[-22,-23], a_8[-29,-30,-31,32], a_7[-4,-5,-6,-7,8], a_6, a_5, a_4, a_3
11	m_{10}		a_{11}[-15], a_{10}, a_9[-22,-23], a_8[-29,-30,-31,32], a_7[-4,-5,-6,-7,8], a_6, a_5, a_4
12	m_{11}	2^{31}	a_{12}, a_{11}[-15], a_{10}, a_9[-22,-23], a_8[-29,-30,-31,32], a_7[-4,-5,-6,-7,8], a_6, a_5
13	m_{12}		a_{13}[1], a_{12}, a_{11}[-15], a_{10}, a_9[-22,-23], a_8[-29,-30,-31,32], a_7[-4,-5,-6,-7,8], a_6
14	m_{13}		a_{14}, a_{13}[1], a_{12}, a_{11}[-15], a_{10}, a_9[-22,-23], a_8[-29,-30,-31,32], a_7[-4,-5,-6,-7,8]
15	m_{14}		a_{15}, a_{14}, a_{13}[1], a_{12}, a_{11}[-15], a_{10}, a_9[-22,-23], a_8[-29,-30,-31,32]
16	m_{15}		a_{16}[18], a_{15}, a_{14}, a_{13}[1], a_{12}, a_{11}[-15], a_{10}, a_9[-22,-23]
17	m_{16}		a_{17}[12], a_{16}[18], a_{15}, a_{14}, a_{13}[1], a_{12}, a_{11}[-15], a_{10}
18	m_{17}		a_{18}, a_{17}[12], a_{16}[18], a_{15}, a_{14}, a_{13}[1], a_{12}, a_{11}[-15]
19	m_{18}	2^3	a_{19}, a_{18}, a_{17}[12], a_{16}[18], a_{15}, a_{14}, a_{13}[1], a_{12}
20	m_{19}		a_{20}, a_{19}, a_{18}, a_{17}[12], a_{16}[18], a_{15}, a_{14}, a_{13}[1]
21	m_{20}		a_{21}[22], a_{20}, a_{19}, a_{18}, a_{17}[12], a_{16}[18], a_{15}, a_{14}
22	m_{21}		a_{22}, a_{21}[22], a_{20}, a_{19}, a_{18}, a_{17}[12], a_{16}[18], a_{15}
23	m_{22}		a_{23}, a_{22}, a_{21}[22], a_{20}, a_{19}, a_{18}, a_{17}[12], a_{16}[18]
24	m_{23}		a_{24}[7], a_{23}, a_{22}, a_{21}[22], a_{20}, a_{19}, a_{18}, a_{17}[12]
25	m_{24}		a_{25}[1,2,3,-4], a_{24}[7], a_{23}, a_{22}, a_{21}[22], a_{20}, a_{19}, a_{18}
26	m_{25}		a_{26}, a_{25}[1,2,3,-4], a_{24}[7], a_{23}, a_{22}, a_{21}[22], a_{20}, a_{19}
27	m_{26}		a_{27}, a_{26}, a_{25}[1,2,3,-4], a_{24}[7], a_{23}, a_{22}, a_{21}[22], a_{20}
28	m_{27}		a_{28}, a_{27}, a_{26}, a_{25}[1,2,3,-4], a_{24}[7], a_{23}, a_{22}, a_{21}[22]
29	m_{28}		a_{29}[11], a_{28}, a_{27}, a_{26}, a_{25}[1,2,3,-4], a_{24}[7], a_{23}, a_{22}
30	m_{29}		a_{30}, a_{29}[11], a_{28}, a_{27}, a_{26}, a_{25}[1,2,3,-4], a_{24}[7], a_{23}
31	m_{30}		a_{31}, a_{30}, a_{29}[11], a_{28}, a_{27}, a_{26}, a_{25}[1,2,3,-4], a_{24}[7]
32	m_{31}		a_{32}, a_{31}, a_{30}, a_{29}[11], a_{28}, a_{27}, a_{26}, a_{25}[1,2,3,-4]
33	m_5		a_{33}[22], a_{32}, a_{31}, a_{30}, a_{29}[11], a_{28}, a_{27}, a_{26}
34	m_{14}		a_{34}, a_{33}[22], a_{32}, a_{31}, a_{30}, a_{29}[11], a_{28}, a_{27}
35	m_{26}		a_{35}, a_{34}, a_{33}[22], a_{32}, a_{31}, a_{30}, a_{29}[11], a_{28}
36	m_{18}		a_{36}, a_{35}, a_{34}, a_{33}[22], a_{32}, a_{31}, a_{30}, a_{29}[11]
37	m_{11}		a_{37}, a_{36}, a_{35}, a_{34}, a_{33}[22], a_{32}, a_{31}, a_{30}
38	m_{28}		a_{38}, a_{37}, a_{36}, a_{35}, a_{34}, a_{33}[22], a_{32}, a_{31}
39	m_7		a_{39}, a_{38}, a_{37}, a_{36}, a_{35}, a_{34}, a_{33}[22], a_{32}
40	m_{16}		a_{40}, a_{39}, a_{38}, a_{37}, a_{36}, a_{35}, a_{34}, a_{33}[22]
86	m_0	2^{10}	a_{86}[11], a_{85}, a_{84}, a_{83}, a_{82}, a_{81}, a_{80}, a_{79}
87	m_{18}	2^3	a_{87}, a_{86}[11], a_{85}, a_{84}, a_{83}, a_{82}, a_{81}, a_{80}
88	m_{27}		a_{88}, a_{87}, a_{86}[11], a_{85}, a_{84}, a_{83}, a_{82}, a_{81}
89	m_{13}		a_{89}, a_{88}, a_{87}, a_{86}[11], a_{85}, a_{84}, a_{83}, a_{82}
90	m_6		a_{90}, a_{89}, a_{88}, a_{87}, a_{86}[11], a_{85}, a_{84}, a_{83}
91	m_{21}		a_{91}, a_{90}, a_{89}, a_{88}, a_{87}, a_{86}[11], a_{85}, a_{84}
92	m_{10}		a_{92}, a_{91}, a_{90}, a_{89}, a_{88}, a_{87}, a_{86}[11], a_{85}
93	m_{23}		a_{93}, a_{92}, a_{91}, a_{90}, a_{89}, a_{88}, a_{87}, a_{86}[11]
94	m_{11}	2^{31}	a_{94}, a_{93}, a_{92}, a_{91}, a_{90}, a_{89}, a_{88}, a_{87}

Table 10. Collision differential path 2 of 3-Pass HAVAL for the 4-collision attack

Step	m'_{i-1}	Δm_i	Output for M'_1
21	m_{20}	2^{27}	$a_{21}[28]$, a_{20}, a_{19}, a_{18}, a_{17}, a_{16}, a_{15}, a_{14}
22	m_{21}		a_{22}, $a_{21}[28]$, a_{20}, a_{19}, a_{18}, $a_{17}[29]$, a_{16}, a_{15}
23	m_{22}		a_{23}, a_{22}, $a_{21}[28]$, a_{20}, a_{19}, a_{18}, $a_{17}[29]$, a_{16}
24	m_{23}		a_{24}, a_{23}, a_{22}, $a_{21}[28]$, $a_{20}[28]$, a_{19}, a_{18}, a_{17}
25	m_{24}		a_{25}, a_{24}, a_{23}, a_{22}, $a_{21}[28]$, a_{20}, a_{19}, a_{18}
26	m_{25}		a_{26}, a_{25}, a_{24}, a_{23}, a_{22}, $a_{21}[28]$, a_{20}, a_{19}
27	m_{26}		a_{27}, a_{26}, a_{25}, a_{24}, a_{23}, a_{22}, $a_{21}[28]$, a_{20}
28	m_{27}		a_{28}, a_{27}, a_{26}, a_{25}, a_{24}, a_{23}, a_{22}, $a_{21}[28]$
29	m_{28}		$a_{29}[-17,-18,19]$, a_{28}, a_{27}, a_{26}, a_{25}, a_{24}, a_{23}, a_{22}
30	m_{29}		a_{30}, $a_{29}[-17,-18,19]$, a_{28}, a_{27}, a_{26}, a_{25}, a_{24}, a_{23}
31	m_{30}		$a_{31}[-12,-13,-14,15]$, a_{30}, $a_{29}[-17,-18,19]$, a_{28}, a_{27}, a_{26}, a_{25}, a_{24}
32	m_{31}		a_{32}, $a_{31}[-12,-13,-14,15]$, a_{30}, $a_{29}[-17,-18,19]$, a_{28}, a_{27}, a_{26}, a_{25}
33	m_5		a_{33}, a_{32}, $a_{31}[-12,-13,-14,15]$, a_{30}, $a_{29}[-17,-18,19]$, a_{28}, a_{27}, a_{26}
34	m_{14}		a_{34}, a_{33}, a_{32}, $a_{31}[-12,-13,-14,15]$, a_{30}, $a_{29}[-17,-18,19]$, a_{28}, a_{27}
35	m_{26}		$a_{35}[7]$, a_{34}, a_{33}, a_{32}, $a_{31}[-12,-13,-14,15]$, a_{30}, $a_{29}[-17,-18,19]$, a_{28}
36	m_{18}		$a_{36}[-8]$, $a_{35}[7]$, a_{34}, a_{33}, a_{32}, $a_{31}[-12,-13,-14,15]$, a_{30}, $a_{29}[-17,-18,19]$
37	m_{11}		a_{37}, $a_{36}[-8]$, $a_{35}[7]$, a_{34}, a_{33}, a_{32}, $a_{31}[-12,-13,-14,15]$, a_{30}
38	m_{28}		a_{38}, a_{37}, $a_{36}[-8]$, $a_{35}[7]$, a_{34}, a_{33}, a_{32}, $a_{31}[-12,-13,-14,15]$
39	m_7		a_{39}, a_{38}, a_{37}, $a_{36}[-8]$, $a_{35}[7]$, a_{34}, a_{33}, a_{32}
40	m_{16}		a_{40}, a_{39}, a_{38}, a_{37}, $a_{36}[-8]$, $a_{35}[7]$, a_{34}, a_{33}
41	m_0		a_{41}, a_{40}, a_{39}, a_{38}, a_{37}, $a_{36}[-8]$, $a_{35}[7]$, a_{34}
42	m_{23}		a_{42}, a_{41}, a_{40}, a_{39}, a_{38}, a_{37}, $a_{36}[-8]$, $a_{35}[7]$
43	m_{20}	2^{27}	a_{43}, a_{42}, a_{41}, a_{40}, a_{39}, a_{38}, a_{37}, $a_{36}[-8]$
44	m_{22}		$a_{44}[-29]$, a_{43}, a_{42}, a_{41}, a_{40}, a_{39}, a_{38}, a_{37}
45	m_1		a_{45}, $a_{44}[-29]$, a_{43}, a_{42}, a_{41}, a_{40}, a_{39}, a_{38}
46	m_{10}		a_{46}, a_{45}, $a_{44}[-29]$, a_{43}, a_{42}, a_{41}, a_{40}, a_{39}
47	m_4		a_{47}, a_{46}, a_{45}, $a_{44}[-29]$, a_{43}, a_{42}, a_{41}, a_{40}
48	m_8		a_{48}, a_{47}, a_{46}, a_{45}, $a_{44}[-29]$, a_{43}, a_{42}, a_{41}
49	m_{30}		a_{49}, a_{48}, a_{47}, a_{46}, a_{45}, $a_{44}[-29]$, a_{43}, a_{42}
50	m_3		a_{50}, a_{49}, a_{48}, a_{47}, a_{46}, a_{45}, $a_{44}[-29]$, a_{43}
51	m_{21}		a_{51}, a_{50}, a_{49}, a_{48}, a_{47}, a_{46}, a_{45}, $a_{44}[-29]$
52	m_9		$a_{52}[-18]$, a_{51}, a_{50}, a_{49}, a_{48}, a_{47}, a_{46}, a_{45}
53	m_{17}		a_{53}, $a_{52}[-18]$, a_{51}, a_{50}, a_{49}, a_{48}, a_{47}, a_{46}
54	m_{24}		a_{54}, a_{53}, $a_{52}[-18]$, a_{51}, a_{50}, a_{49}, a_{48}, a_{47}
55	m_{29}		a_{55}, a_{54}, a_{53}, $a_{52}[-18]$, a_{51}, a_{50}, a_{49}, a_{48}
56	m_6		a_{56}, a_{55}, a_{54}, a_{53}, $a_{52}[-18]$, a_{51}, a_{50}, a_{49}
57	m_{19}		a_{57}, a_{56}, a_{55}, a_{54}, a_{53}, $a_{52}[-18]$, a_{51}, a_{50}
58	m_{12}		a_{58}, a_{57}, a_{56}, a_{55}, a_{54}, a_{53}, $a_{52}[-18]$, a_{51}
59	m_{15}		a_{59}, a_{58}, a_{57}, a_{56}, a_{55}, a_{54}, a_{53}, $a_{52}[-18]$
60	m_{13}		$a_{60}[-7]$, a_{59}, a_{58}, a_{57}, a_{56}, a_{55}, a_{54}, a_{53}
61	m_2		a_{61}, $a_{60}[-7]$, a_{59}, a_{58}, a_{57}, a_{56}, a_{55}, a_{54}
62	m_{25}		a_{62}, m_{61}, $a_{60}[-7]$, a_{59}, a_{58}, a_{57}, a_{56}, a_{55}
63	m_{31}		a_{63}, a_{62}, a_{61}, $a_{60}[-7]$, a_{59}, a_{58}, a_{57}, a_{56}
64	m_{27}		a_{64}, a_{63}, a_{62}, a_{61}, $a_{60}[-7]$, a_{59}, a_{58}, a_{57}
65	m_{19}		a_{65}, a_{64}, a_{63}, a_{62}, a_{61}, $a_{60}[-7]$, a_{59}, a_{58}
66	m_9		a_{66}, a_{65}, a_{64}, a_{63}, a_{62}, a_{61}, $a_{60}[-7]$, a_{59}
67	m_4		a_{67}, a_{66}, a_{65}, a_{64}, a_{63}, a_{62}, a_{61}, $a_{60}[-7]$
68	m_{20}	2^{27}	a_{68}, a_{67}, a_{66}, a_{65}, a_{64}, a_{63}, a_{62}, a_{61}

Table 11. Sufficient conditions for the differential paths given in Table 9 and 10

a_i	Conditions of the chaining variable in each step
$a_1 - a_4$	$a_{1,4} = 1$, $a_{1,5} = 0$, $a_{1,6} = 0$, $a_{1,7} = 0$, $a_{1,8} = 0$, $a_{1,11} = 0$, $a_{2,11} = 0$, $a_{2,29} = 1$, $a_{2,30} = 1$, $a_{2,31} = 0$, $a_{2,32} = 0$, $a_{3,4} = 0$, $a_{3,5} = 0$, $a_{3,6} = 0$, $a_{3,7} = 1$, $a_{3,8} = 0$, $a_{3,11} = 1$, $a_{3,22} = 0$, $a_{3,23} = 0$, $a_{4,4} = a_{2,4} + 1$, $a_{4,7} = 1$, $a_{4,11} = 0$, $a_{4,29} = 0$, $a_{4,30} = 0$, $a_{4,31} = 0$, $a_{4,32} = 0$
$a_5 - a_8$	$a_{5,4} = 1$, $a_{5,7} = 0$, $a_{5,8} = 0$, $a_{5,11} = 1$, $a_{5,15} = 0$, $a_{5,22} = 1$, $a_{5,23} = 0$, $a_{5,29} = a_{3,29}$, $a_{5,30} = a_{3,30}$ $a_{6,4} = a_{5,4}$, $a_{6,5} = a_{5,5}$, $a_{6,6} = a_{5,6}$, $a_{6,7} = 0$, $a_{6,8} = 0$, $a_{6,22} = 0$, $a_{6,29} = 1$, $a_{6,30} = 1$, $a_{7,1} = 0$, $a_{7,4} = 1$, $a_{7,5} = 1$, $a_{7,6} = 1$, $a_{7,7} = 1$, $a_{7,8} = 0$, $a_{7,11} = 0$, $a_{7,15} = 0$, $a_{7,22} = 0$, $a_{7,29} = 1$, $a_{7,30} = a_{6,30}$, $a_{7,31} = a_{6,31}$, $a_{7,32} = a_{6,32}$, $a_{8,4} = 0$, $a_{8,5} = 0$, $a_{8,6} = 0$, $a_{8,7} = 0$, $a_{8,8} = 0$, $a_{8,22} = 0$, $a_{8,23} = a_{7,23}$, $a_{8,29} = 1$, $a_{8,30} = 1$, $a_{8,31} = 1$, $a_{8,32} = 0$
$a_9 - a_{12}$	$a_{9,1} = 0$, $a_{9,4} = 1$, $a_{9,5} = 1$, $a_{9,6} = 1$, $a_{9,7} = 0$, $a_{9,8} = 1$, $a_{9,22} = 1$, $a_{9,23} = 1$, $a_{9,29} = 0$, $a_{9,30} = 0$, $a_{9,31} = 0$, $a_{9,32} = 0$, $a_{10,7} = a_{7,7}$, $a_{10,8} = 0$, $a_{10,15} = a_{9,15}$, $a_{10,18} = 1$, $a_{10,22} = 0$, $a_{10,23} = 0$, $a_{10,29} = 1$, $a_{10,30} = 1$, $a_{10,31} = 1$, $a_{10,32} = 1$, $a_{11,4} = 0$, $a_{11,5} = 0$, $a_{11,6} = 0$, $a_{11,7} = 0$, $a_{11,8} = 1$, $a_{11,12} = 0$, $a_{11,15} = 1$, $a_{11,22} = 1$, $a_{11,23} = 1$, $a_{12,1} = a_{11,1}$, $a_{12,15} = 0$, $a_{12,18} = 0$, $a_{12,29} = 0$, $a_{12,30} = 0$, $a_{12,31} = 0$, $a_{12,32} = 0$
$a_{13} - a_{16}$	$a_{13,1} = 0$, $a_{13,4} = 0$, $a_{13,5} = 0$, $a_{13,6} = 0$, $a_{13,7} = 0$, $a_{13,8} = 0$, $a_{13,12} = 0$, $a_{13,15} = 1$, $a_{13,22} = 0$, $a_{13,23} = 0$, $a_{14,1} = 0$, $a_{14,18} = 0$, $a_{14,29} = 0$, $a_{14,30} = 0$, $a_{14,31} = 0$, $a_{14,32} = 1$, $a_{15,1} = 1$, $a_{15,15} = 0$, $a_{15,18} = 0$, $a_{15,22} = 0$, $a_{15,23} = 0$, $a_{15,28} = 0$ $a_{16,12} = a_{15,12}$, $a_{16,18} = 0$
$a_{17} - a_{20}$	$a_{17,1} = 0$, $a_{17,12} = 1$, $a_{17,15} = 0$, $a_{17,18} = 0$, $a_{17,22} = 0$, $a_{17,28} = 0$, $a_{18,7} = 0$, $a_{18,12} = 0$, $a_{18,18} = 1$, $a_{19,1} = 0$, $a_{19,2} = 0$, $a_{19,3} = 0$, $a_{19,4} = 0$, $a_{19,12} = 1$, $a_{20,7} = 0$, $a_{20,18} = 0$, $a_{20,22} = a_{19,22}$, $a_{20,28} = a_{19,28}$
$a_{21} - a_{24}$	$a_{21,1} = 0$, $a_{21,2} = 0$, $a_{21,3} = 0$, $a_{21,4} = 0$, $a_{21,12} = 0$, $a_{21,22} = 0$, $a_{21,28} = 0$, $a_{22,18} = 0$, $a_{22,22} = 0$, $a_{22,28} = 0$, $a_{23,7} = a_{22,7}$, $a_{23,11} = 0$, $a_{23,12} = 0$, $a_{23,22} = 1$, $a_{23,28} = 1$, $a_{23,17} = 0$, $a_{23,18} = 0$, $a_{23,19} = 0$ $a_{24,1} = a_{23,1}$, $a_{24,2} = a_{23,2}$, $a_{24,3} = a_{23,3}$, $a_{24,4} = a_{23,4}$, $a_{24,7} = 0$
$a_{25} - a_{28}$	$a_{25,1} = 0$, $a_{25,2} = 0$, $a_{25,3} = 0$, $a_{25,4} = 1$, $a_{25,7} = 0$, $a_{25,11} = 0$, $a_{25,12} = 0$, $a_{25,13} = 0$, $a_{25,14} = 0$, $a_{25,15} = 0$, $a_{25,17} = 0$, $a_{25,18} = 0$, $a_{25,19} = 1$, $a_{25,22} = 0$, $a_{25,28} = 0$, $a_{26,4} = 0$, $a_{26,7} = 1$, $a_{26,19} = a_{24,19}$, $a_{26,1} = 0$, $a_{26,2} = 0$, $a_{26,3} = 0$, $a_{27,1} = 1$, $a_{27,2} = 1$, $a_{27,3} = 1$, $a_{27,4} = 1$, $a_{27,11} = 0$, $a_{27,15} = 1$, $a_{27,17} = 0$, $a_{27,18} = 0$, $a_{27,19} = 0$, $a_{27,22} = 0$, $a_{27,28} = 0$, $a_{28,7} = 0$, $a_{28,11} = 0$, $a_{28,12} = 1$, $a_{28,13} = 1$, $a_{28,14} = 0$, $a_{28,15} = 1$, $a_{28,17} = 0$, $a_{28,18} = 0$, $a_{28,19} = 0$, $a_{28,22} = 0$
$a_{29} - a_{32}$	$a_{29,1} = 0$, $a_{29,2} = 0$, $a_{29,3} = 0$, $a_{29,4} = 0$, $a_{29,11} = 0$, $a_{29,12} = 0$, $a_{29,13} = 0$, $a_{29,14} = 0$, $a_{29,15} = 0$, $a_{29,17} = 1$, $a_{29,18} = 1$, $a_{29,19} = 0$, $a_{29,22} = 0$, $a_{30,7} = 0$, $a_{30,11} = 0$, $a_{30,12} = 0$, $a_{30,13} = 0$, $a_{30,14} = 0$, $a_{30,15} = 1$, $a_{30,22} = 1$, $a_{30,17} = 1$, $a_{30,18} = 1$, $a_{30,19} = 1$, $a_{31,1} = 0$, $a_{31,2} = 0$, $a_{31,3} = 1$, $a_{31,4} = 0$, $a_{31,11} = 0$, $a_{31,12} = 1$, $a_{31,13} = 1$, $a_{31,14} = 1$, $a_{31,15} = 0$, $a_{31,17} = 0$, $a_{31,18} = 0$, $a_{31,19} = 0$, $a_{31,22} = 0$, $a_{32,7} = 1$, $a_{32,8} = 0$, $a_{32,11} = 0$, $a_{32,12} = 1$, $a_{32,13} = 1$, $a_{32,14} = 1$, $a_{32,15} = 1$, $a_{32,17} = 1$, $a_{32,18} = 1$, $a_{32,19} = 1$, $a_{32,22} = 0$
$a_{33} - a_{36}$	$a_{33,7} = 0$, $a_{33,8} = 0$, $a_{33,11} = 1$, $a_{33,12} = 0$, $a_{33,13} = 0$, $a_{33,14} = 1$, $a_{33,15} = 0$, $a_{33,17} = 1$, $a_{33,18} = 1$, $a_{33,19} = 1$, $a_{33,22} = 1$, $a_{34,22} = 1$, $a_{34,12} = 1$, $a_{34,13} = 0$, $a_{34,14} = 1$, $a_{34,15} = 0$, $a_{34,7} = 0$, $a_{34,8} = 0$ $a_{35,11} = a_{34,11} + 1$, $a_{35,22} = 0$, $a_{35,12} = 1$, $a_{35,15} = 0$, $a_{35,7} = 1$, $a_{35,8} = 0$, $a_{36,7} = 1$, $a_{36,8} = 1$, $a_{36,22} = 1$, $a_{36,14} = a_{35,14}$
$a_{37} - a_{40}$	$a_{37,7} = 0$, $a_{37,22} = 1$, $a_{37,8} = 1$, $a_{38,7} = 1$, $a_{38,8} = 0$, $a_{39,7} = 1$, $a_{39,8} = 1$, $a_{39,29} = 0$, $a_{40,8} = 1$
$a_{41} - a_{48}$	$a_{44,29} = 1$, $a_{43,29} = 0$, $a_{42,29} = 0$, $a_{41,29} = 1$, $a_{45,29} = 1$, $a_{46,29} = 0$, $a_{47,29} = 1$, $a_{48,29} = 1$
$a_{49} - a_{56}$	$a_{52,18} = 1$, $a_{51,18} = 0$, $a_{50,18} = 0$, $a_{49,18} = 1$, $a_{47,18} = 0$, $a_{53,18} = 1$, $a_{54,18} = 0$, $a_{55,18} = 1$, $a_{56,18} = 1$
$a_{82} - a_{89}$	$a_{82,11} = 1$, $a_{83,11} = 0$, $a_{84,11} = 0$, $a_{85,11} = 1$, $a_{86,11} = 0$, $a_{87,11} = 0$, $a_{89,11} = 0$

Table 12. The near-collision differential path for the 3-Pass HAVAL

Step	m'_{i-1}	Outputs for M'_0	Sufficient conditions
6	m'_5	$a_6[-12,13]$, a_5, a_4, a_3, a_2, a_1, a_0, b_0	$a_{6,12}=1$, $a_{6,13}=0$
7	m_6	a_7, $a_6[-12,13]$, a_5, a_4, a_3, a_2, a_1, a_0	$a_{0,12}=0$, $a_{0,13}=0$
8	m_7	a_8, a_7, $a_6[-12,13]$, a_5, a_4, a_3, a_2, a_1	$a_{2,12}=0$, $a_{2,13}=0$
9	m_8	a_9, a_8, a_7, $a_6[-12,13]$, a_5, a_4, a_3, a_2	$a_{5,12}=a_{4,12}$, $a_{5,13}=a_{4,13}$
10	m_9	a_{10}, a_9, a_8, a_7, $a_6[-12,13]$, a_5, a_4, a_3	$a_{7,12}=0$, $a_{7,13}=0$
11	m_{10}	$a_{11}[-6,-7,-8,9]$, a_{10}, a_9, a_8, a_7, $a_6[-12,13]$, a_5, a_4	$a_{8,12}=1$, $a_{8,13}=0$, $a_{5,13}=0$, $a_{11,6}=1$, $a_{11,7}=1$, $a_{11,8}=1$, $a_{11,9}=0$
12	m_{11}	a_{12}, $a_{11}[-6,-7,-8,9]$, a_{10}, a_9, a_8, a_7, $a_6[-12,13]$, a_5	$a_{10,12}=0$, $a_{10,13}=0$, $a_{6,6}=0$, $a_{6,7}=0$, $a_{6,8}=0$, $a_{6,9}=0$
13	m_{12}	a_{13}, a_{12}, $a_{11}[-6,-7,-8,9]$, a_{10}, a_9, a_8, a_7, $a_6[-12,13]$	$a_{12,12}=0$, $a_{12,13}=0$, $a_{7,6}=0$, $a_{7,7}=0$, $a_{7,8}=0$, $a_{7,9}=0$
14	m_{13}	a_{14}, a_{13}, a_{12}, $a_{11}[-6,-7,-8,9]$, a_{10}, a_9, a_8, a_7	$a_{10,6}=a_{9,6}$, $a_{10,7}=a_{9,7}$, $a_{10,8}=a_{9,8}+1$, $a_{10,9}=a_{9,9}$, $a_{9,8}=0$
15	m_{14}	a_{15}, a_{14}, a_{13}, a_{12}, $a_{11}[-6,-7,-8,9]$, a_{10}, a_9, a_8	$a_{12,6}=0$, $a_{12,7}=0$,$a_{12,8}=0$, $a_{12,9}=0$
16	m_{15}	a_{16}, a_{15}, a_{14}, a_{13}, a_{12}, $a_{11}[-6,-7,-8,9]$, a_{10}, a_9	$a_{13,6}=1$, $a_{13,7}=1$,$a_{13,8}=1$, $a_{13,9}=1$
17	m_{16}	$a_{17}[-2]$, a_{16}, a_{15}, a_{14}, a_{13}, a_{12}, $a_{11}[-6,-7,-8,9]$, a_{10}	$a_{15,6}=0$, $a_{15,7}=0$, $a_{15,8}=0$, $a_{15,9}=1$, $a_{10,9}=0$, $a_{14,9}=1$, $a_{17,2}=1$
18	m_{17}	a_{18}, $a_{17}[-2]$, a_{16}, a_{15}, a_{14}, a_{13}, a_{12}, $a_{11}[-6,-7,-8,9]$	$a_{17,6}=0$, $a_{17,7}=0$, $a_{17,8}=0$, $a_{17,9}=0$, $a_{11,2}=0$
19	m_{18}	a_{19}, a_{18}, $a_{17}[-2]$, a_{16}, a_{15}, a_{14}, a_{13}, a_{12}	$a_{13,2}=1$, $a_{14,2}=0$, $a_{15,2}=0$
20	m_{19}	a_{20}, a_{19}, a_{18}, $a_{17}[-2]$, a_{16}, a_{15}, a_{14}, a_{13}	$a_{16,2}=a_{15,2}$
21	m_{20}	a_{21}, a_{20}, a_{19}, a_{18}, $a_{17}[-2]$, a_{16}, a_{15}, a_{14}	$a_{18,2}=0$
22	m_{21}	a_{22}, a_{21}, a_{20}, a_{19}, a_{18}, $a_{17}[-2]$, $a_{16}[$, a_{15}	$a_{19,2}=1$
23	m_{22}	a_{23}, a_{22}, a_{21}, a_{20}, a_{19}, a_{18}, $a_{17}[-2]$, a_{16}	$a_{21,2}=0$
24	m_{23}	a_{24}, a_{23}, a_{22}, a_{21}, a_{20}, a_{19}, a_{18}, $a_{17}[-2]$	$a_{23,2}=0$
25	m_{24}	$a_{25}[-23]$, a_{24}, a_{23}, a_{22}, a_{21}, a_{20}, a_{19}, a_{18}	$a_{25,23}=1$
26	m_{25}	a_{26}, $a_{25}[-23]$, a_{24}, a_{23}, a_{22}, a_{21}, a_{20}, a_{19}	$a_{19,23}=0$
27	m_{26}	a_{27}, a_{26}, $a_{25}[-23]$, a_{24}, a_{23}, a_{22}, a_{21}, a_{20}	$a_{21,23}=0$
28	m_{27}	a_{28}, a_{27}, a_{26}, $a_{25}[-23]$, a_{24}, a_{23}, a_{22}, a_{21}	$a_{24,23}=a_{23,23}$
29	m_{28}	a_{29}, a_{28}, a_{27}, a_{26}, $a_{25}[-23]$, a_{24}, a_{23}, a_{22}	$a_{26,23}=0$
30	m_{29}	a_{30}, a_{29}, a_{28}, a_{27}, a_{26}, $a_{25}[-23]$, a_{24}, a_{23}	$a_{27,23}=1$
31	m_{30}	a_{31}, a_{30}, a_{29}, a_{28}, a_{27}, a_{26}, $a_{25}[-23]$, a_{24}	$a_{29,23}=0$
32	m_{31}	a_{32}, a_{31}, a_{30}, a_{29}, a_{28}, a_{27}, a_{26}, $a_{25}[-23]$	$a_{31,23}=0$
33	m'_5	a_{33}, a_{32}, a_{31}, a_{30}, a_{29}, a_{28}, a_{27}, a_{26}	
...	...	...	...
95	m'_5	$a_{95}[12]$, a_{94}, a_{93}, a_{92}, a_{91}, a_{90}, a_{89}, a_{88}	$a_{95,12}=0$
96	m_2	a_{96}, $a_{95}[12]$, a_{94}, a_{93}, a_{92}, a_{91}, a_{90}, a_{89}	$a_{92,12}=1$

Table 13. A set of sufficient conditions for the 8-near-collision of 3-Pass HAVAL

Step	Output variable	Varialbe conditions
0	IV	$a_{0,12}=0$, $a_{0,13}=0$, $a_{0,20}=0$, $a_{0,21}=0$, $a_{0,28}=0$, $a_{0,29}=0$
1	a_1	
2	a_2	$a_{2,12}=0$, $a_{2,13}=0$, $a_{2,20}=0$, $a_{2,21}=0$, $a_{2,28}=0$, $a_{2,29}=0$
3	a_3	
4	a_4	$a_{4,13}=0$, $a_{4,21}=0$, $a_{4,29}=0$
5	a_5	$a_{5,6}=0$, $a_{5,7}=0$, $a_{5,8}=0$, $a_{5,9}=0$, $a_{5,12}=a_{4,12}$, $a_{5,13}=a_{4,13}$, $a_{5,14}=0$, $a_{5,15}=0$, $a_{5,16}=0$, $a_{5,17}=0$, $a_{5,20}=a_{4,20}$, $a_{5,21}=a_{4,21}$, $a_{5,22}=0$, $a_{5,23}=0$, $a_{5,24}=0$, $a_{5,25}=0$, $a_{5,28}=a_{4,28}$, $a_{5,29}=a_{4,29}$
6	a_6	$a_{6,12}=1$, $a_{6,13}=0$, $a_{6,20}=1$, $a_{6,21}=0$, $a_{6,28}=1$, $a_{6,29}=0$
7	a_7	$a_{7,6}=0$, $a_{7,7}=0$, $a_{7,8}=0$, $a_{7,9}=0$, $a_{7,12}=0$, $a_{7,13}=0$, $a_{7,14}=0$, $a_{7,15}=0$, $a_{7,16}=0$, $a_{7,17}=0$, $a_{7,20}=0$, $a_{7,21}=0$, $a_{7,22}=0$, $a_{7,23}=0$, $a_{7,24}=0$, $a_{7,25}=0$, $a_{7,28}=0$, $a_{7,29}=0$
8	a_8	$a_{8,12}=1$, $a_{8,13}=0$, $a_{8,20}=1$, $a_{8,21}=0$, $a_{8,28}=1$, $a_{8,29}=0$
9	a_9	$a_{9,8}=0$, $a_{9,9}=0$, $a_{9,16}=0$, $a_{9,17}=0$, $a_{9,24}=0$, $a_{9,25}=0$
10	a_{10}	$a_{10,6}=a_{9,6}$, $a_{10,7}=a_{9,7}$, $a_{10,8}=a_{9,8}+1$, $a_{10,9}=a_{9,9}$, $a_{10,12}=0$, $a_{10,13}=0$, $a_{10,14}=a_{9,14}$, $a_{10,15}=a_{9,15}$, $a_{10,16}=a_{9,16}+1$, $a_{10,17}=a_{9,17}$, $a_{10,20}=0$, $a_{10,21}=0$, $a_{10,22}=a_{9,22}$, $a_{10,23}=a_{9,23}$, $a_{10,24}=a_{9,24}+1$, $a_{10,25}=a_{9,25}$, $a_{10,28}=0$, $a_{10,29}=0$
11	a_{11}	$a_{11,2}=0$, $a_{11,6}=1$, $a_{11,7}=1$, $a_{11,8}=1$, $a_{11,9}=0$, $a_{11,10}=0$, $a_{11,14}=1$, $a_{11,15}=1$, $a_{11,16}=1$, $a_{11,17}=0$, $a_{11,18}=0$, $a_{11,22}=1$, $a_{11,23}=1$, $a_{11,24}=1$, $a_{11,25}=0$
12	a_{12}	$a_{12,6}=0$, $a_{12,7}=0$, $a_{12,8}=0$, $a_{12,9}=0$, $a_{12,12}=0$, $a_{12,13}=0$, $a_{12,14}=0$, $a_{12,15}=0$, $a_{12,16}=0$, $a_{12,17}=0$, $a_{12,20}=0$, $a_{12,21}=0$, $a_{12,22}=0$, $a_{12,23}=0$, $a_{12,24}=0$, $a_{12,25}=0$, $a_{12,28}=0$, $a_{12,29}=0$
13	a_{13}	$a_{13,2}=1$, $a_{13,6}=1$, $a_{13,7}=1$, $a_{13,8}=1$, $a_{13,9}=1$, $a_{13,10}=1$, $a_{13,14}=1$, $a_{13,15}=1$, $a_{13,16}=1$, $a_{13,17}=1$, $a_{13,18}=1$, $a_{13,22}=1$, $a_{13,23}=1$, $a_{13,24}=1$, $a_{13,25}=1$
14	a_{14}	$a_{14,2}=0$, $a_{14,9}=1$, $a_{14,10}=0$, $a_{14,17}=1$, $a_{14,18}=0$, $a_{14,25}=1$
15	a_{15}	$a_{15,2}=0$, $a_{15,6}=0$, $a_{15,7}=0$, $a_{15,8}=0$, $a_{15,9}=1$, $a_{15,10}=0$, $a_{15,14}=0$, $a_{15,15}=0$, $a_{15,16}=0$, $a_{15,17}=1$, $a_{15,18}=0$, $a_{15,22}=0$, $a_{15,23}=0$, $a_{15,24}=0$, $a_{15,25}=1$
16	a_{16}	$a_{16,2}=a_{15,2}$, $a_{16,10}=a_{15,10}$, $a_{16,18}=a_{15,18}$
17	a_{17}	$a_{17,2}=1$, $a_{17,6}=0$, $a_{17,7}=0$, $a_{17,8}=0$, $a_{17,9}=0$, $a_{17,10}=1$, $a_{17,14}=0$, $a_{17,15}=0$, $a_{17,16}=0$, $a_{17,17}=0$, $a_{17,18}=1$, $a_{17,22}=0$, $a_{17,23}=0$, $a_{17,24}=0$, $a_{17,25}=0$
18	a_{18}	$a_{18,2}=0$, $a_{18,10}=0$, $a_{18,18}=0$
19	a_{19}	$a_{19,2}=1$, $a_{19,7}=0$, $a_{19,10}=1$, $a_{19,18}=1$, $a_{19,23}=0$, $a_{19,31}=0$
20	a_{20}	
21	a_{21}	$a_{21,2}=0$, $a_{21,7}=0$, $a_{21,10}=0$, $a_{21,18}=0$, $a_{21,23}=0$, $a_{21,31}=0$
22	a_{22}	
23	a_{23}	$a_{23,2}=0$, $a_{23,10}=0$, $a_{23,18}=0$
24	a_{24}	$a_{24,7}=a_{23,7}$, $a_{24,23}=a_{23,23}$, $a_{24,31}=a_{23,31}$
25	a_{25}	$a_{25,7}=1$, $a_{25,23}=1$, $a_{25,31}=1$
26	a_{26}	$a_{26,7}=0$, $a_{26,23}=0$, $a_{26,31}=0$
27	a_{27}	$a_{27,7}=1$, $a_{27,23}=1$, $a_{27,31}=1$
28	a_{28}	
29	a_{29}	$a_{29,7}=0$, $a_{29,23}=0$, $a_{29,31}=0$
30	a_{30}	
31	a_{31}	$a_{31,7}=0$, $a_{31,23}=0$, $a_{31,31}=0$
92	a_{92}	$a_{92,12}=1$, $a_{92,20}=1$, $a_{92,28}=1$
95	a_{95}	$a_{95,12}=0$, $a_{95,20}=0$, $a_{95,28}=0$
		$(b_0+a_{95})_{12}=0$, $(b_0+a_{95})_{20}=0$, $(b_0+a_{95})_{28}=0$

Differential Cryptanalysis of T-Function Based Stream Cipher TSC-4

Haina Zhang[1] and Xiaoyun Wang[2,⋆]

[1] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan 250100, P.R. China
hnzhang@math.sdu.edu.cn
[2] Tsinghua University, Beijing 100087, P.R. China
xiaoyunwang@tsinghua.edu.cn

Abstract. TSC-4 is a T-function based stream cipher with 80-bit key, and proposed as a candidate for ECRYPT eStream project. In this paper, we introduce a differential method to analyze TSC-4. Our attack is based on the vulnerable differential characteristics in the state initialization of TSC-4, and for the chosen IV pairs, the differential probability is up to $2^{-15.40}$ in the case of weak keys. We show that there are about 2^{72} weak keys among the total 2^{80} keys. To recover 8 bits of a weak key needs about $2^{40.53}$ chosen IV pairs. After that, we can search the other 72 key bits by an exhaustive attack.

Keywords: Differential cryptanalysis, T-function, stream cipher, chosen IV attack, TSC-4.

1 Introduction

For a long time, a classical approach for designing stream ciphers is to utilize the Linear Feedback Shift Registers (LFSR) combined with nonlinear Boolean functions. Recently, a new primitive has been introduced to replace the LFSR, which is the T-function (Triangular-function) proposed by Klimov and Shamir [4,5,6]. T-function is a nonlinear function which has some nice properties, such as the operations both available on processors and software implementations, and one single cycle by the appropriate choice, etc. There are some interesting researches on this area in recent years.

At FSE 2005, Hong *et al*. proposed a new class of single T-functions with S-box application [1], and described two stream ciphers TSC-1 and TSC-2. At Asiacrypt 2004, Mitra and Sarkar described a time-memory trade-off attack [8] to break some stream ciphers proposed by Klimov and Shamir. Recently, Kunzli, Junod and Meier proposed the distinguishing attacks [7] applicable to several T-function based stream ciphers (TSC). Based on the above researches, Hong *et al*. proposed a new TSC version called TSC-3 [2], as a candidate to eStream project.

⋆ Supported by the National Natural Science Foundation of China (NSFC Grant No.90604036) and 973 Project (No.2007CB807902).

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 227–238, 2007.

In 2005 and 2006, Muller and Peyrin [10,11] introduced a linear cryptanalysis attack on the TSC family TSC-1, TSC-2, TSC-3 together with Klimov and Shamir's ciphers. This linear correlation attack can recover the full secret keys of these stream ciphers. Soon, Dukjae Moon *et al*. twisted TSC-3 and proposed the latest version TSC-4 [9] to prevent the distinguishing attack, the time-memory trade-off attack and the linear correlation attack described in above literatures.

At Indocrypt 2006, Fischer, Meier and Berbain *et al*. [3] presented a non-randomness behavior of the full eight-round state initialization for TSC-4. The non-randomness can be detected in the initial state with about 1000 inputs, however, no bias in the keystream of TSC-4 resulting from this non-randomness has been detected yet. So, no substantial attack has been found from the keystream until now.

In this paper, we present an efficient chosen IV differential attack on TSC-4 with weak keys based on two special differential characteristics in the state initialization. The outline of our attack is as follows. Firstly, construct two special differential characteristics in the state initialization of TSC-4. Secondly, investigate what keys and IVs can result in the high occurrence probability of two differential characteristics, and find out the serious non-randomness behavior of the initial state under these weak keys and chosen IVs. Finally, utilizing $2^{40.53}$ chosen IV pairs, we can identify a weak key and recover 8 bits of the key, and recover the full 80-bit weak key by searching exhaustively the other 72 key bits.

This paper is organized as follows. In Section 2, a brief description of TSC-4 is given. Section 3 introduces two special differential characteristics according to the structure of TSC-4. Section 4 presents the chosen IV differential attack on TSC-4. Finally, we conclude the paper in Section 5.

2 A Brief Description of TSC-4

TSC-4 is a synchronous stream cipher optimized for constrained hardware, including the cipher body and the state initialization. The structure of TSC-4 is illustrated in Fig. 1. Its internal state consists of two states X and Y of 4×32 bits each, denoted $X = (x_3, x_2, x_1, x_0)^T$ and $Y = (y_3, y_2, y_1, y_0)^T$, where x_k and y_k $(k = 0, \cdots, 3)$ are 32-bit words. Let $[x]_i$ denote the i-th least significant bit of a 32-bit word x, then the i-th bit-slice of state X and the i-th bit-slice of Y are defined as

$$[X]_i = \sum_{k=0}^{3} [x_k]_i 2^k, \quad [Y]_i = \sum_{k=0}^{3} [y_k]_i 2^k, \quad i = 0, \cdots 31. \tag{1}$$

2.1 Cipher Body

We first describe two functions of X and Y. In the case of state X, a 32-bit parameter $\alpha_1(X)$ is computed as a function of X. It is defined by $\alpha_1(X) = (p + c) \oplus p \oplus 2s$ with $p = x_0 \wedge x_1 \wedge x_2 \wedge x_3$ and $s = x_0 + x_1 + x_2 + x_3$ and constant c_X =0x51291089. Here $\wedge$, $\oplus$ and $+$ denote bitwise AND, bitwise XOR

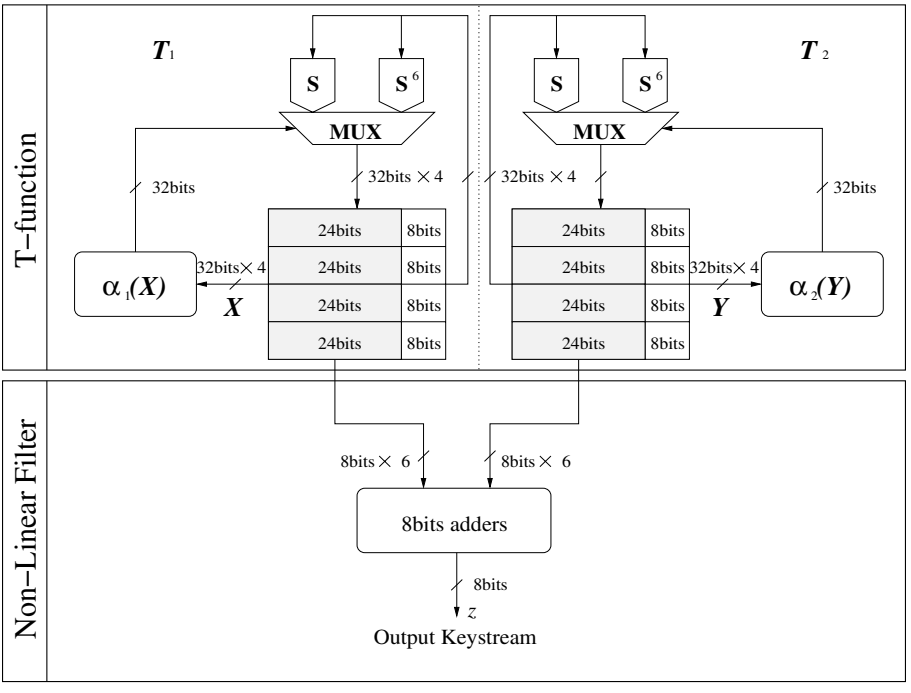


Fig. 1. The structure of TSC-4

and addition operation modulo 2^{32} respectively. The 32-bit parameter $\alpha_2(Y)$ is defined similarly to $\alpha_1(X)$ except for the constant c_Y=0x12910895.

In the cipher body, bit-slices $[X]_i$ and $[Y]_i$ are always mapped by two single-cycle S-boxes. The S-boxes are defined as

$$S[16] = \{9, 2, 11, 15, 3, 0, 14, 4, 10, 13, 12, 5, 6, 8, 7, 1\},$$

$$S^6[16] = \{6, 13, 8, 0, 5, 12, 1, 11, 4, 14, 3, 10, 15, 7, 2, 9\}.$$

Now T-functions T_1 and T_2 on input states X and Y are defined as

$$[T_1(X)]_i = \begin{cases} S([X]_i) & \text{if } [\alpha_1(X)]_i = 1, \\ S^6([X]_i) & \text{if } [\alpha_1(X)]_i = 0. \end{cases}$$

$$[T_2(Y)]_i = \begin{cases} S([Y]_i) & \text{if } [\alpha_2(Y)]_i = 1, \\ S^6([Y]_i) & \text{if } [\alpha_2(Y)]_i = 0. \end{cases}$$

Where the bit-slices $[T_1(X)]_i$ and $[T_2(Y)]_i (i = 0, 1, \cdots, 31)$ are regarded as 4-bit integers similar to equations (1). Then $[X]_i$ will be replaced by $[T_1(X)]_i$, and $[Y]_i$ will be replaced by $[T_2(Y)]_i$ respectively, in this way, the states X and Y are updated by T-function T_1 and T_2.

The non-linear filter in Fig. 1. produces the actual output keystream from the current internal states X and Y. We divide 32-bit word x_k into four bytes

$x_{k,0}, x_{k,1}, x_{k,2}, x_{k,3}$, where $x_{k,0}$ is the least significant byte, and $x_{k,3}$ is the most significant byte. This is similar to $y_{k,l}$ $(0 \leq k, l \leq 3)$. Then we compute six 8-bit temporary variables $a_0, \cdots, a_5$ as follows:

$$\begin{aligned} a_0 &= x_{3,3} + y_{1,1}, \\ a_1 &= x_{0,3} + y_{2,1}, \\ a_2 &= x_{2,2} + y_{3,2}, \\ a_3 &= x_{1,2} + y_{0,2}, \\ a_4 &= x_{3,1} + y_{2,3}, \\ a_5 &= x_{0,1} + y_{1,3}, \end{aligned} \tag{2}$$

where, + denotes the addition operation modulo 2^8. Now the 8-bit keystream z is defined as

$$z = a_0 \oplus (a_1)_{\ggg 5} \oplus (a_2)_{\ggg 2} \oplus (a_3)_{\ggg 5} \oplus (a_4)_{\ggg 6} \oplus (a_5)_{\ggg 2}, \tag{3}$$

where $\ggg$ denotes the right rotation.

2.2 State Initialization

We now describe how the states X and Y are initialized from a given key and IV.

Key/IV Loading: Let $K = (k_{79}, k_{78}, \cdots, k_1, k_0)$ and IV=(iv_{79}, iv_{78}, $\cdots$, iv_1, iv_0) be an 80-bit key and 80-bit IV respectively. Two 128-bit internal states X and Y are initialized as follows:

$$X = \begin{pmatrix} x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix} = \begin{pmatrix} iv_{63} & iv_{62} & \cdots & iv_{33} & iv_{32} \\ iv_{31} & iv_{30} & \cdots & iv_1 & iv_0 \\ k_{63} & k_{62} & \cdots & k_{33} & k_{32} \\ k_{31} & k_{30} & \cdots & k_1 & k_0 \end{pmatrix},$$

$$Y = \begin{pmatrix} y_3 \\ y_2 \\ y_1 \\ y_0 \end{pmatrix} = \begin{pmatrix} k_{47} & k_{46} & \cdots & \cdots & k_{17} & k_{16} \\ k_{15} & \cdots & k_0 & k_{79} & \cdots & k_{64} \\ iv_{47} & iv_{46} & \cdots & \cdots & iv_{17} & iv_{16} \\ iv_{15} & \cdots & iv_0 & iv_{79} & \cdots & iv_{64} \end{pmatrix}.$$

Warm-up: Once the internal states are initialized, the K and IV are mixed by the following process,

- Step 1, run cipher body once to produce a single 8-bit output,
- Step 2, rotate x_1 and y_0 to the left by 8 bits,
- Step 3, XOR the output z to the least significant 8 bits of x_1 and y_0.

The state initialization is completely finished by repeating the above three steps 8 times.

3 Two Special Differential Characteristics in the State Initialization

In this section, we will construct two special differential characteristics which will occur with obvious probability advantage for a large amount of keys.

Suppose that IV and IV' are two different IVs, and we denote them as an IV pair (IV, IV'). Let $X = (x_3, x_2, x_1, x_0)^T$ and $Y = (y_3, y_2, y_1, y_0)^T$ are two 128-bit initial states corresponding to IV. $X' = (x'_3, x'_2, x'_1, x'_0)^T$ and $Y' = (y'_3, y'_2, y'_1, y'_0)^T$ are states related to IV'. Denote the differential $X \oplus X'$ as $\Delta X = (\Delta x_3, \Delta x_2, \Delta x_1, \Delta x_0)^T$, and $Y \oplus Y'$ as $\Delta Y = (\Delta y_3, \Delta y_2, \Delta y_1, \Delta y_0)^T$, where $\Delta x_k = x_k \oplus x'_k$, $\Delta y_k = y_k \oplus y'_k$, $k = 0, \cdots, 3$.

All the 16 bytes of ΔY can be represented as a matrix $\Delta Y = (\Delta y_{k,l})_{4\times 4}$ with $\Delta y_{k,l} = y_{k,l} \oplus y'_{k,l}$, where $0 \le k, l \le 3$.

$$\Delta Y = \begin{pmatrix} \Delta y_3 \\ \Delta y_2 \\ \Delta y_1 \\ \Delta y_0 \end{pmatrix} = \begin{pmatrix} \Delta y_{3,3} & \Delta y_{3,2} & \Delta y_{3,1} & \Delta y_{3,0} \\ \Delta y_{2,3} & \Delta y_{2,2} & \Delta y_{2,1} & \Delta y_{2,0} \\ \Delta y_{1,3} & \Delta y_{1,2} & \Delta y_{1,1} & \Delta y_{1,0} \\ \Delta y_{0,3} & \Delta y_{0,2} & \Delta y_{0,1} & \Delta y_{0,0} \end{pmatrix}.$$

In the Key/IV loading process, it is obvious that the IV bits $iv_{48}, \cdots, iv_{63}$ (to initial state X) and $iv_{64}, \cdots, iv_{79}$ (to initial state Y) occur only one time respectively, and the other IV bits occur twice. This means that one nonzero bit difference in $\{iv_{64}, iv_{49}, \cdots, iv_{79}\}$ results in only one nonzero difference bit in ΔY.

We denote the IV differential $IV \oplus IV'$ as $\Delta IV = (\Delta iv_{79}, \cdots \Delta iv_0)$ with $\Delta iv_j = iv_j \oplus iv'_j$, where $0 \le j \le 79$. Then we announce for the same key K, the chosen IV pair holds that IV and IV' are only different at one bit $\Delta iv_{j_0} = 1$ and the other 79 bits of them are all the same, where $64 \le j_0 \le 79$. After the key/IV loading process, we can get the initial state of X and Y are $\Delta X^0 = (0, 0, 0, 0)^T$ and $\Delta Y^0 = (0, 0, 0, \Delta y_0^0)^T$ respectively, where Δy_0^0 only has one nonzero bit difference (i.e., the hamming weight of Δy_0^0 is 1). In this paper, we select $j_0 = 66$ i.e., Δy_0^0 =0x4. The choice for the position j_0 depends on the occurrence probability of the expected differential characteristic in the state initialization process, the details are described in Subsection 4.1. The state initialization in Section 2.2 includes 8 warm-up processes, and we denote each warm-up process as one round.

Now we construct two differential characteristics Ω_X and Ω_Y, and discuss the interaction between them.

Differential characteristic Ω_X: Ω_X is an 8-round differential in the state initialization such that the difference of initial state X after Key/IV loading process is $\Delta X^0 = (0, 0, 0, 0)^T$, and the output difference of internal state X after Step 3 at the i-th round is $\Delta X^i = (0, 0, 0, 0)^T$, $i = 1, \cdots, 8$, i.e.,

$$\Omega_X: \; X^0 \xrightarrow[Round\ 1]{Step\ 3} X^1 \xrightarrow[Round\ 2]{Step\ 3} X^2 \longrightarrow \; \ldots \xrightarrow[Round\ 8]{Step\ 3} X^8.$$

Differential characteristic Ω_Y: Ω_Y behaves that, the difference of initial state Y after Key/IV loading process is $\Delta Y^0 = (0,0,0,4)^T$, and the output difference of internal state Y after Step 1 before Step 2 at the i-th round is ΔY^i which holds that 6 differential bytes $\Delta Y^i_{3,2}$, $\Delta Y^i_{2,3}$, $\Delta Y^i_{2,1}$, $\Delta Y^i_{1,3}$, $\Delta Y^i_{1,1}$ and $\Delta Y^i_{0,2}$ are all 0. Here $i = 1, \cdots, 8$.

$$\Omega_Y : \ Y^0 \xrightarrow[Round\ 1]{Step\ 1} \Delta Y^1 \xrightarrow[Round\ 2]{Step\ 1} \Delta Y^2 \longrightarrow \cdots \xrightarrow[Round\ 8]{Step\ 1} \Delta Y^8.$$

The differential characteristic Ω_Y is illustrated in Fig. 2. The blank means that the difference byte is 0.

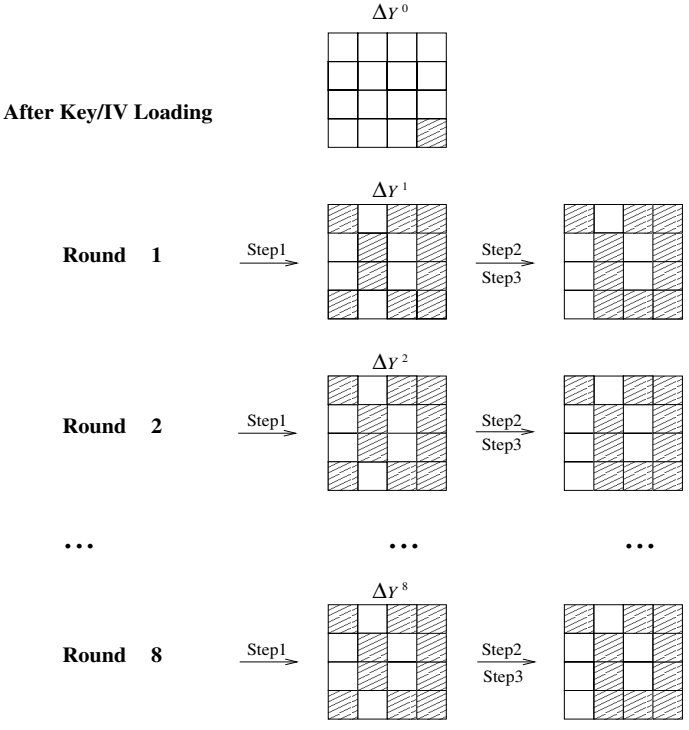


Fig. 2. The differential characteristic Ω_Y in the state initialization

It is obvious that, if IV and IV' with the same key K are different at only one bit $\Delta iv_{66} = 1$, the initial state differential ΔX^0 is $(0,0,0,0)^T$ and ΔY^0 is $(0,0,0,4)^T$. And then, by the equations (2) and (3), it is easy to confirm that, ΔX^0 and the choice of ΔY^1 guarantee the keystream output differential Δz^1_r at the first round is 0. So, from Step 3 of the warm-up process in Subsection 2.2, the difference ΔX^1 is always $(0,0,0,0)^T$ after the first round. Similarly, ΔX^1 and the choice of ΔY^2 guarantee ΔX^2 is always $(0,0,0,0)^T$ after the second round, and so on. This means that, under the condition IV and IV' are different at only one bit $\Delta iv_{66} = 1$, if the differential characteristic Ω_Y holds, then the

differential characteristic Ω_X holds with probability 1. And we can only focus on the occurrence probability of Ω_Y in the next sections.

4 Chosen IV Differential Attack on TSC-4

In this section, we introduce a chosen IV differential attack on TSC-4.

4.1 Weak Keys Resulting in the High Occurrence Probability of Differential Characteristic Ω_Y

We now investigate what keys and IVs can result in the high occurrence probability of differential characteristic Ω_Y in Section 3.

Experiment 1. For each j_0 ($j_0 = 64, \cdots, 79$), we randomly select 2^{38} (K, IV, IV') triplets where IV and IV' are only different at one bit $\Delta iv_{j_0} = 1$. For each (K, IV, IV') triplet, we perform two state initialization processes with (K, IV) and (K, IV') to search for Ω_Y. We find that there are more Ω_Y holding for $j_0 = 66$ or 68. Especially, for $j_0 = 66$, the number of obtained Ω_Y is up to 4983. Since each Ω_Y corresponds to one (K, IV) pair, we can further investigate the statistical property of 4983 (K, IV) pairs obtained above. Firstly, we perform one Key/IV loading process with each obtained (K, IV) pair, then 4983 initial state Y^0 can be obtained. Secondly, we employ a χ^2 test for 16 states of each $[Y^0]_i (i = 0, 1, \cdots, 31)$. Finally, list the χ^2 distributions of $[Y^0]_i$ which are serious unbalance, and the result is illustrated in Table 1.

Table 1. χ^2 distributions and χ^2 values of $[Y^0]_i$ $(i = 0, 1, \cdots, 4)$

k	$[Y^0]_0$	$[Y^0]_1$	$[Y^0]_2$	$[Y^0]_3$	$[Y^0]_4$
0	169	335	0	636	58
1	156	50	0	73	0
2	229	57	4	63	98
3	379	103	5	624	0
4	183	71	0	97	**4594**
5	**376**	101	0	**563**	15
6	**400**	116	0	**602**	1
7	356	839	0	50	7
8	176	63	0	93	30
9	**359**	115	0	**707**	1
10	**406**	92	0	**665**	4
11	323	841	0	47	4
12	322	105	12	596	1
13	318	**865**	9	68	76
14	333	**791**	**2413**	64	1
15	498	439	**2540**	35	93
χ^2	486.2	5201.4	34429.2	4047.9	62874.7

From Table 1, we find that the 5 least significant columns of $[Y^0]_i$ reveal the serious unbalance for χ^2 value which is more than 100, where $\chi^2 = \sum_{k=0}^{15} \frac{(n_k - np_k)^2}{np_k}$, n_k is the frequency of the 4-bit integer $[Y^0]_i$ being k, $n = \sum_{k=0}^{15} n_k$ and $p_k = \frac{1}{16}$. Since K and IV are embedded into Y^0, we hope to deduce some linear correlations between K and IV by the unbalance of $[Y^0]_i$.

For the initial state $Y^0 = (y_3^0, y_2^0, y_1^0, y_0^0)^T$ initialed by IV and K after the Key/IV loading process, we know that $[y_0^0]_i = iv_{64+i}$, $[y_1^0]_i = iv_{16+i}$, $[y_2^0]_i = k_{64+i}$ and $[y_3^0]_i = k_{16+i}$, where $i = 0, 1, \cdots, 4$. So, $[Y^0]_i$ can be represented as

$$[Y^0]_i = iv_{64+i} + iv_{16+i} \cdot 2 + k_{64+i} \cdot 2^2 + k_{16+i} \cdot 2^3, \ i = 0, 1, \cdots, 4. \tag{4}$$

Especially, we can find some strong linear correlations between K and IV. For example, $[Y^0]_4$ concentrates on the point 0x4 with probability $\frac{4594}{4983} = 0.9129$ by the sixth column of Table 1. So from $[Y^0]_4$ =0x4 and equation (4), we get the following conditional probability

$$\Pr(k_{20} = 0, k_{68} = 1, iv_{20} = 0, iv_{68} = 0 | \Omega_Y) = 0.9129. \tag{5}$$

We try to explore some other more complex linear correlations between K and IV by observing the χ^2 distribution of $[Y^0]_i$. We firstly investigate the possible linear equations from $[Y^0]_0$. In order to find the linear correlations easily with high probability, we list the distribution of $[Y^0]_0$ as a matrix in Fig. 3.

cd / ***ab***	00	01	10	11
00	169	156	229	379
01	183	**376**	**400**	356
10	176	**359**	**406**	323
11	322	318	333	498

Fig. 3. 4×4 matrix of $[Y]_0$, ab represents $k_{16}k_{64}$, and cd is $iv_{16}iv_{64}$

From Fig. 3, we know that two linear equations $iv_{64} \oplus iv_{16} = 1$ and $k_{64} \oplus k_{16} = 1$ hold concurrently in the four bold items which correspond to $5 = (0101)_2$, $6 = (0110)_2$, $9 = (1001)_2$ and $10 = (1010)_2$ respectively.

From $\Pr([Y]_0 = 5, 6, 9, 10 | \Omega_Y) = \frac{1541}{4983} = 0.3093$, we get

$$\Pr(iv_{64} \oplus iv_{16} = 1, k_{64} \oplus k_{16} = 1 | \Omega_Y) = 0.3093. \tag{6}$$

Clearly, the probability of (6) is much more than the average value 0.25.

By studying the 3rd, 4-th and 5-th columns of Table 1, we can conclude that

$$\Pr(k_{17} = 1, k_{65} = 1, iv_{65} \oplus iv_{17} = 1 | \Omega_Y) = 0.3323, \quad (7)$$
$$\Pr(k_{18} = 1, k_{66} = 1, iv_{18} = 1 | \Omega_Y) = 0.9939, \quad (8)$$
$$\Pr(k_{19} \oplus k_{67} = 1, iv_{19} \oplus iv_{67} = 1 | \Omega_Y) = 0.5091. \quad (9)$$

Summing up all the linear equations (5)-(9), we define the following sets,

$$A_0 = \{K | k_{64} \oplus k_{16} = 1\},$$
$$A_1 = \{K | k_{17} = 1, k_{65} = 1\},$$
$$A_2 = \{K | k_{18} = 1, k_{66} = 1\},$$
$$A_3 = \{K | k_{19} \oplus k_{67} = 1\},$$
$$A_4 = \{K | k_{20} = 0, k_{68} = 1\}.$$

$$B_0 = \{IV | iv_{64} \oplus iv_{16} = 1\},$$
$$B_1 = \{IV | iv_{65} \oplus iv_{17} = 1\},$$
$$B_2 = \{IV | iv_{18} = 1\},$$
$$B_3 = \{IV | iv_{19} \oplus iv_{67} = 1\},$$
$$B_4 = \{IV | iv_{20} = 0, iv_{68} = 0\}.$$

Let

$$\mathscr{A} = \bigcap_{j=0}^{4} A_j, \quad \mathscr{B} = \bigcap_{j=0}^{4} B_j.$$

We can get the following probability by searching 4983 (K, IV) pairs in Experiment 1.

$$\Pr(K \in \mathscr{A}, IV \in \mathscr{B} | \Omega_Y) = \frac{386}{4983} = 2^{-3.69}. \quad (10)$$

And it is obvious that

$$\Pr(\Omega_Y) = \frac{4983}{2^{38}} = 2^{-25.71}. \quad (11)$$

Because $\Pr(K \in \mathscr{A}, IV \in \mathscr{B}) = 2^{-14}$, by the conditional probability formula with (10) and (11), we get

$$\Pr(\Omega_Y | K \in \mathscr{A}, IV \in \mathscr{B}) = \frac{\Pr(K \in \mathscr{A}, IV \in \mathscr{B} | \Omega_Y) \Pr(\Omega_Y)}{\Pr(K \in \mathscr{A}, IV \in \mathscr{B})} = 2^{-15.40}. \quad (12)$$

We define the key K which falls into set $\mathscr{A}$ as a weak key. For 80 bits key size, the number of weak keys is $2^{80-8} = 2^{72}$. From equation (12), we know that, for a chosen IV pair (IV, IV') with $\{IV, IV'\} \subset \mathscr{B}$, if K is a weak key, Ω_Y occurs with high probability $2^{-15.40}$. We can verify the above probability by the statistical Experiment 2.

Experiment 2. We randomly select 2^{10} weak keys. For each weak key, randomly select 2^{20} (IV, IV') pairs where IV and IV' are only different at one bit $\Delta iv_{66} = 1$, and both belong to set $\mathscr{B}$. Then we get the total occurrence frequency of the differential characteristic Ω_Y is 26086.

Experiment 2 means that $\Pr(\Omega_Y | K \in \mathscr{A}, IV \in \mathscr{B})$ is about $\frac{26086}{2^{10} \times 2^{20}} = 2^{-15.33}$ which confirms our result in (12) again.

4.2 Identifying and Recovering the Weak Keys

From the above subsection, we know that, for a chosen IV pair, if K is a weak key, Ω_Y happens with high probability $2^{-15.40}$. But this is not enough to identify and recover a weak key, we need more information. Denote the first actual keystream output byte which is used to encrypt a plaintext byte as z_s^1, and z_s^1 is generated by IV and K. Similarly, $z_s'^1$ corresponds to IV' and K. Let Δz_s^1 be $z_s^1 \oplus z_s'^1$, and $[\Delta z_s^1]_k$ be the k-th least significant bit of the differential byte Δz_s^1, $k = 0, 1, \cdots 7$. Then the bias of $[\Delta z_s^1]_k$ will provide a solution to identify a weak key.

Experiment 3. In Experiment 2, we have obtained 26086 differential characteristics Ω_Y. And we know each Ω_Y corresponds to one (K, IV, IV') triplet, then we utilize each (K, IV, IV') triplet to run cipher body once and produce two first 8-bit keystream outputs z_s^1 and $z_s'^1$. Then 26086 Δz_s^1 bytes can be obtained.

Denote the number of $[\Delta z_s^1]_k$ being '0' as $n_{k,0}$, and the number of $[\Delta z_s^1]_k$ being '1' as $n_{k,1}$. The 0-1 bias of $[\Delta z_s^1]_k$ can be represented as $\varepsilon_k = |\frac{n_{k,0}}{n_{k,0}+n_{k,1}} - \frac{1}{2}|$. From Experiment 3, for a weak key with chosen IV pairs, if the differential characteristic Ω_Y occurs, the bias of $[\Delta z_s^1]_k$ is listed in Table 2.

Table 2. The bias of $[\Delta z_s^1]_k$ with weak keys $(k = 0, 1, .., 7)$

k	0	1	2	3	4	5	6	7
$\lvert log_2 \varepsilon_k \rvert$	4.38	**3.54**	5.09	5.90	4.91	4.81	6.15	5.02

For a weak key with chosen IV pairs, we adopt $[\Delta z_s^1]_1$ which holds with the highest bias, and get the following conclusion by Experiment 3.

$$\Pr([\Delta z_s^1]_1 = 0 | \Omega_Y, K \in \mathscr{A}, IV \in \mathscr{B}) = \frac{1}{2} + 2^{-3.54}. \tag{13}$$

In an actual attack, we can not get Ω_Y directly except for the actual keystream output. Then we need to know the bias of $[\Delta z_s^1]_1$ without the condition Ω_Y. By further experiments, we can find that if Ω_Y does not happen, for any K and IV, the probability $\Pr([\Delta z_s^1]_1 = 0 | \overline{\Omega}_Y)$ is very close to $\frac{1}{2}$. Then for a weak key and the chosen IV pairs, from (12) and (13), we can get

$$\Pr([\Delta z_s^1]_1 = 0 | K \in \mathscr{A}, IV \in \mathscr{B}) = \frac{1}{2} + 2^{-15.40} \times 2^{-3.54} = \frac{1}{2} + 2^{-18.94}. \tag{14}$$

In order to improve the correctness for identifying a weak key, we need to explore the 0-1 bias of $[\Delta z_s^1]_k$ for a strong key. A strong key means that the key $K \in \overline{\mathscr{A}}$.

Utilizing 4983 keys and IV pairs obtained by Experiment 1, we get the probability

$$\Pr(K \in \overline{\mathscr{A}}, IV \in \mathscr{B} | \Omega_Y) = \frac{151}{4983} = 2^{-5.04}. \tag{15}$$

Since $\Pr(K \in \overline{\mathscr{A}}, IV \in \mathscr{B}) = (1 - 2^{-8}) \times 2^{-6} = 2^{-6.01}$, from (15), it is easy to deduce the probability

$$\Pr(\Omega_Y | K \in \overline{\mathscr{A}}, IV \in \mathscr{B}) = \frac{\Pr(K \in \overline{\mathscr{A}}, IV \in \mathscr{B} | \Omega_Y) \Pr(\Omega_Y)}{\Pr(K \in \overline{\mathscr{A}}, IV \in \mathscr{B})} = 2^{-24.74}. \tag{16}$$

Experiment 4. We randomly select 2^{10} strong keys. For each strong key, randomly select 2^{24} chosen IV pairs, and we can get that the total frequency that Ω_Y occurs is 645. Therefore $\Pr(\Omega_Y | K \in \overline{\mathscr{A}}, IV \in \mathscr{B})$ is about $\frac{645}{2^{10} \times 2^{24}} = 2^{-24.67}$. It is clear that, Experiment 4 verifies the result in (16).

In Experiment 4, we can also get 645 Δz_s^1 corresponding to 645 Ω_Y. By counting the number of '0' among 645 $[\Delta z_s^1]_1$, we calculate the following probability under a strong key and the chosen IV pairs,

$$\Pr([\Delta z_s^1]_1 = 0 | \Omega_Y, K \in \overline{\mathscr{A}}, IV \in \mathscr{B}) = \frac{1}{2} + 2^{-3.68}. \tag{17}$$

By combination of equations (16) and (17), for a strong key and the chosen IV pairs, similarly to (14), we get the probability

$$\Pr([\Delta z_s^1]_1 = 0 | K \in \overline{\mathscr{A}}, IV \in \mathscr{B}) = \frac{1}{2} + 2^{-24.74} \times 2^{-3.68} = \frac{1}{2} + 2^{-28.42}. \tag{18}$$

We use (14) and (18) to identify the weak keys. Approximate the binomial distribution with the normal distribution. Denote the total number of samples as N, the mean as μ, and the standard variance as σ.

From (14), $p = \frac{1}{2} + 2^{-18.94}$, $\mu = Np$ and $\sigma = \sqrt{Np(1-p)}$.

From (18), $p' = \frac{1}{2} + 2^{-28.42}$, $\mu' = Np'$ and $\sigma' = \sqrt{Np'(1-p')}$.

For the normal distribution, the cumulative function gives value $1 - 2^{-9.53}$ at $3\sigma'$, and value 0.023 at -2σ.

If the following relation holds

$$\mu - \mu' \geqslant 3\sigma' + 2\sigma, \tag{19}$$

a strong key will be wrongly identified as a weak key (false positive) with probability $2^{-9.53}$, and each weak key is not identified as a weak key (false negative) with probability 0.023. This means that the weak keys can be successfully identified. By solving (19), the amount of IV pairs required is $N = 2^{40.53}$.

Identifying a weak key means that we can recover 8 key bits. The other 72 key bits can be recovered by searching exhaustively.

5 Conclusion

In this paper, we developed a differential attack on TSC-4. Utilizing the structure of T-function in TSC-4, we constructed a special differential characteristic with high probability which incurs about 2^{72} weak keys. $2^{40.53}$ chosen IV pairs can be used to identify a weak key and recover 8 bits of the weak key. We can recover the other 72 key bits by exhaustive search.

References

1. Hong, J., Lee, D.H., Yeom, Y., Han, D.: New Class of Single Cycle T-functions. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 68–82. Springer, Heidelberg (2005)
2. Hong, J., Lee, D.H., Yeom, Y., Han, D., Chee, S.: T-function Based Stream Cipher TSC-3, available at http://www.ecrypt.eu.org/stream/ciphers/tsc3/tsc3.pdf
3. Fischer, S., Meier, W., Berbain, C., et al.: Non-randomness is eSTREAM Candidates Salsa20 and TSC-4. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 2–16. Springer, Heidelberg (2006)
4. Klimov, A., Shamir, A.: A New Class of Invertible Mappings. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 470–483. Springer, Heidelberg (2003)
5. Klimov, A., Shamir, A.: Cryptographic Application of T-functions. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 248–261. Springer, Heidelberg (2004)
6. Klimov, A., Shamir, A.: New Cryptographic Primitives Based on Multiword T-functions. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 1–15. Springer, Heidelberg (2004)
7. Kunzli, S., Junod, P., Meier, W.: Distinguishing Attacks on T-functions. In: International Conference on Cryptology in Malaysia (2005)
8. Mitra, J., Sarkar, P.: Time-memory Trade-Off Attacks on Multiplications and T-functions. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 468–482. Springer, Heidelberg (2004)
9. Moon, D., Kwon, D., Han, D., et al.: T-function Based Stream Cipher TSC-4, available at http://www.ecrypt.eu.org/stream/p2ciphers/tsc4/tsc4_p2.pdf
10. Muller, F., Peyrin, T.: Linear Cryptanalysis of the TSC Family of Stream Ciphers. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 373–394. Springer, Heidelberg (2005)
11. Muller, F., Peyrin, T.: Linear Cryptanalysis of TSC Stream Ciphers - Applications to the ECRYPT Proposal TSC-3, available at http://www.ecrypt.eu.org/stream/papersdir/042.ps

New Results on Impossible Differential Cryptanalysis of Reduced AES

Wentao Zhang[1], Wenling Wu[2], and Dengguo Feng[2]

[1] State Key Laboratory of Information Security,
Graduate University of Chinese Academy of Sciences, Beijing 100049, P.R. China
zhangwt06@yahoo.com
[2] State Key Laboratory of Information Security,
Institute of Software, Chinese Academy of Sciences, Beijing 100080, P.R. China
{wwl,feng}@is.iscas.ac.cn

Abstract. In this paper, we present some new results on impossible differential cryptanalysis of reduced AES, which update the best known impossible differential attacks on reduced AES. First, we present some new attacks on 6-round AES (for all the three key length). Second, we extend to 7-round AES, also for all the three key variants. Especially for 128-bit keys, the best known results can attack up to 7 rounds using square attack and collision attack respectively, but their complexity are both marginal either on data or on time (ie. require nearly the entire codebook, or close to key exhaustive search). In this sense, our attack is the first non-marginal one on 7-round AES with 128-bit keys. Thirdly, we extend to 8 rounds for 256-bit keys, which is also non-marginal compared with the best non-related-key attacks so far. Finally, we give an improvement of the 7-round attack for 192-bit keys in R.C.W.Phan's paper, which makes the time complexity reduced greatly.

Keywords: AES, cryptanalysis, impossible differentials.

1 Introduction

AES [1] supports 128-bit block size with three different key lengths (128, 192, and 256 bits), which is denoted as AES-128, AES-192 and AES-256 respectively, and we write AES for all the three variants. Ever since the selection of AES, its security has drawn much attention from worldwide cryptology researchers. Because of the importance of AES, it's very necessary to constantly reevaluate its security under various cryptanalytic techniques. In this paper, we study the security of AES against impossible differential attack.

Impossible differential attacks [2] use differentials that hold with probability 0 (or non-existing differentials) to eliminate wrong key material and leave the right key candidate. There have been several impossible differential attacks on AES [3,4,5]. In [3], E.Biham and N.Keller present an impossible differential attack on 5-round AES-128 using some 4-round impossible differentials. Later in [4], J.H.Cheon et al. improved the attack to 6-round AES-128. Note that the attacks

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 239–250, 2007.

in the above two papers didn't exploit the key schedule, so the same attacks can also apply to AES-192 and AES-256. In [5], R.C.W.Phan gave attacks on 7-round AES-192 and AES-256 exploiting weaknesses in the key schedule. From which we can see that the best impossible differential attack on AES-128 reached up to 6 rounds [4], and on AES-192 and AES-256 both up to 7 rounds [5].

In this paper, we present some new results on impossible differential cryptanalysis of AES. First, we present some new attacks on 6-round AES, whose complexity is reduced significantly compared with that in [4]. Next, we extend to 7 rounds, which is also suitable to all the three key variants of AES, especially for 128 bits. Then, we extend to 8-round AES-256. Finally, we present an improvement of R.C.W.Phan's attack [5] on 7-round AES-192. Our results presented here update the best-known impossible differential cryptanalysis on AES to date.

Up to now, there exist only two marginal attacks [6,7] on 7-round AES-128 (ie., require nearly the entire codebook, or time complexity close to key exhaustive search), respectively using square attack (or called partial sum attack) and collision attack, which are the best known attacks on AES-128. In this sense, our result add a new and non-marginal attack on 7-round AES-128, using impossible differential attack.

For comparison, the best attacks on AES-192 and AES-256 not under the related-key model are both square attacks up to 8 rounds [6], which are both

Table 1. Comparison of Some Previous Attacks with Our New Attacks

Cipher	Source	Number of Rounds	Data Complexity	Time Complexity	Attack Type
AES-128	Ref.[3]	5	$2^{29.5}$ CP	2^{31}	Imp.Diff
AES-128	Ref.[4]	6	2^{86} CP	2^{125}	Imp.Diff
AES-192	Ref.[5]	7	2^{92}CP	2^{186}	Imp.Diff
AES-256	Ref.[5]	7	$2^{92.5}$CP	$2^{250.5}$	Imp.Diff
AES	This paper	6	$2^{114.5}$CP	2^{50}	Imp.Diff
		6	$2^{75.5}$CP	2^{104}	
		7	$2^{115.5}$CP	2^{119}	
AES-192	This paper	7	2^{92}CP	2^{162}	Imp.Diff
AES-256	This paper	8	$2^{116.5}$CP	$2^{247.5}$	Imp.Diff
AES-128	Ref.[6]	7	$2^{128} - 2^{119}$CP	2^{120}	Square
AES-128	Ref.[7]	7	2^{32}CP	$\approx 2^{128}$	Collision
AES-192	Ref.[6]	8	$2^{128} - 2^{119}$CP	2^{188}	Square
AES-256	Ref.[6]	8	$2^{128} - 2^{119}$CP	2^{204}	Square

CP – Chosen plaintext.
Time complexity is measured in encryption units.

marginal on data complexity. So we also add a new and non-marginal attack on 8-round AES-256. However, we can't extend to 8 or more rounds on AES-192 at present.

Besides, better cryptanalysis results have been achieved under related-key model for AES-192 and AES-256, but this fact doesn't hold for AES-128 due to the different key schedule. We don't consider the related-key environment in this paper.

We summarize our results along with some previously known ones against AES in Table 1.

Here is the outline. In Section 2, we give a brief description of AES. In Section 3, we present the 4-round impossible differentials. Using these impossible differentials, section 4 presents some new attacks on 6-round AES; section 5 extends to 7-round AES; section 6 extends to 8-round AES-256. Section 7 presents an improvement of R.C.W.Phan's attack on 7-round AES-192. Finally, section 8 summarizes this paper.

2 Description of AES

The AES algorithm encrypts or decrypts data blocks of 128 bits by using keys of 128, 192 or 256 bits. A 128-bit plaintext and the intermediate state are commonly treated as byte matrices of size 4×4, which is shown in Fig.1.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Fig. 1. 4 ×4 Byte Indexing of 128-bit Data Block

Each round is composed of four operations:

- SubBytes(SB): applying the S-box on each byte.
- ShiftRows(SR): cyclically shifting each row (the i'th row is shifted by i bytes to the left, $i = 0, 1, 2, 3$).
- MixColumns(MC): multiplication of each column by a constant 4×4 matrix M over the field $GF(2^8)$, where M is

$$\begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix}$$

- AddRoundKey(ARK): XORing the state and a 128-bit subkey.

The MixColumns operation is omitted in the last round, and an additional AddRoundKey operation is performed before the first round. We also assume

that the MixColumns operation is omitted in the last round of the reduced-round variants.

The number of rounds r is dependent on the key size, 10 rounds for 128-bit keys, 12 for 192-bit keys and 14 for 256-bit keys.

The key schedule of AES takes the secret key and expands it to $r+1$ 128-bit subkeys.

2.1 Notations

In the rest of this paper, we will use the following notations: P denotes the plaintext, and C the ciphertext. x_i^I denotes the input of the i'th round especially, x_1^I denotes the state after the initial whitening subkey addition), while x_i^S, x_i^R, x_i^M and x_i^O respectively denote the intermediate values after the application of SubBytes, ShiftRows, MixColumns and AddRoundKey operations of the i'th round. Obviously, $x_{i-1}^O = x_i^I$ holds.

Let K_i denote the subkey in the i'th round, and the initial whitening subkey is K_0. In some cases, the order of the MixColumns and the AddRoundKey operation in the same round can be interchanged, which is done by replacing the subkey K_i with an equivalent subkey K_i^*, where $K_i^* = MC^{-1}(K_i)$, and we use x_i^W to denote the intermediate value after the application of AddRoundKey operation with K_i^* in the i'th round.

Let $(x_i)_j$ denote the j'th byte of x_i, $j = 1, 2, \ldots 16$. $(x_i)_{Col(l)}$ the l'th column of x_i, $l = 1, 2, 3, 4$. Thus, Column(1) includes bytes 1,5,9 and 13, Column(2) includes bytes 2,6,10 and 14, etc.

3 Four-Round Impossible Differentials of AES

In [3], some 4-round impossible differentials are constructed, and used to attack 5-round AES-128. Later in [4], the same 4-round impossible differentials are used to attack 6-round AES-128. In [5], also using the same impossible differentials, attacks on 7-round AES-192 and AES-256 are presented. It's worth noting that the same 4-round impossible differentials are used in all the above attacks. Moreover, later in all the related-key impossible differential attacks on AES-192 and AES-256 [8,9,10], the related-key impossible differentials used have the same idea with that in [3]. The reason behind is that no longer differentials with probability 1 can be derived due to the good diffusion property of AES, and we suppose that no longer than 4-round impossible differentials exist for AES.

In the following attacks, we also use the same impossible differentials. Firstly, a 2-round differential with probability 1 in the forward direction, then a 2-round differential with probability 1 in the reverse direction, where the intermediate differences contradict each other. More exactly, the 4-round impossible differentials are: given a pair of round inputs which are equal in all bytes except one, then the outputs after 4 rounds can't be equal in any of the following combinations of byte positions: (1,8,11,14),(2,5,12,15), (3,6,9,16) nor (4,7,10,13). Figure 2 shows an illustration of the first case, where a is any non-zero byte, N a non-zero value (possibly distinct), and ? any value.

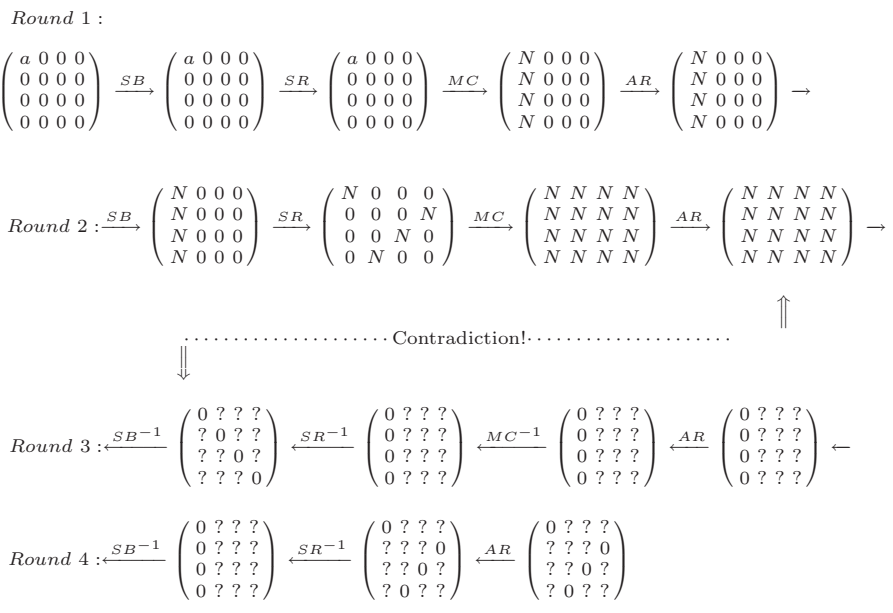


Fig. 2. A 4-round impossible differential of AES

4 Attacking 6-Round AES

In this section, we present some new attacks on 6-round AES using the above impossible differentials. It needs to be pointed that there exists some other-type attacks on 6-round AES, including the designers's square attack [11], partial sum attack [6] and boomerang attack [12], in which the best attacks on 6 rounds are due to [11] and [12].

The main idea of our attacks is as follows: applying the 4-round impossible differentials between the second and the fifth round, assuming some key bytes in the first and the last round for partially decryption, then get rid of all wrong keys using the impossible differentials.

In the attacks, we need to check up whether the difference in at least one of the 4-byte sets (1,8,11,14), (2,5,12,15), (3,6,9,16) or (4,7,10,13) of x_5^W is zero. Because of the next MC operation in the 5'th round, it seems that all the 128 bits of K_6 need to be guessed, however the time complexity exceed the exhaustive key search for AES-128! As in many cipher attacks, the time complexity can be decreased at a cost of more data, eg. in the attacks in [4,5], only those data pairs whose ciphertext pairs have zero difference in certain bytes are chosen, which immediately results in an increase of data amount, but the number of guessing key bytes is decreased greatly. In our following attacks, we make more restrictions on the filtering of data, which is the key point that make our attacks on 6-round AES improved substantially and on 7-round AES-128 succeed.

Assuming a data pair has zero difference in one or several columns of x_5^O, then this property is preserved when rolling back through the MC^{-1} operation. On the other hand, in the rest non-zero-difference columns of x_5^O, we restrict that the difference are non-zero only in two bytes, while zero in the other two bytes (notice that if the difference is non-zero only in one byte while zero in the other three bytes, then the pair will not satisfy the impossible differential in Fig.2 because of the diffusion property of MC). Notice that the above two restrictions are selective in the attacks, thus there will be several combinations to launch a successful attack on 6-round AES. Here, we only present the attack whose time complexity reach the least, which is the basis of our attack on 7-round AES. Furthermore, we briefly introduce a second attack whose data complexity reach the least.

4.1 The First Attack

One can refer to Fig.3 for the following attack, but notice it's only restricted in the former 6 rounds, we can easily conclude that the number of non-zero bytes in x_5^O reach the least possible, ie. only 2 non-zero bytes.

The Attack Procedure
Precomputation: For all the 2^{32} possible pairs of values of the first column of x_1^M with one of the four differences: $(a,0,0,0)$, $(0,a,0,0)$,$(0,0,a,0)$ or $(0,0,0,a)$ (here a can be any non-zero byte), compute the 4 byte values in byte positions (1,6,11,16) of x_1^I. Store these $2^{32} \times 4 \times (2^8 - 1) \approx 2^{42}$ pairs of 4-byte values in a hash table H_p indexed by the XOR differences in these four bytes, then one indexed value corresponds to 2^{10} pairs on average.

The algorithm is as follows:

1. Choose a set of 2^{32} plaintexts which have certain fixed values in all but the four bytes (1,6,11,16). We call this a structure, and one structure can form $2^{32} \times (2^{32} - 1)/2 \approx 2^{63}$ plaintext pairs. Generate m structures, thus $2^{32}m$ plaintexts, and $2^{63}m$ plaintext pairs.
2. Choose only the pairs whose ciphertext pairs have zero difference in all but the two bytes (10,13). The expected number of such pairs is $2^{63} \times m \times 2^{-112} = 2^{-49}m$.
3. Guess the value of the subkey bytes $(K_6)_{10}$ and $(K_6)_{13}$, and perform the followings:
   (a) Initialize a list A of the 2^{32} possible values of the bytes (1,6,11,16) of K_0.
   (b) Decrypt the two bytes (10,13) in all the ciphertext pairs through the 6'th round to get the two bytes (12,16) of x_5^O. Besides, we conclude that all the other bytes in x_5^O have zero difference. Thus, we can calculate the difference in the last column of x_5^W through MC^{-1} operation. If the four bytes are all non-zero, then discard the data pair. The probability that a pair is remained is about $4 \times 2^{-8} = 2^{-6}$, so the expected remaining pairs is $2^{-49}m \times 2^{-6} = 2^{-55}m$. The remained pairs satisfy the following

condition: the difference in at least one of the four 4-byte sets is zero, where the four sets are (1,8,11,14),(2,5,12,15), (3,6,9,16) and (4,7,10,13) of x_5^W.

(c) For every remaining pair, consider their plaintexts (P_1, P_2) and compute $P_1 \oplus P_2$ in the four bytes (1,6,11,16), denote the resulting value by P'. Access the bin P' in H_p. For each pair (x, y) in that bin, remove from the list A the values $P_1 \oplus x$, where P_1 is restricted to four bytes (plaintext bytes (1,6,11,16)).

(d) If A is not empty, output the values in A along with the guess of $(K_6)_{10}$ and $(K_6)_{13}$, there are six key bytes in all.

Analysis of the attack complexity:
From m structures, $2^{63}m$ pairs can be derived. After the filtering in step 2, there remains about $2^{-49}m$ pairs. Then after the filtering in step 3.(b), about $m' = 2^{-55}m$ pairs will remain for a given subkey guess of $(K_6)_{10}$ and $(K_6)_{13}$. Each pair deletes 2^{10} subkey candidates on average, and there are 2^{32} subkey candidates in all, so the expected number of remaining subkeys in A is about $2^{32}(1-2^{10}/2^{32})^{m'}$ after step 3.(c). If $m' = 2^{27.5}$, then only about $2^{32} \times e^{-2^{5.5}} \approx 2^{-33}$ wrong values of the four bytes of K_0 remain, thus in step 3.(d) the wrong value of the six key bytes remains with a very small probability $2^{-33} \times 2^{16} = 2^{-17}$. So we can expect that only the right subkey will remain. Hence, we get the value of $6 \times 8 = 48$ subkey bits. In order to derive $m' = 2^{27.5}$, we need $m = 2^{82.5}$ structures, so the data complexity of this attack is $2^{114.5}$ chosen plaintexts.

Step 3.(b) requires about $233.5 \times 2 \times 2^{16} \times 1/4 = 2^{48.5}$ one round encryptions. Step 3.(c) requires about $2^{27.5} \times 2^{10} \times 2^{16} = 2^{53.5}$ memory accesses to H_p and A, which is equivalent to about 2^{50} 6-round AES encryptions. So the time complexity of the whole attack is about 2^{50} 6-round encryptions. The precomputation requires about $2^{32}/6 \approx 2^{29.5}$ 6-round AES encryptions and the main required memory is dominated by H_p, which is about 2^{45} bytes.

To sum up, the total complexity of the above attack is as follows: The data complexity is $2^{114.5}$ chosen plaintexts, the time complexity is 2^{46} encryptions, and the required memory is 2^{45} bytes.

4.2 The Second Attack

In this subsection, we briefly present another attack on 6-round AES whose data complexity reach the least. The number of non-zero bytes in x_5^O reach the most possible in this attack, ie. 8 non-zero bytes.

Here is the brief procedure. Assuming n structures are needed. Only choosing the pairs whose ciphertext pairs have zero difference in the eight byte positions (1,2,5,8,11,12,14,15). The expected number of such pairs is about $2^{63} \times n \times 2^{-64} = 2^{-1}n$. Guess the eight byte values of subkey K_6 in byte positions (3,4,6,7,9,10,13,16), then decrypt these eight bytes in all the ciphertext pairs through the 6'th round to get the last two columns of x_5^O, we conclude that the first two columns in x_5^O have zero difference. Thus we can calculate the difference in the last two columns of x_5^W, and check up if the two bytes are both

zero in these four possible byte-position combinations: (11,14),(12,15),(3,16) or (4,7). If not, delete the pairs. The probability for a pair to pass this filtering is about $4 \times 2^{-16} = 2^{-14}$, so there remains about $2^{-15}n$ pairs for each guess of the eight bytes of K_6. Thus, we can use the remaining pairs to delete the wrong key candidates in A for each guess of the eight bytes of K_6. The data complexity of this attack is about $2^{75.5}$ chosen plaintexts, the time complexity is about $2^{42.5} \times 2 \times 2^{64}/2/6 = 2^{104}$ encryptions, and the required memory is also 2^{45} bytes.

Finally, we need to point out that the attacks in this subsection can be applicable to all three key variants of AES. If just considering AES-192 or AES-256, there exist attacks whose data complexity are less than this second attack, eg., the attacker guess all the 16 bytes of k_6 and decrypt partially to filter out the data which satisfy the 4-round impossible differentials, the data complexity of this attack is about 2^{30} plaintexts (which can form about 2^{59} plaintext pairs), and the time complexity is about $2^{30} \times 2^{128}/6 \approx 2^{155.5}$ 6-round encryptions.

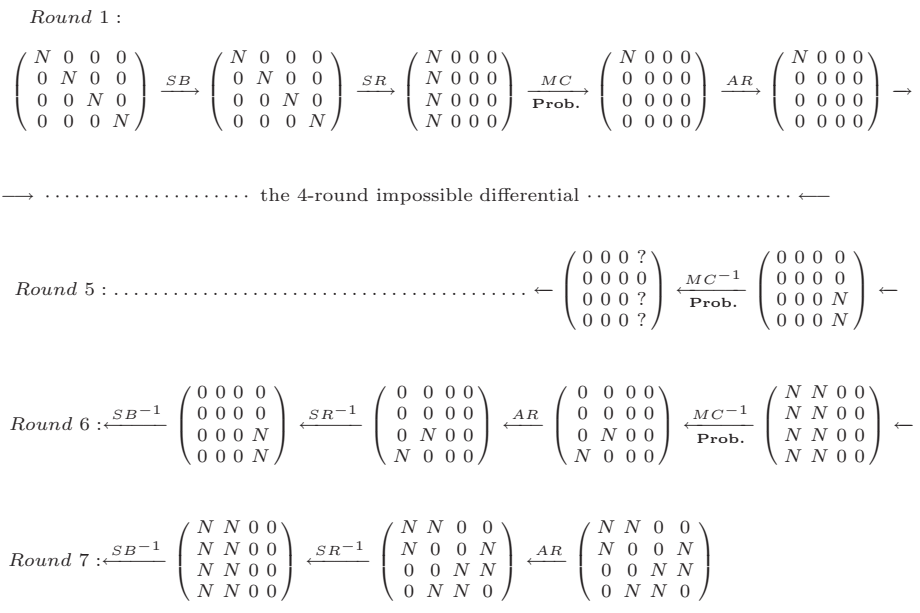


Fig. 3. Impossible Differential Cryptanalysis of 7-round AES

5 Attacking 7-Round AES

Based on the first attack in Section 4, we will present an attack on 7-round AES in this section, which is also effective to all the three key variants of AES. The main idea is: guessing part of the last round subkey K_7, peel off the last round and apply the 6-round attack. In order to reduce the amount of key material

guess, we change the order of the MixColumns and the AddRoundKey operations in the 5'th and 6'th rounds, this is done by replacing the subkey K_5 and K_6 with equivalent subkeys K_5^* and K_6^* respectively.

The attack is illustrated in Figure 3, which corresponds to the first case of the 4-round impossible differentials (ie. the difference in byte positions (1,8,11,14) of x_5^W is zero), in which Prob. means that the event occurs with some probability not equal to 1.

The Attack Procedure

Precomputation: The hash table H_P is just like before.
The algorithm is as follows:

1. Choose m structures, thus $2^{63}m$ plaintext pairs are derived.
2. Choose only the pairs whose ciphertext pairs have zero difference in the 8 byte positions (3,4,6,7,9,10,13,16). [1] The expected number of such pairs is $2^{63} \times m \times 2^{-64} = 2^{-1}m$.
3. Guess the value of 4 bytes in positions (1,8,11,14) of K_7, and perform the followings:
   (a) Initialize a list A of the 2^{32} possible values of the bytes (1,6,11,16) of K_0.
   (b) Decrypt the four bytes (1,8,11,14) in all the ciphertext pairs to get the first column of x_6^W, check up if the difference in three bytes (1,5,9) are all zero. If no, then discard the pair. The probability that a pair is remained is about 2^{-24}, so the expected remaining pairs is $2^{-25}m$.
   (c) Guess the value of another 4 bytes in positions (2,5,12,15) of K_7, and perform the followings:
      i. For each pair remained in step 3.(b), decrypt the four bytes (2,5,12,15) in ciphertext pair to get the second column of x_6^W, check up if the difference in three bytes (2,6,14) are all zero. If no, then discard the pair. The expected remaining pairs is $2^{-49}m$.
      ii. Guess the value of 2 bytes in positions (10,13) of K_6^*, and perform the followings:
         A. For each remaining pair, continue the decryption to get the two bytes in positions (12,16) of x_5^O. Besides, we conclude that the difference of the other two bytes (4,8) in the last column of x_5^O are both zero. Thus, applying MC^{-1} operation, we can get the difference of the last column of x_5^W, check up if the difference in one of the four bytes is zero. If no, then discard the pair, the probability that a pair is remained is about 2^{-6}, so the expected remaining pairs is $2^{-55}m$.
         B. For each remaining pair, consider their plaintexts (P_1, P_2) and compute $P_1 \oplus P_2$ in the four bytes (1,6,11,16), denote the resulting value by P'. Access the bin P' in H_p. For each pair (x, y) in

[1] In fact, it's required that the difference in the other 8 bytes must be non-zero, which occurs with a probability of about 0.969 (very close to 1). So we omit this in step 2.

that bin, remove from the list A the values $P_1 \oplus x$, where P_1 is restricted to four bytes (plaintext bytes (1,6,11,16)).

C. If A is not empty, output the values in A along with the 8 bytes guess in positions (1,2,5,8,11,12,14,15) of K_7 and 2 bytes guess in positions (10,13) of K_6^*, 14 key bytes in all.

Analysis of the attack complexity

After step 2, there remains about $2^{-1}m$ pairs. Then after step 3.(b), about $2^{-25}m$ pairs will remain for a given 4-byte guess of K_7. Next, after step 3.(c).i, about $2^{-49}m$ pairs will remain for a given 8-byte guess of K_7. At last, after step 3.(c).ii.A, about $m' = 2^{-55}m$ pairs will remain for a given 8-byte guess of K_7 and 2-byte guess of K_6^*. Therefor, the expected number of remaining subkeys in A is about $2^{32}(1 - 2^{10}/2^{32})^{m'}$ for a given subkey guess. If $m' = 2^{28.5}$, then only about $2^{32} \times e^{-2^{6.5}} \approx 2^{-98.5}$ wrong values of the four bytes of K_0 remain, and in the last step 3.(c).ii.C, the wrong value of the 14-byte key guess will remain with a very small probability $2^{-98.5} \times 2^{80} = 2^{-18.5}$, so we can expect that only the right subkey will remain. Hence, we get the value of $8 \times 14 = 112$ subkey bits. In order to derive $m' = 2^{28.5}$, we need $m = 2^{83.5}$ structures. So the data complexity of the attack is $2^{115.5}$ chosen plaintexts.

Step 3.(b) requires about $2^{82.5} \times 2 \times 2^{32}/4 = 2^{113.5}$ one round encryptions. Step 3.(c).i requires about $2^{58.5} \times 2 \times 2^{64}/4 = 2^{121.5}$ one round encryptions. Step 3.(c).ii.A requires about $2^{34.5} \times 2 \times 2^{80}/4 = 2^{113.5}$ two round encryptions. Step 3.(c).ii.B requires about $2^{28.5} \times 2^{10} \times 2^{80} = 2^{118.5}$ memory access to A. Thus, the time complexity of the whole attack is about 2^{119} 7-round AES encryptions, and the required memory is also about 2^{45} bytes.

To sum up, the total complexity of the above attack is as follows: The data complexity is $2^{115.5}$ chosen plaintexts, the time complexity is 2^{119} encryptions, and the required memory is 2^{45} bytes.

6 Attacking 8-Round AES-256

For AES-256, guess all the 128 bits of K_8 and exchange the order of the MixColumns and the AddRoundKey operations in the 7'th round. Thus, we can peel off the 8'th round, and apply the 7-round attack in section 5. Because we can calculate one byte of K_6^* from K_8, the number of guessed key bytes is totally 12+14-1=25. The data complexity is $2^{116.5}$ chosen plaintexts, the time complexity is $2^{247.5}$ 8-round AES-256 encryptions, and the required memory is 2^{45} bytes.

For AES-192, if guessing all the 128 bits of K_8, the time complexity will exceed 2^{192}, although several key bytes can be saved because of the 192-bit key schedule. Thus we can't attack 8-round AES-192 using the similar technique.

7 An Improvement of R.C.W.Phan's Attack on 7-Round AES-192

In [5], R.C.W.Phan presented an impossible differential attack on 7-round AES-192 by exploiting weaknesses in the AES key schedule. After careful analysis, we find that 13 key bytes guessing is enough for the attack instead of 16 key bytes in the original attack. Thus, the whole time complexity of R.C.W.Phan's attack can be reduced by a factor of 2^{24}, while the data complexity doesn't change.

Not like the attack in section 5, the order of the MixColumns and the AddRoundKey operations is not changed in the 6'th round, just like in the original attack.

For a pair, if the differences of the last two columns of x_6^O are zero, then along the encryption direction, we conclude that the difference of eight bytes in positions (3,4,6,7,9,10,13,16) of ciphertext pairs must be zero. On the other hand, along the decryption direction, we conclude that the difference of eight bytes in positions (3,4,5,8,9,10,14,15) of x_5^O are all zero. Notice that for a pair, the difference of x_5^O is enough for us to check up whether the difference of x_5^W in the four impossible byte positions are all zero because of the linearity of MC^{-1} operation. Thus, we only need to calculate the difference of the other eight bytes in positions (1,2,6,7,11,12,13,16) of x_5^O.

Herein, in order to know the difference of the other eight bytes of x_5^O from the ciphertexts, we need to guess the value of the eight key bytes (1,2,5,8,11,12,14,15) of K_7 and the first two columns of K_6, ie, the eight key bytes (1,2,5,6,9,10,13,14) of K_6. According to the key schedule of AES-192, we can derive $(k_6)_{10}=(k_7)_{11} \oplus (k_7)_{12}$, also we can calculate $(k_6)_9$ from $(k_7)_{11}$ and $(k_7)_{14}$, $(k_6)_{13}$ from $(k_7)_2$ and $(k_7)_5$. Thus, we only need to guess $8+8-3=13$ key bytes instead of 16 key bytes.

To sum up, the total time complexity of the attack will decreased to 2^{162}, and the data complexity and space complexity don't change.

8 Summary

In this paper, we improved the best known impossible differential attacks on AES. For AES-128, we can reach up to 7 rounds while the previous attacks can only reach up to 6 rounds. For AES-256, we can reach up to 8 rounds while the previous attacks can only reach up to 7 rounds. Furthermore, we present an improvement of the attack on 7-round AES-192 in [5], which makes the time complexity reduced greatly. The comparison of our attack results and related attack results can be found in Table 1. Our results are the best-known impossible differential cryptanalysis on AES to date.

Moreover, the best attack on AES-128 reaches up to 7 rounds, and the best non-related-key attack on AES-256 up to 8 rounds[6,7]. Our impossible differential attacks in this paper add a new and non-marginal attack on 7-round AES-128 and 8-round AES-256 respectively. In contrast, the best non-related-key attack on AES-192 can reach up to 8 rounds [6], however we can't reach 8 rounds on AES-192 using impossible differential attack at present, and we leave it for further work.

Acknowledgment

We would like to thank anonymous referees for their helpful comments and suggestions. The research presented in this paper is supported by the National Natural Science Foundation of China (No.90604036); the National Basic Research 973 Program of China (No.2004CB318004); the National High Technology Research and Development 863 Program of China (No.2007AA01Z470).

References

1. National Institute of Standards and Technology. Advanced Encryption Standard (AES), FIPS Publication 197 (November 26, 2001), available at http://csrc.nist.gov/encryption/aes
2. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 12–23. Springer, Heidelberg (1999)
3. Biham, E., Keller, N.: Cryptanalysis of Reduced Variants of Rijndael, in Official public comment for Round 2 of the AES development effort (2000), available at http://csrc.nist.gov/encryption/aes/round2/conf3/aes3papers.html
4. Cheon, J.H., Kim, M., Kim, K., Lee, J.-Y., Kang, S.: Improved Impossible Differential Cryptanalysis of Rijndael and Crypton. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 39–49. Springer, Heidelberg (2002)
5. Phan, R.C.-W.: Impossible Differential Cryptanalysis of 7-round Advanced Encryption Standard (AES). Information Processing Letters 91(1), 33–38 (2004)
6. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., Whiting, D.: Improved cryptanalysis of Rijndael. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 213–230. Springer, Heidelberg (2001)
7. Gilbert, H., Minier, M.: A collision attack on 7 rounds of Rijndael. In: The Third Advanced Encryption Standard Candidate Conference, pp. 230–241 (April 2000), see http://www.nist.gov/aes
8. Jakimoski, G., Desmedt, Y.: Related-Key Differential Cryptanalysis of 192-bit Key AES Variants. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, Springer, Heidelberg (2004)
9. Biham, E., Dunkelman, O., Keller, N.: Related-Key Impossible Differential Attacks on 8-Round AES-192. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 21–33. Springer, Heidelberg (2006)
10. Zhang, W., Wu, W., Zhang, L., Feng, D.: Improved Related-Key Impossible Differential Attacks on Reduced-Round AES-192. In: SAC 2006. LNCS, vol. 4356, pp. 15–27. Springer-Verlag, Heidelberg (2007)
11. Daemen, J., Rijnmen, V.: AES Proposal: Rijndael, available at http://csrc.nist.gov/envryption/aes/rijndael
12. Biryukov, A.: The Boomerang Attack on 5 and 6-Round Reduced AES. In: Dobbertin, H., Rijmen, V., Sowa, A. (eds.) AES 2004. LNCS, vol. 3373, pp. 11–15. Springer, Heidelberg (2005)

A Note About the Traceability Properties of Linear Codes

Marcel Fernandez[1], Josep Cotrina[1], Miguel Soriano[1,2], and Neus Domingo[3,⋆]

[1] Technical University of Catalonia, Department of Telematic Engineering
C/Jordi Girona 1-3, C3 08034, Barcelona, Spain
[2] CTTC. Centre Tecnolgic de Telecomunicacions de Catalunya
Av. Canal Olmpic S/N 08860 Castelldefels. Barcelona, Spain
[3] I.E.S. J.V Foix
Rubi. Barcelona, Spain

Abstract. We characterize the traceability properties of linear codes. It is well known that any code of length n and minimum distance d is a c-TA code if $c^2 < n/(n-d)$. In this paper, we show that a less restrictive condition can be derived. In other words, there exists a value Z_C, with $n - d \leq Z_C \leq c(n - d)$, such that any linear code is c-TA if $c < n/Z_C$. We also prove that in many cases this condition is also necessary. These results are applied to cyclic and Reed-Solomon codes.

1 Introduction

The concept of traitor tracing was introduced in [1] as a method to discourage piracy. Traitor tracing schemes are useful in scenarios where the distributed content may only be accessible to authorized users, like decrypting broadcast messages, software installation and distribution of multimedia content.

This paper discusses the characteristics of the traceability properties of codes used in traitor tracing and fingerprinting schemes. Before we get into technical matters, we give an intuitive overview. By doing this at the beginning of the paper, we try to separate the concepts from where our work emanates from the intrinsic mathematical development and also hopefully provide the reader an extra motivation for going deep into our results.

The scenario we will deal with is the following one. A distributor D, that sells digital content, wishes to discourage illegal redistribution of its products. To this end, he embeds a [illegible] set of symbols to each copy of the content before it is delivered. This makes each copy unique and therefore if a dishonest user illegally redistributes his copy, he can be unambiguously identified by simply extracting the set of symbols.

A weakness to this scheme can be spotted by noting that a coalition of two or more dishonest users can get together and by comparing their copies they

⋆ This work has been supported in part by the Spanish Research Council (CICYT) Project TSI2005-07293-C02-01 (SECONNET), by CICYT Project TEC2006-04504 and by CONSOLIDER CSD2007-00004 "ARES", funded by the Spanish Ministry of Science and Education.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 251–258, 2007.

perform a [illegible]. This attack consists in detecting the positions in which their copies differ and with this knowledge, they create a new copy that in every detected position contains a symbol of one of the members of the coalition. This new copy is a pirate copy that tries to disguise the identity of the guilty users and is the one they redistribute.

More precisely, the distributor assigns a codeword from a q-ary fingerprinting code to each user. To embed the codeword into each users object, the object is first divided into blocks. The distributor then picks a set of these blocks at random. This set of blocks is kept secret and will be the same for all users. Then using a watermarking algorithm a mark of the fingerprint codeword is embedded in each block. Note that a given user will have one of the q versions of the block. The colluding traitors compare their copies, detect the blocks where their copies differ and with this information at hand, they construct a pirate copy where each block belongs to the corresponding block of one of the traitors. Since each mark is embedded using a different random sequence and these sequences are unknown to the traitors, they cannot create a version of the block that they do not have.

With the above scenario in mind, it is clear that the distributor D, has to embed sets of symbols that are secure against collusion attacks. One way to obtain such sets is by using [illegible] (c-TA). In this case, traitor tracing reduces to search for the codewords that agree in most symbol positions with the pirate.

1.1 Previous Work

The c-TA property of error correcting codes has been studied by several authors.

In [9], sufficient conditions for a linear error correcting code to be a c-TA code are given. Efficient algorithms for the identification of traitors in schemes using c-TA codes are discussed in [8] [3].

In [10], families of c-TA codes with small alphabet are obtained, thus answering a question in [9]. New families of traceability codes, that also answer a question posted in [9], are discussed in [5]. For bounds and constructions of traceability schemes see [7].

More related to our work are the results in [4], where necessary conditions for c-TA maximum distance separable codes (MDS) codes are given, thus closing the problem of establishing traceability conditions for this family of codes.

1.2 Our Contribution

In this paper we discuss the c-TA properties of several families of error correcting codes. The key issue of this paper is to provide necessary conditions for the c-TA property of error-correcting codes, since prior to [4], only sufficient conditions were known.

First of all, we generalize the results in [4] for MDS codes to other types of codes, by giving necessary conditions for a large number of families of linear codes to be c-TA codes. We also show how the sufficient conditions stated in

[1] [2] [9, Theorem 4.4] can, in general, be relaxed. We also specialize our results by developing these results for cyclic and Reed-Solomon codes. Note that in this case, we are able to state the results in [4] as a corollary.

1.3 Organization of the Paper

The paper is organized as follows. In Section 2, we provide the necessary background in coding theory and traceability. The main result of Section 3, comes in the form of a theorem and gives a necessary condition for linear codes to be c-TA codes. Also, a tighter bound for the sufficient conditions of the c-TA property is established, and the results are extended to cyclic and Reed-Solomon codes. We draw our conclusions in Section 4.

2 Definitions and Previous Results

We define a *code* as a set of n-tuples of elements from a set of scalars. The set of scalars is called the *alphabet of the code*. An n-tuple in the code is called a *word* and the elements of the code are called *codewords*. If the code alphabet is a finite field $\mathbb{F}_q$, then a code C is a *linear code* if it forms a vectorial subspace. The dimension of the code is defined as the dimension of the vectorial subspace.

Let $\mathbf{a}, \mathbf{b} \in \mathbb{F}_q^n$ be two words, then the *Hamming distance* $(\mathbf{a}, \mathbf{b})$ between $\mathbf{a}$ and $\mathbf{b}$ is the number of positions where $\mathbf{a}$ and $\mathbf{b}$ differ. Let C be a code, the *minimum distance* of C, $d(C)$, is defined as the smallest distance between two different codewords.

Let $\mathbf{c}$ be a word, then the number of nonzero coordinates in $\mathbf{c}$ is called the *weight* of $\mathbf{c}$ and is commonly denoted $w(\mathbf{c})$.

A linear code with length n, dimension k and minimum distance d is denoted as a $[n, k, d]$-code, or simply as an (n, d) code.

For a linear $[n, k, d]$-code C, we have that the inequality $d \leq n - k + 1$ always holds. This inequality is called the *Singleton bound* [6]. Codes with equality in the Singleton bound are called *maximum distance separable codes* or just MDS codes. A well known class of linear MDS codes are Reed-Solomon codes, that can be defined as follows:

Let $\mathbb{F}_q$ be the finite field of q elements and $\mathbb{F}_q[x]$ the ring of polynomials defined over $\mathbb{F}_q$. Consider the set of polynomials of degree less than k, $\mathbb{F}_q[x]_k \subset \mathbb{F}_q[x]$. Let $\alpha_1, \ldots, \alpha_n$, be n distinct elements of $\mathbb{F}_q$, and $\beta_1, \ldots, \beta_n \in \mathbb{F}_q - \{0\}$. We define a *generalized* Reed-Solomon (RS) code as the vector subspace of $\mathbb{F}_q^n$ determined by the vectors of the form

$$\mathbf{v} = (\beta_1 f(\alpha_1), \ldots, \beta_n f(\alpha_n))$$

where $f \in \mathbb{F}_q[x]_k$.

This is a *general* definition of RS codes, and includes more codes than the classical one, nevertheless we will use it because the results derived in Corollary 2 below are applicable to all of them.

A linear code for which every cyclic shift of a code word is also a code word is called a *cyclic code*.

2.1 Background and Previous Results on c-TA Traceability Codes

Given a code $C(n,d)$ defined over the finite field of q elements, $\mathbb{F}_q$, where n denotes the code length and d the minimum distance of the code, the [illegible] of any subset $T = \{\mathbf{t}^1, \ldots, \mathbf{t}^c\} \subseteq C$, where $\mathbf{t}^i = (t_1^i, \ldots, t_n^i)$, denoted $desc(T)$, is defined as

$$desc(T) = \left\{ \mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{F}_q^n | y_i \in \{t_i^j | \mathbf{t}^j \in T\}, 1 \leq i \leq n \right\}.$$

Definition 1. [illegible] C [illegible] c [illegible] c [illegible] $c > 0$ [illegible] $T \subseteq C$ [illegible] c [illegible] $\mathbf{y} \in desc(T)$ [illegible] $\mathbf{t} \in T$ [illegible] $d(\mathbf{y}, \mathbf{t}) < d(\mathbf{y}, \mathbf{w})$ [illegible] $\mathbf{w} \in C - T$

Definition 2. [illegible] $C(n,d)$ [illegible] $\mathbb{F}_q$ [illegible] c [illegible] c $c > 0$ [illegible] $\mathbf{y} \in \mathbb{F}_q^n$ [illegible] $T \subseteq C$ [illegible] c [illegible] $\mathbf{y} \notin \bigcup_T desc(T)$

$$\bigcap_{\mathbf{y} \in desc(T)} T \neq \emptyset.$$

In [9, Lemma 1.3] it is shown that that a c-TA code is a c-IPP code. In [1] [2] [9, Theorem 4.4] it is proved that any $C(n,d)$ code with $d > n - n/c^2$ is a c-TA code. Moreover, if $C(n,d)$ is a code defined over $\mathbb{F}_q$, in [9, Lemma 1.6] authors show that if $|C| > c \geq q$ then C is not a c-IPP code.

Given a code $C(n,d)$, authors in [8, Section IV], construct unordered sets from the ordered sets that constitute the code as follows: to a codeword $\mathbf{x} = (x_1, \ldots, x_n) \in C$ they associate the set $x' = \{(1, x_1), \ldots, (n, x_n)\}$. Then they define TA set systems (as opposed to TA codes) in the natural way, with the noteworthy difference that a pirate unordered set (unordered false fingerprint) consists of n elements such that each element is a member of some coalition member's set. In [8, Theorem 7] authors prove that if $C(n,d)$ is a Reed-Solomon code with minimum distance $d \leq n - n/c^2$ then the set system corresponding to C is not a c-TA system. Note that this result does not imply that $d > n - n/c^2$ is a necessary condition for RS codes to have the c-TA property.

The work in [4, Theorem 2.2] deals with MDS codes. By developing the concept of group distance, necessary conditions for the c-TA property of MDS codes are given.

3 Sufficient and Necessary Conditions for c-TA Linear Codes

In this section we deal with the c-TA properties of linear codes. First of all, in Proposition 1, we provide a tighter bound for the sufficient conditions of the c-TA property. Armed with this result, we are able to also provide necessary conditions for a linear code to be a c-TA code. Moreover, these conditions are also stated for cyclic and Reed-Solomon codes.

Consider a code $C(n,d)$ (not necessarily a linear code), where n is the length of the code words and d the minimum distance of the code. Then, it is not difficult to prove that a sufficient condition for C to be a c-TA code is that

$$c^2 < n/(n-d) \tag{1}$$

is satisfied.

To see that the condition is sufficient, we consider a coalition of at most c code words that produce a false fingerprint (descendant) $\mathbf{y}$. We can assert that one of the c code words in the coalition agrees with $\mathbf{y}$ in at least $n/c > (n-d)c$ coordinates, but then, if the code is to be a c-TA code, any code word $\mathbf{v}$ ($\mathbf{v}$ not a coalition member) can agree with $\mathbf{y}$ in at most $(n-d)c$ coordinates, otherwise $\mathbf{v}$ would agree in more than $n-d$ coordinates with a coalition code word, and this is not possible by the definition of minimum distance. See also [9, Theorem 4.4].

To verify that a linear code C is not a c-TA code, we only need to find a coalition $T = \{\mathbf{t}_1, \cdots, \mathbf{t}_c\}$ that can produce a false fingerprint $\mathbf{y}$ with $\max_i I(\mathbf{y}, \mathbf{t}_i) \leq \max_{\mathbf{v}} I(\mathbf{y}, \mathbf{v})$ where $\mathbf{v} \in C - T$, and $I(,)$, as defined before, represents the number of coordinates in which both vectors agree. In this case, if we consider $T - \mathbf{v} := \{\mathbf{t}_1 - \mathbf{v}, \cdots, \mathbf{t}_c - \mathbf{v}\}$ it is not difficult to see that coalition $T - \mathbf{v}$ can produce the false fingerprint $\mathbf{y} - \mathbf{v}$, and moreover it satisfies $\max_i I(\mathbf{y} - \mathbf{v}, \mathbf{t}_i - \mathbf{v}) \leq max_{\mathbf{v}} I(\mathbf{y} - \mathbf{v}, \mathbf{0})$. Therefore, to see that a code fails to be a c-TA code, it is enough to analyze coalitions that can produce false fingerprints that are close to the $\mathbf{0}$ code word, and obviously that do not contain code word $\mathbf{0}$.

3.1 The c-TA Property in Linear Codes

Let $C[n,k,d]_q$ be a linear code of block length n, dimension k, minimum distance d, defined over $\mathbb{F}_q$. As we have seen before, if $c^2 < n/(n-d)$, then the code is a c-TA code. This condition is based on the fact that c code words can generate a false fingerprint (descendant) $\mathbf{y}$ with

$$Z(\mathbf{y}) := I(\mathbf{y}, \mathbf{0}) = c(n-d).$$

We now show that for general linear codes this condition is not, in general, necessary and might be relaxed.

We define Z_C as the maximum number of 0's that can be placed on a descendant (false fingerprint) by a c-coalition of $C - \{\mathbf{0}\}$. Note that $n - d \leq Z_C \leq c(n-d)$.

Proposition 1. , $c < n/Z_C$ .. .. $C[n,k,d]_q$ .. c ... ...

Proof: If we suppose that the code is not a c-TA code, then there must exist a c-coalition $T = \{\mathbf{t}_1, \ldots, \mathbf{t}_c\} \subseteq C - \{\mathbf{0}\}$ that can generate a false fingerprint $\mathbf{y}$ such that $I(\mathbf{0}, \mathbf{y}) \geq \max_i I(\mathbf{t}_i, \mathbf{y})$. Obviously $\max_i I(\mathbf{t}_i, \mathbf{y}) \geq n/c$, but we know that $I(\mathbf{0}, \mathbf{y}) = Z(\mathbf{y}) \leq Z_C < n/c$ and the proposition is proved. ∎

We are now in a position to state the main result of this section.

Theorem 1. , $q > n - Z_C + 1$ .. .. .. $C[n,k,d]_q$ .. c .. .. $c < n/Z_C$

Proof: We already know, from Proposition 1, the sufficiency. Here we shall prove the necessary part. In order to do this, we will assume that $c \geq n/Z_C$. This assumption will allow us to construct a coalition that violates the c-TA property.

We begin with a coalition $T = \{\mathbf{t}_1, \ldots, \mathbf{t}_c\} \subseteq C - \{\mathbf{0}\}$, that can generate a false fingerprint $\mathbf{y}$, with $Z(\mathbf{y}) = Z_C$. This coalition exists due to the way Z_C is defined. Moreover, we assume without loss of generality that $Z(\mathbf{t}_1) \geq \cdots \geq Z(\mathbf{t}_c)$. Let $I = \{1, \ldots, n\}$ be the set of indexes that represent the coordinates of a code word. First, we remove from I all the values i corresponding to the coordinates $y_i = 0$ of $\mathbf{y}$, that is $I := I - \{i | y_i = 0\}$.

We define the natural projection $\pi_I : \mathbb{F}_q^n \rightarrow \oplus_{i \in I} \mathbb{F}_q$. Then, for all the code words $\mathbf{t}_j$, $1 < j \leq c$, we choose a field element x_j, such that $I(\pi_I(x_j \mathbf{t}_j), \pi_I(\mathbf{t}_1)) = 0$. These field elements x_j, exist because the coordinates in I of the vectors $\mathbf{t}_j$ are not 0, and because we have $q > n - Z_C + 1 = |I| + 1$. We define $\mathbf{t}_j^* := x_j \mathbf{t}_j$.

The remaining coordinates of the false fingerprint $\mathbf{y}$ (the non 0 coordinates) are produced applying the following process, beginning with $\mathbf{t}_1$, that we describe in a generic way:

For $\mathbf{t}_j^*$ we pick the set of indexes $S \subseteq I$ corresponding to the coordinates s such that $\pi_{\{s\}}(\mathbf{t}_{j+1}) = \pi_{\{s\}}(\mathbf{t}_r)$ for some r such that $j+1 < r \leq c$. Then we add to the false fingerprint $\mathbf{y}$ the coordinates of $\mathbf{t}_j^*$ corresponding to S, and redefine $I := I - S$.

Next, we arbitrarily select a set $L \subseteq I$, defined as $|L| = max\{0, Z_C - |S| - Z(\mathbf{t}_j^*)\}$, and we also place these coordinates from $\mathbf{t}_j^*$ to the false fingerprint $\mathbf{y}$, and again redefine $I := I - L$. We end the process when $I = \emptyset$. In what follows we show that this must happen.

First, the process "fills" all the false fingerprint, or in other words, in the worst case, after applying the process to $\mathbf{t}_c$, we have $I = \emptyset$. Note that in the process we select Z_C coordinates from each coalition code word, thus in the worst case, we need $cZ_C \geq n$, and this is in fact our hypothesis.

Secondly, note that in step j, the coordinates $i \in I$ of $\mathbf{t}_j^*$ always satisfy $\pi_{\{i\}}(\mathbf{t}_j^*) \neq \pi_{\{i\}}(\mathbf{t}_s)$ for $s \neq j$. This is obviously true (by construction) for $j = 1$, and since in step $j - 1$ we erase from I all the indexes corresponding to coordinates satisfying $\pi_{\{s\}}(\mathbf{t}_j^*) = \pi_{\{s\}}(\mathbf{t}_r)$ for any $j < r \leq c$, then it is also true for j. Therefore, the at most Z_C coordinates chosen from $\mathbf{t}_j^*$ to produce the false fingerprint do not agree with the corresponding coordinates of any other coalition code word.

Therefore, if $c \geq n/Z_C$ we can find a coalition T, with $\mathbf{0} \notin T$, that can produce a false fingerprint $\mathbf{y}$ such that $I(\mathbf{y}, \mathbf{t}_j^*) \leq Z_C$, $1 \leq j \leq c$, and $I(\mathbf{y}, \mathbf{0}) = Z_C$, thus proving the theorem. ∎

3.2 The c-TA Conditions for Cyclic Codes

A linear code for which every cyclic shift of a code word is also a code word is called a .. .. Cyclic codes represent an important family of linear codes,

that include Hamming, BCH and RS codes among others. For cyclic codes, applying shifts to a code word of minimum weight, we can ensure that $Z_C = c(n-d)$, thus we have following corollary to Theorem 1.

Corollary 1. $q > n - c(n-d) + 1$ $C[n,k,d]_q$ c $c^2 < n/(n-d)$

We also have the following corollary that has appeared previously in [4].

Corollary 2. $C[n,k,d]_q$ c $c^2 < n/(n-d)$

Note the above corollary holds, because from the definition of Reed-Solomon codes we have that $q \geq n$.

4 Conclusions

The key issue of the work in this paper is to provide necessary conditions for the c-TA property of error-correcting linear codes and Chinese Remainder Theorem codes, since prior to [4], only sufficient conditions were known.

The c-TA properties of linear codes are discussed. Sufficient conditions for a linear code to a be a c-TA code were already known [9]. For the class of maximum distance separable (MDS) codes necessary conditions were established in [4].

In this paper we give necessary conditions for a linear code to have the c-TA property. In a sense we extend the discussion in [4] to all linear codes with a large enough alphabet. We also show that the previously known sufficient conditions can, in many cases, be relaxed. This allows us to express the results in [4] as a corollary.

References

1. Chor, B., Fiat, A., Naor, M.: Tracing traitors. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 480–491. Springer, Heidelberg (1994)
2. Chor, B., Fiat, A., Naor, M., Pinkas, B.: Tracing traitors. IEEE Trans. Inform. Theory 46, 893–910 (2000)
3. Fernandez, M., Soriano, M.: Identification of traitors in algebraic-geometric traceability codes. IEEE Transactions on Signal Processing 52(10), 3073–3077 (2004)
4. Jin, H., Blaum, M.: Combinatorial properties for traceability codes using error correcting codes. IEEE Transactions on Information Theory 53(2), 804–808 (2007)
5. Lindkvist, T., Löfvenberg, J., Svanström, M.: A class of traceability codes. IEEE Transactions on Information Theory 48(7), 2094–2096 (2002)
6. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error-Correcting Codes. North Holland, Amsterdam (1977)
7. Safavi-Naini, R., Wang, Y.: New results on frame-proof codes and traceability schemes. IEEE Transactions on Information Theory 47(7), 3029–3033 (2001)

8. Silverberg, A., Staddon, J., Walker, J.L.: Applications of list decoding to tracing traitors. IEEE Transactions on Information Theory 49(5), 1312–1318 (2003)
9. Staddon, J.N., Stinson, D.R., Wei, R.: Combinatorial properties of frameproof and traceability codes. IEEE Trans. Inform. Theory 47(3), 1042–1049 (2001)
10. van Trung, T., Martirosyan, S.: On a class of traceability codes. Des. Codes Cryptography 31(2), 125–132 (2004)

Power Analysis Attacks on MDPL and DRSL Implementations*

Amir Moradi[1], Mahmoud Salmasizadeh[2], and Mohammad T. Manzuri Shalmani[1,3]

[1] Department of Computer Engineering, Sharif University of Technology, Azadi St., Tehran, Iran
[2] Electronic Research Center, Sharif University of Technology, Azadi St., Tehran, Iran
[3] School of Computer Science, IPM, Tehran, Iran
a_moradi@ce.sharif.edu, {salmasi,manzuri}@sharif.edu

Abstract. Several logic styles such as Masked Dual-Rail Pre-charge Logic (MDPL) and Dual-Rail Random Switching Logic (DRSL) have been recently proposed to make implementations resistant against power analysis attacks. In this paper, it is shown that the circuits which contain sequential elements, flip-flops, and implemented in MDPL or DRSL styles are vulnerable to DPA attacks. Based on our results, the information leakage of CMOS D-flip-flops that are used to construct MDPL and DRSL D-flip-flops is the cause of this vulnerability. To reduce the leakage, a modification on the structure of the MDPL and DRSL flip-flops are proposed; two CMOS D-flip-flops are used in the suggested structure. The proposed technique shows a significant reduction in the information leakage of MDPL and DRSL flip-flops.

Keywords: Side-Channel Attacks, DPA, DRSL, MDPL, flip-flop.

1 Introduction

Since side-channel attacks were introduced as a new aspect of cryptanalysis in [7], the resistance of cryptographic implementations against the new attacks has been involved as a new factor in security evaluation processes. Nowadays the security of the used cryptographic algorithm is not sufficient to undertake the security of a device. Among other types of side-channel attacks, Power Analysis Attacks have been taken into consideration because of their efficiency to find the secret key and their applicability on various types of implementation. Several approaches have been presented to improve the functionality of power analysis attacks. Differential Power Analysis (DPA) attacks [8], second and higher order DPA attacks [10,12], frequency-based DPA attacks [5], correlation power analysis attacks [3], zero-offset DPA attack [22], toggle-count DPA attack [9], and

* This project is partially supported by Iran National Science Foundation and Iran Telecommunication Research Center.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 259–272, 2007.

zero-input DPA attacks [6] are examples of the progress made by researchers in this field. In contrast, others have tried to discover new methods to counteract power analysis attacks. All of them aim at changing the correlation between the instantaneous power consumption of the implementation and the secret intermediate values of the algorithm. Some of them added a noise generation module to make the power traces indistinct [2]; others applied masking methods at algorithm and gate levels [1,11,21]. Also, new logic styles have been presented to make the implementations resistant against power analysis attacks; their goal was to make constant the power consumption of the implementation for every condition of intermediate or other secret values. Pre-charge logics such as Sense Amplifier Based Logic (SABL) [19], Wave Dynamic Differential Logic (WDDL) [20], Dual-Spacer Dual-Rail Pre-charge Logic [16], Masked Dual-Rail Pre-charge Logic (MDPL) [14], which has mixed the masking technique at the gate level and pre-charge logic, and Dual-Rail Random Switching Logic (DRSL) [4] have been proposed to construct resistant implementations.

In this article, we focus on the MDPL and DRSL. The security of combinational circuits which are implemented using these logic styles has been examined previously [14,4]. Although in [17] it has been shown that the leakage occurs in the MDPL gates when input signals have different delay times, it needs a more accurate(high resolution and noise free) power consumption sampling to discover the difference of power traces. The security evaluations having been done so far were at the base of a logic gate or a combinational circuit. In this paper the security of MDPL and DRSL implementations which contain sequential elements, flip-flops, are examined. It is shown that the CMOS flip-flops have a significant information leakage while their inputs are changed; CMOS D-flip-flops were used to build the MDPL D-flip-flop [14] and DRSL D-flip-flop [4]. Accordingly, the logic styles which use CMOS flip-flops are exposed to risk of vulnerability to power analysis attacks even against Simple Power Analysis (SPA) attacks. Two alternative attacks are presented; the first one attacks on a simple MDPL hardware which stores the XOR result of 8-bit plaintext and 8-bit secret key, and the other one attacks on a more complicated circuit, 8-bit XOR followed by an AES S-box, which is constructed according to the DRSL style. The effect of random mask generator which prepares the one-bit masks for MDPL and DRSL circuits on the power consumption traces has been relinquished. These attacks have been done using the simulation results of HSPICE and TSMC $0.18\mu m$ library logic cells. The results show that the DRSL and MDPL implementations containing D-flip-flops are vulnerable to power analysis attacks; finally, a modification on the structure of MDPL and DRSL flip-flops is proposed to reduce their information leakage.

The rest of the paper is organized as follows: MDPL and DRSL styles are reviewed briefly in Section 2. Section 3 illustrates how a CMOS D-flip-flop makes the information leakage occur. Then, two attacks on the MDPL and DRSL implementations based on the given results of Section 3 are presented in Section 4. The proposed modification in the structure of MDPL and DRSL flip-flops is expressed in Section 5. Finally, Section 6 concludes the paper.

2 MDPL and DRSL Styles

In this section, two logic styles having been considered to be attacked are surveyed stressed on the structure of D-flip-flops: (i) MDPL, which has been invented with a mixture of pre-charge concept and gate level masking approach, and (ii) DRSL, which is the dual-rail version of Random Switching Logic (RSL) [18] style. Also, it contains a control box for each gate to generate pre-charge signal separately and automatically.

2.1 MDPL

Figure 1 shows a CMOS majority gate, the basic part of MDPL gates. Two majority gates are applied to build an MDPL AND gate shown in Figure 2. It is noticeable that MDPL gates does not need pre-charge signals. However, input signals of a combinational circuit which was implemented in MDPL style must be pre-charged by two NOR gates as same as the other pre-charge logic styles. Thus, MDPL gates are pre-charged to '0' while the clock signal is HI. The usage of majority gates prevents the glitches in MDPL gates. Clearly, the complement of the MDPL gates can be made by swapping the complementary output wires. Moreover, the structure of the MDPL XOR gate is shown in Figure 3; it is constructed using three MDPL NAND gates to prevent glitches for XOR is not a monotonic function [14]. As mentioned, the MDPL gates are the masked gates, but only one-bit mask is used for all gates in the circuit; therefore, all MDPL gates apply one random bit which can be changed in each clock cycle.

The security evaluation of combinational hardwares which are implemented with the MDPL gates is not our aim in this article; however, we focus on the structure of the MDPL D-flip-flop shown in Figure 4. It is assembled by an MDPL XOR gate and a CMOS D-flip-flop. The XOR gate is used to switch the input values; d_m and $\overline{d_m}$, according to the current and the next mask bits, m and m_n. The fixed 0 and 1 values at the b inputs do not affect the functionality of the pre-charge majority gates which they are connected to. This fact that the complement output of the XOR gate, $\overline{q_m}$, is connected to

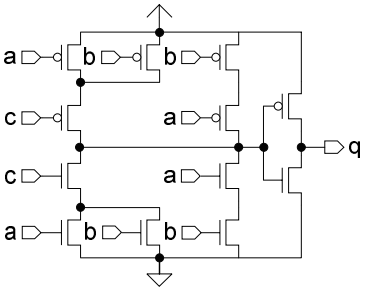


Fig. 1. Schematic of a CMOS majority gate [14]

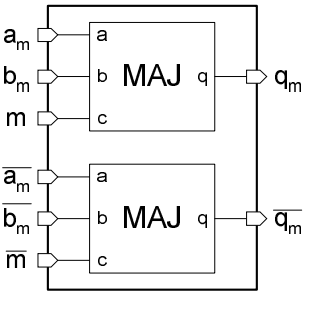


Fig. 2. Schematic of an MDPL AND gate [14]

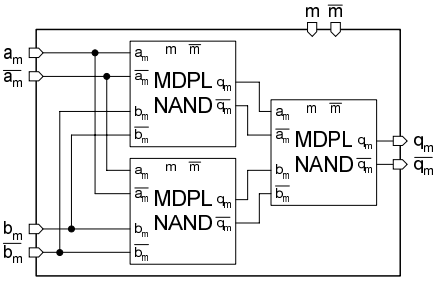


Fig. 3. Schematic of an MDPL XOR gate [14]

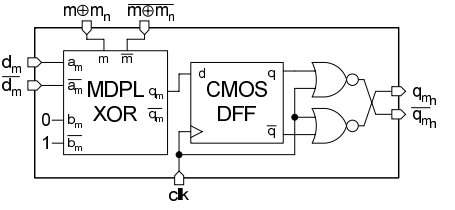


Fig. 4. Schematic of an MDPL D-flip-flop [14]

nowhere causes the risk of information leakage to be highlighted; we have used this concept to attack on MDPL D-flop-flops.

2.2 DRSL

Although the name of DRSL has been extracted from RSL [18] that was duplicated to construct the complementary outputs, it is allied with the MDPL. They have two differences: (i) DRSL gates contain pre-charge signal, but MDPL gates do not and (ii) DRSL gates have a control box which generates the requisite transitions on pre-charge signal for each gate autonomously. As a result, the DRSL implementations do not need a central control unit to schedule the transition time of the pre-charge signals. Figure 5 shows an DRSL AND gate; the control box sink the internal pre-charge signal when all of input signals were evaluated and are stable. Besides, other logic gates and D-flip-flop in DRSL are constructed according to the corresponding MDPL one. Figure 6, for instance, shows the structure of the DRSL D-flip-flop. The risk of information leakage discussed at MDPL subsection is true for DRSL flip-flop, and we use this weakness at the attacks.

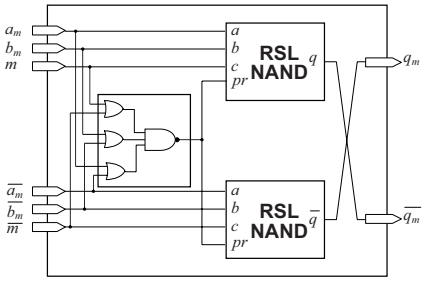


Fig. 5. Schematic of an DRSL AND gate [4]

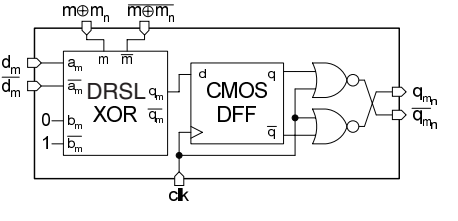


Fig. 6. Schematic of an DRSL D-flip-flop [4]

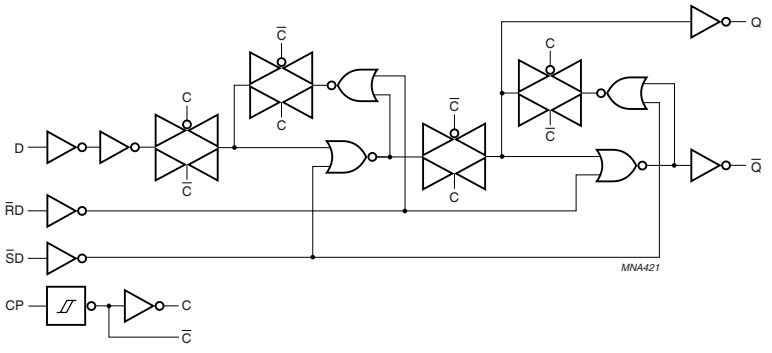


Fig. 7. CMOS D-flip-flop structure[13]

3 Information Leakage of the CMOS D-flip-flop

As described in the previous section, the CMOS D-flip-flops have been used in the MDPL and DRSL D-flip-flops. It is noticeable that we have to use edge triggered flip-flops to build pre-charge compatible flip-flops; otherwise, level sensitive flip-flops; latches, always store zero value in this case. This is due to the fact that their inputs are the outputs of pre-charged gates which are in the pre-charge phase while clock signal remains stable at high. Thus, the usage of edge triggered flip-flops is mandatory.

Clearly, CMOS flip-flops are made using CMOS logic gates. Figure 7 shows the block diagram of a typical D-type positive edge flip-flop [13]. It is not possible to implement this flip-flop with pre-charge logic styles, and we have to use typical CMOS logic styles. Although other more efficient methods have been presented to implement edge triggered CMOS flip-flops, we suppose the typical structure.

It is well known that the usage of the typical CMOS logic gates causes the implementation to leak information through power consumption or electromagnetic channels. Figure 8 shows I_{DD} trace of the presented CMOS D-flip-flop for different changes of input signals. Obviously, the peak of supply current differs for different input changes. Also, Table 1 shows the power consumption of a CMOS D-flip-flop for all possible changes in input signals that were obtained by our simulation. The results show that power consumption values correlate to the change which occurs on the input signals or to the value stored in the flip-flop. For instance, when the clock signal drops, means a negative edge in clock pulse, the power consumption is significantly correlated with the difference between D value and the stored value in F.F.; in other words, if the value of the input signal, D, is different to the internal value of F.F., Q, the power consumption is much more than the condition in which the values of D and Q are equal. This table is the main source of the attacks that are designed to discover the secret key of hardwares implemented in MDPL and DRSL styles.

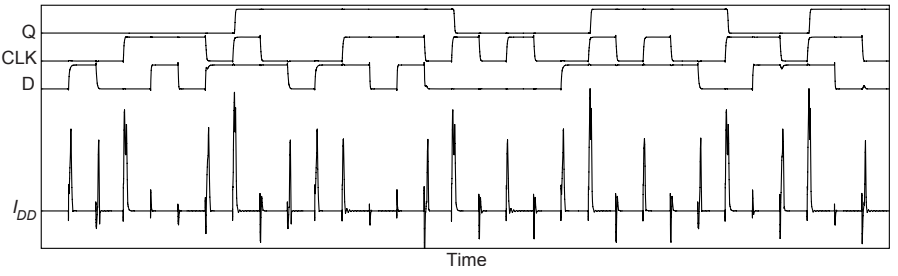


Fig. 8. I_{DD} trace of a CMOS D-F.F. for different changes on input signals

Table 1. The power consumption of a CMOS D-F.F. for every change in input signals

D	CLK	Q	Power (fW)
0→1	0	0	84.2
1→0	0	0	44.7
0→1	1	0	5.3
1→0	1	0	0.8
1→0	0	1	44.7
0→1	0	1	84.2
1→0	1	1	0.8
0→1	1	1	5.4

D	CLK	Q	Power (fW)
0	0→1	1	149.7
0	1→0	0	7.2
0	0→1	0	62.8
1	0→1	0	157.5
1	1→0	1	8.8
1	0→1	1	65.0
1	1→0	0	87.1
0	1→0	1	45.3

4 Attacks

In this section, two attacks are illustrated; they were simulated to discover the secret key of an MDPL and an DRSL implementation using the presented leakage model in the previous section. We show how the CMOS D-flip-flops bring about the MDPL and DRSL hardwares to be vulnerable to power analysis attacks.

4.1 Attack on a Simple MDPL Circuit

For simplicity we start with a simple circuit which stores the XOR result of the plaintext and key in MDPL style. It is supposed that the plaintext and the key are 8-bit data. Figure 9 shows its schematic; the gray part is not accessible, and the plaintext, clock signal, and the current of the gray module are available for using in attack. Furthermore, we assumed that the mask bit is generated by a random number generator such as an LFSR. Neither the mask bit, nor the generator's structure is known for the attacker. Moreover, clock signal plays the pre-charge signal role for the input signals($p_i \oplus m$, $k_i \oplus m$, and m_i) and the output of flip-flops.

It is clearly known that when the clock signal drops from high to low, the input signal of internal CMOS F.F., q_m in Figure 4, stays at zero because the internal MDPL XOR gates is at pre-charge phase. According to Table 1, the power consumption value in this case depends on the value which has been

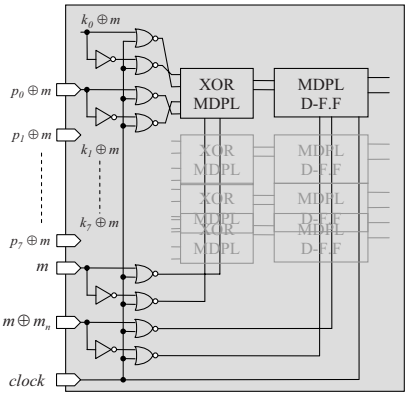


Fig. 9. Schematic of the attacked MDPL hardware

stored in F.F.; Figure 10 shows four different conditions in which it is possible to discover the secret information. As mentioned, in state (1), at the falling clock edge when the internal XOR's output is still '0' and the flip-flops begin to propagate, the stored value in the flip-flop is recognizable, but what is the stored value? $p_i(t-1) \oplus k_i \oplus m(t)$ was stored before in the ith flip-flop.

Also, in state (2) when all pre-charge signals stay at low level and the output of internal XOR modules in flip-flops,. q_m, are evaluated. According to the figure and the related table, there is as obvious difference in power consumption if this output signal is assessed to zero in comparison with the other case. Thus, it is possible to extract the value which will be stored in the flip-flop at the next possible edge of the clock pulse. This value is $p_i(t) \oplus k_i \oplus m(t+1)$ for the ith flip-flop, the addition result of current plaintext and key masked by the next mask bit.

In state (3), when a positive edge appears on the clock signal, according to the related table, when the input signal of flip-flop differs from the stored value, power consumption is more than the condition in which they are the same. In other words, if the flip-flop stores the same value which has been stored previously, the power consumption is less than when the stored value is toggled. Therefore, we can obtain the difference between the stored value, $p_i(t-1) \oplus k_i \oplus m(t)$, and the value which will be stored, $p_i(t) \oplus k_i \oplus m(t+1)$. In fact, we can discover $p_i(t-1) \oplus p_i(t) \oplus m(t) \oplus m(t+1)$ for the ith flip-flop. Although this value does not contain the information about the secret key, it is useful to discover the relation between respective mask bits which are very helpful to find the secret key.

Finally, in state (4), when clock signal stays at high and the internal XOR gate is put on pre-charge phase, the output of the XOR gate is forced to be zero; obviously, if it was zero previously in the evaluation phase, the power consumption will be at the minimum value. Thus, in this case we can extract the value that has been stored in the flip-flop,. $p_i(t) \oplus k_i \oplus m(t+1)$ for the ith flip-flop, the same value which is obtainable in state (2).

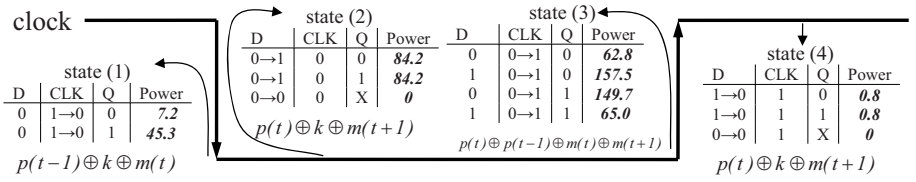


Fig. 10. Four states in which it is possible to use the leakage of flip-flops in the test MDPL circuit

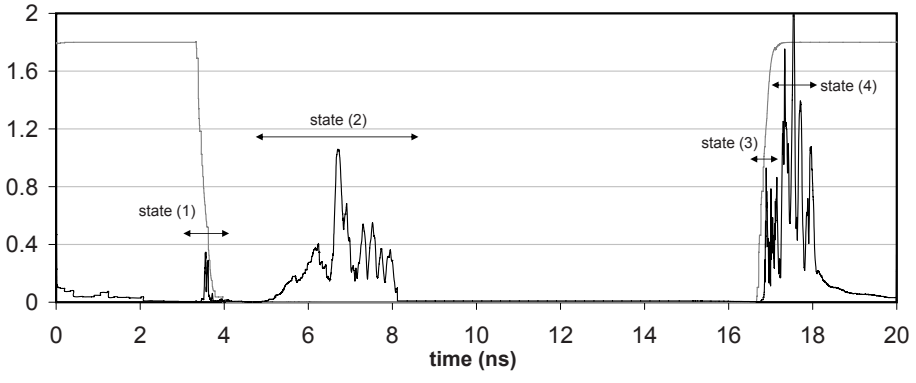


Fig. 11. Sum of absolute differences of power traces in the MDPL circuit to obtain the areas of interest

Now it is time to define the areas of interest in the power traces for each state. We used the approach of Rechberger and Oswald [15]. They have proposed to compute absolute differences of each pair of power traces and sum them up; then, the highest peaks show the most interesting features of the data set. Although they have applied this method in template attacks, it can be used in other attacks which intend to find the points or areas of interest. Figure 11 shows the result of applying this method on the MDPL test circuit with 1000 random plaintexts.

The scenario of the attack consists of two parts. First, we try to extract the value of mask bits; then, the discovered information in the first part is used to find the secret key. As stated above, in state (3) $p_i(t) \oplus p_i(t-1) \oplus m(t) \oplus m(t+1)$ of each flip-flop affects the power consumption dramatically. $p_i(t) \oplus p_i(t-1)$ is known, and $m(t) \oplus m(t+1)$ is recoverable in this state. We used the 8-bit gray code as plaintext because every two respective gray code differ in just one bit. Thus, the effect of $m(t) \oplus m(t+1)$ on power consumption is highlighted. Indeed, we attempt to perform a Simple Power Analysis to discover the difference of respective mask bits,. $m(t) \oplus m(t+1)$. It was done using 256 plaintexts,. 8-bit gray codes. $ms(0)$ to $ms(255)$ were obtained according to equation below.

$$Ms = ms(0),\ ms(1),\ \ldots,\ ms(254),\ ms(255)\ \ ;\ \ ms(t) = m(t) \oplus m(t+1) \quad (1)$$

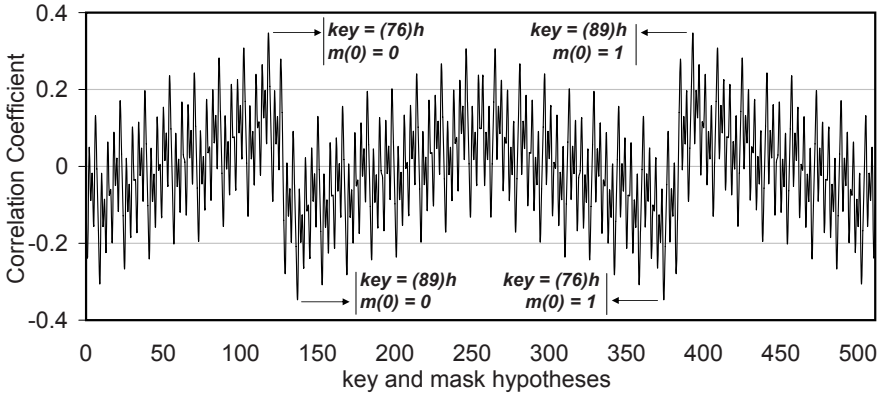


Fig. 12. Correlation coefficient of key and mask hypotheses in the MDPL circuit

Consequently, the guess of $m(0)$(the first mask bit) is sufficient to extract the value of mask bits, $m(i)$; $0 \leq i \leq 255$; finally, there are two 256-bit hypotheses for mask bits. On the other hand, state (2) was selected to apply the mask hypotheses and extract the secret key. We used the power analysis attack based on correlation coefficient which has been introduced in [3]. There are 256 hypotheses for the 8-bit secret key. 512 hypotheses, therefore, exists in the combination of the secret key and mask hypotheses. As mentioned before, $p_i(t) \oplus k_i \oplus m(t+1)$ in each flip-flop affects significantly on the power traces in state (2). The Hamming weight of this term, the number of one bits of $p(t) \oplus k \oplus m(t+1)$, was applied to distinguish the correct key among 512 hypotheses using correlation coefficient. Figure 12 shows the correlation coefficients of all hypotheses. It can be clearly seen that there are two hypotheses for the secret key that one of them is the correct one. In fact, it is shown that leakage of CMOS flip-flops creates the vulnerable implementation even if MDPL style is used to build the combinational part.

4.2 Attack on a AES S-Box Implemented in DRSL Style

A more complicated circuit, key addition followed by substitution box of the AES cryptosystem, was selected to be implemented by DRSL components. Figure 13 shows the block diagram of the considered circuit.

According to the states we discussed for the MDPL circuit, Figure 14 shows the corresponding states of the DRSL circuit. The areas of interest were specified using the same method applied for the MDPL circuit. The scenario of the attack is similar to the procedure illustrated previously. First, we must extract the difference of the mask bits. As mentioned above, the gray codes were used to make the effect of $m(t) \oplus m(t+1)$ out standing, but the value which affects the power consumption is state (3) of the DRSL circuit is $Sbox(p(t-1) \oplus k) \oplus Sbox(p(t) \oplus k) \oplus m(t) \oplus m(t+1)$. The Hamming weight of $Sbox(p(t-1) \oplus k) \oplus Sbox(p(t) \oplus k)$ depends not only on $p(t-1)$ and $p(t)$, but also on the secret

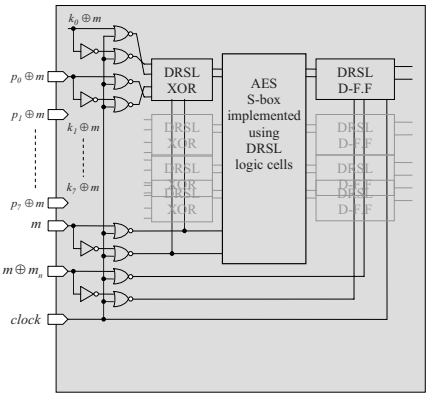


Fig. 13. Schematic of the test circuit implemented in DRSL style

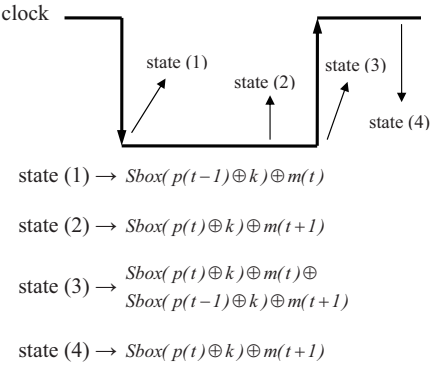


Fig. 14. Four interested status for using the leakage of CMOS flip-flops in the DRSL test circuit

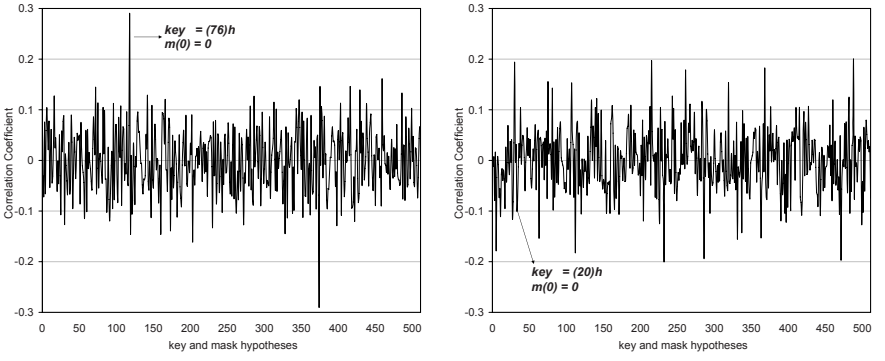
key. Thus, we have to generate p sequences on the base of the key hypothesis as follows:

$$p(t) = Inv\_Sbox\,(GrayCode\,(t)) \oplus k \quad ; \quad 0 \leq t \leq 255 \tag{2}$$

As a result, there are 256 p sequences for each key hypothesis. According to the illustrated attack for MDPL circuit, a stream of $Ms,$ the difference between respective mask bits, is obtained for each p sequence, but which one is correct? The correlation power analysis attacks which are performed to find the secret key specify which hypothesis is correct. In other words, if the correlation coefficient of the key which was selected to generate plaintext sequence is much more than the others, the selected k that equals to the distinguished hypotheses is the correct key. Figure 15 shows the correlation coefficient of hypotheses for two selected keys. In part a, 76_{hex} has been selected to generate the sequence of plaintexts; as it shows, the correlation coefficient of 76_{hex} is much more than the other hypotheses. In contrast, part b shows the correlation coefficient of the hypotheses obtained by the sequence of plaintexts which was generated by selecting 20_{hex} as the secret key. It can be clearly seen that the correlation coefficient of 20_{hex} is not the highest value among the other hypotheses.

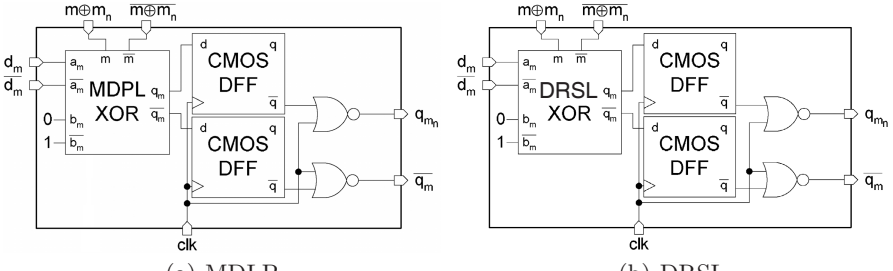
5 A Proposal to Decrease the Leakage of MDPL and DRSL Flip-Flops

As shown in the previous section, the leakage of CMOS flip-flops bring about the MDPL and DRSL flip-flops to be vulnerable to power analysis attacks. We propose to use two CMOS flip-flops in each MDPL and DRSL flip-flips. Figure 16 shows the schematic of the modified flip-flops.



(a) Correct key (76_{hex}) (b) Incorrect key (20_{hex})

Fig. 15. The correlation coefficient of the hypotheses for two alternative selections



(a) MDLP (b) DRSL

Fig. 16. The new structures for the MDPL and DRSL D-flip-flops

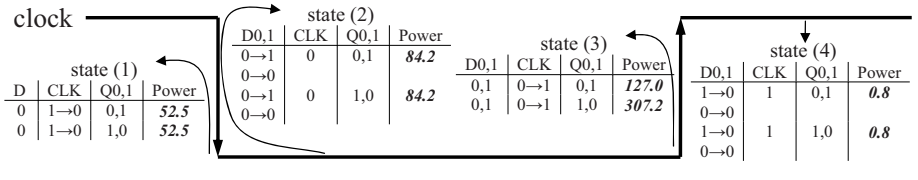


Fig. 17. Four states of information leakage in the new flip-flops

The usage of two flip-flops causes the power consumption which correlated to the modified data to be approximately autonomous from the internal value and the data changed. Figure 17 shows the states for these flip-flops in which the previous structures had information leakage. Obviously, there is no leakage in states (1), (2), and (4), but information leakage still exists at the state (3). Also, the difference between the routing capacitance of F.F inputs, q_m and $\overline{q_m}$, may cause the information leakage in other states. However, according to Figure 17 the information leakage exists only at state (3). Thus, the attacker can find the difference between the consecutive mask bits, and she can not discover any information about the secret key through other states. On the other hand, a better design for CMOS flip-flops helps to avoid the leakage illustrated.

6 Conclusions

First, the leakage model of a CMOS D-flip-flop has been presented. We showed how the power consumption of a CMOS F.F. correlates to the value of input signals changed and to the value stored in the flip-flop. MDPL and DRSL flip-flips are constructed using CMOS D type flip-flops. Thus, they are at the risk of vulnerability to power analysis attacks. Then, two attacks whose aim was to discover the secret key of an MDPL circuit and an DRSL circuit were illustrated. The results that were obtained using simulation show that the usage of CMOS flip-flops leads the MDPL and DRSL implementations to be vulnerable. The attacks were performed in two phases: (i) finding hypotheses for difference between consecutive mask bits using a simple power analysis attack and (ii) finding the secret key using the result obtained in the first phase and a correlation power analysis attack. Finally, a new structure for the MDPL and DRSL flip-flops was proposed. In the suggested method, two CMOS flip-flops are applied to build each MDPL or DRSL flip-flops. The results show that the information leakage of the proposed flip-flops is decreased significantly. In contrast, there is still information leakage at an specific condition, and the attacker would be able to discover the difference between consecutive mask bits. However, the proposed technique is resistant at the other situations. As a result, designing a new CMOS flip-flop to reduce its information leakage can be considered as a future work.

References

1. Akkar, M.-L., Giraud, C.: An Implementation of DES and AES, Secure against Some Attacks. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, Springer, Heidelberg (2001)
2. Benini, L., Omerbegovic, E., Macii, A., Poncino, M., Macii, E., Pro, F.: Energy-aware Design Techniques for Differential Power Analysis Protection. In: Design Automation Conference – DAC 2003, Proceedings, pp. 36–41. ACM Press, New York (2003)
3. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, Springer, Heidelberg (2004)

4. Chen, Z., Zhou, Y.: Dual-Rail Random Switching Logic: A Countermeasure to Reduce Side Channel Leakage. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 242–254. Springer, Heidelberg (2006)
5. Gebotys, C.H., Ho, S., Tiu, C.C.: EM Analysis of Rijndael and ECC on a Wireless Java-Based PDA. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 250–264. Springer, Heidelberg (2005)
6. Golić, J.D., Tymen, C.: Multiplicative Masking and Power Analysis of AES. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 198–212. Springer, Heidelberg (2003)
7. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
8. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
9. Mangard, S., Pramstaller, N., Oswald, E.: Successfully Attacking Masked AES Hardware Implementations. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 157–171. Springer, Heidelberg (2005)
10. Messerges, T.S.: Using Second-Order Power Analysis to Attack DPA Resistant Software. In: Paar, C., Koç, Ç.K. (eds.) CHES 2000. LNCS, vol. 1965, pp. 238–251. Springer, Heidelberg (2000)
11. Oswald, E., Mangard, S., Pramstaller, N., Rijmen, V.: A Side-Channel Analysis Resistant Description of the AES S-box. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 413–423. Springer, Heidelberg (2005)
12. Peeters, E., Standaert, F.-X., Donckers, N., Quisquater, J.-J.: Improved Higher-Order Side-Channel Attacks with FPGA experiments. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 309–323. Springer, Heidelberg (2005)
13. Philips Semiconductors data sheet (July 2003)
14. Popp, T., Mangard, S.: Masked Dual-Rail Pre-Charge Logic: DPA-Resistance without Routing Constraints. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, Springer, Heidelberg (2005)
15. Rechberger, C., Oswald, E.: Practical Template Attacks. In: Lim, C.H., Yung, M. (eds.) WISA 2004. LNCS, vol. 3325, pp. 443–457. Springer, Heidelberg (2005)
16. Sokolov, D., Murphy, J., Bystrov, A., Yakovlev, A.: Improving the Security of Dual-Rail Circuits. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 282–297. Springer, Heidelberg (2004)
17. Suzuki, D., Saeki, M.: Security Evaluation of DPA Countermeasures Using Dual-Rail Pre-charge Logic Style. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 255–269. Springer, Heidelberg (2006)
18. Suzuki, D., Saeki, M., Ichikawa, T.: Random Switching Logic: A Countermeasure against DPA based on Transition Probability. Cryptology ePrint Archive Report 2004/346 (2004), http://eprint.iacr.org/
19. Tiri, K., Akmal, M., Verbauwhede, I.: A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withstand Differential Power Analysis on Smart Cards. In: European Solid-State Circuits Conference, Proceedings, pp. 403–406 (2002)
20. Tiri, K., Verbauwhede, I.: A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In: Design, Automation and Test in Europe Conference and Exposition – DATE 2004, Proceedings, pp. 246–251. IEEE Computer Society, Los Alamitos (2004)

21. Trichina, E., Korkishko, T.: Small Size, Low Power, Side Channel-Immune AES Coprocessor: Design and Synthesis Results. In: Dobbertin, H., Rijmen, V., Sowa, A. (eds.) AES 2004. LNCS, vol. 3373, pp. 113–127. Springer, Heidelberg (2005)
22. Waddle, J., Wagner, D.: Towards Efficient Second-Order Power Analysis. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 1–15. Springer, Heidelberg (2004)

Safe-Error Attack on SPA-FA Resistant Exponentiations Using a HW Modular Multiplier

Chong Hee Kim[1,⋆], Jong Hoon Shin[2],
Jean-Jacques Quisquater[1], and Pil Joong Lee[2]

[1] UCL Crypto Group, Université Catholique de Louvain, Belgium
chong-hee.kim@uclouvain.be, quisquater@dice.ucl.ac.be
[2] Dept. of Electronic and Electrical Eng., POSTECH, Pohang, Korea
jhshin77@gmail.com, pjl@postech.ac.kr

Abstract. The RSA is one of the most widely used algorithms nowadays in smart cards. The main part of RSA is the modular exponentiation composed of modular multiplications. Therefore most smart cards have a hardware modular multiplier to speed up the computation. However, secure implementation of a cryptographic algorithm in an embedded device such as a smart card has now become a big challenge since the advent of side channel analysis and fault attacks. In 2005 Giraud proposed an exponentiation algorithm, which is secure against Simple Power Analysis (SPA) and Fault Attacks (FA). Recently Boscher et al. proposed another SPA-FA resistant exponentiation algorithm. To the authors' best knowledge, only these two provide security against SPA and FA simultaneously in an exponentiation algorithm. Both algorithms are also secure against C safe-error attack and M safe-error attack when they are implemented in a software. However, when they are implemented with a hardware modular multiplier, and this is usual in a smart card, they could be vulnerable to another type of safe error attack. In this paper, we show how this attack is possible on both SPA-FA resistant exponentiation algorithms.

1 Introduction

High speed, small area, and low power consumption have always been important factors in implementing a cryptographic algorithm in a low-resource embedded device such as a smart card. However, after the advent of side channel analysis [15] and fault attacks (FA) [9] we need to balance between the security against these attacks and the above factors. RSA-CRT (RSA based on Chinese Remainder Theorem) is one of the most widely used algorithms in the world. However, it can be broken by simply inducing a random transient fault during its exponentiation computation[9]. Therefore a lot of countermeasures have been proposed [4,19,24] but many of them have been broken [13,21].

In 2005, Giraud proposed a countermeasure against FA and Simple Power Analysis (SPA) for RSA-CRT [12]. SPA is one of the side channel analyses and

⋆ Supported by Walloon Region, Belgium / E.USER project.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 273–281, 2007.

uses the difference in power consumption between different operations [17]. Giraud used the [14] as a basis of his algorithm to defeat both SPA and FA. After that, Boscher et al. proposed another SPA-FA resistant algorithm using a in 2007 [10]. Their algorithm shows the same security level as Giraud's but requires one more register.

Both Giraud's and Boscher et al.' schemes use SPA-FA resistant exponentiation algorithms that use specific relations among temporary variables to defeat FA. Furthermore they are secure against [24] and [22] when they are implemented in a software. However most smart cards have a dedicated hardware to compute a modular multiplier [1,2,3]. Therefore you should check whether these schemes remain secure or not when they are implemented with a hardware multiplier. In this paper, we focus on this issue.

In the next section, we review safe-error attacks. Section 3 shows our fault attack model. We show the vulnerability of the schemes of Giraud and Boscher et al. against our safe error attack in Section 4. Finally we conclude in Section 5.

2 Safe-Error Attacks

There are two types of . The first is the computational safe-error attack (called C safe-error attack) [24]. The second is M safe-error attack [22]. Here we describe both in accordance with the explanation in [14].

C safe-error attack. The C safe-error attack is developed by inducing any temporary random computational fault(s) inside the ALU. It can be mounted against the classical SPA protected exponentiation algorithm of Alg.1. Since the algorithm runs in constant time, an attacker can easily locate the exact moment of the second multiplication, $S[b] \leftarrow S[b] \cdot S[1] \bmod N$, for each iteration. Moreover, when the current exponent bit is 0, the multiplication is a dummy operation and so has no influence on the final result. Therefore, if an attacker induces any kind of computational fault into the ALU during the operation of $S[b] \leftarrow S[b] \cdot S[1] \bmod N$ at ith iteration, then according to whether the final result is incorrect or not, she may deduce if $d_i = 1$ or $d_i = 0$. Therefore, a dummy operation should be avoided to prevent C safe-error attack.

M safe-error attack. The M safe-error attack needs to induce a temporary memory fault inside a register or memory location. Compared to the C safe-error attack, this implies stronger cryptanalytic assumptions, namely a higher controllability of fault location and timing. The M safe-error can be illustrated on the modular multiplication, $B \leftarrow A \cdot B \bmod N$, by calling the program routine listed in Alg.2 as $B \leftarrow MUL(A, B, N)$. In this routine, it is assumed that multiplier B is represented in a 2^T-ary form as $B = \sum_{j=0}^{m-1} B_j (2^T)^j$, and both multiplicand A and multiplier B are sent to the routine MUL by passing their location address (i.e., the call by address technique). This call by address assumption is reasonable since it is popular for both high-level programming language (e.g., C) and all instruction-level language implementations.

Algorithm 1. SPA Resistant left-to-right binary algorithm

INPUT: $M \neq 0, d = (d_{n-1}, ..., d_0)_2, N$
OUTPUT: $M^d \bmod N$

1. $S[0] \leftarrow 1$
2. $S[2] \leftarrow M$
3. for i from $n-1$ to 0 do
4. $\quad b \leftarrow \neg d_i$
6. $\quad S[0] \leftarrow S[0]^2 \bmod N$
7. $\quad S[b] \leftarrow S[b] \cdot S[2] \bmod N$
8. return $S[0]$

The idea behind the M safe-error can be understood as follows. The value of multiplier B will be correct after the assignment operation $B \leftarrow A \cdot B \bmod N$, if some blocks B_j (or Y_j with the notations of Alg.2) of the multiplier are modified after they have been employed in the modular multiplication algorithm. As suggested in [22], this M safe-error can be avoided if B is assigned as the multiplicand, i.e., by calling the routine as $B \leftarrow MUL(B, A, N)$. In this case, since B is always called during each loop operation, any fault induced in B will make the output incorrect regardless of the value of corresponding secret bit.

Algorithm 2. M safe-error on interleaved modular multiplication.

INPUT: X, Y, N
OUTPUT: $R \leftarrow MUL(X, Y, N)$

1. $R \leftarrow 0$
2. for i from $m-1$ to 0 do
3. $\quad R \leftarrow (R \cdot 2^T + X \cdot Y_i) \bmod N$
4. return R

3 Fault Attack Model

Our fault attack model is the same as that of M safe-error attack except that a hardware modular multiplier is used. Since most smart cards have a hardware multiplier to increase the performance during cryptographic operations, it is a reasonable approach. In M safe-error attack, it is assumed that both multiplicand A and multiplier B are sent to the routine MUL by passing their location address. However when a hardware multiplier is used, this assumption is no longer valid. A load of a value of A (or X with the notations of Alg.2) each time during *for* loop is too cumbersome and does not give any advantage of using a hardware multiplier. Because it means that each time it computes $X \cdot Y_i$ it

Algorithm 3. SPA/FA Resistant Right-to-Left Modular Exponentiation

INPUT: $M \neq 0, d = (d_{n-1}, ..., d_0)_2, N$
OUTPUT: $M^d \bmod N$ or "Error"

1. $S[0] \leftarrow 1$
2. $S[1] \leftarrow 1$
3. $A \leftarrow M$
4. for i from 0 to $n-1$ do
5. $\quad S[\overline{d_i}] \leftarrow S[\overline{d_i}] \cdot A \bmod N$
6. $\quad A \leftarrow A^2 \bmod N$
7. if $(M \cdot S[0] \cdot S[1] = A \bmod N)$ and $(A \neq 0)$ then
8. $\quad$ return $(S[0])$
9. else
10. $\quad$ return ("Error")

should load a value of X from memory via a bus. Therefore, a hardware modular multiplier loads either both A and B from the beginning of multiplication or loads A and sequentially B_j during the operation [11,16].

Therefore, our fault attack model assumes that 1) an attacker induces a temporary memory fault inside a register or memory location (the same as in M safe-error attack) and 2) a hardware modular multiplier is used.

4 Safe-Error Attack on SPA-FA Resistant Exponentiations

Recently two SPA-FA resistant exponentiation algorithms have been proposed. The scheme of Boscher et al. is based on right-to-left binary algorithm [10] and the scheme of Giraud uses the Montgomery ladder [12]. Both use the fact that there is a certain relation between the temporary variables in every iteration. They have inherent vulnerabilities against the modification of exponent and skip of some iterations. Therefore they must be used with a countermeasure to check these two vulnerabilities [12]. Hereafter we do not mention it for simplicity purposes.

4.1 Safe-Error Attack on the Scheme of Boscher et al.

Boscher et al. proposed a scheme secure against SPA and FA based on right-to-left binary algorithm [10]. We describe it in Alg.3 where a multiplication and a squaring are done in each iteration. Furthermore in every iteration, the relation $M \cdot S[0] \cdot S[1] = A \bmod N$ holds.

Our attack can be illustrated as follows. Let us call the content of the loop of Alg. 3:

4. for i from 0 to $n-1$ do,
5. $\quad S[\overline{d_i}] \leftarrow S[\overline{d_i}] \cdot A \bmod N$,
6. $\quad A \leftarrow A^2 \bmod N$.

That is, if $d_i = 0$, then it computes:

5. $S[1] \leftarrow S[1] \cdot A \bmod N$,
6. $A \leftarrow A^2 \bmod N$.

If $d_i = 1$, then it computes:

5. $S[0] \leftarrow S[0] \cdot A \bmod N$,
6. $A \leftarrow A^2 \bmod N$.

If we assume that a dedicated hardware is used for a modular multiplication (and this is usual in modern smart cards), then the values of two parameters $S[i]$ and A are employed into a dedicated hardware and its output is returned and stored again in the register $S[i]$. If an error is induced into one of the registers $S[i]$ after its value is loaded (and before storing its result), the error is detected or not depending on the value of d_i.

For example, let us suppose $d_i = 1$ and the attacker induces a temporary memory fault inside $S[0]$ after its value is loaded. Then the output of the dedicated hardware is stored again in $S[0]$. Therefore the induction of an error into $S[0]$ does not influence the final result. Again let us suppose $d_i = 0$ and the attacker induces a temporary memory fault inside $S[0]$ after the load of the value of $S[1]$ into the dedicated hardware. Then, the error in $S[0]$ is propagated until the end of algorithm and finally the algorithm detects the error and outputs "*Error*".

The same situation occurs when the attacker induces a temporary memory fault inside $S[1]$ instead of $S[0]$.

Description of our proposed attack. To describe our attack, we re-write Alg.3 in a more detailed form as in Alg. 4. Therefore the following description is based on Alg. 4.

Procedure of our proposed attack

- From $i = 0$ to $n - 1$ do the following:
  - induce a [illegible] into a register $S[0]$ (or $S[1]$) between Line 5.1 and Line 5.3,
  - if the device outputs "*Error*", then $d_i = 0$ (respectively, $d_i = 1$).

4.2 Practicability of the Attack

Three assumptions are necessary to make our proposed attack possible:

1. Control of the timing of the attack,
2. Possibility of changing the value of a register,
3. Check of the occurrence of the error.

Algorithm 4. (Detail) SPA/FA Resistant Right-to-Left Modular Exponentiation

INPUT: $M \neq 0, d = (d_{n-1}, ..., d_0)_2, N$
OUTPUT: $M^d \bmod N$ or "Error"

1. $S[0] \leftarrow 1$
2. $S[1] \leftarrow 1$
3. $A \leftarrow M$
4. for i from 0 to $n-1$ do
5.1 employment of $S[\overline{d_i}]$ and A into a dedicated HW
5.2 compute $S[\overline{d_i}] \cdot A \bmod N$ (within the dedicated HW)
5.3 update the value of $S[\overline{d_i}]$ with the output of a dedicated HW
6. $A \leftarrow A^2 \bmod N$
7. if $(M \cdot S[0] \cdot S[1] = A \bmod N)$ and $(A \neq 0)$ then
8. return $(S[0])$
9. else
10. return ("Error")

Control of the timing of the attack. Alg. 3 has an iterative computation of a multiplication and a squaring. Therefore we can easily find the timing of a multiplication from a power profile or timing analysis of a device. Aumüller et al. used a power profile to control the timing of their attack [4]. Furthermore, since a dedicated hardware usually spends more power than ALU, we can easily find the interval of modular exponentiation. Besides that, the timing between Line 5.1 and 5.3 is the most time consuming part of the whole operation. Therefore it is not so difficult to control the timing of the attack. Furthermore, we do not need any specific faulty values. We just need to invoke a fault to change the value of the target register into an arbitrary value.

Possibility of changing the value of a register. Possibility of changing the value of a register highly depends on the structure of a device and the attack method. Also it depends on the hardware countermeasures of ICs against fault attacks. There are several experimental results with unprotected hardware. In [4], they succeeded in making faults by producing a spike in the smartcard IC's power supply Vcc. Similar experiment is done by Bar-El el al. [6]. Skorobogatov and Anderson also showed that a value of SRAM memory could be changed [20].

Furthermore there are a lot of published works with the assumption of changing the value of a register. Boneh et al. used a model that assumes a value stored in a register might be corrupted to break certain implementations of RSA and Rabin signatures [9]. Seifert's model assumes that the attack is able to enforce random register faults, resulting in a uniformly chosen register content to break authentication of RSA [18]. We can see also similar fault attack model in [5,7]. Joye and Yen used the same model as ours to describe M safe-error attack [14].

Check of the occurrence of the error. Alg. 3 returns "*Error*" when there is an error. Therefore we can easily figure out whether an error is produced or not. Someone may argue that the attacker can easily be defeated if the hardware re-calculates its output when it detects a fault. However, using the analysis of a timing, we can easily detect it. Indeed, the hardware takes twice as long to re-calculates a value after it detects an 'un-safe' fault. Someone else may argue that the output of a random number instead of outputting "*Error*" when it detects an error can prevent our attack. However the attacker can distinguish it from the correct output after she has received a correct output without giving a fault.

Therefore, we can say our random transient fault model based on the change of the value of a register is applicable.

4.3 Safe-Error Attack on the Scheme of Giraud

There is another exponentiation secure against SPA and FA proposed by Giraud [12]. Here, the equation $S[0] \cdot M = S[1] \bmod N$ holds in each iteration as you can see in Alg. 5.

Algorithm 5. SPA/FA Resistant Modular Exponentiation by Giraud

INPUT: $M \neq 0, d = (d_{n-1}, ..., d_0)_2 odd, N$
OUTPUT: $M^d \bmod N$ or "Error"

1. $S[0] \leftarrow M$
2. $S[1] \leftarrow M^2 \bmod N$
3. for i from $n-2$ to 1 do
4. $\quad S[\overline{d_i}] \leftarrow S[\overline{d_i}] \cdot S[d_i] \bmod N$
5. $\quad S[d_i] \leftarrow S[d_i]^2 \bmod N$
6. $S[1] \leftarrow S[1] \cdot S[0] \bmod N$
7. $S[0] \leftarrow S[0]^2 \bmod N$
8. $S[0] \leftarrow S[0] \cdot M \bmod N$
9. if $(S[0] = S[1])$ then
10. $\quad$ return $(S[0])$
11. else
12. $\quad$ return ("Error")

Let us call the content of loop of Alg. 5. If $d_i = 0$, then it computes:

$$4. \quad S[1] \leftarrow S[1] \cdot S[0] \bmod N,$$
$$5. \quad S[0] \leftarrow S[0]^2 \bmod N.$$

If $d_i = 1$, then it computes:

$$4. \quad S[0] \leftarrow S[0] \cdot S[1] \bmod N,$$
$$5. \quad S[1] \leftarrow S[1]^2 \bmod N.$$

For example, let us suppose $d_i = 0$ and the attacker induces a temporary memory fault inside $S[0]$ after its load in Step 4. Then after $S[1] \leftarrow S[1] \cdot S[0]$, the value of $S[1]$ is not affected by a fault. But the operation $S[0] \leftarrow S[0]^2 \bmod N$ is affected and finally the final output becomes wrong. Again let us suppose $d_i = 1$ and the attacker induces a temporary memory fault inside $S[0]$ after the load of it into the dedicated hardware in Step 4. Then, the error in $S[0]$ is cleared when the output of $S[0] \leftarrow S[0] \cdot S[1]$ is stored. Therefore, the attacker can distinguish the value of d_i according to the output.

The same situation occurs when the attacker induces a temporary memory fault inside $S[1]$ instead of $S[0]$.

5 Conclusions and Future Works

In this paper, theoretically we showed that the two known SPA-FA exponentiation algorithms are vulnerable to safe-error attack when they are implemented with a hardware multiplier. Since nowadays almost all smart cards use a hardware multiplier, this approach is reasonable. Our fault attack model requires a [illegible] that is commonly accepted as possible in the literature. Until now, there is no experimental result about safe-error attack including C safe-error attack and M safe-error attack. Therefore it could be interesting for future works to show safe-error attacks with experiments.

Acknowledgements. This research was supported by BK21 and the MIC of Korea, under the ITRC support program supervised by the IITA.

References

1. Advanced Crypto Engine, Infineon. available at, http://www.infin
2. Fame XE, NXD. available at, http://www.nxp.com
3. TORNATO, Samsung. available at http://www.samsung.com/
4. Aumüller, C., Bier, P., Fischer, W., Hofreiter, P., Seifert, J.-P.: Fault attacks on RSA with CRT: Concrete results and practical countermeasures. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 260–275. Springer, Heidelberg (2003)
5. Bao, F., Deng, R., Han, Y., Jeng, A., Narasimhalu, A., Ngair, T.: Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults. In: Christianson, B., Lomas, M. (eds.) Security Protocols. LNCS, vol. 1361, pp. 115–124. Springer, Heidelberg (1998)
6. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerer's apprentice guide to fault attacks. In: Fault Diagnosis and Tolerance in Cryptography in association with DSN 2004 – The International Conference on Dependable Systems and Networks, pp. 330–342 (2004)
7. Blömer, J., Otto, M.: Wagner's attack on a secure crt-rsa algorithm reconsidered. In: Breveglieri, L., Koren, I., Naccache, D., Seifert, J.-P. (eds.) FDTC 2006. LNCS, vol. 4236, pp. 13–23. Springer, Heidelberg (2006)

8. Boneh, D., DeMillo, R., Lipton, R.: On the importance of checking cryptographic protocols for faults. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997)
9. Boneh, D., DeMillo, R., Lipton, R.: On the importance of eliminating errors in cryptographic computations. Journal of Cryptology 14(2), 101–119 (2001) An earlier version appears in [8]
10. Boscher, A., Naciri, R., Prouff, E.: CRT RSA algorithm protected against fault attacks. In: Sauveron, D., Markantonakis, K., Bilas, A., Quisquater, J.-J. (eds.) WISTP 2007. LNCS, vol. 4462, pp. 237–252. Springer, Heidelberg (2007)
11. Örs, S.B., Batina, L., Preneel, B., Vandewalle, J.: Hardware implementation of a Montgomery modular multiplier in a systolic array. In: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, pp. 1–8. IEEE Computer Society, Los Alamitos (2003)
12. Giraud, C.: Fault resistant RSA implementation. In: D.Walter, C., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 142–151. Springer, Heidelberg (2003)
13. Joye, M., Pailler, P., Yen, S.-M.: Secure evaluation of modular functions. In: International Workshop on Cryptology and Network Security 2001, pp. 227–229 (2001)
14. Joye, M., Yen, S.-M.: The montgomery powering ladder. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 291–302. Springer, Heidelberg (2003)
15. Kocher, P.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
16. Manochehri, K., Pourmozafari, S.: Modified radix-2 montgomery modular multiplication to make it faster and simpler. In: International Conference on Coding and Computing – ITCC 2005, vol. 1, pp. 598–602 (2005)
17. Messerges, T., Dabbish, E., Sloan, R.: Examining smart-card security under the threat of power analysis attack. IEEE Transactions on Computers 51(5), 541–552 (2002)
18. Seifert, J.-P.: On authenticated computing and RSA-based authentication. In: Proc. of ACM conference on computer and communications security 2005, pp. 122–127. ACM Press, New York (2005)
19. Shamir, A.: Method and apparatus for protecting public key schemes from timing and fault attacks. United States Patent ♯5,991,415 (November 23, 1999) Also presented at the rump session of EUROCRYPT 1997
20. Skorobogatov, S., Anderson, R.-J.: Optical fault induction attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 2–12. Springer, Heidelberg (2003)
21. Wagner, D.: Cryptanalysis of a provably secure CRT-RSA algorithm. In: 11th ACM Conference on Computers and Communications Security, pp. 92–97. ACM Press, New York (2004)
22. Yen, S.-M., Joye, M.: Checking before output may not be enough against fault based cryptanalysis. IEEE Transactions on Computers 49(9), 967–970 (2000)
23. Yen, S.-M., Kim, S., Lim, S., Moon, S.: RSA speedup with residue number system immune against hardware fault cryptanalysis. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 397–413. Springer, Heidelberg (2002)
24. Yen, S.-M., Kim, S., Lim, S., Moon, S.: RSA speedup with chinese remainder theorem immune against hardware fault cryptanalysis. IEEE Transactions on Computers 52(4), 461–472 (2003) An earlier version appears in [23]

Generalized MMM-Algorithm Secure Against SPA, DPA, and RPA

Atsuko Miyaji*

Japan Advanced Institute of Science and Technology
miyaji@jaist.ac.jp

Abstract. In the execution on a smart card, elliptic curve cryptosystems have to be secure against side channel attacks such as the simple power analysis (SPA), the differential power analysis (DPA), and the refined power analysis (RPA), and so on. MMM-algorithm proposed by Mamiya, Miyaji, and Morimoto is a scalar multiplication algorithm secure against SPA, DPA, and RPA, which can decrease the computational complexity by increasing the size of a pre-computed table. However, it provides only 4 different cases of pre-computed tables. From the practical point of view, a wider range of time-memory tradeoffs is usually desired. This paper generalizes MMM-algorithm to improve the flexibility of tables as well as the computational complexity. Our improved algorithm is secure, efficient and flexible for the storage size.

Keywords: elliptic curve, DPA, RPA, SPA.

1 Introduction

Elliptic Curve Cryptosystems: The elliptic curve cryptosystem (ECC) chosen appropriately can offer efficient public key cryptosystems [1]. Thus, elliptic curve cryptosystems have been desired in various application such as a smart card, whose memory storage and CPU power are very limited. The efficiency of elliptic curve cryptosystems depends on the implementation of scalar multiplication kP for a secret key k and an elliptic-curve point P.

Side Channel Attacks on ECC: Side channel attacks monitor power consumption and even exploit the leakage information related to power consumption to reveal bits of a secret key k although k is hidden inside a smart card [1]. There are two types of power analysis, the simple power analysis (SPA) and the differential power analysis (DPA). SPA makes use of such an instruction performed during a scalar multiplication that depends on the data being processed. DPA uses correlation between power consumption and specific key-dependent bits. The refined power analysis (RPA) over ECC is one of DPA, which exploits a special point with a zero value such as $(0, y)$ or $(x, 0)$ and reveals a secret key [9]. RPA is also called a Goubin-type attack. Not all elliptic curves are vulnerable

* This study is partly supported by Grant-in-Aid for Scientific Research (B), 17300002. and Yazaki Memorial Foundation for Science and Technology.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 282–296, 2007.

against RPA, but some curves in [18] are vulnerable against these attacks. There exist countermeasures against SPA, DPA, and RPA in [3,16,14]. This paper revisits the algorithm proposed by Mamiya, Miyaji, and Morimoto [14], which is called MMM-algorithm[1], here.

Overview of MMM-algorithm: MMM-algorithm uses a random initial point (RIP) R, computes $kP + R$ by dividing a scalar k into $h \times 1$ blocks with a table of pre-computed points based on the blocks, subtracts R from a result, and, finally, gets kP, where $kP + R$ is computed from left to right without any branch instruction dependent on the data being processed. A random initial point at each execution of kP makes it impossible for an attacker to control a point P as he needs since any point or register used in addition formulae is different at each execution. Thus, MMM-algorithm is not only secure against SPA, DPA, and RPA but also efficient scalar multiplication with a precomputed table. However, it provides only 4 available tables of 9, 15, 27, or 51 field elements[2]. Note that MMM-algorithm with a table of 51 field elements is the most efficient in a 160-bit elliptic curve even if more memory space is allowed to use. From the practical point of view, the memory space allowed to use or the time complexity required for cryptographic functions depends on each individual application. Thus, in some application, MMM-algorithm might not be the best.

Our Contribution: In this paper, we generalize MMM-algorithm by dividing a scalar k into $h \times v$ blocks and optimize the computation method of $kP + R$ to improve flexibility of tables as well as computational complexity, while being secure against SPA, DPA, and RPA. It is called Generalized MMM-algorithm in this paper, GMMM-algorithm in short. We also analyze the computational complexity of GMMM-algorithm theoretically. Furthermore, we explore each best coordinate between affine, (modified) Jacobian, mixed coordinate, etc [6] for each division $h \times v$ of GMMM-algorithm according as the ratio of I/M, where I/M represents the ratio of complexity of modular inversion against modular multiplication. As a result, even in the same division as MMM-algorithm, our optimization on coordinates can reduce the computational and memory complexity since such optimization was not investigated in MMM-algorithm [14]. In facts, GMMM-algorithm with a table of 7 or 38 field elements can reduce the computational complexity of MMM-algorithm with a table of 9 or 51 field elements by 19% or 13.2% over for the range of I/M between 4 and 11, respectively; and GMMM-algorithm with a table of only 19 field elements can work faster than MMM-algorithm with a table of 51 field elements under the above range of I/M. Thus, GMMM-algorithm is significantly efficient and flexible even when the storage available is very small or rather large.

This paper is organized as follows. Section 2 summarizes the known facts on elliptic curves and also reviews MMM-algorithm. Section 3 presents our GMMM-algorithm and analyzes the computational complexity theoretically. Section 4 compares our results with the previous results.

[1] In their paper, MMM-algorithm is called BRIP or EBRIP.

[2] More precisely, it uses 3, 5, 9, 17 elliptic-curve points in Jacobian coordinates.

2 Preliminaries

This section summarizes some facts of elliptic curves such as coordinate systems and side channel attacks against elliptic curves, which refers to [6,1].

2.1 Elliptic Curve

Let $\mathbb{F}_p$ be a finite field, where $p > 3$ is a prime. The Weierstrass form of an elliptic curve over $\mathbb{F}_p$ in affine coordinates is described as

$$E/\mathbb{F}_p : y^2 = x^3 + ax + b \quad (a, b \in \mathbb{F}_p, 4a^3 + 27b^2 \neq 0).$$

The set of all points $P = (x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ satisfying E with the point at infinity $\mathcal{O}$, denoted by $E(\mathbb{F}_p)$, forms an abelian group. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on $E(\mathbb{F}_p)$ and $P_3 = P_1 + P_2 = (x_3, y_3)$ be the sum. Then the addition formula Add (resp. doubling Dbl) in affine coordinate can be described by three modules of $\mathsf{Add_p}(P_1, P_2)$, $\mathsf{Add_I}(\alpha)$, and $\mathsf{Add_{NI}}(P_1, P_2, \lambda)$ (resp. $\mathsf{Dbl_p}(P_1)$, $\mathsf{Dbl_I}(\alpha)$ and $\mathsf{Dbl_{NI}}(P_1, \lambda)$) as in [15]. Each module means preparation for 1 inversion, computation of 1 inversion and computation without inversion, respectively. Then the addition formulae are given as follows.

$\mathsf{Add}(P_1, P_2)$ $(P_1 \neq \pm P_2)$			$\mathsf{Dbl}(P_1)$		
$\mathsf{Add_p}(P_1, P_2)$:	α	$= x_2 - x_1$	$\mathsf{Dbl_p}(P_1)$:	α	$= 2y_1$
$\mathsf{Add_I}(\alpha)$:	λ	$= \frac{1}{\alpha}$	$\mathsf{Dbl_I}(\alpha)$:	λ	$= \frac{1}{\alpha}$
$\mathsf{Add_{NI}}(P_1, P_2, \lambda)$:	γ	$= (y_2 - y_1)\lambda$	$\mathsf{Dbl_{NI}}(P_1, \lambda)$:	γ	$= (3x_1^2 + a)\lambda$
	x_3	$= \gamma^2 - x_1 - x_2$		x_3	$= \gamma^2 - 2x_1$
	y_3	$= \gamma(x_1 - x_3) - y_1$		y_3	$= \gamma(x_1 - x_3) - y_1$

Let us denote the computational complexity of an addition (resp. a doubling) by $t(\mathcal{A}+\mathcal{A})$ (resp. $t(2\mathcal{A})$) in affine coordinate and multiplication (resp. inversion, resp. squaring) in $\mathbb{F}_p$ by M (resp. I, resp. S), where $\mathcal{A}$ means affine coordinates. For simplicity, it is usual to neglect addition, subtraction, and multiplication by small constant in $\mathbb{F}_p$ to discuss the computation amount. Then we see that $t(\mathcal{A}+\mathcal{A}) = I + 2M + S$ and $t(2\mathcal{A}) = I + 2M + 2S$. For the sake of convenience, let us denote the computational complexity of $\mathsf{Add_p}$ and $\mathsf{Add_{NI}}$ (resp. $\mathsf{Dbl_p}$ and $\mathsf{Dbl_{NI}}$) by $t(A + A)_{nI}$ (resp. $t(2A)_{nI}$), which represents the total computational complexity of an addition (resp. doubling) without inversion.

Both addition and doubling formulae in affine coordinate need one inversion over $\mathbb{F}_p$, which is more expensive than multiplication over $\mathbb{F}_p$. Affine coordinate is transformed into Jacobian coordinate, where the inversion is free. We set $x = X/Z^2$ and $y = Y/Z^3$, giving the equation

$$E_{\mathcal{J}} : Y^2 = X^3 + aXZ^4 + bZ^6.$$

Then, two points (X, Y, Z) and (r^2X, r^3Y, rZ) for some $r \in \mathbb{F}_p^*$ are recognized as the same point. The point at infinity is transformed to $(1, 1, 0)$. The doubling and addition formulae in Jacobian coordinate are represented in [6]. The computation time of addition (resp. doubling) in Jacobian coordinate are $t(\mathcal{J}+\mathcal{J}) = 12M+4S$

(resp. $t(2\mathcal{J}) = 4M + 6S$), where $\mathcal{J}$ means Jacobian coordinate. Regarding the iterated doubling, that is the computation of $2^w P$, the iterated doubling formula in Jacobian coordinate [10] can work efficiently with $t(2^w\mathcal{J}) = 4wM + (4w+2)S$.

In addition to affine and Jacobian coordinates, there exists their combination coordinate, called mixed coordinate [6]. In the case of mixed coordinate, let us denote by $t(\mathcal{C}^1 + \mathcal{C}^2 = \mathcal{C}^3)$ the time for addition of points in coordinates $\mathcal{C}^1$ and $\mathcal{C}^2$ giving a result in coordinates $\mathcal{C}^3$, and by $t(2\mathcal{C}^1 = \mathcal{C}^2)$ the time for doubling a point in coordinates $\mathcal{C}^1$ giving a result in coordinates $\mathcal{C}^2$. Their performance is summarized in Table 2.1.

Table 2.1. Computational complexity of addition and doubling of elliptic curve

	computational complexity		computational complexity
$t(\mathcal{A}+\mathcal{A}=\mathcal{J})$	$4M + 2S$	$t(2\mathcal{A}=\mathcal{J})$	$2M + 4S$
$t(\mathcal{A}+\mathcal{J}=\mathcal{J})$	$8M + 3S$	$t(2\mathcal{J})$	$4M + 6S$
$t(\mathcal{A}+\mathcal{A})$	$2M + S + I$	$t(2\mathcal{A})$	$2M + 2S + I$
$t(\mathcal{J}+\mathcal{J})$	$12M + 4S$	$t(2^w\mathcal{J})$	$4wM + (4w+2)S$
$t(\mathcal{J}+\mathcal{J}=\mathcal{A})$	$15M + 5S + I$	$t(\mathcal{J}\to\mathcal{A})^\dagger$	$3M + S + I$

† : The computational complexity of transformation from Jacobian to affine coordinate. The computational complexity without inversion is denoted by $t(\mathcal{J}\to\mathcal{A})_{nI}$.

Let us discuss the difference between these coordinates. Regarding additions, we could roughly estimate that $t(\mathcal{A}+\mathcal{A}=\mathcal{J}) < t(\mathcal{A}+\mathcal{J}=\mathcal{J}) \leq t(\mathcal{A}+\mathcal{A}) \leq t(\mathcal{J}+\mathcal{J})$. This means an addition in mixed coordinate is considerably fast but that of Jacobian coordinate is rather slow. Therefore, an addition of Jacobian coordinate had better be avoided. Regarding doublings, we could roughly estimate that $t(2\mathcal{A}=\mathcal{J}) < t(2\mathcal{J}) \leq t(2\mathcal{A})$. This means a doubling in Jacobian coordinate is considerably fast but that of affine coordinate is rather slow, which had better be avoided.

The major problem in affine coordinate is that it requires 1 inversion when it is executed. However, an addition of affine coordinate should be revisited if we consider the above fact of $t(\mathcal{A}+\mathcal{A}) \leq t(\mathcal{J}+\mathcal{J})$. Furthermore, if several additions or doublings are executed in parallel, then we can make use of the following Montgomery's trick [17] to reduce the total number of inversions.

Algorithm 1 ($\mathsf{Minv}[n]$) `Montgomery's trick`
`Input:` $\alpha_0, \cdots, \alpha_{n-1}, p$
`Output:` $\alpha_0^{-1} \bmod p, \cdots, \alpha_{n-1}^{-1} \bmod p$
`1.` $\lambda_0 = \alpha_0$
`2. For` $i = 1$ `to` $n-1$: $\lambda_i = \lambda_{i-1}\alpha_i \bmod p$.
`3.` $I = \lambda_{n-1}^{-1} \bmod p$
`4. For` $i = n-1$ `to` 0: $\lambda_i = I\lambda_{i-1} \bmod p$. $I = I\alpha_i \bmod p$
`5. Output` $\{\lambda_0, \cdots, \lambda_{n-1}\}$

The Montgomery's trick $\mathsf{Minv}[n]$ works with $3(n-1)$ multiplications and 1 inversion to compute n inversions, whose computation time is denoted by $t(\mathsf{Minv}[n]) =$

$3(n-1)M+I$. Therefore, if several additions or doublings in affine coordinates are executed in parallel, then the total number of inversions can be reduced to 1 by executing $\mathsf{Add_p}(P_1, P_2)$ and $\mathsf{Dbl_p}(P_1)$ in parallel, applying the Montgomery's trick to execute $\mathsf{Add_I}(\alpha)$ and $\mathsf{Dbl_I}(\alpha)$ simultaneously, and finally executing $\mathsf{Add_{NI}}(P_1, P_2, \lambda)$ and $\mathsf{Dbl_{NI}}(P_1, \lambda)$ in parallel.

2.2 Power Analysis

There are two types of power analysis, SPA and DPA, which are described in [1]. RPA is one of DPA, which uses characteristic of some elliptic curve to have a special point [9].

Simple Power Analysis: SPA makes use of such an instruction performed during a scalar multiplication that depends on the data being processed. In order to be resistant to SPA, any branch instruction of scalar multiplication should be eliminated. There are mainly two types of countermeasures: the fixed procedure method and the indistinguishable method. The fixed procedure method deletes any branch instruction conditioned by a secret scalar k such as the add-and-double-always algorithm. The indistinguishable method conceals all branch instructions of scalar multiplication algorithm by using indistinguishable addition and doubling operations, in which dummy operations are inserted.

Differential Power Analysis: DPA uses correlation between power consumption and specific key-dependent bits. In order to be resistant to DPA, power consumption should be changed at each new execution. There are mainly 3 types of countermeasures, the randomized-projective-coordinate method (RPC), the randomized curve method (RC), and the exponent splitting (ES) [3]. RPC uses Jacobian or Projective coordinate to randomize a point $P = (x, y)$ into (r^2x, r^3y, r) or (rx, ry, r) for a random number $r \in \mathbb{F}_p^*$, respectively. RC maps an elliptic curve into an isomorphic elliptic curve by using an isomorphism map of (x, y) to (c^2x, c^3y) for $c \in \mathbb{F}_p^*$. ES splits a scalar and computes $kP = rP + (k-r)P$ for a random integer r.

Refined Power Analysis: RPA reveals a secret key k by using a special elliptic-curve point with a zero value, which is defined as $(x, 0)$ or $(0, y)$. These special points of $(x, 0)$ and $(0, y)$ can not be randomized by RPC or RC since they still have a zero value such as $(r^2x, 0, r)$ (resp. $(rx, 0, r)$) and $(0, r^3y, r)$ (resp. $(0, ry, r)$) in Jacobian (resp. Projective) coordinate after conversion. ES can resist RPA because an attacker cannot handle an elliptic curve point in such a way that a special point with zero value can appear during an execution.

2.3 A Review of Mamiya-Miyaji-Morimoto-Algorithm

We briefly review MMM-algorithm [14]. Assume that the size of the underlying filed and the scalar k are n bits. MMM-algorithm first chooses a random initial point (RIP) R, computes $kP+R$ from left to right without any branch instruction dependent on the data being processed, subtracts R from a result, and gets kP. By using a random initial point at each execution of exponentiation, any point or

any register used in addition formulae changes at each execution, which prevents an attacker from controlling a point P itself as he needs. Thus, it is resistant to SPA, DPA, and RPA[3].

Let us briefly review MMM-algorithm (See Figure 1). First divide n bits into h blocks and choose a random point $R \in E(\mathbb{F}_p)$. Then, MMM-algorithm computes $kP + R$ by representing $kP + R$ to $(1\underbrace{\overline{11}\cdots\overline{11}}_{b})R + (\underbrace{k_{n-1}\cdots}_{b}\cdots\underbrace{\cdots k_1 k_0}_{b})P$ and executing the $(h+1)$-simultaneous scalar multiplication, where $b = \lceil \frac{n}{h} \rceil$. The simultaneous scalar multiplication first constructs a table of $(2^h - 1)$ points $T[0, s]$ for an h-bit integer $s = \sum_{\ell=0}^{h-1} s_\ell 2^\ell$ $(s_\ell \in \{0, 1\})$,

$$T[0, s] = \sum_{\ell=0}^{h-1} s_\ell 2^{b\ell} P - R,$$

and then repeats 1 addition to a table point $T[0, s]$ and 1 doubling from left to right in a b-bit block. The following is the detailed algorithm. In this paper, we also denote the algorithm by h-MMM-algorithm where we want to describe the number of divisions h.

Algorithm 2 (h-MMM-algorithm)
`Input:` $k = \sum_{i=0}^{n-1} k[i]2^i$, P
`Output:` kP
`0.` $k_j = \sum_{\ell=0}^{h-1} k[b\ell + j]2^\ell$ $(j \in \{0, \cdots, b-1\})$.
`1.` $T[0] =$`randompoint()`.
`Table construction`
`2. Compute` $B[i] = 2^{bi}P$ `for` $0 \le i \le (h-1)$.
`3. Compute` $T[0, s] = \sum_{\ell=0}^{h-1} s_\ell B[\ell] - T[0]$ `for an` h`-bit integer` $s = \sum_{\ell=0}^{h-1} s_\ell 2^\ell$ $(s_\ell \in \{0, 1\})$. `Therefore,` $T[0, 0] = -T[0]$.
`Main computation`
`4. For` $j = b - 1$ `to 0:` $T[0] = 2T[0] + T[0, k_j]$.
`5. Output` $T[0] + T[0, 0]$.

3 Generalized MMM-Algorithm

In this section, we give the generalized MMM-algorithm to improve the flexibility of tables as well as computational complexity, while it can resist SPA, DPA, and RPA. Our idea is inspired by an exponentiation algorithm with a fixed point [13,12][4]. Let k be n bits, h and v be positive integers, $\lceil \frac{n}{h} \rceil = a$, and $\lceil \frac{a}{v} \rceil = b$. Let

[3] An SPA in the chosen-message-attack scenario [19] that uses a point P with order 2 is applied to MMM-algorithm. However, we can easily avoid this attack by checking $2P \neq \mathcal{O}$ before computing kP.

[4] The fixed-point exponentiation algorithm divides an exponent k into $h \times v$ blocks and makes a pre-computed table based on the blocks. The application to elliptic curves is discussed in [5]. MMM-algorithm could be considered as a combination of the exponentiation algorithm with $h \times 1$ blocks and a random initial point.

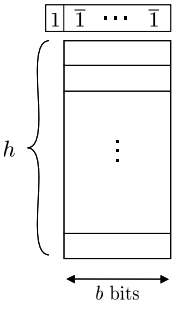


Fig. 1. MMM-algorithm

us also use notation such as $t(2\mathcal{A} = \mathcal{J})$, $t(2^w\mathcal{J})$, $t(\mathcal{J} \to \mathcal{A})_{nI}$, $t(\mathsf{Minv}[n])$, and so on, defined in Section 2.

3.1 Algorithm Intuition

MMM-algorithm computes $kP + R$ by dividing k into $h \times 1$ blocks and executing the $(h+1)$-simultaneous-scalar multiplication of b-bit numbers, where $\lceil \frac{n}{h} \rceil = a$ and $\lceil \frac{a}{1} \rceil = b$. Our target is to generalize MMM-algorithm by dividing k into $h \times v$ blocks and executing the $(h+1)$-simultaneous-scalar multiplication of b-bit numbers. Therefore, MMM-algorithm is a special case of our generalization with $(h, v) = (h, 1)$.

Then, the issues to resolve are: how to embed a random point R into $h \times v$ blocks efficiently; and how to find the optimal (h, v) for the generalized MMM-algorithm together with the best coordinates. Regarding the former issue, one way is to use v random points R_i, compute $kP + R_1 + \cdots R_v$, and subtract $R_1 + \cdots + R_v$ from a result. Another way, which reduces the computational complexity, is to use a random point R in the same way as MMM-algorithm, compute $kP + vR$, and subtract vR. Regarding the latter issue, the best combination of coordinates in table construction, table points themselves, and main computation should be investigated for each (h, v) from the point of view of both computational and memory complexity.

3.2 GMMM-Algorithm

Here we show the generalized MMM-algorithm, which is called GMMM-algorithm in this paper. The brief idea of GMMM-algorithm is described in Figure 2.

Algorithm 3 (**(h, v)-GMMM-algorithm**)
`Input:` $k = \sum_{i=0}^{n-1} k[i]2^i$, P
`Output:` kP
`0.` $k_{i,j} = \sum_{\ell=0}^{h-1} k[a\ell + j + bi]2^\ell$ $((i, j) \in \{0, \cdots, v-1\} \times \{0, \cdots, b-1\})$.
`1. For` $i = 0$ `to` $v - 1$: $R[i]$ =`randompoint()`.

Table Construction
2. Compute $B[i,\ell] = 2^{a\ell+bi}P$ for $0 \le i \le (v-1)$ and $0 \le \ell \le (h-1)$. Then $B[0,0] = P$.
3. Compute $T[i,s] = \sum_{\ell=0}^{h-1} s_\ell B[i,\ell] - R[i]$ for $0 \le i \le (v-1)$ and an h-bit integer $s = \sum_{\ell=0}^{h-1} s_\ell 2^\ell$. Then $T[i,0] = -R[i]$ for $0 \le i \le (v-1)$.
Main computation
4. Initialization: $T[0] = R[0] + \cdots + R[v-1]$. $T[1] = -T[0]$.
5. For $j = b-1$ to 0 by -1
main-loop: $T[0] = 2T[0] + \sum_{i=0}^{v-1} T[i, k_{i,j}]$.
6. Finalization: $T[0] = T[0] + T[1]$.
7. Output $T[0]$.

1. To reduce the computational and memory complexity, 1 random point R would be used instead of v points $\{R[i]\}$. In this case, the initialization step in Algorithm 3 is changed to $T[0] = vR$; and only $T[0,0] = -R$ in the step 3 in Algorithm 3 is kept during the execution.
2. In the current GMMM-algorithm, one fixed power-consumption pattern is observed for any k with a bit length of $n \le bvh$ for the sake of a brief description. However, GMMM-algorithm can be described in such a way that 1 addition is saved for the first $bv - a$ rounds if the $(v-1)$-th block are not full.
3. GMMM-algorithm can also work with two divisions of $(h_1, v_1) \times (h_2, v_2)$ to give the further wide range of time-memory tradeoffs, which will be shown in the final version of this paper.

Theorem 1 (Correctness). kP P $k = \sum_{i=0}^{n-1} k[i]2^i$

Proof: For elliptic-curve points P, $R[0], \cdots, R[v-1]$ and a scalar k, set $B[i,\ell] = 2^{a\ell+bi}P$ for $0 \le i \le (v-1)$ and $0 \le \ell \le (h-1)$, $T[i,s] = \sum_{\ell=0}^{h-1} s_\ell B[i,\ell] - R[i]$ for

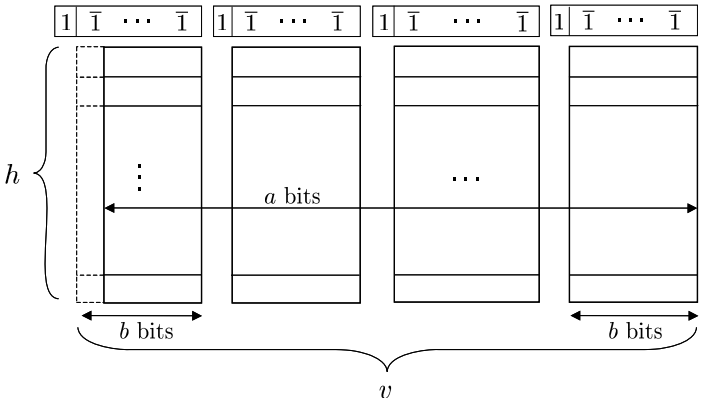


Fig. 2. Generalized MMM-algorithm

$0 \le i \le (v-1)$ and an h-bit integer $s = \sum_{\ell=0}^{h-1} s_\ell 2^\ell$, $k_{i,j} = \sum_{\ell=0}^{h-1} k[a\ell + j + bi]2^\ell$ ($(i,j) \in \{0,\cdots,v-1\} \times \{0,\cdots,b-1\}$), and $T[0] = R[0] + \cdots + R[v-1]$. Then, by describing 1 as a $(b+1)$-bit integer such as $1 = 1\underbrace{\overline{11}\cdots\overline{11}}_{b}$, we get

$$\begin{aligned} kP + R[0] + \cdots + R[v-1] &= T[0] + \sum_{j=0}^{b-1}\sum_{i=0}^{v-1}\left(\sum_{\ell=0}^{h-1} k[a\ell+bi+j]2^{a\ell+bi+j}P - 2^j R[i]\right) \\ &= T[0] + \sum_{j=0}^{b-1} 2^j \sum_{i=0}^{v-1}\left(\sum_{\ell=0}^{h-1} k[a\ell+bi+j]2^{a\ell+bi}P - R[i]\right) \\ &= T[0] + \sum_{j=0}^{b-1} 2^j \sum_{i=0}^{v-1} T[i,k_{i,j}]. \end{aligned}$$

Therefore, the `main-loop` in GMMM-algorithm computes $kP + R[0] + \cdots + R[v-1]$ and, thus, GMMM-algorithm can compute kP correctly. □

Theorem 2 (Security).

Proof: GMMM-algorithm lets the power-consumption pattern be fixed regardless of the bit pattern of a secret key k, and thus it is resistant to SPA. GMMM-algorithm makes use of a random initial point at each execution and let all variables $\{T[i,s]\}$ be dependent on the random point. Thus, an attacker cannot control a point in such a way that it outputs a special point with a zero-value coordinate or zero-value register. Therefore, if $\{R[i]\}$ is chosen randomly by some ways, GMMM-algorithm can be resistant to DPA and RPA. □

In order to enhance the security against address-bit DPA[5] (ADPA) and an SPA in the chosen-message-attack scenario, the same discussion as MMM-algorithm holds in GMM-algorithm.

3.3 The Optimal Division with the Best Coordinate

Both MMM- and GMMM-algorithms aim at a random-point scalar multiplication and, thus, each execution starts with the table construction. Therefore, both are evaluated by the total complexity of the table construction and main computation. MMM-algorithm has employed Jacobian coordinate in the whole procedures. Therefore, any pre-computed point is given in Jacobian coordinate as well as any computation being done in Jacobian coordinate. However, it should be revisited. Because the computational complexity of addition in Jacobian coordinate is considerably large even if iterated doublings in Jacobian coordinate compute the table construction efficiently (see Table 2.1).

Let us investigate the optimal (h,v) with the best coordinates in GMMM-algorithm by separating two phases of table construction and main computation.

[5] ADPA is one of DPA, which uses the leaked information from the address bus and can be applied on such algorithms that fix the address bus during execution.

The table-construction phase computes, first, hv base points $B[i,\ell]$ by iterated doublings and, then, $(2^h - 1 - hv)$ points of $T[i,s] = \sum_{\ell=0}^{h-1} s_\ell B[i,\ell] - R[i]$ by only additions. As for the iterated doublings, Jacobian coordinate is the best coordinate as we have described the above. However, the base points themselves should be converted into affine coordinate to reduce the computational complexity of the next computation of $T[i,s]$. For the conversion from Jacobian to affine coordinate, we can apply Montgomery trick (Algorithm 1). Then, all base points $B[i,\ell]$ can be transformed into affine coordinate with the computational complexity of $3(2hv-3)M + (hv-1)S + I$. The computation of $T[i,s]$ can be executed simultaneously for each hamming weight of $s = \sum_{\ell=0}^{h-1} s_\ell 2^\ell$. Therefore, affine coordinate with the Montgomery trick would be the best. By the above two procedures, we get a pre-computed table with affine coordinate. On the other hand, the main computation repeats additions to each pre-computed point $T[i,s]$ with affine coordinate and 1 doubling. Therefore, there exist two methods. One is mixed coordinate, in which main computation is done in Jacobian coordinate while pre-computed points are given in affine coordinate. The other is affine coordinate with the Montgomery trick, in which main computation is done in affine coordinate by applying the Montgomery trick in each iteration. Then, only 1 inversion is required in each iteration.

Let us summarize the above discussion as follows:

- `Table construction:`
  - `Repeated-doubling phase:` Jacobian coordinate,
  - `Simultaneous-addition phase:` affine coordinate with the Montgomery trick,
- `Main computation:`
  - **case 1:** mixed coordinates of Jacobian and affine coordinates,
  - **case 2:** affine coordinate with the Montgomery trick.

The best combination of coordinates depends on (h,v) and the ratio of I/M, which will be shown in Section 3.4.

3.4 Performance

Let us discuss the memory and computational complexity of (h,v)-GMMM-algorithm in both cases 1 and 2, where the case 1 constructs a table by repeated doublings in Jacobian coordinate and simultaneous additions in affine coordinate with the Montgomery trick and executes the main computation in mixed coordinates of Jacobian and affine coordinates; and the case 2 constructs a table in the same way as the case 1 and executes the main computation in affine coordinate with the Montgomery trick. To make the discussion simple, we assume that 1 random point is used for 1 execution in the same way as MMM-algorithm.

As for the memory complexity, a table with $(2^h - 1)v$ points are required, which are represented in affine coordinate in both cases. In addition, 3 points of $T[0,0]$, $T[0]$, and $T[1]$ are used. All these points are given in affine coordinate in the case 1, while, in the case 2, only $T[0,0]$ is given in affine coordinate and the others are given in Jacobian coordinate.

Let us investigate the computational complexity by separating two phases of the table construction and the main computation. First, let us discuss the table construction phase, in which both cases 1 and 2 employ the same procedure. The table construction consists of the repeated-doubling part and the simultaneous-addition part. The repeated-doubling part computes $\{B[i,\ell]\}$ by executing the iterated doublings in Jacobian coordinate and transforming their results in Jacobian coordinate to affine coordinate. Thus, the total computational complexity of the repeated-doubling part is

$$t(2\mathcal{A}{=}\mathcal{J}){+}t(2^{b(v-1)+a(h-1)-1}\mathcal{J}){+}(hv{-}1)t(\mathcal{J}{\to}\mathcal{A})_{nI}{+}t(\mathsf{Minv}[hv{-}1]) \text{ if } vh \neq 1.$$

In the case of $vh = 1$, we can skip the repeated-doubling part. The simultaneous-addition phase computes $T[i,s] = \sum_{\ell=0}^{h-1} s_\ell B[i,\ell] - R[i]$ simultaneously for each hamming weight of $s = \sum_{\ell=0}^{h-1} s_\ell 2^\ell$. Therefore, it starts with the hamming weight 1 of s, that is, computes $\{B[i,\ell] - R[i]\}_{\ell,i}$ simultaneously by executing additions in affine coordinate together with Montgomery's trick. Thus, the total computational complexity of the simultaneous-addition part is

$$\sum_{i=1}^{h} \left(v\binom{h}{i} t(A+A)_{nI} + t(\mathsf{Minv}[v\binom{h}{i}]) \right),$$

where $\binom{h}{i}$ means the combination to choose i elements from h elements. Then, we've got a precomputed table in affine coordinate.

Let us discuss the main-computation phase, in which each case employs each different procedure. The main-computation phase consists of initialization, main loop, and finalization (step 4, 5, and 6 in Algorithm 3, respectively). Let us focus on the case 1, that is the mixed coordinates of Jacobian and affine coordinates. The computational complexity of the main loop is

$$a \cdot t(\mathcal{J} + \mathcal{A} = \mathcal{J}) + b \cdot t(2\mathcal{J}).$$

The computational complexity of initialization (resp. finalization) is that to compute $T[0] = vR$ for R in affine coordinate giving a result in Jacobian coordinate (resp. $T[0] + T[1]$ of points in Jacobian coordinate giving a result in affine coordinate).

Let us focus on the case 2 of affine coordinate with the Montgomery trick. As for the computation of the main-loop, a tournament structure with $v+2$ leaves is applied, where v points of $T[i,k_{i,j}]$ and doubling points of $T[0]$ are in the leaf of the tournament structure (see Figure 3). We pairwise add points at leaves with a common parent, give its sum to the parent node, and carry out these procedures at each level to the root. By applying the Montgomery's trick to each level, only 1 inversion is required in each level. The simplest case is the binary tree case, which is described in [15]. Then, the computational complexity of the main loop is

$$a \cdot t(\mathcal{A}+\mathcal{A})_{nI} + b \cdot t(2\mathcal{A})_{nI} + b \cdot t(\mathsf{TournaMinv}[v+2]),$$

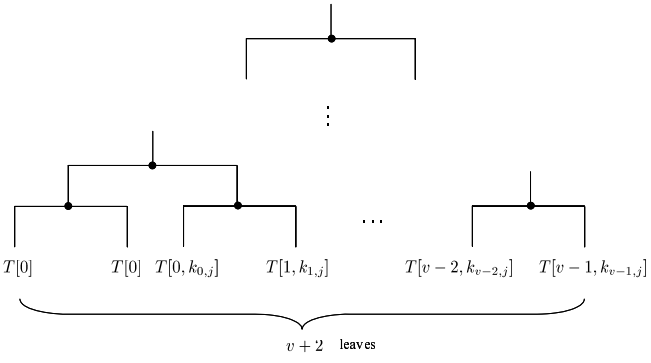


Fig. 3. Main computation of GMMM-algorithm in the case 2

where $t(\mathsf{TournaMinv}[v+2])$ denotes the computational complexity to get all inversions of $(v+2)$-point summation according to the tournament structure with $(v+2)$ leaves. The computational complexity of initialization (resp. finalization) is that to compute $T[0] = vR$ of R in affine coordinate giving a result in affine coordinate (resp. $T[0]+T[1]$ of points in affine coordinate giving a result in affine coordinate).

The above discussion is summarized in the following theorem.

Theorem 3. (h, v)

$$
\begin{aligned}
Comp &= t(2\mathcal{A} = \mathcal{J}) + t(2^{b(v-1)+a(h-1)-1}\mathcal{J}) + (hv-1)t(\mathcal{J} \to \mathcal{A})_{nI} + t(\mathsf{Minv}[hv-1]) \\
&\quad + \textstyle\sum_{i=1}^{h} \left(v\binom{h}{i} t(A+A)_{nI} + t(\mathsf{Minv}[v\binom{h}{i}])\right) + t(v\mathcal{A} = \mathcal{J}) \\
&\quad + a \cdot t(\mathcal{J} + \mathcal{A} = \mathcal{J}) + b \cdot t(2\mathcal{J}) + t(\mathcal{J} + \mathcal{J} = \mathcal{A}) \ \ (\textit{if } vh \neq 1). \\
Comp &= t(\mathcal{A} + \mathcal{A}) + a \cdot t(\mathcal{J} + \mathcal{A} = \mathcal{J}) + b \cdot t(2\mathcal{J}) + t(\mathcal{J} + \mathcal{A} = \mathcal{A}) \ \ (\textit{if } vh = 1).
\end{aligned}
$$

ffi

$$
\begin{aligned}
Comp &= t(2\mathcal{A} = \mathcal{J}) + t(2^{b(v-1)+a(h-1)-1}\mathcal{J}) + (hv-1)t(\mathcal{J} \to \mathcal{A})_{nI} + t(\mathsf{Minv}[hv-1]) \\
&\quad + \textstyle\sum_{i=1}^{h} \left(v\binom{h}{i} t(A+A)_{nI} + t(\mathsf{Minv}[v\binom{h}{i}])\right) + t(v\mathcal{A}) \\
&\quad + a \cdot t(\mathcal{A} + \mathcal{A})_{nI} + b \cdot t(2\mathcal{A})_{nI} + b \cdot t(\mathsf{TournaMinv}[v+2]) + t(\mathcal{A}+\mathcal{A}) \ \ (\textit{if } vh \neq 1),
\end{aligned}
$$

$\lceil \frac{n}{h} \rceil = a \ \lceil \frac{a}{v} \rceil = b$ $t(v\mathcal{A})$, $t(v\mathcal{A} = \mathcal{J})$

v ffi ffi

4 Comparison

In this section, we compare our algorithm with the previous countermeasures to SPA, DPA, and RPA, those are ES [3], randomized window algorithm [16], LRIP [11], and MMM-algorithm [14]. Here, M, S, or I represents the computation amount of modular multiplication, square, or inversion, respectively. In order to make comparison easier, the computation complexity is also estimated in terms

of M and I by assuming that $S = 0.8M$ as usual and that $S = 0.8M$ and $I = 4M$ or $I = 11M$ (typically the ratio[6] I/M is between 4 and 11). Memory complexity is evaluated by the number of finite-field elements, where 1 point in Jacobian (resp. affine) coordinate consists of 3 (resp. 2) field elements.

Table 4.1 shows the computational and memory complexity of GMMM-algorithm with cases 1 and 2 in the case of a 160-bit scalar. These are arranged in ascending order of memory. Therefore, if we focus on either case of GMMM-algorithm, these are also arranged in descending order of computational complexity. However, the case 1 has advantage over the case 2 if the ratio of inversion over multiplication is rather large and, thus, better case depends on the ratio of I/M. We also compute a break-even point for the borderline between both cases. The break-even point shows I/M when the computational complexity of GMMM-algorithm with the case 1 is equal to that with the case 2 under $S = 0.8M$. Thus, if I/M is smaller than the indicated value, GMMM-algorithm with the case 2 is more efficient than that with the case 1. For example, $(3, 1)$-GMMM with the case 1 is more efficient than $(3, 2)$-GMMM with the case 2 if and only if $I/M > 8.4$. Each division of $(1, 1)$, $(2, 1)$, $(3, 1)$ or $(4, 1)$-GMMM-algorithm corresponds to that of 1, 2, 3, or 4-MMM-algorithm, respectively. The difference is: MMM-algorithm uses Jacobian coordinate in the whole execution but GMMM-algorithm with the case 1 employs mixed coordinate. Note that GMMM-algorithm with the case 2 can not be applied to these cases.

Table 4.2 shows the computational and memory complexity of previous algorithms in the case of a 160-bit scalar, where Jacobian coordinate is used for the whole computation and a result is transformed into affine coordinate according to their original proposals.

By generalizing MMM-algorithm to GMMM-algorithm, we see that more flexibility, that is a wider range of time-memory tradeoffs, can be realized. Furthermore, the optimization of coordinates can reduce both the computational complexity and memory even if both MMM and GMMM-algorithms use the same division. In fact, GMMM-algorithm performs better than any previous method under the above realistic assumptions concerning the ratio I/M. For example, (1,1)-GMMM-algorithm with the case 1 can reduce the computational complexity of 1-MMM-algorithm by 19% over for the range of I/M and also reduce the memory size. (4,1)-GMMM-algorithm with the case 1 can reduce the computational complexity of 4-MMM-algorithm by 13.2% over for the range of I/M and also reduce the memory by 25.4 %. Note that 4-MMM-algorithm is the most efficient case in MMM-algorithm[7]. However, even $(2, 2)$-GMMM-algorithm with the case 2 or $(3, 1)$-GMMM-algorithm with the case 1 can work faster than 4-MMM-algorithm under the range of $I/M < 7.9$ or 49.3 and also reduce the memory by 64.8% or 62.8%, respectively.

[6] Generally, the ratio of I/M depends on the algorithm used and the size and type of the finite field. The ratio in [7,2] (resp. [6]) is between 4 and 10 (resp. 4 and 11) . So here we adopt the wider range. A discussion on the ratio can be found in [4].

[7] 5-MMM-algorithm is not as efficient as 4-MMM-algorithm although it requires more memory than 4-MMM-algorithm.

Table 4.1. Performance of GMMM-algorithm (160 bits)

division (h, v) case 1 or 2	Computational complexity				Memory[‡]	$I/M^{\star}$
		$S = 0.8M$	$I = 4M$	$I = 11M$		
$(1,1)^{\dagger}$-case 1	$1933M + 1445S + 2I$	$3089M + 2I$	$3097M$	$3111M$	7	
$(1,2)$-case 2	$1052M + 648S + 164I$	$1570.4M + 164I$	$2226.4M$	$3374.4M$	10	
$(1,2)$-case 1	$1945M + 1294S + 3I$	$2980.2M + 3I$	$2992.2M$	$3013.2M$	12	8.8
$(2,1)^{\dagger}$-case 1	$1305M + 1051S + 4I$	$2145.8M + 4I$	$2161.8M$	$2189.8M$	14	
$(2,2)$-case 2	$881M + 654S + 85I$	$1404.2M + 85I$	$1744.2M$	$2339.2M$	18	
$(2,2)$-case 1	$1334M + 980S + 4I$	$2118M + 4I$	$2134M$	$2162M$	20	8.8
$(3,1)^{\dagger}$-case 1	$1128M + 934S + 5I$	$1875.2M + 5I$	$1895.2M$	$1930.2M$	22	
$(3,2)$-case 2	$873M + 672S + 60I$	$1410.6M + 60I$	$1650.6M$	$2070.6M$	34	8.4
$(4,1)^{\dagger}$-case 1	$1051M + 865S + 6I$	$1743M + 6I$	$1767M$	$1809M$	38	
$(4,2)$-case 2	$919M + 682S + 47I$	$1464.6M + 47I$	$1652.6M$	$1981.6M$	66	6.8

† : They correspond to h-MMM-algorithm. ‡ : # field elements. * : break-even point.

Table 4.2. Comparison of known countermeasures (160 bits)

	Computational complexity			Memory
		$S = 0.8M$	$I = 11M$	
ES [3]	$2563M + 1601S + I$	$3843.8M + I$	$3854.8M$	$14^{\diamond}$
strengthened window [16]	$1643M + 1298S + I$	$2681.4M + I$	$2692.4M$	15
LRIP [11]	$2563M + 1283S + I$	$3589.4M + I$	$3600.4M$	12
1-MMM [14]	$2563M + 1601S + I$	$3843.8M + I$	$3854.8M$	9
2-MMM	$1651M + 1139S + I$	$2562.2M + I$	$2573.2M$	15
3-MMM	$1395M + 1007S + I$	$2200.4M + I$	$2211.4M$	27
4-MMM	$1315M + 947S + I$	$2072.4M + I$	$2073.4M$	51

$^{\diamond}$: Strictly, it needs 4 elliptic curve points and 2 field elements.

5 Conclusion

In this paper, we have generalized MMM-algorithm, which is the secure scalar multiplication with using a random initial point. Our improved algorithm is significantly efficient and flexible and can work efficiently even when the storage available is very small or quite large. We have also given the formulae of the computational complexity for any division of the proposed algorithm theoretically, which helps developers to choose the best division suitable for the storage available.

References

1. Avanzi, R., Cohen, H., Doche, C., Frey, G., Lange, T., Nguyen, K., Vercauteren, F.: Handbook of Elliptic and Hyperelliptic Curve Cryptography. Chapman & Hall/CRC (2006)
2. Blake, I.F., Seroussi, G., Smart, N.P.: Elliptic Curves in Cryptology. In: LMS, vol. 265. Cambridge University Press, Cambridge (1999)

3. Ciet, M., Joye, M.: (Virtually) Free randomization technique for elliptic curve cryptography. In: Qing, S., Gollmann, D., Zhou, J. (eds.) ICICS 2003. LNCS, vol. 2836, pp. 348–359. Springer, Heidelberg (2003)
4. Ciet, M., Joye, M., Lauter, K., Montgomey, P.L.: Trading inversions for multiplications in elliptic curve cryptography. Designs, Codes and Cryptography 39(2), 189–206 (2006)
5. Cohen, H., Miyaji, A., Ono, T.: Efficient elliptic curve exponentiation. In: Han, Y., Quing, S. (eds.) ICICS 1997. LNCS, vol. 1334, pp. 282–290. Springer, Heidelberg (1997)
6. Cohen, H., Miyaji, A., Ono, T.: Efficient elliptic curve exponentiation using mixed coordinates. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 51–65. Springer, Heidelberg (1998)
7. Doche, C., Icart, T., Kohel, D.R.: Efficient scalar multiplication by isogeny decompositions. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 191–206. Springer, Heidelberg (2006)
8. Eisenträger, K., Lauter, K., Montgomey, P.L.: Fast elliptic curve arithmetic and improved Weil pairing evaluation. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 343–354. Springer, Heidelberg (2003)
9. Goubin, L.: A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 199–210. Springer, Heidelberg (2002)
10. Itoh, K., Takenaka, M., Torii, N., Temma, S., Kurihara, Y.: Fast implementation of public-key cryptography on DSP TMS320C6201. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 61–72. Springer, Heidelberg (1999)
11. Itoh, K., Izu, T., Takenaka, M.: Efficient countermeasures against power analysis for elliptic curve cryptosystems. In: Proceedings of CARDIS 2004, pp. 99–114. Kluwer, Dordrecht (2004)
12. Lim, C.H., Lee, P.J.: More flexible exponentiation with precomputation. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 95–107. Springer, Heidelberg (1994)
13. Pippenger, N.: On the evaluation of powers and related problems (preliminary version). In: 17th annual symposium on foundations of computer science, pp. 258–263. IEEE Computer Society, Los Alamitos (1976)
14. Mamiya, H., Miyaji, A., Morimoto, H.: Secure elliptic curve exponentiation against RPA, ZRA, DPA, and SPA. IEICE Trans. Fundamentals E89-A(8), 2207–2215 (2006)
15. Mishra, P.K., Sarkar, P.: Application of Montgomery's trick to scalar multiplication for EC and HEC using fixed base point. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 41–57. Springer, Heidelberg (2004)
16. Möller, B.: Parallelizable elliptic curve point multiplication method with resistance against side-channel attacks. In: Chan, A.H., Gligor, V.D. (eds.) ISC 2002. LNCS, vol. 2433, pp. 402–413. Springer, Heidelberg (2002)
17. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods for factorization. Mathematics of Computation 48, 243–264 (1987)
18. Standard for efficient cryptography group, specification of standards for efficient cryptography, available from: http://www.secg.org
19. Yen, S.M., Lien, W.C., Moon, S., Ha, J.: Power analysis by exploiting chosen message and internal collisions - Vulnerability of checking mechanism for RSA-Decryption. In: Dawson, E., Vaudenay, S. (eds.) Mycrypt 2005. LNCS, vol. 3715, pp. 183–195. Springer, Heidelberg (2005)

Pairing-Friendly Elliptic Curves with Small Security Loss by Cheon's Algorithm

Aya Comuta[1], Mitsuru Kawazoe[2], and Tetsuya Takahashi[2]

[1] Graduate School of Science
Osaka Prefecture University
[2] Faculty of Liberal Arts and Sciences
Osaka Prefecture University
1-1 Gakuen-cho Naka-ku Sakai Osaka 599-8531 Japan
{kawazoe,takahasi}@las.osakafu-u.ac.jp

Abstract. Pairing based cryptography is a new public key cryptographic scheme. An elliptic curve suitable for pairing based cryptography is called a "pairing-friendly" elliptic curve. After Mitsunari, Sakai and Kasahara's traitor tracing scheme and Boneh and Boyen's short signature scheme, many protocols based on pairing-related problems such as the q-weak Diffie-Hellman problem have been proposed. In Eurocrypt 2006, Cheon proposed a new efficient algorithm to solve pairing-related problems and recently the complexity of Cheon's algorithm has been improved by Kozaki, Kutsuma and Matsuo. Due to these two works, an influence of Cheon's algorithm should be considered when we construct a suitable curve for the use of a protocol based on a pairing-related problem. Among known methods for constructing pairing-friendly elliptic curves, ones using cyclotomic polynomials are affected by Cheon's algorithm. In this paper, we study how to reduce a security loss of a cyclotomic family by Cheon's algorithm.

Keywords: Pairing based cryptosystem, Elliptic curves.

1 Introduction

Pairing based cryptography is a new public key cryptographic scheme, which was proposed around 2000 by three important works due to Joux [15], Sakai, Ohgishi and Kasahara [22] and Boneh and Franklin [5]. In these last two papers, the authors constructed an identity-based encryption scheme by using the Weil pairing of elliptic curves. An elliptic curve suitable for pairing-based cryptography is called a "pairing-friendly" elliptic curve. It is very important to find an efficient method to construct pairing-friendly elliptic curves. There are many works on this topic: Miyaji, Nakabayashi and Takano [19], Cocks and Pinch [9], Brezing and Weng [8], Barreto and Naerig [1], Scott and Barreto [21], Freeman, Scott and Teske [12] and so on.

In 2002, Mitsunari, Sakai and Kasahara proposed a new traitor tracing scheme based on the q-weak Diffie-Hellman problem [18]. The q-weak Diffie-Hellman

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 297–308, 2007.

problem is described as follows: Let g be an element of prime order ℓ in an abelian group and $\alpha \in (\mathbb{Z}/\ell\mathbb{Z})^*$. Then the q-weak Diffie-Hellman problem asks $[1/\alpha]g$ for given g, $[\alpha]g$, $[\alpha^2]g$,..., $[\alpha^q]g$. In 2004, Boneh and Boyen proposed a short signature scheme based on the q-strong Diffie-Hellman problem. After these two works, many protocols have been proposed based on q-weak Diffie-Hellman-like problems: [2], [3], [4], [20] and so on.

Before 2006, there had been no known efficient algorithm to solve the discrete logarithm problem related to the above protocols which works faster than the rho method and the square root method. However, in Eurocrypt 2006, Cheon proposed a new efficient algorithm which computes the discrete logarithm of the q-strong Diffie-Hellman problem [7]. Let ℓ be a group order, and g, $[\alpha]g$, $[\alpha^d]g$ given elements where d is a positive divisor of $\ell - 1$. Cheon's algorithm can compute α from these data in $O(\log \ell(\sqrt{\ell/d} + \sqrt{d}))$ group operations. In the same paper, Cheon gave an algorithm which computes α for given g, $[\alpha^i]g$, $i = 1, 2, \ldots, 2d$ in $O(\log \ell(\sqrt{\ell/d}+d))$ group operations when d is a divisor of $\ell+1$. Recently, Kozaki, Kutsuma and Matsuo showed that the complexity of Cheon's algorithm can be reduced to $O(\sqrt{\ell/d} + \sqrt{d})$ for $d \,|\, (\ell - 1)$ and $O(\sqrt{\ell/d} + d)$ for $d \,|\, (\ell + 1)$, respectively [16]. It is obvious that the complexity is reduced when $d(< \sqrt{\ell})$ becomes larger. When $d = O(\ell^{1/2})$, the cost of an $(\ell-1)$-version becomes $O(\ell^{1/4})$. And when $d = O(\ell^{1/3})$, the cost of an $(\ell + 1)$-version becomes $O(\ell^{1/3})$. Those are much smaller than the rho method and the square root method. Hence, one should be careful about a divisor of $\ell \pm 1$ of the order ℓ of an elliptic curve used for protocols based on the q-weak Diffie-Hellman problem, the q-strong Diffie-Hellman problem or other related problems. There are many methods to construct pairing-friendly elliptic curves over a finite field $\mathbb{F}_p$. However, in all methods for constructing pairing-friendly elliptic curves except for the Cocks-Pinch method, the order of an elliptic curve is given by an irreducible polynomial $\ell(x)$. If $\ell(x) \pm 1$ is reducible, there is a big security loss due to Cheon's algorithm. In fact, for an example of the embedding degree $k = 10$ in [11], $k = 12$ in [1], all examples in [8], and curves obtained by using cyclotomic fields in [12], the polynomial $\ell(x) + 1$ or $\ell(x) - 1$ has a polynomial factor of degree 1 or 2. Roughly speaking, if $\ell \pm 1$ has a factor d, the cost for solving the q-strong Diffie-Hellman problem is reduced by a factor $\sqrt{d}$. So when we take ℓ as the k-th cyclotomic polynomial $\Phi_k(x)$ as in [8] and so on, there is at least $(\lg \ell)/(2\varphi(k))$-bit security loss by Cheon's algorithm where $\lg x := \log_2 x$ and φ is the Euler phi function. Though the advantage of a cyclotomic family is that one can take the ratio $\rho = \lg p / \lg \ell$ less than two, these are affected by Cheon's algorithm for the use of protocols based on pairing-related problems.

In this paper, we study how to reduce a security loss of a cyclotomic family by Cheon's algorithm keeping its advantage for the value of ρ. To reduce the security loss by Cheon's algorithm, we propose to take a prime ℓ as a large proper divisor of $\Phi_k(x)$. However, for example, in the Freeman-Scott-Teske method which needs an extension of a cyclotomic field $\mathbb{Q}(\zeta_k)$ and uses $\Phi_{ck}(x)$ as ℓ for $c \geq 2$, it is difficult to find a suitable order ℓ only by taking a proper divisor of $\Phi_{ck}(x)$. It is mainly because the degree of $\Phi_{ck}(x)$ is relatively large. In this paper, we propose

an improved method for the embedding degree $2n$ where n is an odd prime. Since the proposed method does not need a field extension, the probability of finding a pairing-friendly curve by the method is higher than the Freeman-Scott-Teske method. Heuristically, our method gives pairing-friendly elliptic curves whose security loss by Cheon's algorithm is within 5 bits for $k \leq 38$. We note that ρ of constructed curves is still less than two, moreover almost same as the Freeman-Scott-Teske method.

We give the outline of this article. In Section 2, we recall the Weil paring and the condition to construct a secure and efficient pairing based cryptosystem. In Section 3, we recall the q-weak/strong Diffie-Hellman problem and Cheon's algorithm. In Section 4, for known methods of constructing pairing-friendly elliptic curves, we study the effect of Cheon's algorithm on them. In Section 5, we study how to reduce the security loss of cyclotomic methods and give an improved method to construct pairing-friendly elliptic curves with small security loss. We also give examples obtained by using the proposed method. Finally, we summarize our result in Section 6.

2 Pairing Based Cryptosystem

Here, we recall a pairing based cryptosystem using an elliptic curve over a finite prime field. Let p be a prime, $K := \mathbb{F}_p$ a finite field with p elements and E an elliptic curve defined over K. The finite abelian group of K-rational points of E and its order are denoted by $E(K)$ and $\#E(K)$, respectively. Assume that $E(K)$ has a subgroup G of a large prime order. The most simple case is that $E(K) = G$, that is, the order of $E(K)$ is prime. Let ℓ be the order of G. We denote by $E[\ell]$ the group of ℓ-torsion points of $E(\overline{K})$ where $\overline{K}$ is an algebraic closure of K. In the following, we denote $\log_2 x$ by $\lg x$.

For a positive integer ℓ coprime to the characteristic of K, the Weil pairing is a map

$$e_\ell : E[\ell] \times E[\ell] \to \mu_\ell \subset \hat{K}^*$$

where $\hat{K}$ is the field extension of K generated by coordinates of all points in $E[\ell]$, $\hat{K}^*$ is the multiplicative group of $\hat{K}$ and μ_ℓ is the group of ℓ-th roots of unity in $\hat{K}^*$. For the details of the Weil pairing, see [23] for example. The key idea of pairing based cryptography is based on the fact that the subgroup $G = \langle P \rangle$ is embedded into the multiplicative group $\mu_\ell \subset \hat{K}^*$ via the Weil pairing or some other pairing map.

The extension degree of the field extension $\hat{K}/K$ is called the embedding degree of E with respect to ℓ. It is known that E has the embedding degree k with respect to ℓ if and only if k is the smallest integer such that ℓ divides $p^k - 1$. In pairing based cryptography, the following conditions must be satisfied to make a system secure:

- the order ℓ of a prime order subgroup of $E(K)$ should be large enough so that solving a discrete logarithm problem on the group is computationally infeasible and

– p^k should be large enough so that solving a discrete logarithm problem on the multiplicative group $\mathbb{F}_{p^k}^*$ is computationally infeasible.

Moreover for an efficient implementation of a pairing based cryptosystem, the followings are important:

– the embedding degree k should be appropriately small and
– the ratio $\lg p / \lg \ell$ should be appropriately small.

Elliptic curves satisfying the above four conditions are called "pairing-friendly" elliptic curves. In practice, it is currently recommended that ℓ should be larger than 2^{160} and p^k should be larger than 2^{1024}.

3 Protocols Based on Pairing-Related Problem and Cheon's Algorithm

3.1 Pairing-Related Problems

A new traitor tracing scheme proposed by Mitsunari, Sakai and Kasahara [18] in 2002 is based on the q-weak Diffie-Hellman problem. The definition of the q-weak Diffie-Hellman problem is as follows.

Definition 1 (The q-weak Diffie-Hellman problem). Let G be a cyclic group of prime order ℓ. The q-weak Diffie-Hellman problem in G is to compute $[1/\alpha]g \in G$ from a $(q+1)$-tuple $(g, [\alpha]g, [\alpha^2]g, \ldots, [\alpha^q]g)$ where $g \in G$ and $\alpha \in (\mathbb{Z}/\ell\mathbb{Z})^\times$.

In 2004, Boneh and Boyen proposed a short signature scheme based on the q-strong Diffie-Hellman problem. (For the definition of q-strong Diffie-Hellman problem, see [3].) After Mitsunari, Sakai and Kasahara's work [18] and Boneh and Boyen's work [3], many protocols without random oracles have been proposed based on q-weak/strong Diffie-Hellman-like problems, e.g. [2], [4], [20]. In the following, we call such kind of problems the "pairing-related problems".

3.2 Cheon's Algorithm and Its Improvement

In Eurocrypt 2006, Cheon [7] proposed an algorithm to solve the q-weak/strong Diffie-Hellman problem. Very recently, Kozaki, Kutsuma and Matsuo [16] improved the complexity of Cheon's algorithm for the q-weak Diffie-Hellman problem. For an abelian group G of prime order ℓ, if $\ell - 1$ has a positive divisor d less than or equal to q, then their improved algorithm can solve the q-weak Diffie-Hellman problem within $O\left(\sqrt{\ell/d} + \sqrt{d}\right)$ group operations using space for $O\left(\max\left(\sqrt{\ell/d}, \sqrt{d}\right)\right)$ group elements. There also exists an $\ell + 1$ variant of this algorithm. The details of the results in [16] are as follows:

Theorem 1 ([16]). g ℓ d $\ell-1$ g $[\alpha]g$ $[\alpha^d]g$ α $O\left(\sqrt{\ell/d}+\sqrt{d}\right)$ $O\left(\max\left(\sqrt{\ell/d},\sqrt{d}\right)\right)$

Theorem 2 ([16]). g ℓ d $\ell+1$ $[\alpha^i]g$ $i=1,2,\ldots,2d$ α $O\left(\sqrt{\ell/d}+d\right)$ $O\left(\max\left(\sqrt{\ell/d},\sqrt{d}\right)\right)$

In the original result of Cheon [7], the complexity in the above two theorems were given by $O\left(\log \ell\left(\sqrt{\ell/d}+\sqrt{d}\right)\right)$ and $O\left(\log \ell\left(\sqrt{\ell/d}+d\right)\right)$ group operations, respectively.

3.3 The Effect of Cheon's Algorithm for Constructing Pairing-Friendly Elliptic Curves

In this section, we consider the effect of Cheon's algorithm on known methods for constructing pairing-friendly elliptic curves.

With respect to Cocks-Pinch method, the group size ℓ can be randomly chosen. So it is not difficult to avoid the security loss by Cheon's algorithm. See Section 6 of [16] for the details.

Except for Cocks-Pinch method, since the group order ℓ is given by a polynomial $\ell(x)$, we should be careful about the effect of Cheon's algorithm. Roughly speaking, if $\ell \pm 1$ has a factor d, the cost for solving the q-strong Diffie-Hellman problem is reduced by a factor $\sqrt{d}$. So, if a polynomial $\ell(x) \pm 1$ is reducible and its non-trivial polynomial factor $h(x)$ has a small degree, there is at least a $(\lg h(x_0))/2$ bits security loss by Cheon's algorithm when we take $l = l(x_0)$ for an integer x_0. In the following, we see the security loss by Cheon's algorithm for each method using polynomials.

The MNT method and its variant. In the MNT method based on Miyaji, Nakabayashi and Takano's result [19], the following polynomials are used: $\ell(x) = 12x^2 \pm 6x + 1$ for $k = 3$, $\ell(x) = x^2 + 2x + 2$ or $x^2 + 1$ for $k = 4$ and $\ell(x) = 4x^2 \pm 2x + 1$ for $k = 6$. Except for $\ell(x) = \ell^2 + 2\ell + 2$ in the case $k = 4$, $\ell(x) - 1$ is divisible by x. For the generalized MNT method such as Galbraith, McKee and Valença's method [13], there are some cases that $\ell(x) \pm 1$ are reducible. Since the degree of $\ell(x)$ equals two, this fact does not lead directly that Cheon's algorithm affects the security of each curve.

A Cyclotomic Family. There are some methods using a cyclotomic polynomial as $\ell(x)$, e.g. Brezing and Weng's method [8] and Freedman, Scott and Teske's method [12]. We call these methods a "cyclotomic family". The advantage of a

cyclotomic family is that one can take curves with relatively small $\rho = \lg p / \lg \ell$ (< 2).

All of them use a cyclotomic polynomial to set a prime ℓ as $\ell = \Phi_k(x)$ or $\ell = \Phi_{ck}(x)$ for some $c > 1$ where k is the embedding degree. Then, $\ell - 1$ is factored by x at least. Moreover, if $ck = 2^m$, then $\ell - 1$ is factored by $x^{2^{m-1}}$, otherwise $\ell - 1$ is factored by $x(x+1)$ or $x(x-1)$. The size of x is about $(\lg \ell)/\varphi(ck)$ bits, $c \geq 1$, where φ is the Euler phi function. Hence, if $x < q$ (resp. $x(x+1) < q$), the complexity to solve the q-weak Diffie-Hellman problem is reduced to $O(\sqrt{\ell^{1-1/\varphi(ck)}} + \sqrt{\ell^{1/\varphi(ck)}})$ (resp. $O(\sqrt{\ell^{1-2/\varphi(ck)}} + \sqrt{\ell^{2/\varphi(ck)}})$) group operations.

Other methods. For $k = 10$, Freeman gave the following family [11]: $p(x) = 25x^4 + 25x^3 + 25x^2 + 10x + 3$, $\ell(x) = 25x^4 + 25x^3 + 15x^2 + 5x + 1$ and $Dy^2 = 15x^2 + 10x + 3$. For this family, $\ell(x) \pm 1$ factor as $\ell(x) - 1 = 5x(5x^3 + 5x^2 + 3x + 1)$ and $\ell(x) + 1 = (5x^2 + 1)(5x^2 + 5x + 2)$. The following two examples are given in [11].

$$\ell = 5031898990973855325985710847786081764109733 51$$

$$\ell = 6109996327108312874607376956745050221908714591643483962630 1$$

The former is a 149 bit prime and the latter is a 196 bit prime. For each example, $\ell - 1$ factors as

$$\ell - 1 = 2 \cdot 5^2 \cdot 853 \cdot (\text{a 33 bit prime}) \cdot (\text{a 39 bit prime}) \cdot (\text{a 63 bit prime})$$

$$\ell - 1 = 2^2 \cdot 5^2 \cdot 7 \cdot (\text{a 29 bit prime}) \cdot (\text{a 44 bit prime}) \cdot (\text{a 114 bit prime})$$

respectively. Cheon's algorithm affects each case.

For $k = 12$, Barreto and Naerig gave the following family [1]: $\ell(x) = 36x^4 + 36x^3 + 18x^2 + 6x + 1$, $p(x) = 36x^4 + 36x^3 + 24x^2 + 6x + 1$ and $Dy^2 = 3(6x^2 + 4x + 1)$. For this family, $\ell(x) \pm 1$ factor as $\ell(x) - 1 = x(6x^3 + 6x^2 + 3x + 1)$ and $\ell(x) + 1 = (3x^2 + 3x + 1)(6x^2 + 1)$. The following example is given in [1].

$$\ell = 1461501624496790265145447380994971188499300027613 \text{ (160 bit)}$$

For this example, we have

$$\begin{aligned}\ell - 1 =& 2^2 \cdot 3 \cdot (\text{a 24 bit prime}) \cdot (\text{a 38 bit prime}) \cdot (\text{a 39 bit prime}) \\ & \cdot (\text{a 57 bit prime}) \\ \ell + 1 =& 2 \cdot 7 \cdot 13 \cdot 19 \cdot 1279 \cdot 1861 \cdot 21227 \cdot (\text{a 19 bit prime}) \cdot (\text{a 21 bit prime}) \\ & \cdot (\text{a 24 bit prime}) \cdot (\text{a 50 bit prime}).\end{aligned}$$

Hence Cheon's algorithm affects the security of this example.

4 How to Reduce a Security Loss of a Cyclotomic Family

In this section, we consider the way to reduce the security loss by Cheon's algorithm for a cyclotomic family with embedding degree $2n$ where n is an odd

prime. As we see in the previous section, a cyclotomic family uses the ck-th cyclotomic polynomial $\Phi_{ck}(x)$ as $\ell = \ell(x)$ for finding a pairing-friendly elliptic curve of an embedding degree k. Therefore $\ell(x) - 1$ is factored by x. To reduce the security loss by Cheon's algorithm, we propose to take a prime ℓ as a large proper divisor of $\Phi_k(x)$ for an integer x.

In a cyclotomic family, the best asymptotic ρ value for each embedding degree $k = 2n$ for an odd prime n is given by the Freeman-Scott-Teske method. However, since the Freeman-Scott-Teske method uses $\Phi_{ck}(x)$ for $c \geq 2$, it is difficult to find a suitable order ℓ only by taking a proper divisor of $\Phi_{ck}(x)$. We give the search result using the Freeman-Scott-Teske method in Table 1. In this search, for each k we used D and $\Phi_{ck}(x)$ as shown in [12] which gives the best asymptotic ρ value and searched ℓ satisfying the following condition:

- $\Phi_{ck}(x) = \ell \cdot$ (a positive integer less than 10000)
- $\ell \pm 1 =$ (a positive integer $< 2^{10}$) $\cdot$ (a product of primes $\geq 2^{50}$).

Table 1. The size of the smallest $\ell(\leq 2^{400})$ satisfying the condition

k	14	22	26	34	38
The size of ℓ	180 bits	338 bits	Not found	Not found	Not found

As Table 1 shows, it is difficult to find a suitable ℓ of appropriate size except for $k = 14$. It is mainly because the degree of $\Phi_{ck}(x)$ is relatively large for $c \geq 2$. So it is important to use $\Phi_k(x)$ to search a pairing-friendly elliptic curve of the embedding degree k which is not affected by Cheon's algorithm. In the following section, we propose an improved method for the embedding degree $2n$ where n is an odd prime and we show the efficiency of the proposed method by giving the experimental data.

4.1 The Condition of a Large Prime Factor of $\Phi_{2n}(x)$

First, we study the condition for the large prime factor ℓ of $\Phi_{2n}(x)$. Note that when n is an odd prime, $\Phi_{2n}(x) = \Phi_n(-x)$.

Lemma 1. [illegible] n [illegible] ℓ [illegible] x [illegible] $\Phi_n(x) \equiv 0 \pmod{\ell}$ [illegible] $\ell = n$ [illegible] $\ell \equiv 1 \pmod{n}$

[illegible] Assume that $\Phi_n(x) \equiv 0 \pmod{\ell}$ and $\ell \neq n$. Then $\Phi_n(x) \equiv 0 \pmod{\ell}$ yields that x gives a primitive n-th root of unity in $(\mathbb{Z}/\ell\mathbb{Z})^\times$. Hence n divides $\#(\mathbb{Z}/\ell\mathbb{Z})^\times = \ell - 1$; that is, $\ell \equiv 1 \pmod{n}$. □

Theorem 3. [illegible] k [illegible] $k = 2n$ [illegible] n [illegible] [illegible] x [illegible] ℓ [illegible] [illegible] n [illegible] s [illegible] [illegible] $\Phi_k(x) = s\ell$ [illegible]

[illegible] s [illegible] n [illegible] $x \equiv -1 \pmod{n}$ [illegible] s [illegible] [illegible] n^2
[illegible] $s = n$ [illegible] $\ell - 1$ [illegible] $x + 1$
[illegible] s [illegible] n [illegible] $x \not\equiv -1 \pmod{n}$

In Theorem 3, note that by the assumption $\ell > n$ and Lemma 1, $\ell - 1$ is divisible by n. Moreover, it is easy to see that $\ell^2 - 1$ is divisible by 24. Hence $(\ell+1)(\ell-1)$ is divisible by $24n$. We also note that if s is a small prime which divides $\Phi_k(x)$ for an integer x, then $s = n$ or $s \equiv 1 \pmod{n}$.

First, note that $\ell - 1 = \Phi_k(x)/s - 1 = (\Phi_n(-x) - s)/s$. Second, note that if $x \not\equiv -1$, then $\Phi_k(x) = \Phi_n(-x) = ((-x)^p - 1)/(-x-1) \equiv (-x-1)/(-x-1) = 1 \pmod{n}$ and hence, if n divides s, we have $x \equiv -1 \pmod{n}$.

(1) From the above, if n divides s, then $x \equiv -1 \pmod{n}$. Hence, we only have to show that n^2 does not divide s. To show this, we prove that $\Phi_k(x)(= s\ell) \equiv n \pmod{n^2}$ when $x \equiv -1 \pmod{n}$. Write $x = mn - 1$ for some integer m. Since $\Phi_k(x) = \Phi_n(-x) = (-x)^{n-1} + (-x)^{n-2} + \cdots + (-x) + 1$, $\Phi_k(x) = \Phi_n(-x) = \Phi_n(-mn+1) \equiv \left(\sum_{i=1}^{n-1} {}_iC_1\right)(-mn) + n = (-m)(n+1)n^2/2 + n \equiv n \pmod{n^2}$. This calculation shows that n^2 does not divide $\Phi_n(x) = s\ell$ and hence we have that s is not divisible by n^2.

(2) If $s = n$, then since $\Phi_k(-1) - s = \Phi_n(1) - n = 0$, $\Phi_k(x) - s$ has a factor $x + 1$. More precisely, we have $\Phi_k(x) - n = \Phi_n(-x) - n = -(x+1)((-x)^{n-2} + 2(-x)^{n-3} + \cdots + (n-2)x + (n-1))$. Since $x + 1 \equiv 0 \pmod{n}$ in this case and n is an odd prime, $(-x)^{n-2} + 2(-x)^{n-3} + \cdots + (n-2)(-x) + (n-1) \equiv n(n-1)/2 \equiv 0 \pmod{n}$. Hence, we have $\ell - 1 = (\Phi_n(-x) - n)/n$ has a factor $x + 1$.

(3) Suppose that $x \equiv -1 \pmod{n}$. Then $\Phi_k(x) \equiv \Phi_n(1) \equiv 0 \pmod{n}$. This contradicts the assumption that n does not divide s. □

In particular, the case (2) in Theorem 3 is not suitable for the protocols based on pairing-related problems if we consider an effect of Cheon's algorithm.

4.2 Our Construction

From the result of the previous section, we propose a method to construct pairing-friendly elliptic curves with small security loss by Cheon's algorithm.

We consider only the case that the embedding degree k is in the form $k = 2n$ where n is an odd prime. Our construction is an improved version of the Freeman-Scott-Teske method. Since the Freeman-Scott-Teske method needs a field extension, we should use $\Phi_{ck}(x)$ where c is an extension degree. So when we take ℓ as a proper divisor of a cyclotomic polynomial in the Freeman-Scott-Teske method, ℓ and p become much larger. Here, we improve the Freeman-Scott-Teske method such that we can obtain the small ρ value with not so much large ℓ and p even when we take ℓ as a proper divisor of a cyclotomic polynomial.

First note that for $k = 2n$ with an odd prime n, if g is a primitive k-th root of unity in a field K, then $\sqrt{-g} = g^{(n+1)/2}$ belongs to K. Our idea is to use this $\sqrt{-g} = g^{(n+1)/2}$ as $\sqrt{-D}$. The advantage to use such $\sqrt{-D}$ is that we do not need to extend a cyclotomic field $\mathbb{Q}(\zeta_k)$ to obtain a small value of $\rho = \lg p / \lg \ell$. In the following, we describe our method. It is divided into two cases: (1) the case of a general n, (2) the case of $n \equiv 1 \pmod{4}$.

The general case. Let g be a positive integer such that $\Phi_k(g) = s\ell$ for a very small integer s and a large prime ℓ. Then, g is a primitive k-th root of unity modulo ℓ and $\sqrt{-g} \equiv g^{(n+1)/2} \pmod{\ell}$. Take D, a, b $(0 < D, a, b < \ell)$ as follows:

$$D := g, \qquad a := g + 1, \qquad b :\equiv (g-1)g^{(n+1)/2}/g \pmod{\ell}.$$

Then, $p = (a^2 + Db^2)/4 = O(g^{n+2})$ and $\ell = O(g^{\varphi(n)}) = O(g^{n-1})$, where φ denotes the Euler phi function. Since s is very small, we have $\rho \sim (n+2)/(n-1)$ as $p, \ell \to \infty$.

The case of $n \equiv 1 \pmod 4$. When $n \equiv 1 \pmod 4$, we can improve the asymptotic value of ρ.

Let g, $\Phi_k(g) = s\ell$ be as in the general case. Then, g is a primitive k-th root of unity modulo ℓ and $\sqrt{-g} \equiv g^{(n+1)/2} \pmod{\ell}$. Note that $g^{(n+1)/2}$ is also a primitive k-th root of unity modulo ℓ. Take D, a, b $(0 < D, a, b < \ell)$ as follows:

$$D := g, \qquad a := g^{(n+1)/2} + 1, \qquad b :\equiv (g^{(n+1)/2} - 1)g^{(n+1)/2}/g \pmod{\ell}.$$

Then, since $b \equiv g^n - g^{(n-1)/2} \equiv -1 - g^{(n-1)/2} \pmod{\ell}$, we have $p = (a^2 + Db^2)/4 = O(g^{n+1})$ and $\ell = O(g^{\varphi(n)}) = O(g^{n-1})$. Since s is very small, we have $\rho \sim (n+1)/(n-1)$ as $p, \ell \to \infty$.

The algorithm of our construction is given as follows.

Algorithm 1.

Input:	n [illegible] α β q [illegible]
Output:	p ℓ [illegible] $E/\mathbb{F}_p$ [illegible] $\mathbb{F}_p$ [illegible] $\#E(\mathbb{F}_p) = \ell$ [illegible] [illegible] $k = 2n$
Step 1:	[illegible] $g \in \mathbb{Z}_{>0}$ [illegible] $\Phi_k(g) = s\ell$ [illegible] ℓ [illegible] s [illegible] [illegible] $\neq n$ [illegible] n [illegible] $\ell - 1 = 2n($ [illegible] $\leq 2^{\alpha}) \prod($ [illegible] $\geq q)$ $\ell + 1 = 2($ [illegible] $\leq 2^{\beta}) \prod($ [illegible] $\geq q)$
Step 2:	[illegible] $a := g + 1$, $n \equiv 3 \pmod 4$ [illegible] $a := g^{(n+1)/2} + 1$, $n \equiv 1 \pmod 4$ [illegible] b [illegible] $< \ell$ [illegible] $b \equiv (a-2)g^{(n+1)/2}/g \pmod{\ell}$ [illegible] $D := g$ [illegible] $p := (a^2 + Db^2)/4$ [illegible]
Step 3:	[illegible]

The positive integer q in the input is a parameter of the q-weak/strong Diffie-Hellman problem. The size of q depends on a protocol and the ability of attackers. The positive integers α and β in the input are parameters which determine the bound of the security loss by Cheon's algorithm. We take $\alpha = \beta = 6$ for examples in the next section.

Using the CM method, we can construct an ordinary elliptic curve with the complex multiplication by an order of the imaginary quadratic field $K = \mathbb{Q}(\sqrt{-D})$, $D > 0$. Refer to [14] for the details of the calculation. In general, for a large D, it is hard to construct the elliptic curve by the CM method. Therefore we must be careful with the size of D.

In our method, we set $D = g$. If g is not square free, then we set the square free part of g as D. So the size of g is important when we construct the elliptic curve using the CM method. But as stated in [12], we can construct an elliptic curve by using the CM method for $D < 10^{10}$. Hence our method is effective to construct pairing-friendly elliptic curves.

4.3 Examples

Here we show examples of pairing-friendly elliptic curves with small security loss by Cheon's algorithm. The following examples are obtained by using Algorithm 1 with $q = 2^{50}$ and $\alpha = \beta = 6$ for $14 \leq k \leq 38$. The security loss of these examples is within 5 bits, if the parameter q in the weak/strong Diffie-Hellman problem is less than 50 bits. If the parameter q is about 2^{60}, we can find examples for $q = 2^{60}$ by taking $\alpha = \beta = 11$.

k	14
(x, s)	$(1083603511, 29)$
ℓ	558244461317143757104672701626918998407404333205677 39 (176 bit)
p	5149601701498901149849436799809351834489449663566405000139 9\ 1240135020678496405311
ρ	1.53017
$\ell - 1$	$2 \cdot 7 \cdot$ (a 69 bit prime) $\cdot$ (a 103 bit prime)
$\ell + 1$	$2^2 \cdot 3 \cdot 5 \cdot$ (a 65 bit prime) $\cdot$ (a 106 bit prime)

k	22
(x, s)	$(2169245, 67)$
ℓ	3443586908389364671503933551495445912546234980894932315809 9\ 743 (205 bit)
p	5887778651704515848057946195601171633901757087143749298020 1\ 25450311726006289864629
ρ	1.32879
$\ell - 1$	$2 \cdot 11 \cdot$ (a 74 bit prime) $\cdot$(a 127 bit prime)
$\ell + 1$	$2^5 \cdot 3 \cdot$ (a 73 bit prime) $\cdot$ (a 125 bit prime)

k	26
(x, s)	$(83647, 131)$
ℓ	895628588110024088164630713805121667532341241783716653231 (190 bit)
p	2052345035175498040876970342827233281136809297495235578441 6\ 0697479999
ρ	1.19947
$\ell - 1$	$2 \cdot 5 \cdot 13 \cdot$ (a 55 bit prime) $\cdot$ (a 128 bit prime)
$\ell + 1$	$2^4 \cdot 3 \cdot$ (an 84 bit prime) $\cdot$ (a 100 bit prime)

k	34
(x, s)	$(1730735, 17 \cdot 137)$
ℓ	27830402151707213772790243425060710128851524965270716441651\ 113285546630638085671924440248448543 29 (321 bit)
p	48538978648626809809653096381338491065159598631595616079566\ 88321815318124568522625897243485762842754461264104559
ρ	1.15803
$\ell - 1$	$2^3 \cdot 17 \cdot$ (a 92 bit prime) $\cdot$ (a 222 bit prime)
$\ell + 1$	$2 \cdot 3 \cdot 5 \cdot$ (a 102 bit prime) $\cdot$ (a 214 bit prime)
k	38
(x, s)	$(422017, 2281)$
ℓ	79033772326705018830502245444409438041774479438057073363711\ 630220987237178915490932609778746724313 (326 bit)
p	33874025807138240665499623427646024497140999922941667223498\ 12927081355741867650294171908202450963933866119466570911873
ρ	1.20054
$\ell - 1$	$2^3 \cdot 3 \cdot 19 \cdot$(a 66 bit prime)$\cdot$(a 71 bit prime)$\cdot$ (an 83 bit prime)$\cdot$(a 99 bit prime)
$\ell + 1$	$2 \cdot 7 \cdot$ (a 74 bit prime) $\cdot$ (a 118 bit prime) $\cdot$ (a 131 bit prime)

5 Conclusion

In this article, we studied the effect of Cheon's algorithm on known methods of constructing pairing-friendly elliptic curves. We showed that Cheon's algorithm affects a cyclotomic family. We considered the way to reduce the security loss of a cyclotomic family by Cheon's algorithm and proposed a method to construct pairing-friendly elliptic curves with small security loss by Cheon's algorithm. Heuristically, the proposed method gives pairing-friendly elliptic curves whose security loss by Cheon's algorithm is within 5 bits for $k \leq 38$. Moreover, the value of ρ of constructed curves is almost same as the Freeman-Scott-Teske method.

References

1. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006)
2. Boneh, D., Boyen, X.: Efficient selective-ID secure identity-based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004)
3. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004)
4. Boneh, D., Boyen, X., Goh, E.-J.: Hierarchical identity based encryption with constant size ciphertext. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005)

5. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. SIAM Journal of Computing 32(3), 586–615 (2003)
6. Blake, I.-F., Seroussi, G., Smart, N.-P.: Advances in Elliptic Curve Cryptography. Cambridge University Press, Cambridge (2005)
7. Cheon, J.H.: Security Analysis of the Strong Diffie-Hellman Problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 1–11. Springer, Heidelberg (2006)
8. Brezing, F., Weng, A.: Elliptic curves suitable for pairing based cryptography. Design, Codes and Cryptography 37, 133–141 (2005)
9. Cocks, C., Pinch, R.G.E.: Identity-based cryptosystems based on the Weil pairing (Unpublished manuscript)
10. Freeman, D.: Methods for constructing pairing-friendly elliptic curves. In: 10th Workshop on Elliptic Curves in Cryptography (ECC 2006), Toronto, Canada (September 2006)
11. Freeman, D.: Constructing Pairing-Friendly Elliptic Curves with Embedding Degree 10, Cryptology ePrint Archive, Report, 2006/026 (2006), http://eprint.iacr.org/
12. Freeman, D., Scott, M., Teske, E.: A taxonomy of pairing-friendly elliptic curves, Cryptology ePrint Archive, Report 2006/372 (2006), http://eprint.iacr.org/
13. Galbraith, S., McKee, J., Valença, P.: Ordinary abelian varieties having small embedding degree. In: Proc. Workshop on Mathematical Problems and Techniques in Cryptology, pp. 29–45. CRM, Barcelona (2005)
14. IEEE Standard Specifications For Public-Key Cryptography - IEEE Std 1363-2000. IEEE Computer Society, New York, USA (2000)
15. Joux, A.: A one round protocol for tripartite Diffie-Hellman. In: Bosma, W. (ed.) Algorithmic Number Theory. LNCS, vol. 1838, pp. 385–393. Springer, Heidelberg (2000) Full version: Journal of Cryptology 17 263–276 (2004)
16. Kozaki, S., Kutsuma, T., Matsuo, K.: Remarks on Cheon's algorithms for pairing-related problems. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 302–316. Springer, Heidelberg (2007)
17. Kutsuma, T., Matsuo, K.: Remarks on Cheon's algorithms for pairing-related problems. In: 2007 Symposium on Cryptography and Information Security (SCIS2007), Nagasaki, Japan (2007)
18. Mitsunari, S., Sakai, R., Kasahara, M.: A new traitor tracing. IEICE Trans. Fundamentals E85-A(2), 481–484 (2002)
19. Miyaji, A., Nakabayashi, M., Takano, S.: New explicit conditions of elliptic curve traces for FR-reduction. IEICE Transactions on Fundamentals E84-A(5), 1234–1243 (2001)
20. Okamoto, T.: Efficient blind and partially blind signatures without random oracles. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 80–99. Springer, Heidelberg (2006)
21. Scott, M., Barreto, P.S.L.M.: Generating more MNT elliptic curves. Designs, Codes and Cryptography 38, 209–217 (2006)
22. Sakai, R., Ohgishi, K., Kasahara, M.: Cryptosystem based on pairing. In: 2000 Symposium on Cryptography and Information Security (SCIS 2000), Okinawa, Japan (2000)
23. Silverman, J.H.: The Arithmetic of Elliptic Curves. GTM 106. Springer, Heidelberg (1986)

Analysis of Multivariate Hash Functions

Jean-Philippe Aumasson* and Willi Meier**

FHNW, 5210 Windisch, Switzerland

Abstract. We analyse the security of new hash functions whose compression function is explicitly defined as a sequence of multivariate equations. First we prove non-universality of certain proposals with sparse equations, and deduce trivial collisions holding with high probability. Then we introduce a method inspired from coding theory for solving underdefined systems with a low density of non-linear monomials, and apply it to find collisions in certain functions. We also study the security of message authentication codes HMAC and NMAC built on multivariate hash functions, and demonstrate that families of low-degree functions over GF(2) are neither pseudo-random nor unpredictable.

1 Introduction

A fundamental idea of [illegible] was stated by Shannon in 1949 [35]: [illegible]. Multivariate primitives are indeed directly described in terms of multivariate polynomial functions, in order to reduce certain security problems to the presumably hard problem of solving the system, and/or to problems like Polynomial Isomorphism, Minrank, [illegible] At the opposite, primitives without explicit multivariate equations might be attacked by first finding a full or partial description as a system of equations, then exploiting the latter system (ideally, solving it) – this is the principle of [illegible]. A number of multivariate primitives appeared since the early years of modern cryptology, mainly asymmetric schemes (Matsumoto-Imai [28], Ong-Schnorr-Shamir [33], HFE [34], [illegible]), and more recently, the stream cipher QUAD [7]. As the advent of RSA led to a multitude of works on integer factorisation, researchers designed new algorithms for solving multivariate systems of equations, to tackle multivariate primitives [18,24,14]. Note that, although every cipher possesses a characterisation as a system of Boolean equations, this latter is generally at least as hard to compute as breaking the cipher with a brute force attack.

This paper analyses [illegible], that is, iterated hash schemes built upon a function $\mathbb{K}^{m+n} \mapsto \mathbb{K}^n$ explicitly defined as a sequence of multivariate equations. More precisely, we focus our study on the later [illegible]

* Supported by the Swiss National Science Foundation under project number 113329.
** Supported by Hasler Foundation `http://www.haslerfoundation.ch/` under project number 2005.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 309–323, 2007.

, and relate its parameters to the security of the overall primitive. It is well-known that hash functions are essential in numerous real-life cryptographic schemes and protocols, be it digital signatures (in DSA and ECDSA), authentication codes (through HMAC), or for building pseudo-random function and derive keys. After several breakthroughs in the analysis of the previous standards MD5, SHA-0, and SHA-1, the community is particularly attentive to new designs (NIST's competition for a new standard [32]). Surprisingly, multivariate hash functions only appeared in 2007, in works by Billet, Peyrin and Robshaw [8], introducing the constructions MQ-HASH and RMQ-HASH, and simultaneously by Ding and Yang [20]. By reducing the problem of finding a preimage of a given digest to the problem of solving a multivariate system, a security guarantee is given for multivariate hash functions. Other "provably secure" hash functions exist, whose resistance to preimage and/or collision relies on problems as different as syndrome decoding [2,25], (approximate) shortest vector in a lattice [6,29], and non-trivial square root of very smooth numbers [12]. We notice that the term "multivariate hash functions" has already been employed in a non-cryptographic context [36] to denote functions hashing several objects at the same times, and that multivariate hash functions over GF(2) have recently been employed in the context of interactive hashing [27].

Our Work. Section 2 gives security definitions and presents our model of multivariate hash functions, along with the description of the constructions MQ-HASH, RMQ-HASH, and SCC. Section 3 then proves the non-universality of functions based on sparse equations, like SCC, and illustrates this with several trivial collisions holding with high probability. Section 4 introduces a new method for solving underdefined semi-sparse multivariate systems, that we apply to certain kinds of multivariate hash functions. Section 5 studies the security of message authentication codes NMAC and HMAC built on multivariate hash functions, showing a critical attack on NMAC-SCC. Section 6 demonstrates that all multivariate hash functions over GF(2) with small degree are efficiently predictable and distinguishable from random functions. As an aside, we identify weak instances of QUAD in Section 7, and eventually draw some conclusions in Section 8.

2 Preliminaries

Let $n \in \mathbb{N}^\star, m \in \mathbb{Z}$. A (of equations) over a finite field $\mathbb{K}$ with n equations and $(m+n)$ unknowns is denoted as $\{h_i(x) = 0\}_{0 \leq i < n-1}$, or simply $h(x) = 0$, with $x = (x_0, \ldots, x_{m+n-1}) \in \mathbb{K}^{m+n}$. The system is called (respectively) when $m > 0$ (resp. $m < 0$). The degree $\deg(h)$ of the system is $\max_i \deg(h_i)$. We identify Boolean functions with their representative polynomial over GF(2), and the of a polynomial is defined as the number of non-null coefficients in its algebraic normal form. The number of square-free monomials in n variables (that is, considering $x^2 \equiv x$ as over GF(2)) of degree in $[0, d]$ is

$$\mathcal{N}(n,d) = \sum_{i=0}^{d} \binom{n}{i} . \tag{1}$$

The [illegible] of a polynomial of degree d in n variables is the ratio between its weight and $\mathcal{N}(n,d)$, so that a [illegible] of density $\delta \in [0,1]$ has its equations with expected weight $\lfloor \delta \mathcal{N}(n,d) \rfloor$ (for convenience, we omit from now the flooring symbols $\lfloor . \rfloor$), such that each monomial has probability δ to appear in an arbitrary component. We call a random system [illegible] when it has density $\delta \ll 50\%$, and [illegible] when only monomials of certains degrees have a density $\ll 50\%$ (for example, imagine a cubic system where δ only applies to the cubic monomials). Eventually, when we mention "random" objects, it implicitly means with respect to a uniform distribution in the appropriate sample space, unless precised differently.

2.1 Security Definitions

We give the security definitions for [illegible] of hash functions[1], that is, subsets of the set of all functions $\mathbb{K}^{m+n} \mapsto \mathbb{K}^n$ for fixed $\mathbb{K}$, m, and n. These families can also be seen as [illegible] over this superset.

Preimage	
Input	a random hash function $h \in \mathcal{F}$, a random digest $y \in \{0,1\}^n$.
Output	$x \in \{0,1\}^{m+n}$ such that $h(x) = y$.

Second Preimage	
Input	a random hash function $h \in \mathcal{F}$, a random input $x \in \{0,1\}^{m+n}$.
Output	$x' \in \{0,1\}^{m+n}$ such that $h(x) = h(x')$ and $x \neq x'$.

Collision	
Input	a random hash function $h \in \mathcal{F}$.
Output	$x, x' \in \{0,1\}^{m+n}$ such that $h(x) = h(x')$ and $x \neq x'$.

In addition, the term [illegible] designates a collision over only certain bits of the digest. For an ideal hash function h, the problems above have complexity of about 2^n, 2^n and $2^{n/2}$ evaluations of h respectively.

Another crucial notion for the security of hash functions is their [illegible], necessary for building secure key-derivation schemes, and, obviously, to instantiate pseudo-random functions. Since we actually consider distributions of functions rather than single instances, the following definitions from [31] are relevant.

Definition 1. [illegible] $\mathcal{F}$ [illegible] pseudo-random [illegible]

[illegible] *ffi* [illegible]

[1] We further call "hash functions" mappings $\mathbb{K}^{m+n} \mapsto \mathbb{K}^n$, independently of their role of compression function in iterated hash functions.

Definition 2. $\mathcal{F}$ unpredictable

ffi
ffi

(See [31] for more formal definitions 3.1 and 3.2.) Finally, we recall the definition of ε-universality.

Definition 3. $\mathcal{F}$ ε-universal x x' $h \in \mathcal{F}$ h $h(x) = h(x')$ ε

We rather consider the computational version, denoted ε-cAU (computational almost universality, see [4]), such that for a family not ε-cAU, one can efficiently compute such a pair (x, x').

2.2 Multivariate Hash Functions

A $h : \mathbb{K}^{m+n} \mapsto \mathbb{K}^n$ is explicitly defined as a sequence of n polynomial functions $h_i : \mathbb{K}^{m+n} \mapsto \mathbb{K}$ for some finite field $\mathbb{K}$, its A of multivariate hash functions is characterised by a scheme, along with a choice of for this scheme, thereby defining a distribution over the set of all functions $\mathbb{K}^{m+n} \mapsto \mathbb{K}^n$, where $\mathbb{K}$, m, and n are fixed either by the construction itself of by the parameters. An is then randomly picked with respect to that distribution, casting into the framework of [9].

Given an arbitrary family of multivariate hash functions $\mathcal{F}$, solving PREIMAGE reduces (in the Turing sense) to solving the system $h(x) = y$ for random $h \in \mathcal{F}$ and $y \in \mathbb{K}^n$. When $\mathcal{F}$ corresponds to the set of all quadratic systems over $\mathbb{K}$ with $(m + n)$ unknowns and n equations, PREIMAGE reduces to the problem MQ, known to be NP-hard for any finite field $\mathbb{K}$ if m is small [26]:

MULTIVARIATE QUADRATIC (MQ)	
Input	a finite field $\mathbb{K}$, a system $f = \{f_i\}_{0 \leq i < n}$ of n random quadratic equations in $n + m$ variables over $\mathbb{K}$, $n \in \mathbb{N}^\star$, $m \in \mathbb{Z}$.
Output	$x \in \mathbb{K}^{m+n}$ such that $f(x) = 0$.

The problem of solving a multivariate system is also assumed hard for higher degrees (state-of-the art methods are briefly surveyed in Section 2.3). Furthermore, no efficient quantum algorithm is known yet to solve multivariate systems, hence multivariate hash functions have chances to survive in a world with efficient quantum computers.

On the other hand, COLLISION reduces to solving the equation $h(x)-h(x')=0$ with the constraint $x \neq x'$, which will not be an instance of MQ. Another technique to find collisions is to assume that there exists a colliding pair (x, x') with difference $\Delta = x - x' = (x_i - x'_i)_{0 \leq i < m+n}$, then computing that pair by solving the system

$$h(x) - h(x - \Delta) = 0 \ , \tag{2}$$

for a fixed and [illegible] difference $\Delta \neq 0$. This system has degree at most $\deg(h) - 1$, and is expected to have at least one solution for a sufficiently large m. We shall further refer to this technique as the [illegible].

Composed Quadratics Construction. The construction MQ-HASH by Billet, Peyrin and Robshaw [8], and an unnamed construction by Ding and Yang [20], propose to define a quartic (degree 4) system h using two composed quadratic systems f and g, such that $h = g \circ f$. Following the ideas of [1], the first box $f : \mathbb{K}^{m+n} \mapsto \mathbb{K}^r$, for $r > (m+n)$, expands the input, while a second box $g : \mathbb{K}^r \mapsto \mathbb{K}^n$ compresses the intermediate value. Security aspects are much more developed in [8] than in [20], and we will only consider that former reference for composed quadratics. Hereafter we present a succinct overview of the security arguments for MQ-HASH.

First, the main result of [8] is the reduction of PREIMAGE to the problem of inverting f or g, which proves PREIMAGE-resistance, assuming the hardness of MQ for the parameters chosen. Although no reduction is given for COLLISION, several arguments are presented: indeed, a necessary property for h to resist COLLISION is the COLLISION-resistance of the expanding box f; this is expected to hold since f will actually be [illegible] with high probability for well chosen parameters, as stated by Proposition 1 of [8]. In the worst case, when there exists a pair (x, x') such that $f(x) = f(x')$, this can be recovered by solving a linear system only if the difference $(x - x')$ is known. However, since the expected number of colliding pairs is very low, only very few differentials would lead to a collision. Choosing $2(m + n) - r < s$ ensures that a random instance will possess a collision with probability $< 2^{-s}$, let alone the hardness of finding the corresponding difference. Another strategy to find a colliding pair consists in

1. finding a collision (y, y') for g, and
2. computing preimages of y and y' by f,

but this again is not efficient since f is assumed hard to invert.

The iteration mode for MQ-HASH is a basic Merkle-Damgård mode, with standard padding and no output filter. In order the function to meet the 80-bit security level (meaning here a minimum of 2^{80} trials in average to find a collision), the authors of MQ-HASH propose to use the family over $\mathbb{K} = \mathrm{GF}(2)$ with message blocks of $m = 32$ bits, a chaining value of $n = 160$ bits, and an intermediate value of $r = 464$ bits. We will refer to this family along the paper.

An alternative constrution to MQ-HASH called RMQ-HASH [8] proposes to define

$$h(x) = h(x_1 || x_2) = f(x_1, g(x_2)),$$

with the message block represented by x_1, and the previous chaining value by x_2. The functions f and g are quadratic, and defined as $f : \mathbb{K}^{m+r} \mapsto \mathbb{K}^n$, and $g : \mathbb{K}^n \mapsto \mathbb{K}^r$. This construction is just presented as a possible variant of MQ-HASH, and no security analysis is provided. In the remainder of the paper, we will rather concentrate our study on MQ-HASH, which remains the main proposal of [8], and whose analysis partially overlaps RMQ-HASH's. However we can already observe that when $m > n$, one may simply set a random value for x_2, such that PREIMAGE reduces to solving a quadratic system of n equations in m unknown, but COLLISION with a given difference can be computed by solving a linear system with as many equations and unknowns.

Sparse Cubic Construction. This construction introduced by Ding and Yang [20] merely consists in a cubic system $h : \mathbb{K}^{2n} \mapsto \mathbb{K}^n$ (thus $m = n$) of density δ. In other words, every component h_i has exactly $\delta\mathcal{N}(2n, 3)$ monomials. We further use the shortcut "SCC" to refer this construction.

Clearly, PREIMAGE reduces to solving a sparse cubic system, assumed hard for well chosen parameters by the designers. The generic attack against COLLISION directly reduces here to solving a , quadratic system. Although assumed hard, the problem of solving sparse systems is not as hard as the general case (, Section 2.3). On the other hand, sparse systems provide a considerable speed-up, as well as reduced storage requirements. Several families are suggested, characterised by their parameters hereafter (recall $m = n$).

1. For 160-bit digests:
   - $\mathbb{K} = \mathrm{GF}(2)$, $n = 160$, $\delta = 0.1\%$
   - $\mathbb{K} = \mathrm{GF}(2^4)$, $n = 40$, $\delta = 0.1\%$
   - $\mathbb{K} = \mathrm{GF}(2^8)$, $n = 20$, $\delta = 0.2\%$
2. For 256-bit digests:
   - $\mathbb{K} = \mathrm{GF}(2)$, $n = 256$, $\delta = 0.1\%$
   - $\mathbb{K} = \mathrm{GF}(2^4)$, $n = 64$, $\delta = 0.1\%$
   - $\mathbb{K} = \mathrm{GF}(2^8)$, $n = 32$, $\delta = 0.1\%$
   - $\mathbb{K} = \mathrm{GF}(2^{16})$, $n = 16$, $\delta = 0.2\%$

2.3 Solving Multivariate Systems

To solve a system of multivariate equations, cryptanalysts mainly employ methods derived from Buchberger's algorithm to compute a Gröbner basis of a polynomial ideal. The most efficient ones are Faugère's F_4 and F_5 [22,23], and the algorithms of the XL family [16,13,17,19]. Those algorithms perform better on overdefined systems, and F_4 and F_5 are know to take advantage of sparse systems. Some work concentrates on the particularities of underdefined [15], overdefined [16], or sparse systems [38,37]. More recently, sparse systems derived from cryptographic primitives were solved by converting the system into a SAT instance, then solving this instance using an efficient SAT-solver (, MINISAT [21]), and finally converting the solution to a solution of the system [3,30]. Unfortunately, the complexity of multivariate solvers is often hard to estimate.

Empirical results are here probably more significant for cryptanalysts. For instance, the algorithm XL-Wiedemann was demonstrated [40] to solve MQ over $\mathrm{GF}(2^8)$ with $n = 40$ equations and $n + m = 20$ unknowns in less than 2^{45} cycles of a 64-bit AMD Opteron processor (a few hours of computation).

3 Non-universality of Sparse Function Families

In this section we give a simple result on the universality of sparse multivariate hash functions independently of the degree of the components, and deduce collisions holding with high probability.

3.1 General Case

Consider a family $\mathcal{F}$ of multivariate hash functions $\mathbb{K}^{m+n} \mapsto \mathbb{K}^n$ of density δ. Then for a random $h \in \mathcal{F}$, any given monomial appears in an arbitrary component h_i with probability δ. In particular, a given degree 1 monomial x_i appears in no single component (let's call such a x_i an [illegible]) with probability $(1-\delta)^n$. When this event occurs, it is easy to see that

$$h(0,\ldots,0,x_i=0,0,\ldots,0) = h(0,\ldots,0,x_i=1,0,\ldots,0)\ . \tag{3}$$

Consequently, for any such pair of inputs, a collision occurs in a random $h \in \mathcal{F}$ with probability $(1-\delta)^n$. In other words, $\mathcal{F}$ is not $(1-\delta)^n$-cAU. Moreover, by trying all possible such pairs of input, one gets at least one collision with probability

$$\rho = 1 - (1 - (1-\delta)^n)^{n+m}\ . \tag{4}$$

For all the parameters of SCC proposed in [20], $\rho \approx 1$, hence with high probability at least one such collision exists.

When no isolated variable exists in the original system, one might be found in a derived system, obtained by suitably fixing values of a subset of the variables, such that there exists an isolated variable in the new system. To find a derived system with x_j isolated, one can rewrite all components as

$$h_i(x) = x_j \cdot d_i(x) + e_i(x)\ , \tag{5}$$

for polynomial functions d_i and e_i, such that $e_i(x)$ does not contain the variable x_j in any monomial. Consequently, $\deg(d_i) \leq (\deg(h)-1)$ and $\deg(e_i) \leq \deg(h)$. Consider now the system $\{d_i(x) = 0\}_{0 \leq i < n}$, with $(m+n-1)$ unknowns: a solution gives a valuation such that the output is independent of x_j. This is an alternative manner to find collisions by solving a system of reduced degree, equivalent to the generic collision attack.

Finally, observe that when the monomial x_i appears in exactly k equations, then $h(0,\ldots,0,x_i=0,0,\ldots,0)$ and $h(0,\ldots,0,x_i=1,0,\ldots,0)$ collide over exactly $(n-k)$ bits, thus bringing near-collisions when k is small.

3.2 Case of Even Components over GF(2)

Now consider a family of multivariate hash functions $\mathrm{GF}(2)^{m+n} \mapsto \mathrm{GF}(2)^n$ of density δ, such that δ imposes an [illegible] number of monomials in each component h_i. Since the constant monomial 1 appears with probability δ in a given h_i, the collision

$$h(0,\ldots,0) = h(1,\ldots,1) \tag{6}$$

will hold for a random h with probability $(1-\delta)^n$. It is a different method to see that such families are not $(1-\delta)^n$-cAU. For a random instance of SCC with 160-bit digest, the collision in Eq. (6) holds with probability 0.73, and for 256-bit digests, with probability 0.60.

When the system does not have only "even" equations, one might look for a suitable derived system where all components have an even number of non-constant monomials. One can observe that finding such a system is equivalent to finding a preimage of $h(0,\ldots,0)$.

Analogously to the previous observations, a near-collision over $(n-k)$ bit occurs when exactly k equations have an even number of non-constant monomials.

4 Solving Underdefined Semi-sparse Systems

In a multivariate hash function, replacing a sparse system of equations by a semi-sparse one, where the density $\delta \ll 50\%$ only applies to monomials of degree > 1, avoids the weaknesses of Section 3. However collisions might be found in semi-sparse systems, as shown in the present section: we introduce a method for solving underdefined quadratic systems with density of quadratic terms $\delta \lll 50\%$, then apply it to find collisions in semi-sparse cubic systems, based on the generic attack.

4.1 Description of the Method

Consider a [illegible] quadratic system $h(x) = 0$ in $(m+n)$ variables with n equations, $m > 0$, such that each equation contains each degree 1 monomial with probability 1/2, but with a density of quadratic terms δ, and null constant terms (homogeneous system). The linear system $h'(x) = 0$ obtained by removing the quadratic monomials then describes the parity-check matrix of a random linear code C of length $(m+n)$, dimension m (assuming linear independency of the equations), and unknown minimal distance $d_{\min}$. Each solution of the system $h'(x) = 0$ then corresponds to a codeword of C. The key observation is that a low-weight solution of this system will be a solution of $h(x) = 0$ if in each component the sum of all quadratic monomials happens to vanish. For a random word of weight w this latter event has probability (piling-up lemma in Appendix)

$$\left(\frac{1}{2} + 2^{\delta\binom{m+n}{2}-1}\left|\frac{1}{2} - \frac{w}{m+n}\right|^{\delta\binom{m+n}{2}}\right)^n, \tag{7}$$

where $\delta\binom{m+n}{2}$ is the expected number of quadratic monomials in an equation.

The best algorithm known so far for finding a low-weight codeword in a random linear code [10] requires a "work factor" estimated to

$$\exp_2 \left\{ 0.12 \times (m+n-1) H \left(\frac{m}{m+n-1} + 2^{-5} \right) + 10 \right\} , \tag{8}$$

where H is the binary entropy function,

$$H(\varepsilon) = -\varepsilon \log(\varepsilon) - (1-\varepsilon) \log(1-\varepsilon) . \tag{9}$$

Before applying this algorithm, we need to compute the generating matrix of the linear code C from the parity-check matrix derived from the system $h'(x) = 0$, which adds a cost in $\mathcal{O}((m+n)^3)$.

The expected efficiency of this technique cannot be precisely established, since the distance $d_{\min}$ of the code is unknown, as well as its expected value. Useful results are the Gilbert-Varshamov bound

$$\sum_{i=0}^{d_{\min}-2} \binom{m+n-1}{i} < 2^n , \tag{10}$$

and an upper bound on the number of codewords of weight $\leq \varepsilon(m+n)$, equal to $2^{(m+n)H(\varepsilon)}$.

Note that this method can also be applied to systems of degree larger than two, in which case it succeeds as soon as all the sums of monomials of degree at least two happen to vanish in each equation of the system.

4.2 Application to Multivariate Hash Functions

Consider a variant of SCC over GF(2) with $n = 160$, where the density $\delta \ll 50\%$ only applies to the monomials. Using the generic collision attack with a differential of weight 1, a colliding pair of inputs can be computed by solving a quadratic system with in average $\delta\binom{2n-1}{2}$ quadratic monomials, inherited from the cubic monomials of the original system (since there are exactly $\binom{2n-1}{2}$ cubic monomials containing a given x_j). We then consider the system built by removing those quadratic terms, in order to apply the method of the previous subsection. The expected work factor to find a minimal weight word in the associated linear code is about 2^{48} (cf. Eq. (8)), and there are at most $\approx 2^{14}$ codewords of weight ≤ 40. Assume that a word with weight ≤ 40 is found. Then a collision is found for $\delta = 0.2\%$ with probability ≥ 0.0017, for $\delta = 0.1\%$ with probability ≥ 0.0402, and for $\delta = 0.05\%$ with probability ≥ 0.1988. The ratios "success probability over complexity" are then clearly higher than for a birthday attack. Nonetheless, one should be careful by mixing asymptotic estimates and assertions on concrete instances; for instance, the effective computation time of the word-finding algorithm of "work factor" of 2^{48} is probably much higher than the cost of computing 2^{48} digests.

Finally, note that we considered a homogeneous system, whereas the one we need in SCC is not necessarily; we may easily convert this system to a homogeneous one, by introducing a dummy variable X as soon as the constant 1 appears. Then the words obtained will have $X = 1$ with probability $w/(m+n)$, hence the attack has to be repeated about $(m+n)/w$ times (that is, with as many different codewords), to succeed – assuming a uniform distribution of the non-null offsets in those words. An alternative solution is to directly modify the algorithm of [10] to suit non-homogeneous systems.

5 Key Recovery for NMAC and HMAC

In this section we consider a concrete application of hash functions: we show that the message authentication codes NMAC and HMAC [5] built on multivariate hash functions can be attacked by solving large overdefined systems. This alternative to exhaustive search directly follows from the explicit structure of such hash functions.

Let $\mathcal{F}$ be a multivariate hash function $\mathbb{K}^{m+n} \mapsto \mathbb{K}^n$ of degree d. For an arbitrary known $h \in \mathcal{F}$, we consider $h_k^\star : \mathbb{K}^\star \mapsto \mathbb{K}^n$ the corresponding iterated hash function with initial value $k \in \mathbb{K}^n$, no padding rule, and no output filter. For $x \in \mathbb{K}^\star$, the NMAC construction with secret key (k_1, k_2), $k_i \in \mathbb{K}^n$, is described as follows:

$$\mathrm{NMAC}_{k_1,k_2}(x) = h_{k_1}^\star(h_{k_2}^\star(x)) \ . \tag{11}$$

Let an attacker have access to NMAC_{k_1,k_2} as a black box. With N queries of $\mathrm{NMAC}_{k_1,k_2}(x)$ for N distinct x's long of one message block (thus for $x \in \mathbb{K}^m$), she gets nN equations in $2n$ unknowns, of degree d^{b+1}, where b is the number of message blocks of $h_{k_2}^\star(x)$. That is, $b = \lceil n/m \rceil$. If $m \geq n$, then $b = 1$, thus the key (k_1, k_2) can be recovered by solving a system of nN degree d^2 equations in $2n$ unknowns. If k_2 is known, the same observations apply to recover k_1 except that the system has now only n unknowns and degree d^b.

The HMAC construction with key k is defined by

$$\mathrm{HMAC}_k(x) = h_{\mathsf{iv}}^\star(k \oplus \mathrm{OPAD} || h_{\mathsf{iv}}^\star(k \oplus \mathrm{IPAD} || x)) \ , \tag{12}$$

with constant OPAD and IPAD long of one message block, k at most as long as the constants, and a fixed initial value $\mathsf{iv} \in \mathbb{K}^n$. The input of the outer $h_{\mathsf{iv}}^\star$ is then an element of $\mathbb{K}^{m+n}$, and the input of the inner function is an element of $\mathbb{K}^{m+|x|}$, where $|x|$ is the number of field elements of x. Hence both the inner and the outer $h_{\mathsf{iv}}^\star$ run the compression function at least twice. In this best-case scenario (when $n \leq m$), with N queries with a m-element x, one gets nN equations of degree d^3 in $|k|$ unknowns.

Are those attacks faster than exhaustive search of the key(s) ? This depends on the construction, and on the parameters chosen. For instance, for the MQ-HASH proposal we have $\mathbb{K} = \mathrm{GF}(2)$, $n/m = 160/32 = 5$, so the attack on NMAC requires to solve a system of degree 320 with 2^{32} equations in 320 unknowns, certainly hard. For NMAC-SCC with $\mathbb{K} = \mathrm{GF}(2)$ and $m = n = 160$, with

$N \leq 2^{160}$ queries one gets $320N$ equations of degree 9 in 320 unknowns (k_1 and k_2). Thus with about 2^{48} queries, one obtains enough equations to solve the system by linearisation (about 2^{56} variables): this gives a complexity about $(2^{56})^3 = 2^{168}$ (higher than exhaustive search's cost 2^{160}). If $\mathbb{K} = \mathrm{GF}(2^8)$ and $m = n = 20$, with $N \leq 2^{160}$ queries one gets $40N$ equations of degree 9 in 40 unknowns. Thus with about 2^{23} queries, one obtains enough equations to solve the system by linearisation (about 2^{28} variables): complexity is then about $(2^{28})^3 \approx 2^{74}$ (smaller than exhaustive search's cost 2^{160}). However, in both cases, memory requirements for a practical implementation seem unrealistic.

Note that the complexity evaluations above are independent of the density of the system. For sparse systems, the cost of linearisation can be reduced (since certain monomials might not appear in the system), as well as other methods as F_5 or SAT-solvers can take advantage of low-density systems.

6 Pseudo-randomness and Unpredictability

We show here that all families of low-degree multivariate hash functions over GF(2) are neither pseudo-random nor unpredictable. This is a consequence of Fact 1, holding for an arbitrary family $\mathcal{F}$ of degree d multivariate hash functions.

Fact 1. [illegible] $h \in \mathcal{F}$ [illegible] h [illegible] $\mathcal{N}(m+n,d)$ [illegible]

This obviously holds for any function family, not only multivariate ones. However for low-degree multivariate hash functions $\mathcal{N}(m+n,d)$ is much lower than for a random function, whose degree, though unknown, is maximal ($= m+n$) with almost certainty. In this case $\mathcal{N}(m+n,d) = 2^{m+n}$.

We now briefly justify Fact 1. Let us call B the challenge box with components $\{B_i\}_{0 \leq i < n}$ ([illegible] unknown), then $B_i(0,\ldots,0)$ is equal to the constant term of the algebraic normal form of B_i. By querying B with all inputs of weight 1, one then recovers all the linear terms of the algebraic normal forms of the B_i's, using the knowledge of the constant terms. Once all the linear terms are known, all weight 2 queries give the quadratic monomials, which are used to deduce the list of cubic monomials, and so on, until degree d. As a consequence, for a family $\mathcal{F}$ of degree $d < m+n$, we have the following facts.

Fact 2. [illegible] $h \in \mathcal{F}$ [illegible] $(2)^{m+n} \mapsto (2)^n$ [illegible] $\geq (1-2^{-n})\mathcal{N}(m+n,d)$ [illegible]

Fact 3. [illegible] $h \in \mathcal{F}$ [illegible] $h(x)$ [illegible] x [illegible] $> d$ [illegible] x [illegible] $\mathcal{N}(m+n,d)$ [illegible]

In Fact 2, the box is identified by computing its algebraic normal form up to degree d, then evaluating the system obtained, and querying the box with a same input of degree $> d$. Since a random function will have an output distinct from

the degree d system's with probability $\geq (1 - 2^{-n})$, one identifies the box with high probability. The result of Fact 3 is also quite simple: in order to find $h(x)$, one simply has to compute the algebraic normal form of the function using black box queries, then evaluates the digest of any input without an explicit query.

Consequently, all the families of MQ-HASH and SCC over GF(2) with reasonable parameters fail to be pseudo-random and unpredictable, since $\mathcal{N}(m+n, d)$ shall be much lower than 2^n. For the parameters proposed, one distinguishes a random instance of MQ-HASH and SCC from a random function within respectively $2^{25.74}$ and $2^{22.38}$ black box queries. Note that in the iterated version, the padding rule makes those techniques not applicable.

Another noteworthy property of multivariate hash functions (over an arbitrary $\mathbb{K}$) is that, given a random $x = (x_1, \ldots, x_{m+n-1})$ and a random $h \in \mathcal{F}$, one can easily find a distinct h' in $\mathcal{F}$ such that $h'(x) = h(x)$, by adding and/or removing monomials in one or several equations. Although we see no impact on security [illegible], this property must be kept in mind when designing protocols involving multivariate hash functions.

7 Weak Instances of the Stream Ciphers QUAD

QUAD [7] is a construction of multivariate stream ciphers, based on two random quadratic systems $P : \mathbb{K}^n \mapsto \mathbb{K}^r$ (output function) and $Q : \mathbb{K}^n \mapsto \mathbb{K}^n$ (update function). Given an initial state $x_0 \in \mathbb{K}^n$ derived from a key and a nonce, the i-th internal state is $x_i = Q^i(x)$, and the i-th r-bit output is $y_i = P(x_i)$, so that $\{y_i\}_{0 \leq i}$ is the keystream of the cipher.

From the observations of Section 3, we can see that if Q contains an isolated variable, then two distinct initial states producing identical keystreams can be found, and if $P \circ Q^i$ contains an isolated variable, we can find distinct states whose keystreams collide on the i-th output block. Analogously, a trivial collision with all-zero and all-one inputs holds if $\mathbb{K} = \mathrm{GF}(2)$ when all components of Q or $P \circ Q^i$ have an even number of non-constant monomials. When $\mathbb{K} = \mathrm{GF}(2)$, the techniques of Section 6 apply as well to distinguish a random instance of QUAD from a random oracle when given as a black box fed with an initial state. Note that this is not a distinguisher in the usual sense for stream ciphers, in which the instance is known, but not the initial state.

Finally, those observations are [illegible] for the security of QUAD, since weak instances appear with very low probability, and the distinguisher assumes as known the secret information of the cipher.

8 Conclusion

We have studied several security aspects of multivariate hash functions, both in the general case and for the specific constructions MQ-HASH and SCC. Our main results are summarised below.

- Multivariate hash functions over GF(2) of degree $\ll (m+n)$ are neither pseudo-random nor unpredictable.

- NMAC message authentication codes built on certain cubic multivariate hash functions allow key recovery faster than by exhaustive search.
- Families of multivariate hash functions of given density are not ρ-universal, for ρ given by Eq. (4).
- Constructions based on sparse systems are vulnerable to trivial collisions and near-collisions.
- Collisions in certain semi-sparse hash functions can be solved by using an efficient algorithm to find a low-weight word in a random linear code.

The first result applies to both MQ-HASH and SCC, while the second to the fourth results only apply to SCC, and the last one to a semi-sparse variant of SCC.

Further work seems necessary to establish whether multivariate hash functions can be competitive with conservative designs in terms of performances, as well as to design more elaborate constructions improving on security and/or on efficiency.

References

1. Aiello, W., Haber, S., Venkatesan, R.: New constructions for secure hash functions. In: Vaudenay, S. (ed.) FSE 1998. LNCS, vol. 1372, pp. 150–167. Springer, Heidelberg (1998)
2. Augot, D., Finiasz, M., Sendrier, N.: A family of fast syndrome based cryptographic hash functions. In: Dawson, E., Vaudenay, S. (eds.) Mycrypt 2005. LNCS, vol. 3715, pp. 64–83. Springer, Heidelberg (2005)
3. Bard, G.V., Courtois, N.T., Jefferson, C.: Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-solvers. Cryptology ePrint Archive, Report 2007/024 (2007)
4. Bellare, M.: New proofs for NMAC and HMAC: Security without collision-resistance. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 602–619. Springer, Heidelberg (2006)
5. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996)
6. Bentahar, K., Page, D., Saarinen, M.-J.O., Silverman, J.H., Smart, N.: LASH. In: Second NIST Cryptographic Hash Function Workshop (2006)
7. Berbain, C., Gilbert, H., Patarin, J.: QUAD: A practical stream cipher with provable security. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 109–128. Springer, Heidelberg (2006)
8. Billet, O., Robshaw, M.J.B., Peyrin, T.: On building hash functions from multivariate quadratic equations. In: Pieprzyk, J., Ghodosi, H., Dawson, E. (eds.) ACISP. LNCS, vol. 4586, pp. 82–95. Springer, Heidelberg (2007)
9. Canetti, R., Micciancio, D., Reingold, O.: Perfectly one-way probabilistic hash functions (preliminary version). In: STOC, pp. 131–140 (1998)
10. Canteaut, A., Chabaud, F.: A new algorithm for finding minimum-weight words in a linear code: application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. IEEE Transactions on Information Theory 1(44), 367–378 (1998)

11. Contini, S., Lenstra, A.K., Steinfield, R.: VSH, an efficient and provable collision-resistant hash function. In Vaudenay [39] pp. 165–182
12. Contini, S., Lenstra, A.K., Steinfield, R.: VSH, an efficient and provable collision-resistant hash function. Cryptology ePrint Archive, Report, 2006/193. Extended version of [11]
13. Courtois, N.: Higher order correlation attacks, XL algorithm and cryptanalysis of Toyocrypt. In: Lee, P.J., Lim, C.H. (eds.) ICISC 2002. LNCS, vol. 2587, pp. 182–199. Springer, Heidelberg (2003)
14. Courtois, N.: Algebraic attacks over GF(2^k), application to HFE challenge 2 and Sflash-v2. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 201–217. Springer, Heidelberg (2004)
15. Courtois, N., Goubin, L., Meier, W., Tacier, J.-D.: Solving underdefined systems of multivariate quadratic equations. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 211–227. Springer, Heidelberg (2002)
16. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 392–407. Springer, Heidelberg (2000)
17. Courtois, N., Patarin, J.: About the XL algorithm over GF(2). In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 141–157. Springer, Heidelberg (2003)
18. Courtois, N., Pieprzyk, J.: Cryptanalysis of block ciphers with overdefined systems of equations. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 267–287. Springer, Heidelberg (2002)
19. Diem, C.: The XL-algorithm and a conjecture from commutative algebra. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 323–337. Springer, Heidelberg (2004)
20. Ding, J., Yang, B.-Y.: Multivariates polynomials for hashing. Cryptology ePrint Archive, Report 2007/137 (2007)
21. Eén, N., Sörensson, N.: MINISAT. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/
22. Faugère, J.-C.: A new efficient algorithm for computing Gröbner bases (F_4). Journal of Pure and Applied Algebra 139(61), 88 (1999)
23. Faugère, J.-C.: A new efficient algorithm for computing Gröbner bases without reductions to zero (F_5). In: ISSAC, pp. 75–83. ACM Press, New York (2002)
24. Faugère, J.-C., Joux, A.: Algebraic cryptanalysis of hidden field equation (HFE) cryptosystems using Gröbner bases. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 44–60. Springer, Heidelberg (2003)
25. Finiasz, M., Gaborit, P., Sendrier, N.: Improved fast syndrome based cryptographic hash functions. In: ECRYPT Workshop on hash functions (2007)
26. Garey, M., Johnson, D.: Computers and Intractability, a guide to the theory of NP-completeness, p. 251. Freeman, San Francisco (1979)
27. Haitner, I., Reingold, O.: A new interactive hashing theorem. In: IEEE Conference on Computational Complexity (2007)
28. Imai, H., Matsumoto, T.: A class of asymmetric crypto-systems based on polynomials over finite rings. In: IEEE International Symposium on Information Theory, pp. 131–132 (1983)
29. Lyubashevsky, V., Micciancio, D., Peikert, C., Rosen, A.: Provably secure FFT hashing. In: 2nd NIST Cryptographic Hash Function Workshop (2006)
30. McDonald, C., Charnes, C., Pieprzyk, J.: Attacking Bivium with MiniSat. Cryptology ePrint Archive, Report 2007/129

31. Naor, M., Reingold, O.: From unpredictability to indistinguishability: A simple construction of pseudo-random functions from MACs (extended abstract). In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 267–282. Springer, Heidelberg (1998)
32. NIST. Plan for new cryptographic hash functions http://www.nist.gov/hash-function/
33. Ong, H., Schnorr, C.-P., Shamir, A.: Efficient signature schemes based on polynomial equations. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 37–46. Springer, Heidelberg (1985)
34. Patarin, J.: Hidden fields equations (HFE) and isomorphisms of polynomials (IP). In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 33–48. Springer, Heidelberg (1996)
35. Shannon, C.E.: Communication theory of secrecy systems. Bell systems technical journal 28, 646–714 (1949)
36. Tamassia, R., Triandopoulos, N.: Computational bounds on hierarchical data processing with applications to information security. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 153–165. Springer, Heidelberg (2005)
37. Tang, X., Feng, Y.: A new efficient algorithm for solving systems of multivariate polynomials equations. Cryptology ePrint Archive, Report 2005/312 (2005)
38. Raddum, H.v., Semaev, I.: New technique for solving sparse equation systems. Cryptology ePrint Archive, Report 2006/475 (2006)
39. Vaudenay, S. (ed.): EUROCRYPT 2006. LNCS, vol. 4004. Springer, Heidelberg (2006)
40. Yang, B.-Y., Chen, O.C.-H., Bernstein, D.J., Chen, J.: Analysis of QUAD. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, Springer, Heidelberg (2007)

Appendix

Lemma 4 (Piling-up). $\{X_i\}_{0\le i<n}$ $\varepsilon_i = |\frac{1}{2} - P(X_i = 0)|$ $0 \le i < n$

$$P(X_0 \oplus \cdots \oplus X_{n-1} = 0) = \frac{1}{2} + 2^{n-1} \prod_{0\le i<n} \varepsilon_i \ .$$

Colliding Message Pair for 53-Step HAS-160

Florian Mendel* and Vincent Rijmen

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria
{Florian.Mendel,Vincent.Rijmen}@iaik.tugraz.at

Abstract. HAS-160 is an iterated cryptographic hash function that is widely used in Korea. In this article, we present a collision attack on the hash function HAS-160 reduced to 53-steps. The attack has a complexity of about 2^{35} hash computations. It is based on the work of Cho *et al.* presented at ICISC 2006. We improve the attack complexity of Cho *et al.* by a factor of about 2^{20} using a slightly different strategy for message modification in the first 20 steps of the hash function and present the first actual colliding message pair for 53-step HAS-160. Furthermore, we show how the attack can be extended to 59-step HAS-160 by using a characteristic spanning over two message blocks.

1 Introduction

HAS-160 is an iterated cryptographic hash function that is widely used in Korea and standardized by Korean government (TTAS.KO-12.0011/R1) [1]. It is an iterated cryptographic hash function that produces a 160-bit hash value. The design of HAS-160 is based on design principles of the MD4 family. Recently, weaknesses have been shown for several members of the MD4 family such as MD5 [5] and SHA-1 [6]. These breakthrough results by Wang *et al.* in the cryptanalysis of hash functions, are the motivation for intensive research in the design and analysis of hash functions. In [4], Yun *et al.* applied the techniques invented by Wang *et al.* in the cryptanalysis of MD5 and SHA-1 to the HAS-160 hash function. In their article they show that a collision can be found for HAS-160 reduced to 45 (out of 80) steps with a complexity of about 2^{12} 45-step HAS-160 computations. This attack was later extended by Cho *et al.* to HAS-160 reduced to 53 steps. Their attack has a complexity of about 2^{55} 53-step HAS-160 computations. This is not feasible on an ordinary PC in practice.

In this article, we show how to improve their attack by using a slightly different message modification technique to fulfill the conditions on the state variables in the first 20 steps of the hash function. With our method, we find a colliding message pair for 53-step HAS-160 with a complexity of about 2^{35} hash computations. This improves the attack complexity of the original attack of Cho *et al.* by a factor of 2^{20}, which makes the attack feasible in practice. Furthermore, we show how the attack can be extended to 59 steps of HAS-160. The attack has a complexity of about 2^{55} hash computations.

* This author is supported by the Austrian Science Fund (FWF), project P18138.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 324–334, 2007.

The remainder of this article is structured as follows. A description of the hash function is given in Section 2. The collision attack of Cho . , is described in Section 3. In Section 4, we describe the new improved collision attack. A sample colliding message pair is given in Section 5. Section 6 shows how the attack can be extended to 59 steps of HAS-160 by using a characteristic spanning over two message blocks. Finally, conclusions are presented in Section 7.

2 Description of the HAS-160

HAS-160 is an iterative hash function that processes 512-bit input message blocks and produces a 160-bit hash value. The design of HAS-160 is similar to the design principles of MD5 and SHA-1. In the following, we briefly describe the hash function. It basically consists of two parts: message expansion and state update transformation. A detailed description of the HAS-160 hash function is given in [1]. Throughout the remainder of this article, we will follow the notation given in Table 1.

Table 1. Notation

Notation	Meaning
$A \vee B$	logical OR of two bit-strings A and B
$A \wedge B$	logical AND of two bit-strings A and B
$A \oplus B$	logical XOR of two bit-strings A and B
$A \lll n$	bit-rotation of A by n positions to the left
M_j	message block j (512-bits)
m_i	message word i (32-bits)
w_i	expanded message word i (32-bits)

Message Expansion. The message expansion of HAS-160 is a permutation of 20 expanded message words w_i in each round. The 20 expanded message words w_i used in each round are constructed from the 16 input message words m_i as follows.

	Round 1	Round 2	Round 3	Round 4
w_0	m_0	m_0	m_0	m_0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
w_{15}	m_{15}	m_{15}	m_{15}	m_{15}
w_{16}	$w_0 \oplus w_1 \oplus w_2 \oplus w_3$	$w_3 \oplus w_6 \oplus w_9 \oplus w_{12}$	$w_{12} \oplus w_5 \oplus w_{14} \oplus w_7$	$w_7 \oplus w_2 \oplus w_{13} \oplus w_8$
w_{17}	$w_4 \oplus w_5 \oplus w_6 \oplus w_7$	$w_{15} \oplus w_2 \oplus w_5 \oplus w_8$	$w_0 \oplus w_9 \oplus w_2 \oplus w_{11}$	$w_3 \oplus w_{14} \oplus w_9 \oplus w_4$
w_{18}	$w_8 \oplus w_9 \oplus w_{10} \oplus w_{11}$	$w_{11} \oplus w_{14} \oplus w_1 \oplus w_4$	$w_4 \oplus w_{13} \oplus w_6 \oplus w_{15}$	$w_{15} \oplus w_{10} \oplus w_5 \oplus w_0$
w_{19}	$w_{12} \oplus w_{13} \oplus w_{14} \oplus w_{15}$	$w_7 \oplus w_{10} \oplus w_{13} \oplus w_0$	$w_8 \oplus w_1 \oplus w_{10} \oplus w_3$	$w_{11} \oplus w_6 \oplus w_1 \oplus w_{12}$

For the ordering of the expanded message words w_i the following permutation is used:

step i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Round 1	18	0	1	2	3	19	4	5	6	7	16	8	9	10	11	17	12	13	14	15
Round 2	18	3	6	9	12	19	15	2	5	8	16	11	14	1	4	17	7	10	13	0
Round 3	18	12	5	14	7	19	0	9	2	11	16	4	13	6	15	17	8	1	10	3
Round 4	18	7	2	13	8	19	3	14	9	4	16	15	10	5	0	17	11	6	1	12

State Update Transformation. The state update transformation of HAS-160 starts from a (fixed) initial value IV of five 32-bit registers and updates them in 4 rounds of 20 steps each. Figure 1 shows one step of the state update transformation of the hash function.

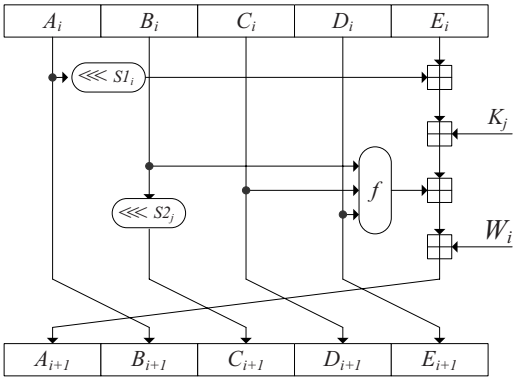


Fig. 1. The step function of HAS-160

Note that the function f is different in each round: f_0 is used in the first round, f_1 is used in round 2 and round 4, and f_2 is used in round 3.

$$f_0(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$$
$$f_1(x, y, z) = x \oplus y \oplus z$$
$$f_2(x, y, z) = (x \vee \neg z) \oplus y$$

A step constant K_j is added in every step; the constant is different for each round. For the actual values of the constants we refer to [1]. While rotation value $s_2 \in \{10, 17, 25, 30\}$ is different in each round of the hash function, the rotation value s_1 is different in each step of a round. The rotation value s_1 for each step of a round is given below.

After the last step of the state update transformation, the initial value and the output values of the last step are combined, resulting in the final value of one iteration known as Davies-Meyer hash construction (feed forward). In detail,

step i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
s_1	5	11	7	15	6	13	8	14	7	12	9	11	8	15	6	12	9	14	5	13

the feed forward is a word-wise modular addition of the IV and the output of the state update transformation. The result is the final hash value or the initial value for the next message block.

3 The Attack on 53-Step HAS-160

In this section, we will briefly describe the attack of Cho on 53-step HAS-160. A detailed description of the attack is given in [3]. For a good understanding of our results it is recommended to study it carefully.

The attack is based on recent results in cryptanalysis of hash functions [5,6]. It can be basically described as follows.

1. Find a characteristic for the hash function that holds with high probability after the first round of the hash function.
2. Find a characteristic (not necessarily with high probability) for the first round of the hash function.
3. Use message modification techniques [5] to fulfill conditions for the characteristic in the first round. This increases the probability of the characteristic.
4. Use random trials to find values for the message bits such that the message follows the characteristic.

3.1 Characteristic for 53-Step HAS-160

Finding a characteristic was the most difficult part of the attack. In Table 2, the characteristic which is used by Cho in the collision attack on 53-step HAS-160 is given. Note that we use the same differential path for our improved collision-attack in Section 4.

Signed-bit differences [5] are used to describe the differential path for 53-step HAS-160. To improve readability, only the differences in the expanded message words and state variable A for each step are given. Note that the differences of the state variables B, C, D, E are defined by the differences in state variable A.

3.2 Set of Sufficient Conditions

In order to guarantee that the message follows the characteristic given in Table 2, a set of conditions on the state variables has to be fulfilled. In Table 3, the set of sufficient conditions for the first 25 steps of the hash function is given.

3.3 Finding a Colliding Message Pair

In order to find a colliding message pair, we have to find a message that fulfills all the conditions given in Table 3. In total there are 434 conditions on the state variables. Therefore, the probability that a random message follows the characteristic can be estimated by 2^{-434}.

Table 2. Characteristic for 53-step HAS-160 (*cf.* [3])

step	ΔA	Δw
1	-32	32
2	$11, -12$	.
3	$18, \ldots, 21, -22$	.
4	$1, \ldots, 16, -17$	.
5	$7, 8, -9, 18, -19, 32$	32
6	$3, -4, 17, -20, -21, 22$	32
7	$-14, \ldots, -17, 18, 22, 29$	.
8	$4, -10, 11$	.
9	$13, -15, 16, 18, -19, 30, 31, -32$	32
10	$-10, -11, 12, -17, -24, 25$	.
11	$-1, 2, -13, \ldots, -15, 16, 28, -32$	32
12	$-8, 9, -17, -21, 22, 26$	32
13	$8, 10, 11, -12, 26, 27, -28$	.
14	$-10, 17, 18, -19, 28, \ldots, 30, -31$	.
15	$11, \ldots, 14, -15, -16, 20, -23, 26, -27$	.
16	$3, 5, 6, -7, -11, -12, 13, -20, 21, -22, -31, 32$	32
17	$-11, 18, -20, -26, 27$	.
18	$1, 5, 18, 20, 22, 27$	.
19	$4, 13, 30$	.
20	$-4, 11$	32
21	11	.
22	-30	32
23	.	32
24	15	.
25	-15	.
26	.	.
27	.	32
28	.	.
29	.	.
30	.	32
31	.	.
$\vdots$	$\vdots$	$\vdots$
53	.	.

However, the probability can be improved by using message modification techniques. The main idea of message modification is to use the degrees of freedom one has in the choice of the message words to fulfill conditions on the state variables. This improves the probability of the characteristic. It is clear that the number of conditions that can not be fulfilled by message modification techniques determine the final attack complexity.

In [3], Cho . . describe an algorithm for finding a colliding message pair for 53-step HAS-160. The algorithm has a complexity of about 2^{55} 53-step HAS-160 computations. It can be summarized as follows:

Table 3. A set of sufficient conditions on A_i for the differential path given in Table 2, where '**a**' denotes a condition $A_{i,j} = A_{i-1,j}$, '**b**' denotes a condition $A_{i,j} \neq A_{i-1,j}$, '**c**' denotes a condition $A_{i,j} \neq A_{i-1,j+7}$, '**d**' denotes a condition $A_{i,j} = A_{i-1,j-10}$, '**e**' denotes a condition $A_{i,j} = A_{i-1,j-17}$, and '**f**' denotes a condition $A_{i,j} \neq A_{i-1,j-17}$.

state variable	condition on bits 32-25	24-17	16-9	8-1	#conditions
A_1	1-------	--------	----110-	1-----aa	7
A_2	0100---1	--------	----100a	01---1--	12
A_3	1100aaa0	aa10000-	------10	10aaa0aa	25
A_4	11000-11	--10---1	00000000	00000000	26
A_5	01110111	00110100	00100111	00-11---	28
A_6	0--00111	110111-0	0000001-	00a010-0	27
A_7	1010101-	--0-1001	1111--10	-------1	19
A_8	100-0000	1-1a0---	-111-011	a0aa0a11	25
A_9	100-0101	10-0010-	0110----	-11-0-00	22
A_{10}	--000-10	10a01-01	1-1-0110	000010--	24
A_{11}	1a100010	-1001-10	01110001	01---001	27
A_{12}	1-1--100	10011111	1---0000	1-0--110	23
A_{13}	00--1001	11-00000	1---1001	0a110-11	25
A_{14}	010001-0	--101100	0--a101-	--111a00	24
A_{15}	---11101	-1000-10	11000001	01100a00	27
A_{16}	01010111	00101--0	1--01110	010010-1	27
A_{17}	111--011	01011-00	00-10110	0--10--1	24
A_{18}	01--00--	-00-0-00	11-1--11	---00b-0	18
A_{19}	--0-----	---d-c--	-0-0-1--	----0---	7
A_{20}	----0-b-	---b----	-----0--	---f1---	6
A_{21}	--f-0---	--------	-f-a-0--	--------	5
A_{22}	--1-----	---e----	--------	--------	2
A_{23}	----f---	--------	-e------	--------	2
A_{24}	--b-----	--------	-0------	--------	2
A_{25}	--------	--------	-1------	--------	1

1. Use basic message modification techniques to fulfill all conditions on state variables $A_1, \ldots, A_{10}$. This determines the message words m_0, m_1, m_2, m_3, m_4, m_5 ,m_6, m_7 and the values for $m_8 \oplus m_9 \oplus m_{10} \oplus m_{11}$ and $m_{12} \oplus m_{13} \oplus m_{14} \oplus m_{15}$. This adds only small additional cost to the attack complexity. Only 10 steps of HAS-160 have to be computed to determine all these message words.
2. At step 11, the dependent expanded message word $w_{16} = m_0 \oplus m_1 \oplus m_2 \oplus m_3$ is already fixed by the first step of the attack. Since there are 27 conditions on A_{11} in step 11, we may fulfill them with a probability of 2^{-27}. Hence, we have to repeat step 1 of the attack about 2^{27} times to find message words satisfying all conditions in steps 1-11.

   Finishing this step of the attack has a complexity of about $2^{27} \cdot 11$ step computations of HAS-160.

3. Use again basic message modification techniques to fulfill all conditions on A_{12}, A_{13}, A_{14}. This determines the message words m_8, m_9, m_{10}. Since these message words can be chosen freely, this adds only small additional cost (3 step computations of HAS-160) to the attack complexity.
4. At step 15 and 16, the dependent words m_{11} and $m_4 \oplus m_5 \oplus m_6 \oplus m_7$ are already fixed by the previous steps of the attack. Since there are $27+27 = 54$ conditions on A_{15} and A_{16}, we may fulfill these conditions with a probability of 2^{-54}. Since there are about 2^{24} possible choices for m_8, m_9, m_{10} in the third step of the attack (see Table 3), step 3 of the attack can only be repeated 2^{24} times. Hence, the whole attack has to be repeated about 2^{30} times to find message words following the characteristic in steps 1-16.
   Finishing this step of the attack has a complexity of about $2^{30} \cdot (2^{27} \cdot 11 + 2^{24} \cdot 5)$ step computations of HAS-160. This is approximately about 2^{55} 53-step HAS-160 computations.
5. To fulfill the conditions on A_{17}, A_{18}, A_{19} basic message modification techniques are used again. This determines the message words m_{12}, m_{13}, m_{14}. Since these values can be chosen freely, this adds only small additional cost (3 step computations of HAS-160) to the attack complexity.
6. After step 19, all the message words have been determined. Since there are still 18 conditions on $A_{20}, \ldots, A_{25}$ (see Table 3), we can satisfy them with a probability of 2^{-18}. Therefore, step 5 of the attack has to be repeated about 2^{18} times to fulfill all the conditions in steps 20-25. This adds negligible cost to the final attack complexity.

With this method a collision can be found in 53-step HAS-160 with a complexity of about 2^{55} 53-step HAS-160 computations. For a detailed description of the attack we refer to [3].

4 Improved Collision Attack

In this section, we show how the attack complexity can be reduced to 2^{35}. In the attack, we use the differential path of Cho [illegible] given in Section 3. To improve the attack complexity, we use a slightly modified strategy for message modification in the first 16 steps of the hash function. The main idea of our new method is to reduce the complexity of the collision search algorithm in steps 1-16. Therefore, we use the fact that an additional (first) message block can be used to generate an arbitrary IV (for the second block). This additional degree of freedom we use to reduce the complexity of the attack. The attack can be summarized as follows.

1. Choose arbitrary values for A_2, A_3, A_4, A_5, A_6 satisfying all conditions.
2. Apply message modification techniques to steps 7-15. This determines the message words $m_4, m_5, m_6, m_7, m_8, m_9, m_{10}, m_{11}$ and the value for $m_0 \oplus m_1 \oplus m_2 \oplus m_3$. At step 16, the dependent words m_4, m_5, m_6, m_7 are already used. Since there are 27 conditions at that step (see Table 3), we may satisfy all the conditions from step 7 up to step 16 with a probability of 2^{-27}. Hence, we have to compute about $2^{27} \cdot 10$ steps of HAS-160 to find message words that follow the characteristic from step 7 to 16.

3. Repeat step 2 of the attack about 2^7 times to get about 2^7 different values for $m_8 \oplus m_9 \oplus m_{10} \oplus m_{11}$ and save them in a list L. We will need these values in the next step of the attack.

   Finishing this step of the attack has a complexity of about $2^7 \cdot 2^{27} \cdot 10$ step computations of HAS-160. This is equivalent to about $2^{31.6}$ 53-step HAS-160 computations.
4. Use an arbitrary (first) message block to get a suitable IV (for the second block). Note that we have to calculate on average 2 IVs, since 1 condition on the IV has to be fulfilled to guarantee that the characteristic holds in the following steps.

   Calculate A_1 (for all values in L) and check if the 7 conditions on A_1 are satisfied. Since there are 7 conditions on A_1, we always expect to meet the conditions after trying all 2^7 values in the list L.

   Once, we have determined A_1, this also determines m_0, m_1, m_2, m_3 and $m_{12} \oplus m_{13} \oplus m_{14} \oplus m_{15}$. Since $m_0 \oplus m_1 \oplus m_2 \oplus m_3$ has already been fixed in the second step of the attack, this step of the attack succeeds with probability 2^{-32}.

   Hence, we have to repeat this step of the attack about 2^{32} times to find message words following the characteristic in steps 1-16. Finishing this step of the attack has a complexity of about $2^{32} \cdot (2^7 + 5 + 53 \cdot 2)$ step computations of HAS-160. This is approximately $2^{34.2}$ 53-step HAS-160 computations.
5. Do steps 17 to 25 as described in the original attack (see Section 3). The complexity of these steps can be neglected for the final attack complexity.

Hence, we can find a colliding message pair for 53-step HAS-160 with a complexity of about $2^{31.6} + 2^{34.2} \approx 2^{35}$ 53-step HAS-160 computations. Note that the complexity of the attack can be slightly improved by increasing the size of the list L. A colliding message pair for 53-step HAS-160 is given in the next section.

5 A Colliding Message for 53-step HAS-160

With our improved collision attack, we can construct a collision with a complexity of about 2^{35} 53-step HAS-160 computations. The colliding message pair is given in Table 4. Note that h_0 is the initial value, h_1 is the intermediate hash value after the first block, and h_2 is the final hash value after the second block.

Table 4. A colliding message pair for HAS-160

h_0	67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0
M_0	34338ECF ED111A03 EB2EE891 763594E3 96080160 4558A929 EC731044 B7BADD0B BC637C76 B21FA220 47493D4D B2AEAB79 A68354CF 5833D227 46DE18D7 F9FF5F3B
h_1	40D4B34F F1185C20 ADE02611 9B666A7E 34769338
M_1	4E8F4717 D8E79F84 89D8FE81 04B34CA7 01EA3C40 A364A502 059F6AB9 22774031 9F3E80CE D647A926 1F61242A A1E224AB 901A5AEE 1BCEEEB1 EDEAA891 31BDFF9A
M_1'	4E8F4717 D8E79F84 89D8FE81 84B34CA7 01EA3C40 A364A502 859F6AB9 22774031 1F3E80CE D647A926 1F61242A A1E224AB 901A5AEE 1BCEEEB1 EDEAA891 B1BDFF9A
ΔM_1	00000000 00000000 00000000 80000000 00000000 00000000 80000000 00000000 80000000 00000000 00000000 00000000 00000000 00000000 00000000 80000000
h_2	96D30020 DA815BDF DF265AB5 819CDE2E 5B887F3E
h_2'	96D30020 DA815BDF DF265AB5 819CDE2E 5B887F3E

6 Extending the Attack to 59 Steps of HAS-160

In this section, we show how the attack of Cho , , on 53-step HAS-160 can be extended to 59 steps. In [5], Wang , , show a collision attack for MD5 spanning over two message blocks. In the attack, they use a second message block to turn a near-collision after the first message block into a collision after feed forward of the second message block. Hence, two related near-collisions can be used to produce a two-block collision for the hash function. Therefore, a different characteristic is required in the first round of the second block. This is depicted in Fig. 2.

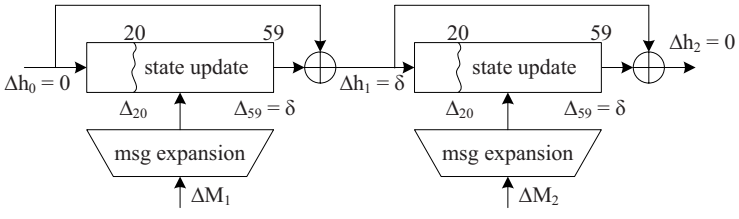


Fig. 2. A two-block collision in the hash function

For the attack on 59-step HAS-160, we use the characteristic given in Table 5. Note that the characteristic from step 20 to step 53 is equal to the characteristic

Table 5. Near-collision producing characteristic for 59-step HAS-160

step	ΔA	Δw
20	$-4, 11$	32
21	11	.
22	-30	32
23	.	32
24	15	.
25	-15	.
26	.	.
27	.	32
28	.	.
29	.	.
30	.	32
31	.	.
⋮	⋮	⋮
53	.	.
54	32	32
55	$6, 32$	32
56	$12, 18$	.
57	$21, -25, 27$	32
58	$3, -7, 9, 31$	.
59	$-4, 8, 11, 13$	.

used by Cho [3] in the attack on 53-step HAS-160. Hence, we can use the same characteristic as Cho [3] in the first 20 steps for the first message block. Since the characteristic has a probability of 2^{-32} to hold in steps 54 to 59, we have an attack complexity for the first message block of about 2^{50} (respectively 2^{55}) hash computations following the attack strategy described in Section 4 (respectively Section 3).

However, for the second message block a slightly different characteristic is needed for the first 20 steps of the hash function. In the analysis, we assume that such a characteristic can be found (once the first message block has been fixed) like it was done for 64-step SHA-1 [2]. Furthermore, it seems to be reasonable to assume that this characteristic has roughly the same probability as the characteristic used in the first message block. Thus, the complexity of the second block is about 2^{55} hash computations. This results in a complexity of about $2^{50} + 2^{55} \approx 2^{55}$ hash computations to construct a two-block collision for HAS-160 reduced to 59 steps.

7 Conclusion

In this article, we presented an improved collision attack on 53-step HAS-160 and an actual colliding message pair. Our new improved attack has a complexity of about 2^{35} 53-step HAS-160 computations. In the attack, we used a slightly modified strategy to do message modification in HAS-160. With this method, we can improve the previous attack by a factor of 2^{20}, which makes the attack feasible in practice. Furthermore, we show how the attack can be extended to 59 steps of HAS-160 by using a characteristic spanning over two message blocks. The attack has a complexity of about 2^{55} hash computations. However, whether or not the attack of Cho [3] can be extended to the full HAS-160 hash function remains a topic of further research.

Acknowledgment

The authors would like to thank Aaram Yun for fruitful discussions on this article.

References

1. Telecommunications Technology Association. Hash Function Standard Part 2: Hash Function Algorithm Standard (HAS-160), TTAS.KO-12.0011/R1 (December 2000)
2. De Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer, Heidelberg (2006)
3. Cho, H.-S., Park, S., Sung, S.H., Yun, A.: Collision Search Attack for 53-Step HAS-160. In: Rhee, M.S., Lee, B. (eds.) ICISC 2006. LNCS, vol. 4296, pp. 286–295. Springer, Heidelberg (2006)

4. Yun, A., Sung, S.H., Park, S., Chang, D., Hong, S., Cho, H.-S.: Finding Collision on 45-step HAS-160. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 146–155. Springer, Heidelberg (2006)
5. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
6. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)

Weaknesses in the HAS-V Compression Function

Florian Mendel* and Vincent Rijmen

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria
{Florian.Mendel,Vincent.Rijmen}@iaik.tugraz.at

Abstract. HAS-V is a cryptographic hash function which processes 1024-bit message blocks and produces a hash value of variable length. The design of the hash function is based on design principles of the MD4 family. Recently, weaknesses have been shown in members of this family. Therefore, the analysis of the HAS-V hash function is of great interest. To the best of our knowledge this is the first article that investigates the security of the HAS-V hash function. In this article, we point out several structural weaknesses in HAS-V which lead to pseudo-collision attacks on HAS-V with tailored output. Furthermore, we show that (second) preimages can be found for HAS-V with a complexity of about 2^{162} hash computations.

1 Introduction

Recently, weaknesses in many commonly used hash functions, such as MD5 and SHA-1 have been found [1,2,9,10]. These breakthrough results in the cryptanalysis of hash functions are the motivation for intensive research in the design and analysis of hash functions. In this article, we will study the HAS-V hash function in detail. It is an iterated hash function that processes 1024-bit message blocks and produces a hash value of variable length: 128, 160, 192, 224, 256, 288, and 320 bits. The HAS-V hash function was proposed by Park [illegible] at SAC 2000 [7]. The design of the hash function is very similar to the design principles of HAS-160, HAVAL, and RIPEMD. Since in all of these hash functions (recently) weaknesses have been shown [3,5,8,11], a detailed analysis of the HAS-V hash function is needed to get a good view on the security margins of the hash function. We are not aware of any other security analysis of the HAS-V hash function.

In this article, we show that the HAS-V hash function has several weaknesses that can be exploited to construct pseudo-collisions for the HAS-V hash function for all output sizes. Furthermore, we show that by using an alternative description of the hash function, which gives more insights in the design of HAS-V, we can construct (second) preimages with a complexity of about 2^{162} hash computations. Note that for an ideal hash function with an n-bit output size, one would expect a complexity of about 2^n. Hence, the security margins for HAS-V with output size 192, 224, 256, 288, and 320 bits are not as high as one would expect.

* This author is supported by the Austrian Science Fund (FWF), project P18138.

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 335–345, 2007.

The remainder of this article is structured as follows. A description of the hash function is given in Section 2. In Section 3, we give an alternative description of HAS-V, which gives more insights in the design of the hash function. We will use this description in Section 4 to show how pseudo-collisions can be constructed. In Section 5, we present a (second) preimage attack with a complexity of about 2^{162} hash computations. Finally, we present conclusions in Section 6.

2 Description of the HAS-V Hash Function

HAS-V is an iterative hash function that processes 1024-bit input message blocks and produces a hash value of variable length: $128, 160, \cdots, 288$, and 320 bits. The design of HAS-V is similar to the design principle of HAVAL, RIPEMD and HAS-160. It consists of two parallel streams. In each stream the state variables are updated (in 5 rounds) according to the expanded message words. After each round the state variables of the left and the right stream are interchanged, see Fig. 1. After the last round, the initial values and the output values of each stream are combined, resulting in the final value of one iteration.

After the last message block has been processed an output tailoring method is applied to construct a hash value of length 128, 160, 192, 224, 256, 288 or 320 bits. For a detailed description of this method we refer to [7].

In the following, we briefly describe the hash function. It basically consists of two parts: the message expansion and the state update transformation.

2.1 Message Expansion

The message expansion of HAS-V is a permutation of 20 expanded message words w_i in each round of a stream. First, the message block M_i of 1024 bits is split into 2 parts $\underline{M_i}$ and $\overline{M_i}$, where $\underline{M_i}$ are the lower 512 bits and $\overline{M_i}$ are

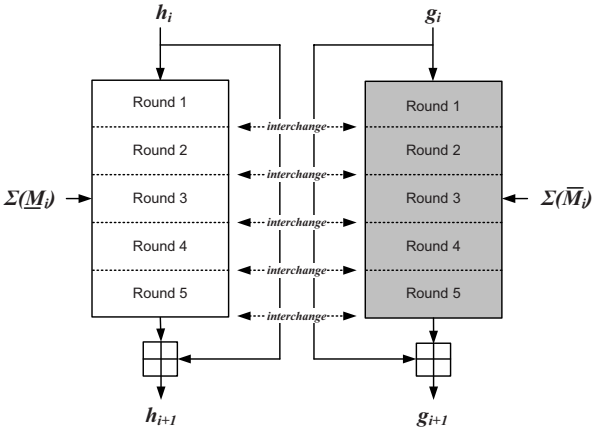


Fig. 1. The HAS-V compression function

the upper 512 bits of the message block M_i. $\underline{M_i}$ is then used to generate the expanded message words for the left stream and $\overline{M_i}$ is used to generate the expanded message words for the right stream. The 20 expanded message words w_i used in each round are constructed from the 16 input message words m_i as follows. The 16 message words m_i are copied to w_i. The remaining 4 expanded message words w_{16}, w_{17}, w_{18} and w_{19} are generated as shown below.

	w_{16}	w_{17}	w_{18}	w_{19}
Round 1	$w_0 \oplus w_1 \oplus w_2 \oplus w_3$	$w_4 \oplus w_5 \oplus w_6 \oplus w_7$	$w_8 \oplus w_9 \oplus w_{10} \oplus w_{11}$	$w_{12} \oplus w_{13} \oplus w_{14} \oplus w_{15}$
Round 2	$w_3 \oplus w_6 \oplus w_9 \oplus w_{12}$	$w_{15} \oplus w_2 \oplus w_5 \oplus w_8$	$w_{11} \oplus w_{14} \oplus w_1 \oplus w_4$	$w_7 \oplus w_{10} \oplus w_{13} \oplus w_0$
Round 3	$w_{12} \oplus w_5 \oplus w_{14} \oplus w_7$	$w_0 \oplus w_9 \oplus w_2 \oplus w_{11}$	$w_4 \oplus w_{13} \oplus w_6 \oplus w_{15}$	$w_8 \oplus w_1 \oplus w_{10} \oplus w_3$
Round 4	$w_7 \oplus w_2 \oplus w_{13} \oplus w_8$	$w_3 \oplus w_{14} \oplus w_9 \oplus w_4$	$w_{15} \oplus w_{10} \oplus w_5 \oplus w_0$	$w_{11} \oplus w_6 \oplus w_1 \oplus w_{12}$
Round 5	$w_{15} \oplus w_9 \oplus w_5 \oplus w_3$	$w_{12} \oplus w_8 \oplus w_6 \oplus w_2$	$w_{13} \oplus w_{11} \oplus w_7 \oplus w_1$	$w_{14} \oplus w_{10} \oplus w_4 \oplus w_0$

For the ordering of the expanded message words w_i in each round the following permutation is used:

step i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Round 1	18	0	1	2	3	19	4	5	6	7	16	8	9	10	11	17	12	13	14	15
Round 2	18	3	6	9	12	19	15	2	5	8	16	11	14	1	4	17	7	10	13	0
Round 3	18	12	5	14	7	19	0	9	2	11	16	4	13	6	15	17	8	1	10	3
Round 4	18	7	2	13	8	19	3	14	9	4	16	15	10	5	0	17	11	6	1	12
Round 5	18	15	9	5	3	19	12	8	6	2	16	13	11	7	1	17	14	10	4	0

2.2 State Update Transformation

The state update transformation of HAS-V starts from a (fixed) initial value IV of ten 32-bit registers (5 for each stream) and updates them in 5 rounds of 20 steps each. Figure 2 shows one step of the state update transformation of the HAS-V hash function.

$$
\begin{aligned}
A_{i+1} &= A_i \lll s + f(B_i, C_i, D_i, E_i) + W_i + K_i \\
B_{i+1} &= A_i \\
C_{i+1} &= B_i \ggg 2 \\
D_{i+1} &= C_i \\
E_{i+1} &= D_i
\end{aligned}
$$

The function f is different in each round. f_j is used for the j-th round in the left stream, f_{4-j} is used for the j-th round in the right stream ($j = 0, \ldots, 4$).

$$
\begin{aligned}
f_0(B, C, D, E) &= (B \wedge C) \oplus (\neg B \wedge D) \oplus (C \wedge E) \oplus (D \wedge E) \\
f_1(B, C, D, E) &= (B \wedge D) \oplus C \oplus E \\
f_2(B, C, D, E) &= (B \wedge C) \oplus (\neg B \wedge E) \oplus D \\
f_3(B, C, D, E) &= B \oplus (C \wedge D) \oplus E \\
f_4(B, C, D, E) &= (\neg B \wedge C) \oplus (B \wedge D) \oplus (C \wedge E) \oplus (D \wedge E)
\end{aligned}
$$

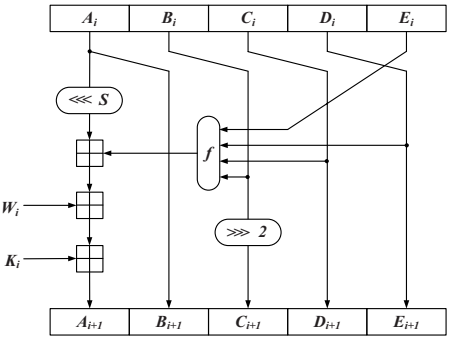


Fig. 2. The step function of HAS-V

A step constant K_i is added in every step; the constant is different for each round. For the actual values of the constants we refer to [7]. The rotation value s is different in each step of a round.

step i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
s	5	11	7	13	15	6	13	9	5	11	7	12	8	15	13	8	15	6	7	14

After the last step of the state update transformation, the initial value and the output values of the last step are combined, resulting in the final value of one iteration. In detail, the feed forward is a word-wise modular addition of the IV and the output of the state update transformation. The result is the final hash value or the initial value for the next message block.

3 Alternative Description of HAS-V

We present here a slightly different description of the HAS-V hash function, which gives more insights in the design of the hash function. By interchanging round 2 and round 4 between the two streams instead of interchanging the state variables after each round, we get an alternative description of the HAS-V hash function, where the two streams are independent, see Fig. 3. Note that a new (different) message expansion is used for the left and the right stream, denoted by Σ^1 and Σ^2.

Let $h(M_i, h_i)$ and $g(M_i, g_i)$ denote the state update function in the left and right stream. Then the compression function of HAS-V can be described as follows:

$$h_{i+1} = h_i + g(M_i, g_i)$$
$$g_{i+1} = g_i + h(M_i, h_i)$$

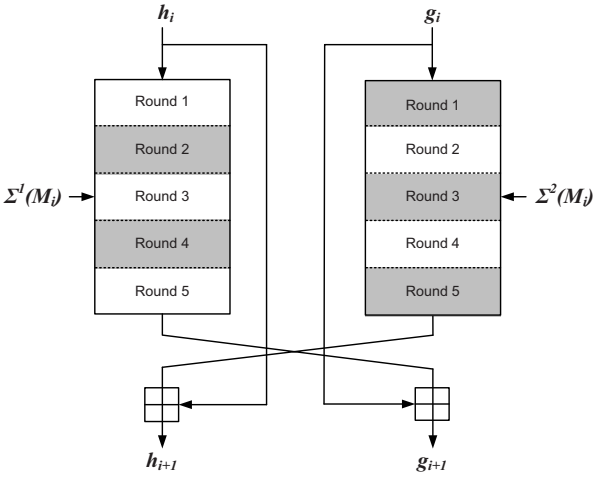


Fig. 3. Alternative description of the HAS-V compression function

For the remainder of this article, we will use this description of the HAS-V hash function. First, we show that a pseudo-collision can be constructed for HAS-V with a complexity lower than $2^{n/2}$ (which one would expect for an ideal hash function with an n-bit hash value). Second, we show that (second) preimages can be found for HAS-V with a complexity of about 2^{162} hash computations.

4 Pseudo-near-Collisions and Pseudo-collisions for the HAS-V Hash Function

In this section, we will show how pseudo-near-collisions and pseudo-collisions can be constructed for the HAS-V compression function by exploiting structural weaknesses in the step function of the hash function. The attack is based on the following two observations.

Observation 1. [illegible] f [illegible] ff [illegible]

Observation 2. [illegible] E [illegible] f [illegible]

Hence, a difference δ in state variable E can easily be canceled by exploiting the properties of the Boolean function f to absorb differences at its input. It is easy to see, that a difference δ in state variable E in the first round of the left stream will always cancel out if $C = D$.

$$f_0(B, C, D, E) = C \quad \Leftrightarrow \quad C = D$$

In a similar way also a difference δ' in E in the first round of the right stream will cancel out if $C = D$.

$$f_4(B, C, D, E) = C \quad \Leftrightarrow \quad C = D$$

We can use this to construct a pseudo-near-collision in the HAS-V hash function. By choosing the initial value of the left and right stream in the following way

$$h_0 = A_0^1||B_0^1||C_0^1||D_0^1||E_0^1 \quad \text{with} \quad C_0^1 = D_0^1$$

and

$$g_0 = A_0^2||B_0^2||C_0^2||D_0^2||E_0^2 \quad \text{with} \quad C_0^2 = D_0^2$$

we will always get a collision after the first step of the state update transformation in both streams for an arbitrary difference δ in E_0^1 in the left stream and δ' in E_0^2 in the right stream. Of course, the feed forward after the last step of the state update transformation will destroy the collision, resulting in a pseudo-near-collision for the HAS-V hash function:

$$(0, 0, 0, 0, \delta, 0, 0, 0, 0, \delta') \rightarrow (0, 0, 0, 0, \delta, 0, 0, 0, 0, \delta')$$

Note that this holds with probability 1 and is independent of the message M. The pseudo-near-collision for HAS-V can be turned into a pseudo-collision for HAS-V with tailored output. This is described in more detail in the next section.

4.1 Pseudo-collisions in HAS-V with Tailored Output

The designers of HAS-V defined several variants of HAS-V with tailored output. For instance, to get a hash value of 160 bits the output of the left an right stream are combined in the following way:

$$h + g = A^1 + A^2||B^1 + B^2||C^1 + C^2||D^1 + D^2||E^1 + E^2$$

For this variant, we can construct a pseudo-collision by using the pseudo-near-collision described in the previous section. By choosing a difference δ in E_0^1 in the left stream and a difference $-\delta$ in E_0^2 in the right stream, we get a pseudo-collision after adding the values of both streams (output tailoring).

$$E^1 + E^2 = (E^1 + \delta) + (E^2 - \delta)$$

This holds with probability 1 in the HAS-V hash function and is independent of the message M. Note that pseudo-collisions can be constructed for the other variants of HAS-V (with tailored output) in a similar way.

4.2 Pseudo-collisions in HAS-V

We can turn the pseudo-near-collision for HAS-V into a pseudo-collision by using a generic birthday attack. The main idea is to use the degree of freedom we have in the choice of the differences in E_0^1 and E_0^2 in the left and right stream to decrease the work needed for a birthday attack. The attack can be summarized as follows:

1. Choose random values for the initial value of the left and right stream of the hash function with $C_0^1 = D_0^1$ and $C_0^2 = D_0^2$.

2. Do a generic birthday attack to find a message pair (M, M^*) such that $\Delta A^1 = \cdots = \Delta D^1 = 0$ and $\Delta A^2 = \cdots = \Delta D^2 = 0$. This has a complexity of about 2^{128} evaluations of the compression function of HAS-V.
3. To cancel the difference in E^1 and E^2 we use the fact that we can inject an arbitrary difference δ in E_0^1 and δ' in E_0^2 to cancel the differences in E^1 and E^2. This leads to a pseudo-collision in HAS-V hash function.

Hence, we can construct a pseudo-collision in the HAS-V hash function with a complexity of about 2^{128} instead of 2^{160} hash computations.

5 (Second) Preimages for the HAS-V Hash Function

In this section, we will describe two methods to construct a (second) preimage for the HAS-V hash function. Both attacks are based on structural weaknesses in the HAS-V hash function. For the analysis, we will use the description of HAS-V given in Section 3. With the first method a (second) preimage can be constructed with a complexity of about 2^{241} hash computations instead of 2^{320}, as one would expect for a hash function with a 320-bit hash value. The first attack is based on the following observation.

Observation 3.

Algorithm 1. A way to invert the compression function

Input: The final hash value $h_{i+1}||g_{i+1}$ and an arbitrary intermediate hash value h_i.
Output: The intermediate hash value g_i such that HAS-V$(M_i, h_i||g_i) = h_{i+1}||g_{i+1}$.

– Guess M_i and calculate:

$$g_i = g_{i+1} - h(M_i, h_i)$$

– Check if the following equation holds:

$$h_{i+1} = h_i + g(M_i, g_i).$$

Hence, a correct choice for M_i can be found in 2^{160} hash computations.

Based on Algorithm 1, the HAS-V hash function is invertible with respect to the chaining variables. We can find the intermediate hash value g_i and the message block M_i in about 2^{160} applications of the compression function of the HAS-V hash function. We can use this to construct a (second) preimage in the HAS-V hash function with a complexity of about 2^{241} hash computations using the following observation.

Observation 4.

Algorithm 2. A (second) preimage in the HAS-V hash function

Input: The final hash value $h_{i+2}||g_{i+2}$ and the intermediate hash value $h_i||g_i$
Output: The message $M = M_i||M_{i+1}$ such that HAS-V$(M, h_i||g_i) = h_{i+2}||g_{i+2}$.

- Calculate and store 2^{80} candidates for $h_{i+1}||g_{i+1}$ in the list L resulting from a backward computation of the compression function using Algorithm 1. ($2^{80} \cdot 2^{160}$ applications of the compression function)
- Calculate $h_{i+1}||g_{i+1}$ resulting from a forward computation of the compression function an check for a match in L.
- After calculating at most 2^{240} candidates for $h_{i+1}||g_{i+1}$ one expect find a matching entry (collision) in L and hence a (second) preimage for HAS-V. Note that a collision is likely to exist due to the birthday paradox.

Based on Algorithm 2, we can find a (second) preimage for the HAS-V hash function with complexity of about $2 \cdot 2^{240}$ applications of the compression function. Note that a similar attack was applied in the past on MDC-2 [4]. However, we can further improve the attack by using the following observation.

Observation 5. [illegible] h_i

Algorithm 3. Constructing a fixed-point in the left stream of HAS-V

Input: The intermediate hash value h_i.
Output: The pair (g_i, g_{i+1}) and M_i such that HAS-V$(M_i, h_i||g_i) = h_i||g_{i+1}$.

- Choose an arbitrary value for M_i and calculate:

$$g_i = g^{-1}(M_i, 0)$$

$$g_{i+1} = g_i + h(M_i, h_i)$$

Based on Algorithm 3, we can construct a fixed-point in the left stream of HAS-V for an arbitrary value of h_i. We can use this to construct a (second) preimage for the HAS-V hash function with a complexity of about 2^{162} hash computations. The main idea of the attack is to use Algorithm 3 to construct many fixed-points for the left stream of the hash function (with the same value of h_i) and save them in a list L and then combine these entries in such a way that we get a (second) preimage for the HAS-V hash function.

Assume we are given the final hash value $h_i||g_i$ and let $h_0||g_0$ denote the initial value of HAS-V. Then the attack can be summarized as follows:

1. Use Algorithm 1 with input $h_i||g_i$ and $h_{i-1} = h_0$ to get g_{i-1} and M_{i-1} such that HAS-V$(M_{i-1}, h_{i-1}||g_{i-1}) = h_i||g_i$. This step of the attack has a complexity of about 2^{160} hash computations and ensures that $h_{i-1} = h_0$. Note that this is needed for the attack to work.

2. Calculate and store $2 \cdot 2^{160}$ candidates for (g_{j-1}, g_j) in the list L using Algorithm 3 with input h_0. Note that each entry in the list L has a fixed-point in the left stream: HAS-V$(M_{j-1}, h_0||g_{j-1}) = h_0||g_j$. We will use this in the next step of the attack to construct a (second) preimage in the HAS-V hash function. Constructing the list L has a complexity of about 2^{161} hash computations. Furthermore, we expect to have always two entries in L where the first component g_{j-1} is equal and we also expect to have always two entries in L where the second component g_j is equal.
3. To construct a (second) preimage in the HAS-V hash function, we use the entries in the list L. Starting from g_0, we construct a tree using the entries in the list L. For each node in the tree we expect to get two new nodes on the next level (see Fig. 4), since we have two entries in list L where the first component is the same. Hence, the number of nodes at level k is 2^k.

   To construct a preimage for HAS-V we need to meet g_{i-1} fixed in the first step of the attack. Therefore, we construct a second tree starting from g_{i-1} also using the entries in the list L. Since we have always two entries in the list L, where the second component is equal, we get for each node on level k two new nodes on the next level. Hence, the number of nodes at level k is also 2^k. It is easy to see, that for $k = 80$ we get 2^{80} nodes in each of the two trees. Therefore, we expect to find a common node in both trees and hence, a (second) preimage for the HAS-V hash function. Note that a common node in both trees is likely to exist due to the birthday paradox.

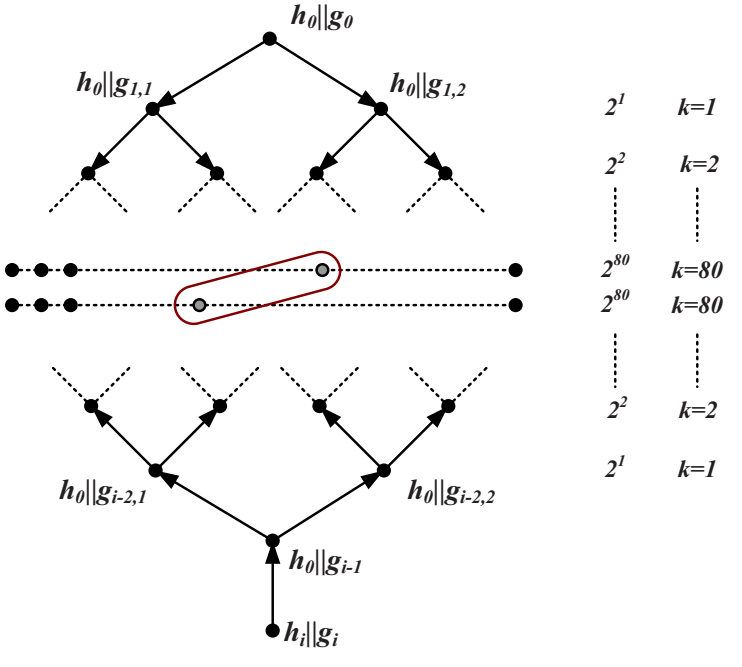


Fig. 4. Constructing a (second) preimage for the HAS-V hash function

With this method we can find a (second) preimage for HAS-V with a complexity of about 2^{162} hash computations and a message consisting of 161 (80+80+1) message blocks. Note that the same attack can be used to produce (second) preimages for HAS-V with tailored output. Hence, the security margins for the HAS-V hash function with output size of 192, 224, 256, 288, and 320 bits are not as high as one would expect for a hash function with that output size.

6 Conclusion

In this article, we showed several weaknesses in the HAS-V hash function. Based on these weaknesses, we presented a pseudo-collision for the HAS-V hash function with tailored output. As an example we show a pseudo-collision for HAS-V with a (tailored) output size of 160 bits. It has probability 1 to hold in the HAS-V hash function and is independent of the message M. Note that similar attacks (also with probability 1 and independent of the message) can be applied for the other variants of HAS-V with tailored output. Furthermore, we showed that also for the 320-bit variant of HAS-V a pseudo-collision can be found with a complexity of about 2^{128} hash computations.

Moreover, we presented a (second) preimage attack on the HAS-V hash function. In the attack we exploit the fact, that due to the design of the HAS-V hash function it is easy to construct a fixed-point in the left stream of the hash function for an arbitrary input value. The attack has a complexity of about 2^{162} hash computations. Hence, the security margins of HAS-V (with output size of 192, 224, 256, 288 and 320 bits) are not as high as one would expect for a hash function with an n-bit output size.

Acknowledgement

The authors wish to thank Norbert Pramstaller, and the anonymous referees for useful comments and discussions.

References

1. Black, J., Cochran, M., Highland, T.: A Study of the MD5 Attacks: Insights and Improvements. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 262–277. Springer, Heidelberg (2006)
2. De Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer, Heidelberg (2006)
3. Cho, H.-S., Park, S., Sung, S.H., Yun, A.: Collision Search Attack for 53-Step HAS-160. In: Rhee, M.S., Lee, B. (eds.) ICISC 2006. LNCS, vol. 4296, pp. 286–295. Springer, Heidelberg (2006)
4. Lai, X., Massey, J.L.: Hash Function Based on Block Ciphers. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 55–70. Springer, Heidelberg (1993)

5. Mendel. F.: Colliding Message Pair for 53-Step HAS-160. Cryptology ePrint Archive, Report 2006/334 (2006), http://eprint.iacr.org/
6. Merkle, R.C., Hellman, M.E.: On the Security of Multiple Encryption. Commun. ACM 24(7), 465–467 (1981)
7. Park, N.K., Hwang, J.H., Lee, P.J.: HAS-V: A New Hash Function with Variable Output Length. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012, pp. 202–216. Springer, Heidelberg (2001)
8. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 1–18. Springer, Heidelberg (2005)
9. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
10. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
11. Yu, H., Wang, X., Yun, A., Park, S.: Cryptanalysis of the Full HAVAL with 4 and 5 Passes. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 89–110. Springer, Heidelberg (2006)

Security-Preserving Asymmetric Protocol Encapsulation

Raphael C.-W. Phan and Serge Vaudenay

Ecole Polytechnique Fédérale de Lausanne (EPFL),
CH-1015 Lausanne, Switzerland
{raphael.phan,serge.vaudenay}@epfl.ch

Abstract. Query-response based protocols between a client and a server such as SSL, TLS, SSH are asymmetric in the sense that the querying client and the responding server play different roles, and for which there is a need for two-way linkability between queries and responses within the protocol. We are motivated by the observation that though results exist in other related contexts, no provably secure scheme has been applied to the setting of client-server protocols, which differ from conventional communications on the above points. We show how to secure the communication of queries and responses in these client-server protocols in a provably secure setting. In doing so, we propose a new primitive: a query-response encapsulation scheme; we give an instantiation, and we demonstrate how this primitive can be used for our purpose. In our proof of secure encapsulation, we show how to preserve the notion of "local-security".

1 Introduction

In this paper, we show how to secure the communication of generic client-server type protocols. These are protocols which involve query and response messages for arbitrary number of rounds, and are asymmetric in the sense that parties on either side of the protocol have differing roles.

Asymmetric protocols. Client-server type protocols exist in many settings [8,9,17,18,19,22,25], e.g. in networks where clients are connected to application servers, database servers, file servers, or mail servers etc. A typical example of such a protocol is a client browser running on a personal computer interacting with a web server sitting on a remote machine. In this setting, it is desired that at the worst case, having the client access the server is at least as secure as just having the client run in isolation. This in fact shows that access to the server does not compromise the security of the client. We denote this notion as [illegible]

Security preservation. By [illegible], we mean the security of the client when it is in isolation. In the generic sense, this can be formally modelled as a

K.-H. Nam and G. Rhee (Eds.): ICISC 2007, LNCS 4817, pp. 346–366, 2007.

game following some rules, played between an adversary and a judge challenger who determines if the adversary succeeded in winning the game.

We want to show that when the client is additionally allowed access to a server, then this can be done such that the local security of the client is preserved, i.e. additional server access does not have adverse affects on the client's security.

In the ideal case, interaction with the server is not observable by the adversary. In practice, we show how this can be done via a form of asymmetric protocol encapsulation that preserves local security. This result applies to any asymmetric protocol of arbitrary rounds.

Examples. One example of an asymmetric client-server protocol is database systems where the client system $\mathcal{S}$ represents the application querying the database server $\mathcal{O}$. So far, existing security models for databases [12,14] are different from our context because they only consider security on the database server $\mathcal{O}$ side. Our focus is on the local security on the client system $\mathcal{S}$ side and how this is affected by the $\mathcal{S} \leftrightarrow \mathcal{O}$ interaction. For instance, let us assume a company $\mathcal{A}$ making queries to a low-cost airline company's online database. A competitor $\mathcal{B}$ to $\mathcal{A}$ who suspects that $\mathcal{A}$ is interested in some type of event can forge a flyer for a fake event at some given location and date, check on the air ticket cost, send the fake event advertisement to $\mathcal{A}$. When $\mathcal{B}$ sees that some request to the airline company was made by $\mathcal{A}$, it can look at the prices again and see if the price increased because $\mathcal{A}$ has just taken the low cost seat. If the link between $\mathcal{A}$ and the airline company is perfectly secure, $\mathcal{B}$ does not even see when the request is made. But when the link is implemented in practice e.g. using SSL through an insecure channel such as the Internet, $\mathcal{B}$ gets this private information. Here, we can see that the encapsulated system is insecure even though the implementations of the client and the server are secure.

Another example is an EMV protocol [11] where the credit-card payment terminal $\mathcal{S}$ makes queries to a bank $\mathcal{O}$ to verify a message authentication code (MAC) issued by a credit card. Here, the response of the bank is a YES/NO reply. Clearly, this system can reach pretty good security if the client-server communication is made through a perfectly secure link. When instantiated in practice with an insecure channel, things could still easily go wrong even though some encapsulation of the channel is applied that achieves some strong cryptographic notion of confidentiality and authentication. For instance, an adversary could try to replay a positive response from the bank without breaking the underlying cryptographic primitives. Indeed, if the primitives do not provide strong linkability between the query and its response, a fake credit card can easily be used for payment. This example shows that some extra cryptographic property must be assumed on the encapsulation scheme used to secure the client-server interaction.

In this paper, our goal is to make explicit this property, to show how to achieve it, and to show that local security can be preserved while moving from a secure (ideal) channel to an insecure but encapsulated one.

Related work. The seminal introduction of public-key encryption by Diffie and Hellman [10] initiated serious work in the public domain into solving the problem of securely encapsulating insecure channels. In fact, the Diffie-Hellman key exchange of [10] was the first to allow the establishment of secure channels without pre-sharing any common secrets between parties, thus solving the key distribution problem inherent in conventional symmetric encryption. With a key exchange protocol, parties can securely establish common shared keys that are then used to encrypt subsequent communications. In a sense, major research has focused on initializing the secure channel hence the study of key exchange protocols, in contrast to the subsequent step after initialization: the encapsulation of the communication channel.

In fact, the study of how to encapsulate an insecure channel is an interesting problem in itself because it is often that a channel needs not only be encapsulated in terms of secrecy, authentication and integrity, but also resistant against other forms of malicious manipulation e.g. reordering or replaying of messages, exploitation of one legitimate party to answer challenges from the other party etc. Indeed, carefully analyzing attacks applied to key exchange protocols reveals some of these problems too, because essentially, it can be seen by looking at the internals of a key exchange protocol that it is a sequence of message transmissions over an insecure channel.

Work initiated by Canetti and Krawczyk [5,6,18,20,21] showed how secure encapsulated channels can be obtained as a by-product of secure key establishment, for the particular setting where each transmitted message is independent of other messages in the encapsulated channel.

Their line of work differs on several points from the context of client-server communication considered in this paper. Client-server communication is asymmetric i.e. each party on either side has a different role, so it matters which side the party is on. More importantly, this leads to adversely different security requirements e.g. there is a need for two-way linkability between the transmitted query and the received response, a requirement not needed for conventional secure channels in [5,6,18,20,21]. Furthermore, we need only communication, i.e. we consider both ends separately and where it is only required to authenticate the server side (see [17]). Or alternatively see [19,22] for a kind of asymmetric authentication where variable levels of authentication apply depending on which side the party is on.

In essence, the motivation behind our considering semi-authentication is that our local security consideration essentially reduces the security of the client-server interaction to the security on the client side. This also models the client-server systems in practice where a single server provides equal access of its resources to any client.

A different line of work related to client-server systems is the study of provably secure schemes the server e.g. provable security of the database server in the case of database systems [12,14].

Our contributions. Our main result is a generic notion for provably secure encapsulation of asymmetric client-server protocols. To the best of our knowledge, our paper is the first work to treat this problem in a provably secure setting. Our results generalize to asymmetric protocols with arbitrary number of rounds. To this end, we give a generic construction for a new primitive, so called the [illegible], that provides confidentiality and semi-authentication of query and response messages. We also give an example instantiation and prove its security.

This Q/REM primitive allows to design client-server protocol communications in a modular fashion. On the one hand, specific details of a client-server protocol are abstracted away during the security proof of encapsulation. On the other, once it is established that a secure encapsulated channel is in place between client and server, the intrinsic details of the protocol can be designed in this ideal encapsulated setting without any further need to treat requirements for confidentiality, integrity and authentication.

In some sense, this modular abstraction bears resemblance to the approach in [5], where key establishment protocols are constructed assuming the existence of encapsulated (only in the sense of authentication) channels, while at a different design level it is studied how protocols designed in the encapsulated model relate to the practical model by the use of [illegible].

In our proof of secure asymmetric protocol encapsulation, we show that the local security on the client side is preserved provided that the communication protocol is locally secure when the adversary knows when the communication happens and its length, and that the server is protected against replay attacks or is a stateless oracle.

2 A Generic Construction for Query-Response Encapsulation

We give here a generic construction for a query/response encapsulation mechanism (Q/REM) primitive and define its corresponding security notions of indistinguishability and authentication. This result in itself is of independent interest; for instance we know of no related primitive that achieves the notion of authentication. The QEM is a regular message transmission primitive yielding a symmetric key which can be re-used to encapsulate a response. The REM re-uses this key. It is then shown how this Q/REM primitive can be used to build an asymmetric encapsulation protocol.

2.1 Query/Response Encapsulation Mechanism (Q/REM)

To be precise, the Q/REM primitive is given in Definition 1.

Definition 1 (Q/REM).

A *query/response encapsulation mechanism (Q/REM)* is a 5-tuple of algorithms.

$\langle pk, sk\rangle \leftarrow QEM.Key(1^\lambda)$:	a probabilistic polynomial time (PPT) algorithm taking as input a security parameter λ, and returns a pair $\langle pk, sk\rangle$ of matching public and private keys.
$\langle K, A\rangle \leftarrow QEM.Enc(pk, q)$:	a PPT algorithm taking as input a public key pk and query q; outputs an ephemeral key/encapsulation pair $\langle K, A\rangle$.
$\langle K, q\rangle \leftarrow QEM.Dec(sk, A)$:	a deterministic polynomial time (DPT) algorithm taking as input a private key sk and an encapsulation A; outputs an ephemeral key K and a decapsulated query q, or a special symbol $\perp$ implying A was invalid.
$B \leftarrow REM.Enc(K, i, r)$:	a PPT algorithm taking as input an ephemeral key K, counter i and a response r, and outputs an encapsulation B.
$r \leftarrow REM.Dec(K, i, B)$:	a DPT algorithm taking as input an ephemeral key K, counter i and an encapsulation B, and outputs a decapsulated response r or a special symbol $\perp$ implying B was invalid.

For , it is required that for any $\langle pk, sk\rangle$ output by $QEM.Key(\cdot)$ and for any $\langle K, A\rangle$ output by $QEM.Enc(pk, q)$, then $QEM.Dec(sk, A) = \langle K, q\rangle$; and furthermore, for any output B by $REM.Enc(K, i, r)$, then $REM.Dec(K, i, B) = r$.

Security notions. The security of Q/REM is captured by two notions: in a IND-CCA vein (Definition 2), and (Definition 3).

Definition 2 (Indistinguishability of Q/REM).

$QEM.Key(1^\lambda)$ pk sk pk $OQEM.Dec(A)$ $OQEM.Dec(\cdot)$ QEM $QEM.Dec(sk, A)$ q^* $b \in \{0,1\}$ $q_0 = q^*$ q_1 $QEM.Enc(pk, q_b)$ $\langle K, A^*\rangle$ A^*

$OQEM.Dec(A^*)$ $OIREM.Enc(i,r)$ $OIREM.Enc(\cdot)$ $B \leftarrow REM.Enc(K,i,r)$ $b=0$ $B \leftarrow REM.Enc(K,i,r^*)$ r^* b $\tilde{b} \in \{0,1\}$

$\pi_{\mathtt{Q/REM}}$ $\mathcal{A}$ $\pi_{\mathtt{Q/REM}}$ $\mathcal{A}$

$$Adv^{\mathtt{IND-CCA}}_{\mathcal{A},\pi_{\mathtt{Q/REM}}}(\lambda) = |\Pr[\tilde{b}=b] - 1/2|.$$

$\pi_{\mathtt{Q/REM}}$ $(\varepsilon(\lambda), c(\lambda))$ $Adv^{\mathtt{IND-CCA}}_{\mathcal{A},\pi_{\mathtt{Q/REM}}}(\lambda)$ $\varepsilon(\lambda)$ $\mathcal{A}$ $c(\lambda)$ $c(\lambda)$ $\varepsilon(\lambda)$ $(\varepsilon(\lambda), c(\lambda))$

Definition 3 (Authentication of Q/REM).

$QEM.Key(1^\lambda)$ pk sk pk $OQEM.Dec(\cdot)$ q $QEM.Enc(pk,q)$ $\langle K, A^* \rangle$ A^* $OQEM.Dec(A^*)$ $OREM.Enc(\cdot)$ $OREM.Enc(i,r)$ $REM.Enc(K,i,r)$ B^* $\langle i, B \rangle$ $REM.Dec(K,i,B)$ B $OREM.Enc(\cdot)$

$\pi_{\mathtt{Q/REM}}$ $\mathcal{A}$ $\pi_{\mathtt{Q/REM}}$ $\mathcal{A}$

$$Adv^{\mathtt{AUTH-CCA}}_{\mathcal{A},\pi_{\mathtt{Q/REM}}}(\lambda) = \Pr[\mathcal{A} \text{ succeeds}].$$

$\pi_{\mathtt{Q/REM}}$ $(\varepsilon(\lambda), c(\lambda))$ $Adv^{\mathtt{AUTH-CCA}}_{\mathcal{A},\pi_{\mathtt{Q/REM}}}(\lambda)$ $\varepsilon(\lambda)$ $\mathcal{A}$ $c(\lambda)$ $c(\lambda)$ $\varepsilon(\lambda)$ $(\varepsilon(\lambda), c(\lambda))$

Definition 4. $(\varepsilon_{sem}, \varepsilon_{auth}, c)$ (ε_{sem}, c) (ε_{auth}, c)

2.2 Asymmetric Protocol Encapsulation with Q/REM

We can secure any 2-party asymmetric protocol between a Client and a Server for arbitrary number of rounds, by using a Q/REM. For this protocol, given input q the Client system should obtain the output $r = \mathcal{O}(q)$ from the Server oracle $\mathcal{O}$. We construct an asymmetric (query-response) encapsulation protocol π_{QR}. A trusted communication channel (e.g. using a trusted third party) is required at the initialization stage to authenticate the public key pk of the Server to the Client.

Initialization: Server runs $QEM.Key(1^\lambda)$, obtains $\langle pk,sk\rangle$, and stores $\langle pk, sk\rangle$. Client obtains pk in a trusted way.

Query Generation: Upon input query q, Client runs $QEM.Enc(pk, q)$, obtaining $\langle K, A\rangle$, where K is an ephemeral (secret) key and A is the encapsulated query. Client sends A to the Server.

Response Generation: Server runs $QEM.Dec(sk, A)$ and obtains $\langle K, q\rangle$, where K is the ephemeral key, or obtains $\perp$ if A is invalid and therefore aborts. Server inputs the query q to its internal oracle $\mathcal{O}$ and obtains the response $r = \mathcal{O}(q)$. Server runs $REM.Enc(K, 1, r)$ and obtains B the encryption of response r under secret key K. Server sends B to the Client.

Response Verification: Client runs $REM.Dec(K, 1, B)$ and obtains decapsulated response r or $\perp$ if B is invalid and therefore aborts. Otherwise output r.

Values of $i > 1$ in REM will be used in protocols with several rounds.

2.3 Instantiating Q/REMs

The generic Q/REM primitive in Definition 1 can in fact be instantiated in several ways using public-key encryption or signcryption with one-pass authenticated encryption, or symmetric encryption with message authentication etc.

For completeness, we give a concrete instantiation example based on a secure KEM [7] and secure Authenticated Encryption (AE) [2,24], and prove its security. Here, we use AE instead of conventional DEM [7] because we need the authentication property and the ability to include a header to re-use a key. Basically, our QEM instantiation is a KEM together with an AE with header 0; subsequent REMs use AE with header set to the counter i.

To the best of our knowledge, this is the first time that a KEM+AE composition is applied for the [illegible] context whereas previous work used KEM+DEM [7,1] or KEM+AE [16] for achieving indistinguishability, and that for symmetric independent-message communication.

Q/REM Instantiation based on KEM and AE

1. $\langle pk, sk\rangle \leftarrow QEM.Key(1^\lambda)$: This is exactly $KEM.Key(1^\lambda)$.
2. $\langle K, A\rangle \leftarrow QEM.Enc(pk, q)$: First, $\langle K, C_0\rangle \leftarrow KEM.Enc(pk)$. This generates the encapsulation in C_0 of an AE key K. Then, $C \leftarrow AE.Enc(K, 0, q)$. The output of $QEM.Enc(\cdot)$ is A where $A = C_0||C$ is the encapsulation output of QEM.

3. $\langle K, q\rangle \leftarrow QEM.Dec(sk, A)$: Parse $A = C_0||C$. Then, $K \leftarrow KEM.Dec(sk, C_0)$: Given as input a private key sk and a key encapsulation C_0, it outputs the decapsulated secret key K, or $\perp$ if the encapsulation C_0 is invalid. Finally, $q \leftarrow AE.Dec(K, 0, C)$: Given as input a secret key K and a message ciphertext C, it outputs the decrypted query q, or outputs $\perp$ if the ciphertext C is not authenticated. The output of $QEM.Dec(\cdot)$ is q or $\perp$.
4. $B \leftarrow REM.Enc(K, i, r)$: Do $B \leftarrow AE.Enc(K, i, r)$: Given a secret key K, a counter i and a response r, it outputs the authenticated ciphertext B. The output of $REM.Enc(\cdot)$ is B.
5. $r \leftarrow REM.Dec(K, i, B)$: Do $r \leftarrow AE.Dec(K, i, B)$ or $\perp$ if B is not authenticated. The output of $REM.Dec(\cdot)$ is r or $\perp$.

The following result shows that this Q/REM instantiation is secure; a detailed proof is in Appendix A.

Theorem 1.

3 Provably Secure Encapsulation

We prove here how to secure asymmetric protocols with arbitrary rounds, involving two parties: a client system $\mathcal{S}$ issuing an initial query q (resp. subsequent query denoted r_t) to a server oracle $\mathcal{O}$ who replies with response r (resp. r_{t+1}). Both parties interact with an (a priori untrusted) environment $\mathcal{E}$. We term as "system" the $\mathcal{S} \leftrightarrow \mathcal{O}$ combination which lives with environment $\mathcal{E}$.

Ideally, the environment does not see the $\mathcal{S} \leftrightarrow \mathcal{O}$ interaction, nor even see when it occurs. In reality, this interaction is necessarily insecure, for which we will show here how to securely model this ideal interaction by encapsulation. An adversary against $\mathcal{S}$ is a malicious environment who interacts with the system, for which we can tell if he succeeds in following the rules of a local game Γ. The notion of "rule of a local game" Γ should be understood as a given judge algorithm Γ who determines from the interactions between $\mathcal{E}$ and $\mathcal{S}$ only (and not other interactions e.g. between $\mathcal{E}$ and $\mathcal{O}$) if the attack succeeded, i.e if the adversary won the game against $\mathcal{S}$. (See Fig. 1.) Not every security notions for a system can be expressed locally. When it is possible to express security this way, it is quite a strong security property because we are saying that having the extra interaction with $\mathcal{O}$ does not give the adversary any additional advantage over just having interaction with $\mathcal{S}$ alone. Our goal is to show how to preserve this local security when the ideal $\mathcal{S} \leftrightarrow \mathcal{O}$ interaction is in reality instantiated with a secure protocol encapsulation.

Definition 5 (Local security).

We assume the complexity of the adversary includes that of the environment $\mathcal{E}$ and system $\mathcal{S}$.

Local security applies in the examples given in Section 1. For database queries, Γ checks if $\mathcal{E}$ guessed a bit $f(e)$ telling whether Company $\mathcal{A}$ is interested in some chosen event e. For EMV, Γ checks if the payment system accepted a fake credit card.

Here, our notion of local security ideally assumes that the $\mathcal{S} \leftrightarrow \mathcal{O}$ interaction is done through a perfectly secure channel. In reality, this channel is an insecure one going through $\mathcal{E}$, for which we would like to secure by means of encapsulation while preserving local security. See Fig. 2 for the resultant encapsulate-system.

Definition 6 (Encapsulate-system). $\mathcal{S} \leftrightarrow \mathcal{O}$ encapsulate-system $\mathcal{S}^{enc} \leftrightarrow \mathcal{O}$

- $\mathcal{S}^{enc}$ sk pk
- $\mathcal{S}^{enc}$ $\mathcal{S}$ $\mathcal{S}$ $\mathcal{S}^{enc}$
- $\mathcal{S}$ q $\mathcal{O}$ $QEM.Enc(pk, q) \rightarrow \langle K, A\rangle$ A" $\mathcal{I}_{\mathcal{S}}$ $\mathcal{I}_{\mathcal{S}}$ $\langle K, 0\rangle$
- A'" $\mathcal{S}^{enc}$ $\mathcal{I}_{\mathcal{O}}$ $QEM.Dec(sk, A') \rightarrow \langle K', q'\rangle$ $\mathcal{O}$ q' r' $REM.Enc(K', 1, r') \rightarrow B'$ B'" $\mathcal{I}_{\mathcal{O}}$ $\langle K', 1\rangle$ $QEM.Dec$ $\mathcal{S}^{enc}$
- B" $\mathcal{S}^{enc}$ $\mathcal{I}_{\mathcal{S}}$ $\langle K, t\rangle$ t $REM.Dec(K, t, B) \rightarrow r$ r $\mathcal{S}$ $REM.Dec$ $\mathcal{S}^{enc}$
- $\mathcal{S}$ r_t $\mathcal{I}_{\mathcal{S}}$ $\langle K, t\rangle$ t $REM.Enc(K, t, r) \rightarrow B$ B_t"
- B_t" $\mathcal{S}^{enc}$ $\mathcal{I}_{\mathcal{O}}$ $\langle K', t\rangle$ t $REM.Dec(K', t, B) \rightarrow r'$ r' $\mathcal{O}$ r'' t $REM.Enc(K', t, r'') \rightarrow B'$

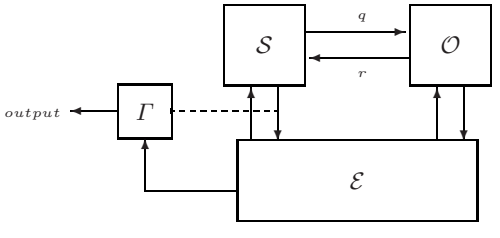


Fig. 1. An oracle-system $\mathcal{S}$ interacting with an oracle $\mathcal{O}$ and an environment $\mathcal{E}$

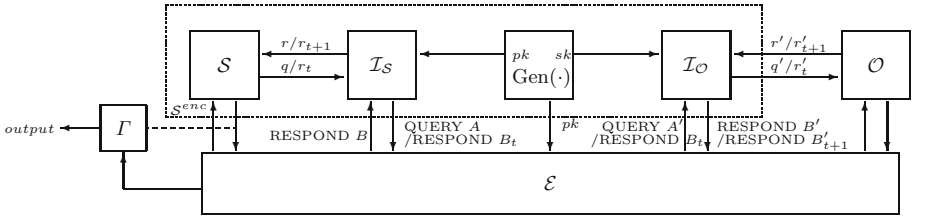


Fig. 2. An encapsulate-system including oracle-system $\mathcal{S}$, interfaces $\mathcal{I}$ and generator Gen

B'_{t+1}"

Any notion of Γ-security on $\mathcal{S} \leftrightarrow \mathcal{O}$ naturally extends to $\mathcal{S}^{enc} \leftrightarrow \mathcal{O}$ by making Γ ignore all communication through the $\mathcal{I}_\mathcal{S}$ and $\mathcal{I}_\mathcal{O}$ ports.

Definition 7 (Encapsulate-security). $\mathcal{S} \leftrightarrow \mathcal{O}$ (ε, c) Γ encapsulate-secure $\mathcal{S}^{enc} \leftrightarrow \mathcal{O}$ (ε, c) Γ

The notion of encapsulate-security captures the fact that local security (with respect to a local game Γ) is preserved when the oracle-system in the ideal case as per Fig. 1 is instantiated in reality with an encapsulate-system as per Fig. 2.

In reality, encapsulation necessary leaks side information:

- [Timing] the point in time at which $\mathcal{S}$ makes its queries and gets its responses, and
- [Length] the length of those query/response messages.

In fact, strong notions of encryption secrecy e.g. IND-CCA also assume that length information is necessarily leaked.

Thus, to prove that encapsulation preserves local security in the practical setting, we must assume that this kind of leakage is harmless. To this end, we define the notion of "toll-security", which captures the oracle-system in the setting similar to Fig. 1 but for which the above side information are inevitably leaked. This is the system that our encapsulate-system needs to emulate.

Definition 8 (Toll-system). $\mathcal{S} \leftrightarrow \mathcal{O}$ toll-system $\mathcal{S}^{toll} \leftrightarrow \mathcal{O}$ □

– $\mathcal{S}^{toll}$ $\mathcal{S}$ $\mathcal{S}$ $\mathcal{S}^{toll}$
– $\mathcal{S}$ q r_t $\mathcal{S} : |q|$" $\mathcal{S} : |r_t|$" $|q|$ q $|r_t|$ r_t $\mathcal{S}^{toll}$ $\mathcal{I}$ q r_t $\mathcal{O}$
– q r_t $\mathcal{O}$ q r_t r r_{t+1} $\mathcal{O} : |r|$" $\mathcal{O} : |r_{t+1}|$"

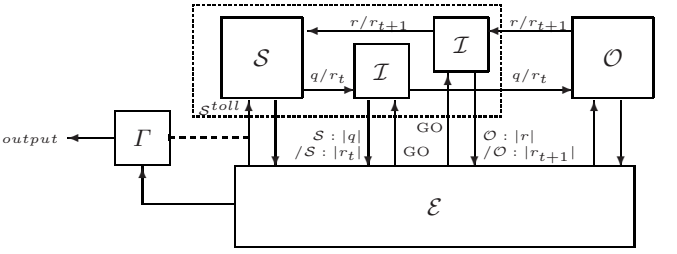


Fig. 3. A toll-system including the oracle-system $\mathcal{S}$ and the interface $\mathcal{I}$

[illegible] $|r|$ [illegible] r [illegible] $|r_{t+1}|$ [illegible] r_{t+1} [illegible] $\mathcal{S}^{toll}$ [illegible] $\mathcal{I}$ [illegible] r [illegible] r_{t+1} [illegible] – [illegible] $\mathcal{S}$" [illegible] $\mathcal{O}$" [illegible] $\mathcal{S}^{toll}$ [illegible] $\mathcal{I}$ [illegible] $\mathcal{O}$ [illegible] $\mathcal{S}$ [illegible]

Definition 9 (Toll-security). [illegible] $\mathcal{S} \leftrightarrow \mathcal{O}$ [illegible] (ε, c) Γ toll-secure [illegible] $\mathcal{S}^{toll} \leftrightarrow \mathcal{O}$ [illegible] (ε, c) Γ [illegible]

Note that many secure oracle-systems are trivially toll-insecure, i.e. they become insecure when side information are leaked. For instance, given a secure signature scheme in the random oracle model, we define a new signature scheme for which the signature algorithm on message m first computes some Boolean $\mathsf{bit}(m, sk)$ function depending on the private key sk and, if the bit is 0, does a query to the random oracle with an empty input, otherwise does nothing, then simulates the original signature scheme. Obviously, this is equivalent to the original scheme in the random oracle model, but leaks $\mathsf{bit}(m, sk)$ to the environment in the toll-system variant. This helps the adversary to reconstruct sk after a few queries when bit is well chosen.

To prove that encapsulation preserves local security with [illegible] oracles, extra treatment is needed: the oracle must be immune to replay attacks.

Definition 10 (Immunity to replay attacks). [illegible] $\mathcal{O}$ [illegible] $\mathcal{O}$ [illegible]

This requirement is necessary otherwise a toll-secure system may not be encapsulate-secure if the oracle is stateful and allows repeated queries. To motivate this assumption, we consider that we have a stateful oracle who simulates a perfect random oracle except when a query is repeated, in which case the oracle answers with a constant value (e.g. 0) to the repeated query. An adversary can actually repeat any query from the system and make the system accept answers to the second query, which provokes responses (in this case a constant value) to be perfectly predictable. In a case where queries by the Enquirer are unknown to $\mathcal{E}$, we can have loss of local security when encapsulating the protocol. This can be avoided with an oracle $\mathcal{O}$ that is immune to replay attacks as per Definition 10. For instance, a stateless oracle or an oracle who repeats the same response to repeating queries without modifying its state would satisfy this condition.

Finally, $\mathcal{S}$ must not make concurrent queries to $\mathcal{O}$. Concretely, $\mathcal{S}$ never sends a new message to $\mathcal{O}$ if he is still waiting for a response. Otherwise, $\mathcal{E}$ could get advantage in modifying the order in which they are sent to a stateful oracle.

We now state our main result about secure asymmetric protocol encapsulation. The proof of Theorem 2 is in Appendix B, and shows that if for any Q/REM secure in the sense defined in section 2, then the encapsulate-system is computationally indistinguishable from the ideal toll-system.

Theorem 2. $\mathcal{S} \leftrightarrow \mathcal{O}$ $\mathcal{O}$ $\mathcal{S}$ Q $\mathcal{O}$ Γ μ $\mathcal{S} \leftrightarrow \mathcal{O}$ (ε, c) Γ $(\varepsilon_{sem}, \varepsilon_{auth}, c)$ $\mathcal{S} \leftrightarrow \mathcal{O}$ $(Q(\varepsilon_{sem} + \varepsilon_{auth}) + \varepsilon, c - \mu)$ Γ

4 Conclusion

We have put forth a generic notion of security for asymmetric (query-response) protocols, that is set up to permit leakages of timing and message length to closely model types of information leakages in practice. In doing so, we proposed a generic construction of a provably secure query-response encapsulation scheme that can be used to this respect, and our results apply to any arbitrary-round asymmetric protocol.

In our proof of encapsulation, we have also modeled the time when a query is made because otherwise it may not properly capture the leakage of side information that would break the system. We prove the security of encapsulation with the assumption

1. that our underlying primitives are secure,
2. that the query-response protocol remains secure when the existence and time of the query and response, and their length leak,
3. that the server is a stateless machine or a stateful one that is immune to replay attacks.

We further demonstrate that the last two conditions are necessary.

Acknowledgement

The second author thanks Tom Shrimpton for discussions on [24] and on AE in general. We thank the anonymous referees for comments that contributed to the clarity of this paper. Une partie de ce travail a été réalisée pendant les longs soirs d' été à St. Sulpice.

References

1. Abe, M., Gennaro, R., Kurosawa, K., Shoup, V.: Tag-KEM/DEM: A New Framework for Hybrid Encryption and a New Analysis of Kurosawa-Desmedt KEM. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 128–146. Springer, Heidelberg (2005)

2. Bellare, M., Namprempre, C.: Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000)
3. Bellare, M., Rogaway, P.: Random Oracles are Practical: a Paradigm for Designing Efficient Protocols. In: Proc. ACM-CCS 1993, pp. 62–73 (1993)
4. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In: Proc. IEEE FOCS 2001, pp. 136–145 (2001), Full version available at IACR ePrint Archive http://eprint.iacr.org/2000/067
5. Canetti, R., Krawczyk, H.: Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001)
6. Canetti, R., Krawczyk, H.: Universally Composable Notions of Key Exchange and Secure Channels. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 337–351. Springer, Heidelberg (2002)
7. Cramer, R., Shoup, V.: Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack. SIAM Journal of Computing 33(1), 167–226 (2004)
8. Dierks, T., Allen, C.: The TLS Protocol: Version 1 (1999)
9. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol: Version 1.1, RFC 4346 (April 2006)
10. Diffie, W., Hellman, M.E.: New Directions in Cryptography. IEEE Transactions on Information Theory 22(6), 644–654 (1976)
11. EMVCo, LLC, EMV 4.1 Specifications (June 2004), Available online at http://www.emvco.com/specifications.asp?show=3
12. Evdokimov, S., Fischmann, M., Gunther, O.: Provable Security for Outsourcing Database Operations. In: Proc. IEEE ICDE 2006, pp. 117 (2006)
13. Freier, A.O., Karlton, P., Kocher, P.C.: The SSL Protocol: Version 3.0. Internet Draft (March 1996)
14. Ge, T., Zdonik, S.: Fast, Secure Encryption for Indexing in a Column-Oriented DBMS. In: Proc. IEEE ICDE 2007, pp. 676–685 (2007)
15. Goldwasser, S., Micali, S., Rivest, R.L.: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. SIAM Journal of Computing 17(2), 281–308 (1988)
16. Hofheinz, D., Kiltz, E.: Secure Hybrid Encryption from Weakened Key Encapsulation. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 553–571. Springer, Heidelberg (2007)
17. Krauß, C., Stumpf, F., Eckert, C.: Detecting Node Compromise in Hybrid Wireless Sensor Networks using Attestation Techniques. In: Proc. ESAS 2007, LNCS 4572, pp. 203–217 (2007)
18. Krawczyk, H.: The Order of Encryption and Authentication for Protecting Communications (or: How Secure is SSL?). In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 310–331. Springer, Heidelberg (2001)
19. Leung, A., Mitchell, C.J.: Ninja: Non Identity Based, Privacy Preserving Authentication for Ubiquitous Environments. In: Krumm, J., Abowd, G.D., Seneviratne, A., Strang, T. (eds.) UbiComp 2007. LNCS, vol. 4717, pp. 73–90. Springer, Heidelberg (2007)
20. Nagao, W., Manabe, Y., Okamoto, T.: A Universally Composable Secure Channel Based on the KEM-DEM Framework. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 426–444. Springer, Heidelberg (2005)

21. Nagao, W., Manabe, Y., Okamoto, T.: A Universally Composable Secure Channel Based on the KEM-DEM Framework. IEICE Trans. Fund. Electronics, Communications & Computer Sciences E89-A(1), 28–38 (2006)
22. Pashalidis, A., Mitchell, C.J.: Single Sign-On using Trusted Platforms. In: Boyd, C., Mao, W. (eds.) ISC 2003. LNCS, vol. 2851, pp. 54–68. Springer, Heidelberg (2003)
23. Rackoff, C., Simon, D.: Non-interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 433–444. Springer, Heidelberg (1992)
24. Rogaway, P., Shrimpton, T.: A Provable-Security Treatment of the Key-Wrap Problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 373–390. Springer, Heidelberg (2006)
25. Ylonen, T., Lonvick, C.: The Secure Shell (SSH) Transport Layer Protocol, RFC 253 (January 2006)

A Proof of Theorem 1

To obtain the proof for Theorem 1, we need to recall the definitions of KEM, AE and their corresponding security notions.

KEM. A key encapsulation mechanism (KEM) [7] is given by the triple of algorithms $KEM.Key(1^\lambda), KEM.Enc(pk)$ and $KEM.Dec(sk, C_0)$, where:

1. $\langle pk, sk\rangle \leftarrow KEM.Key(1^\lambda)$: this is a PPT algorithm that takes a security parameter 1^λ and returns a pair $\langle pk, sk\rangle$ of matching public and private keys.
2. $\langle K, C_0\rangle \leftarrow KEM.Enc(pk)$: this is a PPT algorithm that takes as input a public key pk and outputs a key/ciphertext pair $\langle K, C_0\rangle$.
3. $K \leftarrow KEM.Dec(sk, C_0)$: this is a deterministic polynomial time algorithm that takes as input the private key sk and ciphertext C_0, and outputs key K or a special symbol $\perp$ implying the ciphertext was invalid.

It is required that [illegible] be fulfilled, i.e. for all (pk, sk) output by $KEM.Key(\cdot)$, and for all C_0 output by $KEM.Enc(pk)$, then $KEM.Dec(sk, C_0) = K$.

The security notion of IND-CCA-KEM is as follows:

1. $KEM.Key(\cdot)$ is run to generate the public pk and private key sk for the protocol, and pk is given to the adversary $\mathcal{A}$.
2. $\mathcal{A}$ generates some ciphertext queries and sends them to the challenger. He calls the decryption oracle $KEM.Dec(\cdot)$ who decrypts them and the results are returned to $\mathcal{A}$.
3. The challenger runs $KEM.Enc(\cdot)$ to generate $\langle K^*, C_0^*\rangle$. He generates a random string $\tilde{K}$ where $|\tilde{K}| = |K^*|$. He chooses $b \in \{0, 1\}$. If $b = 0$, he outputs $\langle K^*, C_0^*\rangle$, otherwise he outputs $\langle \tilde{K}, C_0^*\rangle$.
4. $\mathcal{A}$ generates further ciphertext queries, but cannot query the challenge ciphertext C_0^*.
5. $\mathcal{A}$ outputs a guess $\tilde{b} \in \{0, 1\}$.

Let $\Pi_{\mathtt{KEM}}$ be a KEM. The advantage of $\Pi_{\mathtt{KEM}}$ for adversary $\mathcal{A}$, is defined as:

$$Adv^{\mathtt{IND-CCA}}_{\mathcal{A},\Pi_{\mathtt{KEM}}}(\lambda) = |\Pr[\tilde{b} = b] - 1/2|.$$

$\Pi_{\mathtt{KEM}}$ is secure if $Adv^{\mathtt{IND-CCA}}_{\mathcal{A},\Pi_{\mathtt{KEM}}}(\lambda)$ is negligible for any PPT adversary $\mathcal{A}$.

AE. A non-deterministic Authenticated-Encryption (AE) scheme is composed of a key generator and symmetric encryption which use an extra input called the "header" h. [24] follows a series of papers on authenticated-encryption, notably [2] and proposes an all-in-one definition which is adapted below. The decryption algorithm may return $\perp$ if the ciphertext is not consistent with the header. An AE is ε-secure if for any distinguisher with access to two oracles: the left and the right oracle, the advantage for distinguishing the two sets of oracles below is at most ε.

- The left oracle is an encryption oracle. The right oracle is a decryption oracle. Both oracles are first set up with a random key prior to any query.
- The left oracle is a random generator. The right oracle returns $\perp$ except for (y, h) queries such that y was previously output by the left oracle with a (x, h) query: in this case, the right oracle returns x.

Proof of Theorem 1

We first reduce the IND-CCA-QREM game, i.e. we construct an adversary $\mathcal{A}_{\mathtt{KEM}}$ against IND-CCA-KEM using an adversary $\mathcal{A}$ that breaks IND-CCA-QREM. When $\mathcal{A}_{\mathtt{KEM}}$ receives pk, this is forwarded to $\mathcal{A}$. Queries $A = C_0||C$ for $OQEM.Dec(A)$ from $\mathcal{A}$ to $\mathcal{A}_{\mathtt{KEM}}$ are answered by forwarding C_0 to $KEM.Dec(\cdot)$ oracle. The returned K is used by $\mathcal{A}_{\mathtt{KEM}}$ to perform $AE.Dec(K, 0, C)$ and result q is returned to $\mathcal{A}$ as $\langle K, q\rangle$.

During the challenge stage, $\mathcal{A}$ chooses q^* and sends to $\mathcal{A}_{\mathtt{KEM}}$. Meanwhile, $KEM.Enc(pk)$ is run by the KEM challenger in challenge phase of the IND-CCA-KEM game to generate the key-encapsulation pair $\langle K^\dagger, C_0^*\rangle$. Then depending on a flipped bit b, either the computed $K^\dagger$ or a random one $\tilde{K}$ is returned as K^* with C_0^* to $\mathcal{A}_{\mathtt{KEM}}$ as the challenge $K^*||C_0^*$. $\mathcal{A}_{\mathtt{KEM}}$ sets $q_0 = q^*$ and selects a random q_1 of same length. Then $\mathcal{A}_{\mathtt{KEM}}$ flips a bit b' and performs $C^* = AE.Enc(K^*, 0, q_{b'})$. $A^* = C_0^*||C^*$ is returned to $\mathcal{A}$ as a challenge.

After this, queries for $OQEM.Dec(A)$ by $\mathcal{A}$ are answered similar to before, except for $A = C_0||C$ with $C_0 = C_0^*$ (and $C \neq C^*$). In such a case, $\mathcal{A}_{\mathtt{KEM}}$ runs $AE.Dec(K^*, 0, C)$ directly since it knows the decapsulated K^*. Queries for $OIREM.Enc(i, r)$ can be easily answered by $\mathcal{A}_{\mathtt{KEM}}$ since it knows K^* and b', i.e. it sets $r_0 = r$ and selects a random r_1. Then it returns $B = AE.Enc(K^*, i, r_{b'})$ to $\mathcal{A}$. Finally, when $\mathcal{A}$ outputs a bit $\tilde{b}$, $\mathcal{A}_{\mathtt{KEM}}$ outputs $\tilde{b} \oplus b'$.

Let IND-CCA-QREM* be the modified game which behaves just like the original IND-CCA-QEM game except that instead of the secret K^* used in $AE.Enc(\cdot)$ and $AE.Dec(\cdot)$, a completely independent and random key $\tilde{K}$ is used. Note that for $b = 0$, $\mathcal{A}$ plays the IND-CCA-QREM game with $\mathcal{A}_{\mathtt{KEM}}$. When

$b = 1$, $\mathcal{A}$ is essentially playing the IND-CCQ-QREM* game with $\mathcal{A}_{\texttt{KEM}}$. Let X and X^* be events that $\mathcal{A}$ wins the IND-CCA-QREM and IND-CCA-QREM* games respectively. We obtain

$$\texttt{Pr}[\mathcal{A}_{\texttt{KEM}}\ \texttt{wins}] - \frac{1}{2} = \texttt{Pr}[\tilde{b} \oplus b' = b] - \frac{1}{2} = \frac{1}{2}(\texttt{Pr}[\tilde{b} = b'|b = 1] - \texttt{Pr}[\tilde{b} = b'|b = 0])$$
$$= \frac{1}{2}(\texttt{Pr}[X^*] - \texttt{Pr}[X]).$$

Since $|\texttt{Pr}[\mathcal{A}_{\texttt{KEM}}\ \texttt{wins}] - \frac{1}{2}| \leq Adv^{\texttt{IND-CCA}}_{\pi_{\texttt{KEM}}}(\lambda)$, thus $|\texttt{Pr}[X^*] - \texttt{Pr}[X]| \leq 2Adv^{\texttt{IND-CCA}}_{\pi_{\texttt{KEM}}}(\lambda)$.

We now reduce the IND-CCA-QREM* game to the AE game, i.e we use an adversary $\mathcal{A}$ against IND-CCA-QREM* to construct an adversary $\mathcal{A}_{\texttt{AE}}$ against AE security. Note that KEM is no longer relevant in the IND-CCA-QREM* game because the key $\tilde{K}$ used to key AE is random instead of being generated by KEM.

$\mathcal{A}_{\texttt{AE}}$ runs $KEM.Key(\cdot)$ obtaining $\langle pk, sk\rangle$. pk is given to $\mathcal{A}$. Queries $A = C_0||C$ for $OQEM.Dec(A)$ from $\mathcal{A}$ to $\mathcal{A}_{\texttt{AE}}$ are trivially answered by $\mathcal{A}_{\texttt{AE}}$ since it has sk to run $QEM.Dec(sk, C_0)$ obtaining K which is then used in $AE.Dec(K, 0, C)$. The result q is returned to $\mathcal{A}$ as $\langle K, q\rangle$ for $q \neq \perp$.

During the challenge stage, $\mathcal{A}$ chooses q^* and sends to $\mathcal{A}_{\texttt{AE}}$. $\mathcal{A}_{\texttt{AE}}$ sets $q_0 = q^*$ and selects a random q_1. Then flipping a bit b', it sends $\langle 0, q_{b'}\rangle$ to the AE challenger. The challenger selects a random key $\tilde{K}$ and flips a bit b. If $b = 0$, it runs $AE.Enc(\tilde{K}, 0, q_{b'})$ and returns the result as C^*. Else, a random C^* is returned. $\mathcal{A}_{\texttt{AE}}$ runs $KEM.Enc(pk)$ to obtain $\langle K^*, C_0^*\rangle$, discards K^* and returns $A^* = C_0^*||C^*$ to $\mathcal{A}$ as the challenge.

Further queries for $OQEM.Dec(A = C_0||C)$ by $\mathcal{A}$ are answered similarly as before, except for $C_0 = C_0^*$, where $\langle 0, C\rangle$ is sent to $AE.Dec(\cdot)$. (Note that C must differ from C^*). Queries for $OIREM.Enc(i, r)$ are answered by $\mathcal{A}_{\texttt{AE}}$ who sets $r_0 = r$ and selects a random r_1 and gives $\langle i, r_{b'}\rangle$ to the AE challenger, where b' is that chosen by $\mathcal{A}_{\texttt{AE}}$ during the challenge phase. AE challenger returns $B = AE.Enc(\tilde{K}, i, r_{b'})$ if b chosen by it during the previous challenge phase is 0, else it returns a random string. When $\mathcal{A}$ outputs a bit $\tilde{b}$, $\mathcal{A}_{\texttt{AE}}$ outputs $\tilde{b} \oplus b'$.

For $b = 0$, $\mathcal{A}$ plays the IND-CCA-QREM* game with $\mathcal{A}_{\texttt{AE}}$. Further, let X^{**} be the event that $\mathcal{A}$ wins conditioned to $b = 1$. We obtain

$$\texttt{Pr}[\mathcal{A}_{\texttt{AE}}\ \texttt{wins}] - \frac{1}{2} = \texttt{Pr}[\tilde{b} \oplus b' = b] - \frac{1}{2} = \frac{1}{2}(\texttt{Pr}[\tilde{b} = b'|b = 1] - \texttt{Pr}[\tilde{b} = b'|b = 0])$$
$$= \frac{1}{2}(\texttt{Pr}[X^{**}] - \texttt{Pr}[X^*]).$$

Since $|\texttt{Pr}[\mathcal{A}_{\texttt{AE}}\ \texttt{wins}] - \frac{1}{2}| \leq Adv^{\texttt{AE}}_{\pi_{\texttt{AE}}}(\lambda)$, thus $|\texttt{Pr}[X^{**}] - \texttt{Pr}[X^*]| \leq 2Adv^{\texttt{AE}}_{\pi_{\texttt{AE}}}(\lambda)$. Also note that for $b = 1$ no information on b' leaks, so $\texttt{Pr}[X^{**}] = \frac{1}{2}$. Rearranging, we have

$$\texttt{Pr}[X] - \frac{1}{2} \leq 2Adv^{\texttt{IND-CCA}}_{\pi_{\texttt{KEM}}}(\lambda) + 2Adv^{\texttt{AE}}_{\pi_{\texttt{AE}}}(\lambda)$$
$$Adv^{\texttt{IND-CCA-QREM}}_{\pi_{\texttt{QREM}}}(\lambda) \leq 2Adv^{\texttt{IND-CCA}}_{\pi_{\texttt{KEM}}}(\lambda) + 2Adv^{\texttt{AE}}_{\pi_{\texttt{AE}}}(\lambda).$$

We consider reducing the AUTH-CCA-QREM game. This is similar to the previous reduction for IND-CCA-QREM, so we will only highlight

the differences. The simulation proceeds similarly, except during the challenge phase. In more detail, the IND-CCA-KEM challenger runs $KEM.Enc(pk)$ to obtain $\langle K^\dagger, C_0^* \rangle$. It then flips a bit b and if $b = 0$ it returns $\langle K^* = K^\dagger, C_0^* \rangle$ to $\mathcal{A}_{\mathtt{KEM}}$. Else it returns a random key $\tilde{K}$ as K^*. When $\mathcal{A}_{\mathtt{KEM}}$ receives q^* from $\mathcal{A}$ it computes $C^* = AE.Enc(K^*, 0, q^*)$ and returns $A^* = C_0^* || C^*$ as the challenge.

Queries for $OREM.Enc(i, r^*)$ can be easily answered by $\mathcal{A}_{\mathtt{KEM}}$ since it knows K^*, i.e. it returns $B^* = AE.Enc(K^*, i, r^*)$; additionally each of these B^* are recorded.

Finally, when $\mathcal{A}$ outputs a $\langle i, B \rangle$, $\mathcal{A}_{\mathtt{KEM}}$ runs $AE.Dec(K^*, i, B)$ and checks the resulting r. If no $\perp$ is obtained (meaning B is valid) and if B does not match with a previously recorded B^*, then it is clear that $\mathcal{A}$ wins, and thus $\mathcal{A}_{\mathtt{KEM}}$ outputs a guess $\tilde{b} = 0$. Else it outputs $\tilde{b} = 1$.

When $b = 0$, $\mathcal{A}$ plays the AUTH-CCA-QREM game with $\mathcal{A}_{\mathtt{KEM}}$. Let AUTH-CCA-QREM* be the modified game corresponding to $b = 1$, and X and X^* be the corresponding events that $\mathcal{A}$ wins in these AUTH-CCA-QREM and AUTH-CCA-QREM* games respectively:

$$\begin{aligned}\mathtt{Pr}[\mathcal{A}_{\mathtt{KEM}}\ \mathtt{wins}] - \frac{1}{2} = \mathtt{Pr}[\tilde{b} = b] - \frac{1}{2} &= \frac{1}{2}(\mathtt{Pr}[\tilde{b} = 0 | b = 1] - \mathtt{Pr}[\tilde{b} = 0 | b = 0]) \\ &= \frac{1}{2}(\mathtt{Pr}[\mathcal{A}\ \mathtt{wins} | b = 1] - \mathtt{Pr}[\mathcal{A}\ \mathtt{wins} | b = 0]) = \frac{1}{2}(\mathtt{Pr}[X^*] - \mathtt{Pr}[X]).\end{aligned}$$

Since $|\mathtt{Pr}[\mathcal{A}_{\mathtt{KEM}}\ \mathtt{wins}] - \frac{1}{2}| \leq Adv^{\mathtt{IND-CCA}}_{\pi_{\mathtt{KEM}}}(\lambda)$,thus $|\mathtt{Pr}[X^*] - \mathtt{Pr}[X]| \leq 2Adv^{\mathtt{IND-CCA}}_{\pi_{\mathtt{KEM}}}(\lambda)$.

Consider now reducing AUTH-CCA-QREM* to the AE game. The steps are similar to the case for IND-CCA-QREM* so we only mention the differences.

During the challenge stage, when $\mathcal{A}_{\mathtt{AE}}$ receives q^* from $\mathcal{A}$ it forwards $\langle 0, q^* \rangle$ to the AE challenger, who selects a random key $\tilde{K}$ and flips a bit b. If $b = 0$, it runs $AE.Enc(\tilde{K}, 0, q^*)$ and returns the result as C^*. Else, a random C^* is returned. $\mathcal{A}_{\mathtt{AE}}$ runs $KEM.Enc(pk)$ to obtain $\langle K^*, C_0^* \rangle$, discards K^* and returns $A^* = C_0^* || C^*$ to $\mathcal{A}$ as the challenge.

Finally, when $\mathcal{A}$ outputs a B, $\mathcal{A}_{\mathtt{AE}}$ checks that B does not equal the output of any previous $OREM.Enc(i, r)$ query, and passes r to the AE challenger who depending on the bit b selected during the challenge phase either returns the result $r = AE.Dec(\tilde{K}, i, B)$ or $\perp$. If no $\perp$ is obtained meaning $\mathcal{A}$ wins, then $\mathcal{A}_{\mathtt{AE}}$ outputs a guess $\tilde{b} = 0$. Else it outputs $\tilde{b} = 1$.

Let X^{**} be the event that $\mathcal{A}$ wins the game conditioned to $b = 1$. We obtain

$$\begin{aligned}\mathtt{Pr}[\mathcal{A}_{\mathtt{AE}}\ \mathtt{wins}] - \frac{1}{2} = \mathtt{Pr}[\tilde{b} = b] - \frac{1}{2} &= \frac{1}{2}(\mathtt{Pr}[\tilde{b} = 0 | b = 1] - \mathtt{Pr}[\tilde{b} = 0 | b = 0]) \\ &= \frac{1}{2}(\mathtt{Pr}[\mathcal{A}\ \mathtt{wins} | b = 1] - \mathtt{Pr}[\mathcal{A}\ \mathtt{wins} | b = 0]) = \frac{1}{2}(\mathtt{Pr}[X^{**}] - \mathtt{Pr}[X^*]).\end{aligned}$$

Since $|\mathtt{Pr}[\mathcal{A}_{\mathtt{AE}}\ \mathtt{wins}] - \frac{1}{2}| \leq Adv^{\mathtt{AE}}_{\pi_{\mathtt{AE}}}(\lambda)$, and $\mathtt{Pr}[X^{**}] = 0$; thus $|\mathtt{Pr}[X^{**}] - \mathtt{Pr}[X^*]| \leq 2Adv^{\mathtt{AE}}_{\pi_{\mathtt{AE}}}(\lambda)$.

Rearranging, we have

$$\begin{aligned}\mathtt{Pr}[X] &\leq 2Adv^{\mathtt{IND-CCA}}_{\pi_{\mathtt{KEM}}}(\lambda) + 2Adv^{\mathtt{AE}}_{\pi_{\mathtt{AE}}}(\lambda) \\ Adv^{\mathtt{AUTH-CCA-QREM}}_{\pi_{\mathtt{QREM}}}(\lambda) &\leq 2Adv^{\mathtt{IND-CCA}}_{\pi_{\mathtt{KEM}}}(\lambda) + 2Adv^{\mathtt{AE}}_{\pi_{\mathtt{AE}}}(\lambda).\end{aligned}$$

□

B Proof of Theorem 2

Let Q be the number of protocol sessions initiated by $\mathcal{S}$ in the encapsulate-system; and $l+1$ be the number of messages within each protocol session. We take an adversary $\mathcal{E}$ against the encapsulate-system $\mathcal{S}^{enc} \leftrightarrow \mathcal{O}$ of complexity $c-\mu$ where $\mu = \mathtt{max}(\mu_1, \mu_2, \mu_3)$ with μ_1, μ_2, μ_3 to be later defined. We construct a sequence of hybrid systems $\mathcal{S}_i$ and $\mathcal{S}'_i$ for $i = 1, \ldots, Q$ with $\mathcal{S}_0 = \mathcal{S}^{enc}$, and $\mathcal{S}'_0 = \mathcal{S}_Q$ interacting with the adversary $\mathcal{E}$ and oracle $\mathcal{O}$. $\mathcal{S}'_Q$ will almost be the toll-system $\mathcal{S}^{toll}$.

$\mathcal{S}_i$ and $\mathcal{S}'_i$ keep record of all $\langle A', B'_1, B'_2, B'_3, \ldots, B'_l, q', r'_1, r'_2, r'_3, \ldots, r'_l \rangle$. System $\mathcal{S}'_i$ further keeps additional record of some $\langle K, A, q \rangle$ triplets.

We define the oracle-system $\mathcal{S}_i \leftrightarrow \mathcal{O}$ which slightly differs from $\mathcal{S}_{i-1} \leftrightarrow \mathcal{O}$. Let A be the ith encapsulated query from $\mathcal{S}$, i.e. the value of the ith "QUERY A" message. It encapsulates the ith query q. Let B_t (for $t = 1, 2, \ldots, l$) be the encapsulations of the corresponding responses r_t to this ith query q. If the environment submits a "RESPOND B_t" message ($t \in \{1, 2, \ldots, l\}$) to $\mathcal{S}_i$, the system checks if some $\langle A', B'_1, B'_2, B'_3, \ldots, q', r'_1, r'_2, r'_3, \ldots \rangle$ with $A' = A$ and $B'_s = B_s$ (for $s \leq t$) exists in its record. r'_t is returned if this is so. In other cases, S_i aborts.

$\mathcal{S}_{i-1}$ and $\mathcal{S}_i$ only differ in the treatment on the "RESPOND B_t" (for $t = 1, 2, \ldots, l$) messages initiated by the ith "QUERY A" message. They can be simulated without the initial secret key sk provided that we get pk and we can use an $OQEM.Dec(\cdot)$ oracle to treat "QUERY A'" with $A' \neq A$ messages. We define an adversary $\mathcal{A}_i$ against the AUTH-CCA-QREM game this way (See Fig.4): The adversary first receives the public key pk and simulates the interaction with adversary $\mathcal{E}$ and $\mathcal{O}$ until $\mathcal{S}$ issues its ith query q. Then, $\mathcal{A}_i$ submits q^* to the challenger and receives an encapsulation A^*. The adversary can call the $OQEM.Dec(\cdot)$ oracle except for input A^* to treat the "QUERY A'" message with $A' \neq A^*$. To treat the "QUERY A'" and subsequent "RESPOND B'_t" messages with $A' = A^*$ and $B'_s = B_s$ (for $s \leq t$), recall that $\mathcal{A}_i$ knows q^* so it simply queries $\mathcal{O}$ with q^* as many times as necessary to get r^*_{t+1} and calls $OREM.Enc(t+1, r^*_{t+1})$ to get B^*_{t+1}. Note that it is important to query $\mathcal{O}$ every time since the oracle may not be deterministic, e.g. stateful oracles. The final B_l in the "RESPOND B_l" message to the system is the final B_l in the AUTH-CCA-QREM game, so the simulation is perfect. In the event E that B_l equals one of the obtained B^*_t from $OREM.Enc$, then $\mathcal{A}_i$ fails to win the AUTH-CCA-QREM game; in which case the behavior of $\mathcal{E}(\mathcal{S}_{i-1}, \mathcal{O})$ and $\mathcal{E}(\mathcal{S}_i, \mathcal{O})$ are identical, and $\mathcal{A}_i$ perfectly simulates $\mathcal{S}_{i-1}$ or $\mathcal{S}_i$ to $\mathcal{E}$. Hence $\Pr[\mathcal{E}(\mathcal{S}_{i-1}, \mathcal{O})\ \mathtt{wins} | E] = \Pr[\mathcal{E}(\mathcal{S}_i, \mathcal{O})\ \mathtt{wins} | E]$. Otherwise, $\mathcal{A}_i$ wins. If Q/REM is AUTH-CCA secure, the advantage for $\mathcal{A}_i$ is negligible, meaning that $\Pr[\mathcal{A}_i\ \mathtt{wins}] \leq \varepsilon_{auth}$. We deduce $|\Pr[\mathcal{E}(\mathcal{S}_{i-1}, \mathcal{O})\ \mathtt{wins}] - \Pr[\mathcal{E}(\mathcal{S}_i, \mathcal{O})\ \mathtt{wins}]| \leq \varepsilon_{auth}$ since the complexity of $\mathcal{A}_i$ is $c - \mu + \mu_1$ for some small overhead cost μ_1.

We define the oracle-system $\mathcal{S}'_i \leftrightarrow \mathcal{O}$ which slightly differs from $\mathcal{S}'_{i-1} \leftrightarrow \mathcal{O}$. Now, instead of encapsulating q to produce $\langle K, A \rangle$ in the ith query from $\mathcal{S}$, $\mathcal{S}'_i$ encapsulates a random query $\tilde{q}$ of the same length and keeps record of $\langle K, A, q \rangle$. Similarly, for any "QUERY A'" message from the environment with $A' = A$

$\mathcal{E}$		$\mathcal{A}_i$ simulates S_i		QREM Challenger
	$\xleftarrow{pk}$		$\xleftarrow{pk}$	$QEM.Key(1^\lambda) = \langle pk, sk\rangle$.
	$\xleftarrow{\text{QUERY } A}$	Case: Receive jth query q (for $j \neq i$) from $\mathcal{S}$. $QEM.Enc(pk, q) = \langle K, A\rangle$. Keep $\langle j, K\rangle$ in memory.		
	$\xrightarrow{\text{QUERY } A'}$ $\xleftarrow{\text{RESPOND } B'_1}$	If $q' = \perp$ then halt. Else $\mathcal{O}(q') = r'_1$. $REM.Enc(K', 1, r'_1) = B'_1$. Record $\langle K', A', B'_1, q', r'_1\rangle$.	$\xrightarrow{OQEM.Dec(A')}$ $\xleftarrow{\langle K', q'\rangle}$.	$QEM.Dec(sk, A') = \langle K', q'\rangle$.
	$\xrightarrow{\text{RESPOND } B_t (t=1,\ldots)}$ $\xleftarrow{\text{RESPOND } B'_{t+1}}$	For $(j < i)$, if $\langle K', A', B'_1, B'_2, \ldots, q', r'_1, r'_2, \ldots,\rangle$ exists s.t. $A' = A$ and $B'_s = B_s (s \leq t)$, answer r'_t to $\mathcal{S}/\mathcal{O}$ else halt. For $(j > i)$, $REM.Dec(K, t, B_t) = r_t$. If $r_t = \perp$ then halt. Else answer r_t to $\mathcal{S}/\mathcal{O}$ and get r'_{t+1}. $REM.Enc(K', t+1, r'_{t+1}) = B'_{t+1}$. Add $\langle B'_{t+1}, r'_{t+1}\rangle$ to the record.		
	$\xleftarrow{\text{QUERY } A^*}$	Case: Receive ith query q^* from S. Keep $\langle q^*, A^*\rangle$ in memory.	$\xrightarrow{q^*}$ $\xleftarrow{A^*}$	$QEM.Enc(pk, q^*) = \langle K^*, A^*\rangle$.
	$\xrightarrow{\text{QUERY } A'}$ $\xleftarrow{\text{RESPOND } B'_1}$	If $A' \neq A^*$, then proceed as for normal QUERY A' :	:	:
	$\xrightarrow{\text{QUERY } A'}$ $\xleftarrow{\text{RESPOND } B^*_1}$	If $A' = A^*$, then $\mathcal{O}(q^*) = r^*_1$. Record $\langle A^*, B^*_1, q^*, r^*_1\rangle$.	$\xrightarrow{OREM.Enc(1, r^*_1)}$ $\xleftarrow{B^*_1}$	$REM.Enc(K^*, 1, r^*_1) = B^*_1$.
	$\xrightarrow{\text{RESPOND } B'_t (t=1,\ldots)}$ $\xleftarrow{\text{RESPOND } B^*_{t+1}}$	If $\langle A', B'_1, B'_2, \ldots, q', r'_1, r'_2, \ldots\rangle$ exists s.t. $A' = A^*$ and $B'_t = B^*_t$, then answer r'_t to $\mathcal{S}/\mathcal{O}$ else halt. Get r^*_{t+1}. Add $\langle B^*_{t+1}, r^*_{t+1}\rangle$ to the record.	$\xrightarrow{OREM.Enc(t+1, r^*_{t+1})}$ $\xleftarrow{B^*_{t+1}}$	$REM.Enc(K^*, t+1, r^*_{t+1}) = B^*_{t+1}$.
	$\xrightarrow{\text{RESPOND } B_l}$	If $\langle A', B'_1, B'_2, \ldots, B'_l, q', r'_1, r'_2, \ldots, r'_l\rangle$ exists s.t. $A' = A^*$ and $B'_l = B_l$, answer r'_l to $\mathcal{S}$. Else, output B_l to QREM challenger.	$\xrightarrow{B_l}$	

Fig. 4. Reduction from Γ-encapsulate-secure* to AUTH-CCA-QREM

for some $\langle K, A, q\rangle$ record, $\mathcal{S}'_i$ gets q from the record, queries $\mathcal{O}$ with $q' = q$, and obtains the response $r'_1 = r_1$. Nevertheless, instead of encapsulating r_1, the system directly encapsulates a random response $\tilde{r}_1$ to produce B'_1. The record $\langle A', B'_1, q, r_1\rangle$ is inserted. This querying of the oracle $\mathcal{O}$ is just to update the internal state of $\mathcal{O}$ to handle the case of stateful oracles. The same occurs when producing subsequent response messages r_{t+1} $(t \in \{1, 2, \ldots, l\})$ triggered by a preceeding message r_t i.e. a random $\tilde{r}_{t+1}$ is encapsulated instead of querying $\mathcal{O}(r_t)$ and $\langle B'_{t+1}, r_{t+1}\rangle$ are added to the record.

$\mathcal{S}'_{i-1}$ and $\mathcal{S}'_i$ only differ in the treatment on the ith "QUERY A" message and related "QUERY A'" messages with $A' = A$; and corresponding "RESPOND B_t" and "RESPOND B'_t" messages. They can be simulated without the initial secret key sk provided that we get pk and we can use an $OQEM.Dec(\cdot)$ oracle to treat "QUERY A'" with $A' \neq A$ messages; and $OREM.Dec(\cdot)$ oracle to treat "RESPOND B_t" with $B'_s \neq B_s (s \leq t)$ messages. We define an adversary $\mathcal{A}'_i$ against the IND-CCA-QREM game this way (See Fig. 5): The adversary first receives the public key pk and simulates the interaction with adversary $\mathcal{E}$ and $\mathcal{O}$ until $\mathcal{S}$ issues its ith query q^*. Then, $\mathcal{A}'_i$ submits q^* to the challenger and receives an encapsulation A^*. The adversary can call the $OQEM.Dec(\cdot)$ oracle except for input A^* to treat the "QUERY A'" message with $A' \neq A^*$. To treat

$\mathcal{E}$		$\mathcal{A}'_i$ simulates $\mathcal{S}'_i$		QREM Challenger
	$\xleftarrow{pk}$		$\xleftarrow{pk}$	$QEM.Key(1^\lambda) = \langle pk, sk\rangle$.
		Receive jth query q from $\mathcal{S}$. If $(j < i)$: Set $q^\dagger = \tilde{q}$ for random $\tilde{q}$. $QEM.Enc(pk, q^\dagger) = \langle K, A\rangle$. Record $\langle K, A, q\rangle$. If $(j > i)$: Set $q^\dagger = q$. $QEM.Enc(pk, q^\dagger) = \langle K, A\rangle$.		
		If $(j = i)$: :	$\xrightarrow{q^*=q}$	Choose $b \in \{0, 1\}$. Set $q_0 = q^*$. Select random q_1.
	$\xleftarrow{\texttt{QUERY } A}$	Record $\langle \perp, A^*, q\rangle$.	$\xleftarrow{A^*}$	$QEM.Enc(pk, q_b) = \langle K^*, A^*\rangle$.
	$\xrightarrow{\texttt{QUERY } A'}$	If $\langle K, A, q\rangle$ exists s.t. $A' = A$, then: $\mathcal{O}(q) = r_1$. If $K \neq \perp$ then set $r^\dagger = \tilde{r}$ for random $\tilde{r}$. $REM.Enc(K, 1, r^\dagger) = B'_1$.		
		Else if $K = \perp$ call $OIREM.Enc(1, r_1)$. Set $B'_1 = B^*$. Record $\langle A', B'_1, q, r_1\rangle$.	$\xrightarrow{OIREM.Enc(1,r_1)}$ $\xleftarrow{B^*}$	Set $r_0 = r_1$. Select random r_1. $REM.Enc(K^*, 1, r_b) = B^*$.
		Else call $OQEM.Dec(A')$. If $q' = \perp$ then halt. $\mathcal{O}(q') = r'_1$.	$\xrightarrow{OQEM.Dec(A')}$ $\xleftarrow{\langle K', q'\rangle}$	$QEM.Dec(sk, A') = \langle K', q'\rangle$.
	$\xleftarrow{\texttt{RESPOND } B'_1}$	$REM.Enc(K', 1, r'_1) = B'_1$. Record $\langle A', B'_1, q', r'_1\rangle$.		
	$\xrightarrow{\texttt{RESPOND } B_t}$	If $\langle K, A, B_1, B_2 \ldots, q, r_1, r_2, \ldots\rangle$ exists s.t. $A' = A$ and $B'_s = B_s (s \leq t)$, then: Answer r_t to $\mathcal{S}/\mathcal{O}$ and get r_{t+1}. If $K \neq \perp$ then set $r^\dagger = \tilde{r}$ for random $\tilde{r}$. $REM.Enc(K, t+1, r^\dagger) = B'_{t+1}$.		
		Else if $K = \perp$ call $OIREM.Enc(t+1, r_{t+1})$. Set $B'_{t+1} = B^*$. Add $\langle B'_{t+1}, r_{t+1}\rangle$ to record.	$\xrightarrow{OIREM.Enc(t+1,r_{t+1})}$ $\xleftarrow{B^*}$	Set $r_0 = r_{t+1}$. Select random r_1. $REM.Enc(K^*, t+1, r_b) = B^*$.
		Else call $OREM.Dec(t+1, B'_{t+1})$.	$\xrightarrow{OREM.Dec(t+1,B'_{t+1})}$	$REM.Dec(K^*, t+1, B'_{t+1}) = r'_{t+1}$.
	$\xleftarrow{\texttt{RESPOND } B'_{t+1}}$	If $r'_{t+1} = \perp$ then halt. Add $\langle B'_{t+1}, r'_{t+1}\rangle$ to record.	$\xleftarrow{r'_{t+1}}$	
	$\xrightarrow{\texttt{RESPOND } B_l}$	If $\langle A', B'_1, \ldots, B'_l, q', r'_1, \ldots, r'_l\rangle$ exists s.t. $A' = A$ and $B'_l = B_l$, answer r'_l to $\mathcal{S}$.		
	$\xrightarrow{\Gamma-\texttt{Judge: end}}$	If $\mathcal{A}$ wins, output $\tilde{b} = 1$, else $\tilde{b} = 0$.	$\xrightarrow{\tilde{b}}$	

Fig. 5. Reduction from Γ-encapsulate-secure** to IND-CCA-QREM

the "QUERY A'" message with $A' = A^*$, $\mathcal{A}'_i$ knows q^* so it simply queries $\mathcal{O}$ with q^* as many times as necessary to get r^*_t (for $t = 1, 2, \ldots, l$). and calls $OIREM.Enc(t, r^*)$ to get B^*. The success of the Γ-adversary against the system yields the final guess bit in the IND-CCA-QREM game. If Q/REM is IND-CCA secure, the advantage for $\mathcal{A}'_i$ is negligible, meaning that $\Pr[\mathcal{A}'_i\ \texttt{wins}] \leq \varepsilon_{sem}$. We deduce $|\Pr[\mathcal{E}(\mathcal{S}'_{i-1}, \mathcal{O})\ \texttt{wins}] - \Pr[\mathcal{E}(\mathcal{S}'_i, \mathcal{O})\ \texttt{wins}]| \leq \varepsilon_{sem}$ since the complexity of $\mathcal{A}'_i$ is $c - \mu + \mu_2$ for some small overhead cost μ_2.

$\mathcal{S}'_Q$ is such that whenever $\mathcal{E}(\mathcal{S}'_Q, \mathcal{O})$ wins for any i, between any ith QUERY A and ith "RESPOND B_t" messages, there is at least one "QUERY A'" with $A' = A$ and $B'_s = B_s (s \leq t)$ from $\mathcal{E}$ (otherwise some modified treatment of the "RESPOND B_t" message brought by $\mathcal{S}_i$ fails, since if $A' = A$ and and $B'_s = B_s (s \leq t)$ then $\mathcal{S}_i$ will not answer r'_t to $\mathcal{S}$). Due the assumption on $\mathcal{O}$, discarding repeating queries is harmless. We define a new system $\mathcal{S}'' \leftrightarrow \mathcal{O}$ which differs from $\mathcal{S}'_Q \leftrightarrow \mathcal{O}$ in the sense that for any "QUERY A'" or "RESPOND B'_t" message such that there is one record $\langle K, A, B_1, B_2, \ldots, q, r_1, r_2, \ldots\rangle$ with $A' = A$ then $\mathcal{O}$ is not queried but $\mathcal{S}''$ picks a random $\tilde{r}$ of same length as r', does $REM.Enc(K^*, t, \tilde{r}) = B'_t$ and answers "RESPOND B'_t". Obviously, $\Pr[\mathcal{E}(S'', \mathcal{O})\ \texttt{wins}] \geq \Pr[\mathcal{E}(S'_Q, \mathcal{O})\ \texttt{wins}]$.

$\mathcal{E}$		$\mathcal{A}'$		$\mathcal{S}^{toll}$
	$\xleftarrow{pk}$	$QEM.Key(1^\lambda) = \langle pk, sk\rangle$.		
	$\xrightarrow{\texttt{QUERY } A'}$ $\xleftarrow{\texttt{RESPOND } B'_1}$	$QEM.Dec(sk, A') = \langle K', q'\rangle$. $\mathcal{O}(q') = r'_1$. $REM.Enc(K', 1, r'_1) = B'_1$.		
	$\xrightarrow{\texttt{RESPOND } B_t}$ $\xleftarrow{\texttt{RESPOND } B'_{t+1}}$	$REM.Dec(K', t, B_t) = r'_t$. $\mathcal{O}(r'_t) = r'_{t+1}$. $REM.Enc(K', t+1, r'_{t+1}) = B'_{t+1}$.		
	$\xleftarrow{\texttt{QUERY } A}$ $\xrightarrow{\texttt{QUERY } A'}$ $\xleftarrow{\texttt{RESPOND } B'_1}$	Pick random $\tilde{q}$ of length l. $QEM.Enc(pk, \tilde{q}) = \langle K, A\rangle$. If $A' \neq A$, do as before. Else if $A' = A$, pick random $\tilde{r}$ of length m. $REM.Enc(K, 1, \tilde{r}) = B'_1$.	$\xleftarrow{\texttt{QUERY } l}$ $\xrightarrow{\texttt{GO}}$ $\xleftarrow{\texttt{RESPOND } m}$	
	$\xrightarrow{\texttt{RESPOND } B_t}$	If $B'_t \neq B_t$, do as before. Else pick random $\tilde{r}$ of length m. $REM.Enc(K, t+1, \tilde{r}) = B'_{t+1}$.	$\xrightarrow{\texttt{GO}}$	

Fig. 6. Going from $\mathcal{S}''$ to $\mathcal{S}^{toll}$

Finally we construct a new adversary $\mathcal{E}'$ against $\mathcal{S}^{toll}$ from the adversary $\mathcal{E}$ attacking $\mathcal{S}''$. See Fig. 6. $\mathcal{E}'$ first generates $QEM.Key(1^\lambda) = \langle pk, sk\rangle$ and simulates $\mathcal{E}$ with input pk. Interactions between $\mathcal{E}$ and $\mathcal{S}$ remain unchanged. When $\mathcal{E}'$ receives a "QUERY l" message from $\mathcal{S}^{toll}$, $\mathcal{E}'$ picks a random $\tilde{q}$ of length l, runs $QEM.Enc(pk, \tilde{q}) = \langle K', A'\rangle$ and sends a "QUERY A" message to $\mathcal{E}$. When $\mathcal{E}$ yields a "QUERY A'" message with $A' = A$, a "GO" message is given to $\mathcal{S}^{toll}$ and a random response $\tilde{r}_1$ is encapsulated. When $\mathcal{S}^{toll}$ issues a "RESPOND m" message to $\mathcal{E}'$, then $\mathcal{E}'$ answers "RESPOND B'_t" to $\mathcal{E}$. Finally when $\mathcal{E}$ sends a "RESPOND B'_t" message to $\mathcal{E}'$, then $\mathcal{E}'$ sends a "GO" message to $\mathcal{S}^{toll}$. Obviously, $\Pr[\mathcal{E}'(S^{toll}, \mathcal{O})\ \texttt{wins}] = \Pr[\mathcal{E}(S'', \mathcal{O})\ \texttt{wins}]$. Since $\mathcal{S}$ is ε-Γ-toll-secure with $\mathcal{O}$, we know that $\Pr[\mathcal{E}(S^{toll}, \mathcal{O})\ \texttt{wins}] \leq \varepsilon$ since the complexity of $\mathcal{E}'$ is $c - \mu + \mu_3$ for some small overhead cost μ_3. Hence, $\Pr[\mathcal{E}(S^{enc}, \mathcal{O})\ \texttt{wins}] \leq (Q(\varepsilon_{sem} + \varepsilon_{auth}) + \varepsilon)$. □

Author Index