Bart Preneel
Stafford Tavares (Eds.)

LNCS 3897

Selected Areas in Cryptography

12th International Workshop, SAC 2005
Kingston, ON, Canada, August 2005
Revised Selected Papers

Springer

Lecture Notes in Computer Science 3897

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

Editorial Board

Bart Preneel Stafford Tavares (Eds.)

Selected Areas in Cryptography

12th International Workshop, SAC 2005
Kingston, ON, Canada, August 11-12, 2005
Revised Selected Papers

Springer

Volume Editors

Bart Preneel
Katholieke Universiteit Leuven
Department of Electrical Engineering-ESAT
Kasteelpark Arenberg 10, 3001 Leuven-Heverlee, Belgium
E-mail: Bart.Preneel@esat.kuleuven.be

Stafford Tavares
Queen's University Kingston
Department of Electrical and Computer Engineering
Kingston, Ontario, K7L 3N6, Canada
E-mail: stafford.tavares@queensu.ca

Library of Congress Control Number: 2006922554

CR Subject Classification (1998): E.3, D.4.6, K.6.5, F.2.1-2, C.2, H.4.3

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-540-33108-5 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-33108-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11693383 06/3142 5 4 3 2 1 0

Preface

SAC 2005 was the 12th in a series of annual workshops on Selected Areas in Cryptography. This was the 5th time the workshop was hosted by Queen's University in Kingston (the previous workshops were held here in 1994, 1996, 1998 and 1999). Other SAC workshops have been organized at Carleton University in Ottawa (1995, 1997 and 2003), the Fields Institute in Toronto (2001), Memorial University of Newfoundland in St. John's (2002) and the University of Waterloo (2000 and 2004). The workshop provided a relaxed atmosphere in which researchers in cryptography could present and discuss new work on selected areas of current interest.

The themes for SAC 2005 were:

- design and analysis of symmetric key cryptosystems;
- primitives for symmetric key cryptography, including block and stream ciphers, hash functions, and MAC algorithms;
- efficient implementations of symmetric and public key algorithms;
- cryptographic algorithms and protocols for ubiquitous computing (sensor networks, RFID).

A total of 96 papers were submitted. Three papers were not considered because they were identified as being multiple submissions. After an extensive double-blind reviewing process, the program committee accepted 25 papers for presentation at the workshop.

We were very fortunate to have two invited speakers at SAC 2005, who both delivered thought-provoking and entertaining talks:

- Alfred Menezes: Another Look at Provable Security;
- Mike Wiener: The Full Cost of Cryptanalytic Attacks.

First and foremost we would like to thank the members of the program committee for the many days spent on reviewing and discussing the papers – thereby producing more than 600 Kbytes of comments – and for helping us with the difficult decisions. We would also like to thank the numerous external reviewers for their assistance. We are also indebted to Queen's University, Stantive Solutions Inc. and Sun Microsystems for their financial support of the workshop. We also wish to thank Sheila Hutchison for her administrative support, Tom Harper for developing and maintaining the SAC 2005 website and Thomas Herlea and Robert Maier for managing the Web-based review site. Special thanks to Jasper Scholten for his technical assistance during the editing of the preproceedings and this volume.

Finally we would like to thank all the participants, submitters, authors and presenters who all together made SAC 2005 a great success.

December 2005 Bart Preneel, Stafford Tavares

12th Annual Workshop on Selected Areas in Cryptography

August 11-12, 2005, Kingston, Ontario, Canada

Program and General Co-chairs

Bart Preneel ..... Katholieke Universiteit Leuven, Belgium
Stafford Tavares ..... Queen's University, Canada

Program Committee

Roberto Avanzi ..... Ruhr-University Bochum, Germany
John Black ..... University of Colorado at Boulder, USA
Henri Gilbert ..... France Telecom R&D, France
Guang Gong ..... University of Waterloo, Canada
Louis Granboulan ..... Ecole Normale Supérieure, France
Helena Handschuh ..... Gemplus, France
Howard Heys ..... Memorial University, Canada
Antoine Joux ..... DGA and University of Versailles, France
Ari Juels ..... RSA Laboratories, USA
Kaoru Kurosawa ..... Ibaraki University, Japan
Ilya Mironov ..... Microsoft Research, USA
Sean Murphy ..... Royal Holloway, University of London, UK
Vincent Rijmen ..... Graz University of Technology, Austria
Doug Stinson ..... University of Waterloo, Canada
Michael Wiener ..... Cryptographic Clarity, Canada
Amr Youssef ..... Concordia University, Canada

Local Arrangements Committee

Sheila Hutchison ..... Queen's University, Canada
Tom Harper ..... Queen's University, Canada

Sponsoring Institutions

Queen's University
Sun Microsystems
Stantive Solutions Inc.

External Referees

Masayuki Abe
Steve Babbage
Lejla Batina
Côme Berbain
Olivier Billet
Alex Biryukov
An Braeken
Carlos Cid
Mathieu Ciet
Christophe Clavier
Christophe De Cannière
Jacques Fournier
Steven Galbraith
Kenneth Giuliani
Aline Gouget
Kishan Gupta
Swee-Huay Heng
Katrin Hoeper
Tetsu Iwata
Tetsuya Izu
Shaoquan Jiang
Liam Keliher
Kazukuni Kobara
Takeshi Koshiba
Ted Krovetz
Tanja Lange
Joseph Lano
Dong Hoon Lee
Jooyoung Lee
Kerstin Lemke
Yi Lu
Subhamoy Maitra
Stefan Mangard
Keith Martin
Alexander May
Preda Mihăilescu
Atsuko Miyaji
Bodo Möller
David Molnar
Yassir Nawaz
Khanh Nguyen
Miyako Ohkubo
Yasuhiro Ohtaki
Akira Ohtsuka
Francis Olivier
Elisabeth Oswald
Jan Pelzl
Norbert Pramstaller
Christian Rechberger
Matt Robshaw
Rei Safavi-Naini
Palash Sarkar
Erkay Savas
Martin Schläffer
Junji Shikata
Thomas Shrimpton
Francesco Sica
Jessica Staddon
Gelareh Taban
Tsuyoshi Takagi
Duong Quang Viet
Frederik Vercauteren
Dai Watanabe
Christopher Wolf
Johannes Wolkerstorfer
Lu Xiao
Nam Yul Yu

Table of Contents

Stream Ciphers II

Key Establishment Protocols and Access Control

Hash Functions

Protocols for RFID Tags

Efficient Implementations

Conditional Estimators: An Effective Attack on A5/1

Elad Barkan and Eli Biham

Computer Science Department,
Technion – Israel Institute of Technology,
Haifa 32000, Israel
{barkan, biham}@cs.technion.ac.il
http://www.technion.ac.il/∼barkan/
http://www.cs.technion.ac.il/∼biham/

Abstract. Irregularly-clocked linear feedback shift registers (LFSRs) are commonly used in stream ciphers. We propose to harness the power of conditional estimators for correlation attacks on these ciphers. Conditional estimators compensate for some of the obfuscating effects of the irregular clocking, resulting in a correlation with a considerably higher bias. On GSM's cipher A5/1, a factor two is gained in the correlation bias compared to previous correlation attacks. We mount an attack on A5/1 using conditional estimators and using three weaknesses that we observe in one of A5/1's LFSRs (known as $R2$). The weaknesses imply a new criterion that should be taken into account by cipher designers. Given 1500–2000 known-frames (about 4.9–9.2 conversation seconds of known keystream), our attack completes within a few tens of seconds to a few minutes on a PC, with a success rate of about 91%. To complete our attack, we present a source of known-keystream in GSM that can provide the keystream for our attack given 3–4 minutes of GSM ciphertext, transforming our attack to a ciphertext-only attack.

1 Introduction

Correlation attacks are one of the prominent generic attacks on stream ciphers. There were many improvements to correlation attacks after they were introduced by Siegenthaler [13] in 1985. Many of them focus on stream ciphers composed of one or more regularly clocked linear feedback shift registers (LFSRs) whose output is filtered through a non-linear function. In this paper, we discuss stream ciphers composed of irregularly-clocked linear feedback shift registers (LFSRs), and in particular, on stream ciphers whose LFSRs' clocking is controlled by the mutual value of the LFSRs. The irregular clocking of the LFSRs is intended to strengthen the encryption algorithm by hiding from the attacker whether a specific register advances or stands still. Thus, it should be difficult for an attacker to correlate the state of an LFSR at two different times (as he does not know how many times the LFSR has been clocked in between).

Assume the attacker knows the number of clocks that the LFSRs have been clocked until a specific output bit has been produced. The attacker can guess the

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 1–19, 2006.

number of clocks that the LFSRs are clocked during the generation of the next output bit with some success probability $p < 1$. A better analysis that increases the success probability of guessing the number of clocks for the next output bit could prove devastating to the security of the stream cipher. Our proposed conditional estimators are aimed at increasing this success probability.

In this paper, we introduce conditional estimators, aimed to increase the probability of guessing the clockings of the LFSRs correctly. We apply conditional estimators to one of the most fielded irregularly clocked stream ciphers — A5/1, which is used in the GSM cellular network. GSM is the most heavily deployed cellular phone technology in the world. Over a billion customers world-wide own a GSM mobile phone. The over-the-air privacy is currently protected by one of two ciphers: A5/1 — GSM's original cipher (which was export-restricted), or A5/2 which is a weakened cipher designated for non-OECD (Organization for Economic Co-operation and Development) countries. As A5/2 was discovered to be completely insecure [3], the non-OECD countries are now switching to A5/1. The internal design of A5/1 and A5/2 was kept secret until Briceno [6] reverse engineered their internal design in 1999. Contrary to A5/1 and A5/2, the internal design of the future GSM cipher A5/3 was officially published.

The first attacks on A5/1 were proposed by Golic [9] in 1997, when only a rough design of the cipher was leaked. He proposed two known-keystream attacks: the first is a guess and determine attack, and the second is a time-memory tradeoff attack. In 2000, the second attack was significantly improved by Biryukov, Shamir, and Wagner [5]. In some scenarios the improved attack can find the key in less than a second. However, the attack requires four 74-gigabyte disks and a lengthy precomputation. At the same time, Biham and Dunkelman [4] took a different approach. Their attack requires a few tens of seconds of known-keystream and recovers the key with a time complexity of about 2^{40} A5/1 cycles. In 2003, Barkan, Biham, and Keller [3] showed a ciphertext-only attack that finds the encryption key of A5/2, using the fact that in GSM the error-correction codes are employed before encryption. They converted this attack to an active attack on A5/1 networks, and also presented a ciphertext-only time-memory tradeoff attack on A5/1. However, the latter requires a very lengthy precomputation step. As for correlation attacks, in 2001, Ekdahl and Johansson [7] applied ideas from correlation attacks to A5/1. Their attack requires a few minutes of known-keystream, and finds the key within minutes on a personal computer. In 2004, Maximov, Johansson, and Babbage [11] discovered a correlation between the internal state and the output bits and used it to improved the attack. Given about 2000–5000 frames (about 9.2–23 seconds of known-plaintext), their attack recovers the key within 0.5–10 minutes on a personal computer.

These attacks demonstrate that fielded GSM systems do not provide an adequate level of privacy for their customers. However, breaking into fielded A5/1 GSM systems using these attacks requires either active attacks (e.g., man in the middle), a lengthy (although doable) precomputation step, a high time complexity, or a large amount of known keystream.

One advantage of correlation attacks on A5/1 over previous attacks is that they require no long-term storage and no preprocessing, yet given a few seconds of known-keystream, they can find the key within minutes on a personal computer. Another advantage of correlation attacks over some of the previous attacks is the immunity to transmission errors. Some of the previous attacks are susceptible to transmission errors, e.g., a single flipped bit defeats Golic's first attack. Correlation attacks can naturally withstand transmission errors, and even a high bit-error-rate can be accommodated for.

In this paper, we introduce conditional estimators, which can compensate for some of the obfuscating effects caused by the irregular clocking. Using conditional estimators, we improve the bias of the correlation equation that was observed in [11] by a factor of two. In addition, we discover three weaknesses in one of A5/1's registers. We mount a new attack on A5/1 based on the conditional estimators and the three weaknesses. Finally, we describe a source for known keystream transforming our attack to a ciphertext-only attack.

One of the weaknesses relates to the fact that register $R2$ of A5/1 has only two feedback taps, which are adjacent. This weakness enables us to make an optimal use of the estimators by translating the problem of recovery of the internal state of the register to a problem in graph theory. Thus, unlike previous attacks [7, 11], which were forced to use heuristics, we can exactly calculate the list of most probable internal states. We note that in 1988, Meier and Staffelbach [12] warned against the use of LFSRs with few feedback taps. However, it seems that their methods are difficult to apply to A5/1.

An alternative version of our attack can take some advantage of the fact that many operators set the first bits of the key to zero (as reported in [6]); this alternative version slightly simplifies the last step of our attack, and results with a somewhat higher success rate. We are not aware of any other attack on A5/1 (except for exhaustive search) that could benefit from these ten zero bits.

Our last contribution is a new source for known-plaintext in GSM. We point at the Slow Associated Control CHannel (SACCH) and show that its content can be derived. We also discuss the frequency hopping in GSM and how to overcome it. Using this new source for known-plaintext, our attacks can be converted to ciphertext-only attacks. However, this is a slow channel, that provides only about eight known frames each second.

We have performed simulations of our attacks. Given 2000 frames, our simulations take between a few tens of seconds and a few minutes on a PC to find the key with a success rate about 91%. For comparison, the simulations of [11] with a similar number of frames take about four times longer to run and achieve a lower success rate of about only 5%. A comparison of some of the results of previous works and our results is given in Table 1. With our new source for known keystream, the required 1500–2000 *known frames* can be obtained from the *ciphertext* of about 3–4 minutes of conversation.

This paper is organized as follows: We give a short description of A5/1 in Section 2, then, we set our notations and review some of the main ideas of previous works in Section 3. In Section 4 we describe the conditional estimators

Table 1. Comparison Between the Our Attacks and Previous Works. Only passive attacks are included, i.e., the active attack of [3] is not shown. The attack time for [3, 4, 5] is our estimate. As [3, 5] are time/memory/data tradeoff attacks, we give the tradeoff point that uses data that is equivalent to four minutes of ciphertext.
* based on error-correction codes as described in [3] (not on Section 7).
♯ preprocessing time

Attack: (Configuration explained in Section 6)	Required Frames: Known Keystream	Required Frames: Ciphertext Only (by Section 7)	Average Time on a single PC (range)	Success Rate
Ekdahl & Johansson [7]	70000 (322 s)	140 min	5 min	76%
[7]	50000 (230 s)	99 min	4 min	33%
[7]	30000 (138 s)	60 min	3 min	3%
Biham & Dunkelman [4]	20500 (95 s)	40.8 min	≈ 1.5 days	63%
Maximov et al. [11]	10000 (46 s)	20 min	10 min	99.99%
[11]	10000 (46 s)	20 min	76 s	93%
[11]	5000 (23 s)	10 min	10 min	85%
[11]	5000 (23 s)	10 min	44 s	15%
[11]	2000 (9.2 s)	4 min	10 min	5%
[11]	2000 (9.2 s)	4 min	29 s	1%
Biryukov et al. [5]	2000 (9.2 s)	4 min	♯ > 5 years	
Ciphertext only attack of Barkan et al. [3]	—	4 min*	♯ > 2300 years	
This Paper	2000 (9.2 s)	4 min	(6–10 min)	64%
early filtering (220000, 40000, 2000, 5200)	2000 (9.2 s)	4 min	(55–300 s)	64%
early filtering (100000, 15000, 200, 300)	2000 (9.2 s)	4 min	(32–45 s)	48%
improved estimators, (200000, 17000, 900, 2000)	2000 (9.2 s)	4 min	74 s (50–145 s)	86%
improved estimators, (200000, 36000, 1400, 11000)	**2000 (9.2 s)**	**4 min**	**133 s (55–626s)**	**91%**
early filtering (120000, 35000, 1000, 800)	1500 (6.9 s)	3 min	(39–78 s)	23%
improved estimators, (88000, 52000, 700, 1200)	1500 (6.9 s)	3 min	82 s (44–105 s)	48%
improved estimators, (88000, 52000, 3200, 15000)	**1500 (6.9 s)**	**3 min**	**7.2 min (44–780 s)**	**54%**

and three weaknesses, and then use them in our new attack in Section 5. The results of our simulations are presented in Section 6. We describe the new source of known-plaintext in Section 7. Finally, the paper is summarized in Section 8.

2 A Description of A5/1

The stream cipher A5/1 accepts a 64-bit session key K_c and a 22-bit publicly-known frame number f. GSM communication is performed in frames, where a frame is transmitted every 4.6 millisecond. In every frame, A5/1 is initialized with the session key and the frame number. The resulting 228 bit output (keystream) is divided into two halves: the first half is used to encrypt the data from the network to the mobile phone, while the second half is used to encrypt the data from the mobile phone to the network. The encryption is performed by XORing the data with the appropriate half of the keystream.

A5/1 has a 64-bit internal state, composed of three maximal-length Linear Feedback Shift Registers (LFSRs): $R1$, $R2$, and $R3$, with linear feedbacks as shown in Figure 1. Before a register is clocked the feedback is calculated (as the XOR of the feedback taps). Then, the register is shifted one bit to the right (discarding the rightmost bit), and the feedback is stored into the leftmost location (location zero).

A5/1 is initialized with K_c and f in three steps, as described in Figure 2. This initialization is referred to as the *key setup*.

Observe that the key setup is linear in the bits of both K_c and f, i.e., once the key setup is completed, every bit of the internal state is an XOR of bits in fixed locations of K_c and f. This observation is very helpful in correlation attacks.

A5/1 works in cycles, where in each cycle one output bit is produced. A cycle is composed of irregularly clocking $R1$, $R2$, and $R3$ according to a clocking mechanism that we describe later, and then outputting the XOR of the rightmost bits of the three registers (as shown in Figure 1). The first 100 bits of output are discarded, i.e., the 228 bits that are used in GSM are output bits 100,...,327. The keystream generation can be summarized as follows:

1. Run the key setup with K_c and f (Figure 2).
2. Run A5/1 for 100 cycles and discard the output.
3. Run A5/1 for 228 cycles and use the output as keystream.

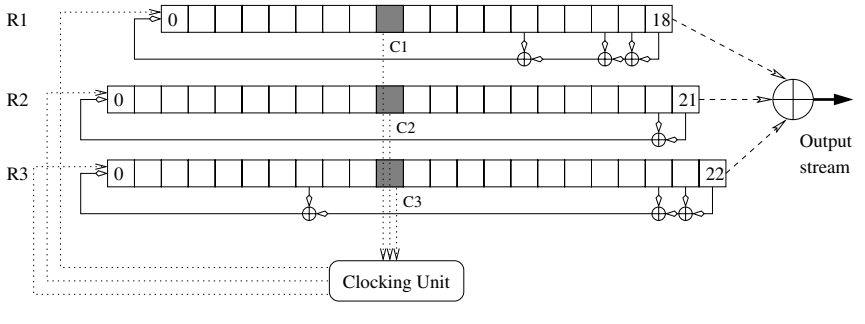


Fig. 1. The A5/1 internal structure

1. Set $R1 = R2 = R3 = 0$.
2. For $i = 0$ to 63
   - Clock all three registers.
   - $R1[0] \leftarrow R1[0] \oplus K_c[i]$; $R2[0] \leftarrow R2[0] \oplus K_c[i]$; $R3[0] \leftarrow R3[0] \oplus K_c[i]$.
3. For $i = 0$ to 21
   - Clock all three registers.
   - $R1[0] \leftarrow R1[0] \oplus f[i]$; $R2[0] \leftarrow R2[0] \oplus f[i]$; $R3[0] \leftarrow R3[0] \oplus f[i]$.

Fig. 2. The key setup of A5/1. The i'th bit of K_c is denoted by $K_c[i]$, and the i'th bit of f is denoted by $f[i]$, where $i = 0$ is the least significant bit. We denote the internal state after the key setup by $(R1, R2, R3) = \text{keysetup}(K_c, f)$.

It remains to describe the clock control mechanism, which is responsible for the irregular clocking. Each register has a special *clocking tap* near its middle (in locations $R1[8]$, $R2[10]$, and $R3[10]$). The clocking mechanism algorithm:

1. Calculate the majority of the values in the three clocking taps.
2. Then, clock a register if and only if its clocking tap agrees with the majority.

For example, assume that $R1[8] = R2[10] = c$ and $R3 = 1 - c$ for some $c \in \{0, 1\}$. Clearly, the value of the majority is c. Therefore, $R1$ and $R2$ are clocked, and $R3$ stands still.

Note that in each cycle of A5/1, either two or three registers are clocked (since at least two bits agree with the majority). Assuming that the clocking taps are uniformly distributed, each register has a probability of 1/4 for standing still and a probability of 3/4 for being clocked.

3 Notations and Previous Works

In this section, we set our notations, and describe some of the main ideas of the previous works. Let S_1, S_2, and S_3 be the initial internal state of registers $R1$, $R2$, and $R3$ after the key-setup (using the correct K_c), where the frame number is chosen to be zero, i.e., $(S_1, S_2, S_3) = \text{keysetup}(K_c, 0)$. For $i = 1, 2, 3$, denote by $\tilde{S}_i[l_i]$ the output bit of Ri after it is clocked l_i times from its initial state S_i.[1] Similarly, let F_1^j, F_2^j, and F_3^j be the initial internal state of registers $R1$, $R2$, and $R3$ after a key setup using all zeros as the key, but with frame number j, i.e., $(F_1^j, F_2^j, F_3^j) = \text{keysetup}(0, j)$. For $i = 1, 2, 3$, denote by $\tilde{F}_i^j[l_i]$ the output of Ri after it is clocked l_i times from its initial state F_i^j. Ekdahl and Johansson [7] observed that due to the linearity of the key setup, the initial internal value of Ri at frame j is given by $S_i \oplus F_i^j$, i.e., $\text{keysetup}(K_c, j) = \text{keysetup}(K_c, 0) \oplus \text{keysetup}(0, j) = (S_1 \oplus F_1^j, S_2 \oplus F_2^j, S_3 \oplus F_3^j)$. Furthermore,

[1] Note that as a register has a probability of 3/4 of being clocked in each cycle, it takes about $l_i + l_i/3$ cycles to clock the register l_i times.

due to the linear feedback of the shift register, the output of LFSR i at frame j after being clocked l_i times from its initial state is given by $\tilde{S}_i[l_i] \oplus \tilde{F}_i^j[l_i]$.

Maximov, Johansson, and Babbage [11] made the following assumptions:

1. *clocking assumption* (j, l_1, l_2, t): Given the keystream of frame j, registers $R1$ and $R2$ were clocked exactly l_1 and l_2 times, respectively, until the end of cycle t. The probability that this assumption holds is denoted by $Pr((l_1, l_2)$ at time $t)$ (this probability can be easily computed, see [11]).
2. *step assumption* (j, t): Given the keystream of frame j, both $R1$ and $R2$ are clocked in cycle $t+1$, but $R3$ stands still. Assuming the values in the clocking taps are uniformly distributed, this assumption holds with probability 1/4 (the clocking mechanism ensures that if the values of the clocking taps are uniformly distributed, each register stands still with probability 1/4).

They observed that under these two assumptions, $R3$'s contribution to the output is fixed in output bits t and $t+1$. Thus, $R3$ does not affect the difference between these two output bits, and the following equation holds:

$$(\tilde{S}_1[l_1] \oplus \tilde{S}_2[l_2]) \oplus (\tilde{S}_1[l_1+1] \oplus \tilde{S}_2[l_2+1]) = \\ \tilde{Z}^j[t] \oplus \tilde{Z}^j[t+1] \oplus (\tilde{F}_1^j[l_1] \oplus \tilde{F}_2^j[l_2]) \oplus (\tilde{F}_1^j[l_1+1] \oplus \tilde{F}_2^j[l_2+1]), \quad (1)$$

where $\tilde{Z}^j[t]$ is the output bit of the cipher at time t of frame j. Thus, the value of $(\tilde{S}_1[l_1] \oplus \tilde{S}_2[l_2]) \oplus (\tilde{S}_1[l_1+1] \oplus \tilde{S}_2[l_2+1])$ can be estimated from the known keystream and the publicly available frame numbers.

Equation (1) holds with probability 1 if both the clocking assumption and the step assumption hold. If either or both assumptions do not hold, then Equation (1) is assumed to hold with probability 1/2 (i.e., it holds by pure chance). Therefore, Equation (1) holds with probability $(1 - Pr((l_1, l_2)$ at time $t))/2 + Pr((l_1, l_2)$ at time $t)((3/4)/2 + 1/4) = 1/2 + Pr((l_1, l_2)$ at time $t)/8$. The bias $Pr((l_1, l_2)$ at time $t)/8$ is typically two to three times higher compared to the bias shown in [7]. Such a difference in the bias is expected to result in an improvement of the number of frames needed by a factor between four and ten, which is indeed the case in [11].

We simplify Equation (1) by introducing the notation $\tilde{S}'_i[l_i]$ defined as $\tilde{S}_i[l_i] \oplus \tilde{S}_i[l_i+1]$. Similarly denote $\tilde{F}_i^j[l_i] \oplus \tilde{F}_i^j[l_i+1]$ by $\tilde{F}_i'^j[l_i]$, and denote $\tilde{Z}^j[t] \oplus \tilde{Z}^j[t+1]$ by $\tilde{Z}'^j[t]$. Thus, Equation (1) can be written as:

$$(\tilde{S}'_1[l_1] \oplus \tilde{S}'_2[l_2]) = \tilde{Z}'^j[t] \oplus (\tilde{F}_1'^j[l_1] \oplus \tilde{F}_2'^j[l_2]) \quad (2)$$

Observe that due to the linearity of the LFSR, $\tilde{S}'_i[l_i]$ can be viewed as the output of Ri after it has been clocked l_i times from the initial state $S'_i \triangleq S_i^+ \oplus S_i$, where S_i^+ denotes the internal state of Ri after it has been clocked once from the internal state S_i. Note that there is a one-to-one correspondence between S_i and S'_i, therefore, when we recover S'_i, we find S_i.

In [11] it was observed that better results are obtained by working with d consecutive bits of the output of S'_i, where d is a small integer. A *symbol* is defined to be the binary string of d consecutive bits $S'_i[l_i] \triangleq \tilde{S}'_i[l_i]||\tilde{S}'_i[l_i+1]|| \cdots ||\tilde{S}'_i[l_i+d-1]$,

where "||" denotes concatenation. For example, $S'_2[81] = \tilde{S}'_2[81]$ is a 1-bit symbol, and $S'_1[90] = \tilde{S}'_1[90]||\tilde{S}'_1[91]$ is a 2-bit symbol.

In the first step of [11], estimators are calculated based on the above correlation and on the available keystream. For every pair of indices l_1 and l_2 for which estimators are computed, and for every possible symbol difference $\delta = S'_1[l_1] \oplus S'_2[l_2]$, the estimator $E_{l_1,l_2}[\delta]$ is the logarithm of the a-posteriori probability that $S'_1[l_1] \oplus S'_2[l_2] = \delta$. For example, for $d = 1$, the symbol is a single bit, thus, the symbol difference can be either zero or one. Then, for $l_1 = 80$ and $l_2 = 83$, the estimator $E_{80,83}[0]$ is the logarithm of the probability that $S'_1[80] \oplus S'_2[83] = 0$, and $E_{80,83}[1]$ is the logarithm of the probability that $S'_1[80] \oplus S'_2[83] = 1$. For $d = 2$, there are four estimator for every pair of indices, e.g., $E_{80,83}[00_2]$, $E_{80,83}[01_2]$, $E_{80,83}[10_2]$, and $E_{80,83}[11_2]$ (where "$_2$" denotes the fact that the number is written in its binary representation, e.g., 11_2 is the binary representation of the number 3). The value of $E_{80,83}[10_2]$ is the logarithm of the probability that $S'_1[80] \oplus S'_2[83] = 10_2$, and so on. Note that the higher d is — the better the estimators are expected to be (but the marginal benefit drops exponentially as d grows).

In order to save space, we do not describe here how to calculate the estimators given the known-keystream and d. See [11] for the details. We would only note that the time complexity of this step is proportional to 2^d. With 2000 frames, the simulation in [11] takes about eleven seconds to complete this step with $d = 1$, and about 40 seconds with $d = 4$.

The rest of the details of previous works deal with how to decode the estimators and to recover candidate values for S_1, S_2, and S_3 (and thus recovering the key). These methods are basically heuristic methods that decode the estimators in short intervals of l_i of the output of S'_i, and then intersect the resulting candidates to find candidates for S_1, S_2, and S_3.

4 The New Observations

In this section, we describe tools and observations that we later combine to form the new attack.

4.1 The New Correlation — Conditional Estimators

In Section 3, we reviewed the correlation equation used by Maximov, Johansson, and Babbage. This correlation equation is based on two assumptions, the clocking assumption and the step assumption. Recall that the step assumption (i.e., that the third register stands still) holds in a quarter of the cases (assuming that the values in the clocking taps are independent and uniformly distributed).

Consider registers $R1$ and $R2$, and assume that for a given frame j and output bit t the clocking assumption holds, i.e., we know that $R1$ and $R2$ were clocked l_1 and l_2 times, respectively, from their initial state. Also assume that we know the value of $\tilde{S}_1[l_1 + 10]$ and $\tilde{S}_2[l_2 + 11]$. We use the publicly known frame number j to find the value of the clocking taps $C_1 = \tilde{S}_1[l_1 + 10] \oplus \tilde{F}_1^j[l_1 + 10]$ of $R1$ and $C_2 = \tilde{S}_2[l_2 + 11] \oplus \tilde{F}_2^j[l_2 + 11]$ of $R2$ at output bit t.

We observe that the bias of the correlation can be improved by a factor of two by dividing the step assumption into two distinct cases. The first of the two distinct cases is when $C_1 \neq C_2$. Due to the clocking mechanism, $R3$ is always clocked in this case along with either $R1$ or $R2$. The step assumption does not hold, and therefore, Equation (2) is assumed to hold in half the cases. In other words, the case where $C_1 \neq C_2$ provides us no information.

However, in the second case, when $C_1 = C_2$, we gain a factor two increase in the bias. In this case, both $R1$ and $R2$ are clocked (as $c = C1 = C2$ is the majority), and $R3$ is clocked with probability 1/2, in case its clocking tap $C3 = c$ (we assume that the values of the clocking taps are uniformly distributed). Therefore, when $C_1 = C_2$, the step assumption holds with probability 1/2 compared to probability 1/4 in [11].

We analyze the probability that Equation (2) holds when $C1 = C2$. If either the step assumption or the clocking assumption do not hold, then we expect that Equation (2) holds with probability 1/2 (i.e., by pure chance). Together with the probability that the assumptions hold, Equation (2) is expected to hold with probability $Pr((l_1, l_2) \text{ at time } t)(1/2{+}1/2{\cdot}1/2){+}1/2(1{-}Pr((l_1, l_2) \text{ at time } t)) = 1/2 + Pr((l_1, l_2) \text{ at time } t)/4$ compared to $1/2 + Pr((l_1, l_2) \text{ at time } t)/8$ in [11]. Therefore, when $C_1 = C_2$, we gain a factor two increase in the bias compared to [11].[2]

We use the above observation to construct *conditional estimators* (which are similar to conditional probabilities). We define a d-bit *clock symbol* $S_i[l_i]$ in index l_i as the d-bit string: $\tilde{S}_i[l_i]||\tilde{S}_i[l_i{+}1]||\cdots||\tilde{S}_i[l_i{+}d{-}1]$, where "||" denotes concatenation. The conditional estimator $E_{l_1,l_2}[x|Sc]$ for indices l_1, l_2 is computed for every possible combination of a clock symbol difference $Sc = S_1[l_1{+}10] \oplus S_2[l_2{+}11]$ and a symbol difference $x = S'_1[l_1] \oplus S'_2[l_2]$. The estimator $E_{l_1,l_2}[x|Sc]$ is the logarithm of the a-posteriori probability that the value of the symbol difference is x, given that the value of the clock symbol difference is Sc. The computation of conditional estimators is similar to the computation of the estimators as described in [11], taking into account the above observations. The complete description of the calculation of conditional estimators will be given in the full version of this paper.

One way of using conditional estimators is to remove the conditional part of the estimators, and use them as regular estimators, i.e., compute $E_{l_1,l_2}[x] = \log\left(\frac{1}{2^d}\sum_y e^{E_{l_1,l_2}[x|y]}\right)$. Nevertheless, the benefit would not be large. A better use of the conditional estimators is to use them directly in the attack as is shown in Section 5.1.

4.2 First Weakness of $R2$ — The Alignment Property

The first weakness of $R2$ uses the fact that the feedback taps of $R2$ coincide with the bits that are estimated by the correlation equation. Assume that the

[2] As a refinement of these observations, note that it suffices to know the value of $\tilde{S}_1[l_1 + 10] \oplus \tilde{S}_2[l_2 + 11]$, since we only consider $C_1 \oplus C_2$ rather than the individual value of $C1$ and $C2$.

value of S_1 is known. Then for every index i, the correlation equation estimates the value of $S_2[i] \oplus S_2[i+1]$. On the other hand the linear feedback of $R2$ forces $S_2[i] \oplus S_2[i+1] = S_2[i+22]$. Thus, the correlation equation actually estimates bits which are 22 bits away. Using our notations, this property can be written as

$$S'_2[i] = S_2[i+22].$$

4.3 Second Weakness of $R2$ — The Folding Property

The second weakness of $R2$ is that it has only two feedback taps, and these taps are adjacent. Let $X[*]$ be a bit-string which is an output of $R2$, and let $cost(i, x)$ be a cost function that sets a cost for every possible d-bit string x in index i of the string $X[*]$ (the cost function is independent of the specific stream $X[*]$). We calculate the total cost of a given string $X[*]$ (i.e., calculate its "score") by

$$\sum_i cost(i, X[i]||X[i+1]||\cdots||X[i+d-1]). \tag{3}$$

Given the cost function, we can also ask what is the string X_{max} that maximizes the above sum, i.e., the string with the highest score.

The folding property allows to create a new cost function $cost'(i, x)$, where i is one of the first 22 indices. The special property of $cost'$ is that the score calculated on the first 22 indices using $cost'$ is equal to the score using Equation (3) over all the indices (using $cost$). $cost'$ is very helpful in finding the highest scored string X_{max} for a given cost function $cost$. However, the transition from $cost$ to $cost'$ has the penalty that $cost'(i, x)$ operates on d'-bit strings x that are slightly longer than d. In general, every 22 additional indices (beyond the first 22 indices) in $X[*]$ add one bit of length to x (in our simulation we work with strings of 66 indices, therefore, our $cost'$ operates on strings of length $d' = d + 2$).

For every index i, it holds that $X[i+22] = X[i] \oplus X[i+1]$, due to the linear feedback taps of $R2$. Therefore, the d'-bit string at index i determines a $(d'-1)$-bit string at index $i+22$, a $(d'-2)$-bit string at index $i+2\cdot 22$, a $(d'-3)$-bit string at index $i+3\cdot 22$, etc. Clearly, the contribution to the score of the strings in these indices is also determined by the value of the d'-bit string at index i, and thus can be "folded" into the cost function for index i.

For simplicity, we assume that the number of indices is divisible by 22, i.e., $22k + d - 1$ bits of $X[*]$ are included in the score computation (the attack can easily be extended to cases where the number of indices is not divisible by 22). The calculation of $cost'$ from $cost$ is given in Figure 3. We call the d'-bit strings *representative symbols*. Note that not every choice of 22 representative symbols is a consistent output of $R2$, as the 22 representative symbols span $22 + d' - 1$ bits (and thus there are $2^{22+d'-1}$ possibilities for these bits), while $R2$'s internal state has 22 bits. Specifically, the last $d' - 1$ bits are determined by the first d' bits through the linear feedback. Denote these last $d' - 1$ bits by w.

The linear feedback of $R2$ is actually calculating the difference between adjacent bits. We denote this operation using the difference operator D, i.e., $D(\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_{d'}) = (\alpha_1 \oplus \alpha_2, \alpha_2 \oplus \alpha_3, \ldots, \alpha_{d'-1} \oplus \alpha_{d'})$.

For each $i \in \{i_s, \ldots, i_s + 21\}$
  For each $e \in \{0,1\}^{d+k-1}$
    $cost'(i,e) \triangleq \sum_{j=0}^{k-1} cost(i + 22k, \mathrm{lsb}_d(D^j(e))$

Fig. 3. The folding property; calculating $cost'$ from $cost$. Denote the first index of $X[*]$ by i_s, and the number of indices by $22k$. $D(x)$ is the difference function that calculates the difference string — the parity of each two adjacent bits in x; $D^j(x)$ is j applications of D on x. $\mathrm{lsb}_d(x)$ returns the d least significant bits of x, thus, $\mathrm{lsb}_d(D^j(e))$ is the d-bit string in index $i + 22k$ that is determined by e.

For the first bits to be consistent with the last bits w, we require that the first bits are equal to $D_0^{-1}(w)$ or $D_1^{-1}(w)$, where $D_0^{-1}(w)$ is the value such that $D(D_0^{-1}(w)) = w$, and the first bit of $D_0^{-1}(w)$ is zero (i.e., D_0^{-1} is one of two inverses of D). $D_1^{-1}(w)$ is the 1-complement of $D_0^{-1}(w)$ (it also satisfies $D(D_1^{-1}(w)) = w$, i.e., D_1^{-1} is the other inverse of D).

4.4 Third Weakness of *R2* — The Symmetry Property

The third weakness in $R2$ is that its clock tap is exactly in its center. Combined with the folding property, a symmetry between the clocking tap and the output tap of $R2$ is formed. The symmetry property allows for an efficient attack using conditional estimators. Assume that S_1 is known. $S_2[i]$ is at the output tap of $R2$ when $S_2[i+11]$ is at the clock tap. When $S_2[i+11]$ reaches the output tap, $S_2[i+11+11] = S_2[i+22]$ is at the clock tap. However, the representative symbol at i determines both the bits of $S_2[i]$ and $S_2[i+22]$. Therefore, the representative symbols are divided into pairs, where each pair contains a representative symbol of some index i and a representative symbol of index $i+11$. When the representative symbols of index i serve for clocking, the other representative symbol is used for the output, and vice versa, i.e., the representative symbols in the pair control the clocking of each other. If the clocking taps were not in the middle, we could not divide the representative symbols into groups of two.

5 The New Attack

The attack is composed of three steps:

1. Compute the conditional estimators.
2. Decode the estimators to find list of best candidate pairs for S_1, S_2 values, by translating the problem of finding the best candidates to a problem in graph-theory.
3. For each candidate in the list, recover candidates for S_3. When a triplet S_1, S_2, S_3 is found, the key is recovered and verified through trial encryptions.

The computation of conditional estimators is based on Section 4.1, and similar to the computation of estimators in [11]. We will give a full description of this computation in the full version of the paper. Step 2 is described in Section 5.1.

In Step 3, given candidate pairs for S_1 and S_2, we work back candidates for S_3 from the keystream. The method is similar to the one briefly described by Ross Anderson in [1]. However, some adjustments are needed as the method in [1] requires the internal state right at the beginning of the keystream (after discarding 100 bits of output), whereas Step 2 provides candidates for the internal state after the key setup but before discarding 100 bits of output (the candidates for S_1 and S_2 XORed with F_1^j and F_2^j, respectively, are the internal state right after the key-setup and before discarding 100 bits of output). An alternative Step 3 exhaustively tries all 2^{23} candidate values for S_3. Taking into account that many operators set ten bits of the key to zero (as reported in [6]), we need to try only the 2^{13} candidate values for S_3 which are consistent with the ten zero bits of the key. A more detailed description of Step 3 will be given in the full version of this paper.

5.1 Step 2 — Decoding of Estimators

The aim of Step 2 is to find the list of best scored candidates for S_1 and S_2, based on the conditional estimators. The score of s_1 and s_2 (candidate values for S_1 and S_2, respectively) is simply the sum of their estimators (which is the logarithm of the product of the a-posteriori probabilities), i.e.,

$$score(s_1, s_2) = \sum_{l_1, l_2} E_{l_1,l_2}[s'_1[l_1] \oplus s'_2[l_2] \quad | \quad s_1[l_1 + 10] \oplus s_2[l_2 + 11]].$$

The list of best candidates is the list of candidates $\{(s_1, s_2)\}$ that receive the highest values in this score. For the case of non-conditional estimators, the *score* is defined in a similar manner but using non-conditional estimators (instead of conditional estimators).

Surprisingly, the list of best candidate pairs can be efficiently computed using the three weaknesses of $R2$. We translate the problem of calculating the list of best scored candidates into a problem in graph theory. The problem is modeled as a huge graph with a source node s and target node t, where each path in the graph from s to t corresponds to a candidate value for S_1 and S_2, with the score of the pair being the sum of the costs of the edges along the path (also, for every candidate pair s_1, s_2, there is a single path in the graph from s to t). Thus, the path with the heaviest score ("longest" path) corresponds to the highest scored pair. A Dijkstra-like algorithm [2] (for finding shortest path) can find the longest path, since the weights on the edges in our graph are negative (logarithm of probability). The list of best candidates corresponds to the list of paths which are near the heaviest. The literature for graph algorithms dealt with finding N-shortest paths in a graph (e.g., [10]); these algorithms can be adapted to our graph, and allow to find the heaviest paths.

Our graph contains 2^{19} subgraphs, one for each candidate value for S_1. All the subgraphs have the same structure, but the weights on the edges are different. Each such subgraph has one incoming edge entering the subgraph from the source node s, and one outgoing edge from the subgraph to the target node t. Both edges have a cost of zero.

5.1.1 The Structure of the Sub-graph Using Non-conditional Estimators

Our method for decoding the estimators can be used with non-conditional estimators, and in fact the structure of the subgraph is best understood by first describing the structure of the subgraph for the case of non-conditional estimators. In this case, the subgraph for the j^{th} candidate of S_1 has a source node s_j and a target node t_j. The subgraph is composed of $2^{d'-1}$ mini-subgraphs. Each mini-subgraph corresponds to one combination w of the last $d'-1$ bits of the representative symbol in index i_s+21 (last representative symbol). Figure 4 shows an example of a subgraph for $d'=3$, in which only the mini-subgraph for $w=01$ is shown. The full subgraph contains a total of four mini-subgraphs, which differ only in the locations of the two incoming edges (and their weight) and the outgoing edge. For each index $i \in \{i_s, \ldots, i_s+21\}$, the mini-subgraph includes $2^{d'-1}$ nodes: one node for each combination of last $d'-1$ bits of the representative symbols in index i. A single outgoing edge connects the mini-subgraph relevant node ${}^0_1 01$ in index i_s+21 to t_j (the other nodes in index i_s+21 can be erased from the mini-subgraph). Two incoming edges (for $D_0^{-1}(w)$ and $D_1^{-1}(w)$) connect s_j to relevant nodes in index i_s, which in our example are $D_0^{-1}(01)=001$ and $D_1^{-1}(01)=110$ (the nodes ${}^0_1 00$ and ${}^0_1 11$ in index i_s can be erased from the mini-subgraph). Thus, any path that goes through the mini-subgraph must include one of these incoming edges and the outgoing edge. This fact ensures that each path corresponds to a consistent choice of representative symbols (as discussed at the end of Section 4.3).

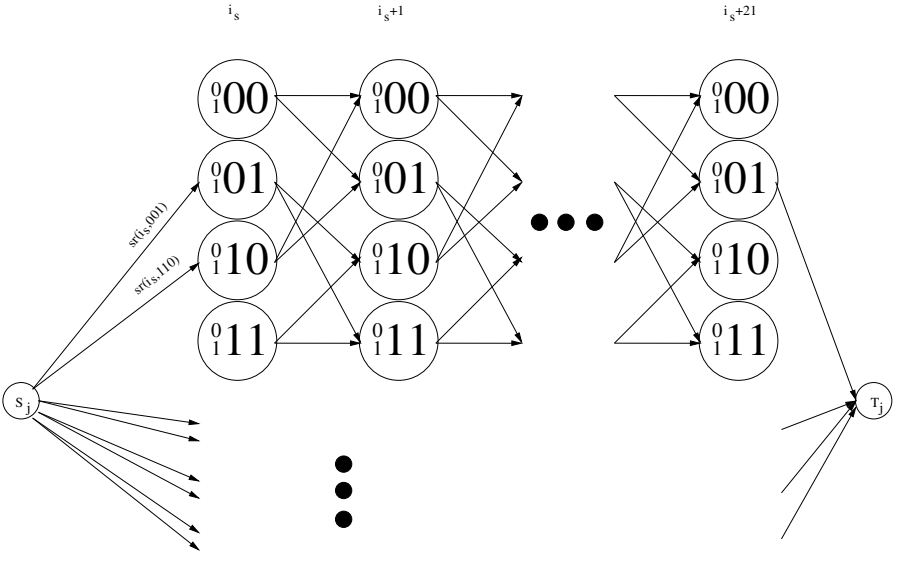


Fig. 4. The subgraph for the j^{th} candidate value of S_1

Consistent transitions between representative symbols in adjacent indices are modeled by edges that connect nodes of adjacent indices (in a way that reminds a de-Bruijn graph). There is an edge from a first node to a second node if and only if the last $d'-1$ bits of the first node are the same as the first $d'-1$ bits of the second node, which is the requirement for consistent choice of representative symbols. For example, a transition between a representative symbol $a_0a_1\dots a_{d'-1}$ in index i and a representative symbol $a_1a_2\dots a_{d'}$ in index $i+1$ is modeled by an edge from node ${}^0_1a_1\dots a_{d'-1}$ to node ${}^0_1a_2\dots a_{d'}$. The cost of the edge is $sr(i+1,a_1a_2\dots a_{d'}) \triangleq cost'(i+1,a_1a_2\dots a_{d'})$, where $cost'$ is folded using the folding property from $cost(i,x) \triangleq func_{i,s_1}[x] \triangleq \sum_{l_1} E_{l_1,i}[s'_1[l_1]\oplus x]$, as described in Section 4.3, and s'_1 is fixed for the given subgraph.

The total cost of edges along a path is $\sum_i func_{i,s_1}[s'_2[i]] = \sum_{l_1,i} E_{l_1,i}[s'_1[l_1]\oplus s'_2[i]] = score(s_1,s_2)$, where s'_2 is the candidate for S'_2 that is implied by the path, and s'_1 is the appropriate value for the j^{th} candidate for S_1. After a quick precomputation, the value of $func_{i,s_1}[x]$ can be calculated using a few table lookups regardless of the value of s_1.

5.1.2 The Structure of the Sub-graph Using Conditional Estimators

Similarly to the case of non-conditional estimators, in case conditional estimators are used, the subgraph for candidate j has a source node s_j, a target node t_j, and the subgraph is composed of several mini-subgraphs, which differ only in the location of the incoming edges (and their cost) and the location of the outgoing edge. However, with conditional estimators, the structure of the mini-subgraphs is different: each pair of indices $i, i+11$ are unified to a single index, denoted by $i|i+11$.

We would like to combine the nodes in index i with nodes in index $i+11$ by computing their cartesian product: for each node a in index i and for each node b in index $i+11$, we form the unified node $a|b$ in unified index $i|i+11$. However, there is a technical difficulty: while (given S_1) a non-conditional estimator depends on a symbol candidate $s'_2[i]$, a conditional estimator depends on both a symbol candidate $s'_2[i]$ and a clock symbol candidate $s_2[i+11]$. As a result, we must apply the D^{-1} operator on nodes in index $i+11$ (to transform them from symbols to clock symbols). This operation divides node $b = {}^0_1 b_1b_2\dots b_{d'-1}$ in index $i+11$ into two nodes ${}^0_1D_0^{-1}(b_1b_2\dots b_{d'-1})$ and ${}^0_1D_1^{-1}(b_1b_2\dots b_{d'-1})$. Only then, we can perform the cartesian product between the nodes in index i and the nodes that results from applying D^{-1}. Thus, from a pair of a and b of the above form, we have two nodes in the product (in index $i|i+11$): $a|{}^0_1D_0^{-1}(b_1b_2\dots b_{d'-1})$ and $a|{}^0_1D_1^{-1}(b_1b_2\dots b_{d'-1})$. We refer to the bits on the left of the "|" in the node as symbol bits, and the bits on the right of the "|" as clock bits. In total, there are $2^{d'-1}(2\cdot 2^{d'-1}) = 2^{2d'-1}$ nodes in each index $i|i+11$.

There is an edge from node $x_1|y_1$ in index $i|i+11$ to node $x_2|y_2$ in index $i+1|i+12$ if and only if the last $d'-1$ bits of x_1 are equal to the first $d'-1$ bits of x_2 and the last d' bits of y_1 are equal to the first d' bits of y_2. Figure 5 depicts four nodes of a mini-subgraph using conditional estimators.

What should be the cost of an edge? the basic cost function is $cost(i,x|y) \triangleq func_{i,s_1}[x|y] \triangleq \sum_{l_1} E_{l_1,i}[s'_1[l_1]\oplus x|s_1[l_1+10]\oplus y]$, which is folded to the cost

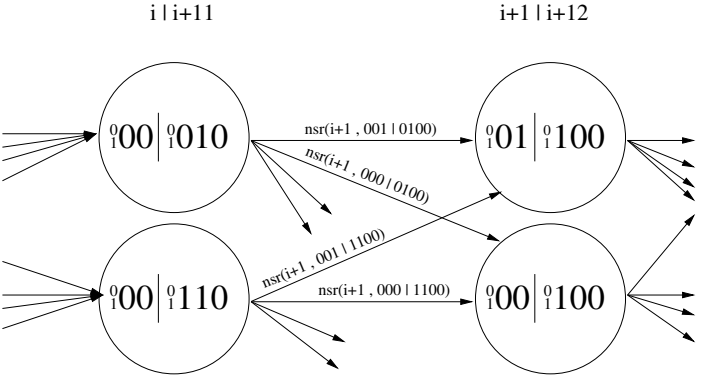


Fig. 5. Four nodes of the mini-subgraph using conditional estimators for $d' = 3$

function $cost'(i, x|y)$. Since each index $i|i+11$ unifies two indices, the edge that enters $i|i+11$ should contain the sum of contribution of indices i and $i+11$, i.e., the cost of the edge is $nsr(i, s_2'[i]|s_2[i+11]) \triangleq cost'(i, s_2'[i]||\mathrm{lsb}_{d'}(s_2[i+11])) + cost'(i+11, s_2'[i+11]|s_2[i+22])$, where $\mathrm{lsb}_{d'}(x)$ returns the d' first bits of x. Note that $s_2'[i+11] = D(s_2[i+11])$, and (due to the alignment property) $s_2[i+22] = s_2'[i]$. Therefore, $nsr(i, s_2'[i]|s_2[i+11]) = cost'(i, s_2'[i]||\mathrm{lsb}_{d'}(s_2[i+11])) + cost'(i+11, D(s_2[i+11])|s_2'[i])$.

Like the case of non-conditional estimators, we create several mini-subgraphs to ensure that the paths in the subgraph represent consistent choices for S_1 and S_2. We include in the subgraph a mini-subgraph for each combination v of the last $d'-1$ symbol bits and each combination w of the last d' clock bits of the last node (the node near t_j). A single edge (with cost zero) connects the mini-subgraph to t_j from node ${}^0_1v|^0_1w$. For consistency with the linear feedback, the bits w must be identical to the symbol bits of the first node (both w and the first symbol bits are d'-bit long). The bits v must be identical to the difference of the first d' bits of the first clock symbol. As v is $(d'-1)$-bit long, and as the clock bits of the first symbol are $(d'+1)$-bit long, there are four possibilities for the clock bits: $D_0^{-1}(v)||0$, $D_1^{-1}(v)||0$, $D_0^{-1}(v)||1$, and $D_1^{-1}(v)||1$. Therefore, four edges $w|D_0^{-1}(v)0$, $w|D_1^{-1}(v)0$, $w|D_0^{-1}(v)1$, and $w|D_1^{-1}(v)1$ connect s_j to the mini-subgraph (the concatenation mark "||" was removed for clarity). Their costs are $nsr(i_s, w|D_0^{-1}(v)0)$, $nsr(i_s, w|D_1^{-1}(v)0)$, $nsr(i_s, w|D_0^{-1}(v)1)$, and $nsr(i_s, w|D_1^{-1}(v)1)$, respectively.

To reconstruct s_2' from a path in the mini-subgraph, we first concatenate the symbol bits to form the first half of the path, and separately concatenate the clock bits to form the second half of the path. Then, we compute the difference between the clock bits, and combine the result with the symbol bits to obtain a path of s_2' (similar to the path in the case of the mini-subgraph using unconditional estimators).

Note that in an efficient implementation there is no need to keep the entire graph in memory, and needed parts of the graph can be reconstructed on-the-fly.

6 Simulations of Our Attacks

We have implemented our attack, and simulated it under various parameters. Our simulations focus on 2000 frames of data, which is the lowest amount of data that gives a non-negligible success rate in the simulations of Maximov, Johansson, and Babbage [11]. We also simulated the attack with 1500 frames. A comparison of simulations of previous attacks and simulations of our new attacks is given in Table 1.

In the simulations we use $d = 1$, $l_1 \in \{61, \ldots, 144\}$, $l_2 \in \{70, \ldots, 135\}$, and calculate estimators for $\mid l_1 - l_2 \mid < 10$. We use the first version of Step 3 with 64-bit keys.

We ran the simulations on a 1.8GHz Pentium-4 Mobile CPU with 512MB of RAM. The operating system was Cygwin under Windows XP. In comparison, the simulations of [11] were performed on a 2.4GHz Pentium-4 CPU with 256MB of RAM under Windows XP, and the simulations of [7] were performed on a 1.8GHz Pentium-4 CPU with 512MB of RAM under Linux.

In one simulation, we limited the size of the list of top (s_1, s_2) pairs to 5200. The key was found in about 64 percent of the cases, compared to about 5 percent in previous attacks with 2000 frames. Our attack takes about 7 seconds to complete Step 1. Step 2 takes about 340 seconds for the first pair, after which it can generate about 1500 pairs of candidates per second. Step 3 scans about 20.4 candidate pairs per second. Therefore, the total time complexity varies depending on the location of the correct pair in the list. It takes about 350 seconds (six minutes) in the best case, and up to ten minutes in the worst case.

For better results, we employ two methods: *early filtering* and *improved estimators*.

6.1 Early Filtering

In early filtering, we perform Step 2 several times, using less accurate (and faster) methods. Thus, we discard many candidate values of S_1 that are highly unlikely, and we do not need to build a subgraph for these values. For example, we score all the candidates of S_1 (a score of a candidate s_1 of S_1 is $\max_{s_2} score(s_1, s_2)$) using non-conditional estimators and a less accurate but faster method. Then, we recalculate the score for the 220000 top candidates, using a similar method, but with conditional estimators. The 40000 top scored candidates are re-scored using conditional estimators with a variation using only one mini-subgraph. Finally, we perform Step 2 of Section 5.1 with subgraphs only for the 2000 scored candidates of S_1. The list of the 5200 top candidates of S_1 and S_2 is generated and passed to Step 3. We denote this kind of configuration in a tuple $(220000, 40000, 2000, 5200)$. Simulation results using other configurations for both 2000 and 1500 frames are given in Table 1.

6.2 Improved Estimators

A disadvantage of the described attack is that only information from the estimators $E_{l_1,l_2}[\cdot|\cdot]$ is taken into consideration, while estimators involving $R3$, i.e.,

$E_{l_1,l_3}[\cdot|\cdot]$ and $E_{l_2,l_3}[\cdot|\cdot]$, are disregarded. In *improved estimators*, we improve our results by adding to each estimator $E_{l_1,l_2}[x|y]$ the contributions of the estimators of the other registers, i.e., we add to it

$$\sum_{l_3} \log \left(\sum_{\alpha,\beta \in \{0,1\}^d} e^{E_{l_1,l_3}[\alpha|\beta]+E_{l_2,l_3}[x\oplus\alpha|y\oplus\beta]} \right).$$

The resulting estimators include more information, and thus, are more accurate. They significantly improve the success rate with a modest increase in the time complexity of Step 1 (mostly, since we need to calculate three times the number of estimators). This increase in time complexity is compensated by a large decrease in the time complexity of Step 3 (as the correct S_1, S_2 are found earlier). The results are summarized in Table 1.

7 New Source for Known-Keystream

Every traffic channel between the handset and the network is accompanied by a slower control channel, which is referred to as the Slow Associated Control CHannel (SACCH). The mobile uses the SACCH channel (on the uplink) to report its reception of adjacent cells. The network uses this channel (on the downlink) to send (general) system messages to the mobile, as well as to control the power and timing of the current conversation.

The contents of the downlink SACCH can be inferred by passive eavesdropping: The network sends power-control commands to the mobile. These commands can be inferred from the transmission power of the mobile. The timing information that the network commands the mobile can be inferred from the transmission timing of the mobile. The other contents of the SACCH is a cyclical transmission of 2–4 "system messages" (see [8, Section 3.4.1]). These messages can be obtained from several sources, for example by passively eavesdropping the downlink at the beginning of a call (as the messages are not encrypted at the beginning of a call), or by actively initiating a conversation with the network using another mobile and recover these messages (these messages are identical for all mobiles). There is no retransmission of messages on the SACCH, which makes the task of the attacker easier, however, it should be noted that an SMS received during an on-going conversation could disrupt the eavesdropper, as the SMS can be transferred on the SACCH, when system messages are expected.

An attacker would still need to cope with the Frequency Hoping (FH) used by GSM. Using a frequency analyzer the attacker can find the list of n frequencies that the conversation hops on. Given n, GSM defines only $64n$ hopping sequences (n cannot be large since the total number of frequencies in GSM is only about 1000, of which only 124 belong to GSM 900). Thus, the hopping sequence can be determined through a quick exhaustive search.

As the name of SACCH implies, it is a slow channel. Only about eight frames are transmitted every second in each direction of the channel. Therefore, to collect 1500–2000 SACCH frames transmitted from the network to mobile, about 3–4 minutes of conversation are needed.

8 Summary

Our contribution in this paper is multi-faced. We begin by introducing conditional estimators that increase the bias of the correlation equation. Then, we present three weaknesses in $R2$, which were not reported previously. The first weakness — the alignment property — utilizes the fact that the correlation equation coincides with the feedback taps of $R2$. The second weakness — the folding property — uses the fact that $R2$ has only two feedback taps, and they are adjacent. We use the folding property to decode the estimators in an optimal way. In contrast, previous attacks were forced to use heuristics to decode the estimators. Using this weakness, we present a novel method to efficiently calculate the list of best candidate pairs for S_1 and S_2. Given S_1 and S_2, the value S_3 can be worked back from the keystream.

The last weakness that we report — the symmetry property — is based on the fact that $R2$'s clocking tap is exactly in its middle, which together with the folding property causes a symmetry between the clocking tap and the output of $R2$. This property enables us to efficiently decode the *conditional* estimators.

Finally, we describe a new source for known-plaintext in GSM. This source of known-plaintext transforms our attack to a practical ciphertext-only attack. With 3–4 minutes of raw ciphertext, we can extract (from the SACCH) the required amount of about 1500–2000 frames of known-plaintext.

We compare some of the previous results and our current simulation results in Table 1. Compared to previous attacks on 1500–2000 frames, it can be seen that our new attack has a significantly higher success rate (91% compared to 5%), it is faster, and it does not require any precomputation.

Acknowledgments

We are pleased to thank Alexander Maximov for providing early versions of [11].

References

1. Ross J. Anderson, *On Fibonacci Keystream Generators*, proceedings of Fast Software Encryption: Second International Workshop, Lecture Notes in Computer Science 1008, Springer-Verlag, pp. 346–352, 1995.
2. Edsger W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik, Vol. 1, pp. 269–271, 1959.
3. Elad Barkan, Eli Biham, Nathan Keller, *Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communications*, Advances in Cryptology, proceedings of Crypto'03, Lecture Notes in Computer Science 2729, Springer-Verlag, pp. 600–616, 2003.
4. Eli Biham, Orr Dunkelman, *Cryptanalysis of the A5/1 GSM Stream Cipher*, Progress in Cryptology, proceedings of Indocrypt'00, Lecture Notes in Computer Science 1977, Springer-Verlag, pp. 43–51, 2000.
5. Alex Biryukov, Adi Shamir, David Wagner, *Real Time Cryptanalysis of A5/1 on a PC*, Advances in Cryptology, proceedings of Fast Software Encryption'00, Lecture Notes in Computer Science 1978, Springer-Verlag, pp. 1–18, 2001.

6. Marc Briceno, Ian Goldberg, David Wagner, *A pedagogical implementation of the GSM A5/1 and A5/2 "voice privacy" encryption algorithms*, `http://cryptome.org/gsm-a512.htm` (originally on www.scard.org), 1999.
7. Patrik Ekdahl, Thomas Johansson, *Another Attack on A5/1*, IEEE Transactions on Information Theory, Volume 49, Issue 1, pp. 284-289, 2003.
8. European Telecommunications Standards Institute (ETSI), *Digital cellular telecommunications system (Phase 2+); Mobile radio interface; Layer 3 specification*, TS 100 940 (GSM 04.08), `http://www.etsi.org`.
9. Jovan Golic, *Cryptanalysis of Alleged A5 Stream Cipher*, Advances in Cryptology, proceedings of Eurocrypt'97, LNCS 1233, pp. 239–255, Springer-Verlag, 1997.
10. Walter Hoffman, Richard Pavley, *A Method for the Solution of the Nth Best Path Problem*, Journal of the ACM (JACM), Volume 6, Issue 4, pp. 506–514, 1959.
11. Alexander Maximov, Thomas Johansson, Steve Babbage, *An improved correlation attack on A5/1*, proceedings of SAC'04, LNCS 3357, pp. 1–18, Springer-Verlag, 2005.
12. Willi Meier, Othmar Staffelbach, *Fast Correlation Attacks on Certain Stream Ciphers*, Journal of Cryptology, Volume 1, Issue 3, pp. 159–176, Springer-Verlag, 1989.
13. Thomas Siegenthaler, *Decrypting a Class of Stream Ciphers Using Ciphertext Only*, IEEE Transactions on Computers, Volume 49, Issue 1, pp. 81–85, 1985.

Cryptanalysis of the F-FCSR Stream Cipher Family

Éliane Jaulmes and Frédéric Muller

DCSSI Crypto Lab,
51, boulevard de La Tour-Maubourg,
75700 Paris-07 SP
{Eliane.Jaulmes, Frederic.Muller}@sgdn.pm.gouv.fr

Abstract. This paper focuses on F-FCSR, a new family of stream ciphers proposed by Arnault and Berger at FSE 2005. It uses a non-linear primitive called the Feedback with Carry Shift Register (FCSR) as a building block. Its security relies on some properties of the 2-adic numbers. The F-FCSR family contains several stream ciphers, each of them proposing different features.

First, we show a resynchronization attack that breaks algorithms in the family that support initialization vectors. The attack requires at most 2^{16} chosen IV's and a little offline processing to recover the full secret key. We have implemented it with success on a standard PC.

Secondly, we show a time/memory/data trade-off attack which breaks several algorithms in the F-FCSR family, even when initialization vectors are not supported. Its complexity ranges from 2^{64} to 2^{80} operations (depending on which algorithm in the family we consider), while the internal state has size 196 bits at least. Therefore this attack is better than generic attacks.

Keywords: FCSR, Time/memory/data trade-off, stream cipher, resynchronization attack.

1 Introduction

Stream ciphers are a special class of secret key cryptosystems. Their principle is to generate a long pseudo-random sequence which is then XORed bitwise to the message in order to produce the ciphertext. Thus, compared to a block cipher, a secure stream cipher is expected to be much faster.

Yet the design of secure stream ciphers also seems to be more difficult. Indeed, over the years, few proposals have withstood cryptanalysis. Many of the attacks staged over stream ciphers exploit the mathematical structure of Linear Feedback Shift Registers (LFSR) which are often used as building blocks. To avoid these pitfalls, alternative constructions have been proposed recently. For instance, it has been suggested to use irregularly-clocked registers or efficient non-linear mapping. One of theses suggestions is to replace LFSR by Feedback with Carry Shift Registers (FCSR). A FCSR is a binary register, similar to a LFSR, except

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 20–35, 2006.

that it performs operations with carries. This brings non-linearity, which is an interesting property to thwart algebraic attacks [8] or correlation attacks [19, 20]. The idea of using FCSR in cryptography was originally proposed by Klapper *et al.* in 1994 [16], but it was shown that they are not secure when used alone [17]. FCSR came back into flavor with recent works by Arnault and Berger [2, 3]. Some of their proposals were broken in 2004 [21]. Later, at FSE 2005, they proposed a concrete family of stream ciphers based on FCSR. It is referred to as the **F-FCSR family** and several variants are suggested. In this paper, we investigate the security of this new family of stream ciphers.

The paper is constructed as follows: in a first section, we recall the principle of the FCSR primitive and describe the proposals of [3].

Secondly, we describe resynchronization attacks against all variants that support Initialization Vectors (IV). We describe how to learn some information about the secret key, by comparing the keystream generated with two related IV's. We manage to recover the full secret key with about 2^{15} pairs of chosen IV's and little offline processing. These attacks have been implemented and take time ranging from a few seconds to a few hours on a standard PC.

Finally, we describe a time/memory/data trade-off against some algorithms in the family. Since the internal state has a size of $n = 196$ bits, the best generic attack is expected to cost about $2^{n/2} = 2^{98}$ computation steps. However we show that the real entropy is only of 128 bits, due to the cycle structure of the state-update function. Using this property, we attack some of the algorithms with time, memory and data of the order of 2^{64}.

2 Stream Ciphers Based on FCSR

2.1 The FCSR Primitive

Feedback with Carry Shift Registers were introduced by Goresky and Klapper [16]. The underlying theory is related to the 2-adic fractions and more details can be found in [3]. Here, we simply recall the main characteristics.

Let q be a negative integer such that $-q$ is prime and let $0 \leq p < -q$. Let $q = 1 - 2d$. We write $d = \sum_{i=0}^{k-1} d_i 2^i$.

The FCSR generator with feedback prime q and initial value p produces the 2-adic expression of the fraction p/q. This corresponds to the infinite sequence of bits $a_i \in \{0, 1\}$, where $p = q \cdot \sum_{i=0}^{\infty} a_i 2^i$.

This sequence has good statistical properties (relative to its period in particular), provided q is prime and 2 is of order $|q| - 1$ modulo q. The sequence of a_i can also be computed in an iterative way through the sequence of integers p_i, defined as:

$$p_0 = p \ , \tag{1}$$

$$a_i = p_i \bmod 2 \quad \text{and} \quad p_{i+1} = \tfrac{p_i - q a_i}{2} \equiv \tfrac{p_i}{2} \bmod q \ . \tag{2}$$

It is easy to verify that $\forall i, 0 \leq p_i < -q$. Also, the following relation holds $\forall i \geq 0$, $\dfrac{p_i}{q} = \displaystyle\sum_{j \geq i} a_j 2^{j-i}$.

The sequence of integers a_i can be obtained in a register-oriented fashion. Consider:

- a main register M with k binary memory cells,
- a carry register C with $\ell - 1$ binary memory cells, where ℓ is the Hamming weight of d. The set $I_d = \{i | 0 \leq i \leq k-2 \text{ and } d_i = 1\}$ denotes the positions of these cells.

The main register M is said to **contain** the integer $m = \sum_{i=0}^{k-1} m_i 2^i$ when the values $(m_0 \dots m_{k-1})$ appear in the k cells of the register. Similarly, the carry register is said to **contain** the integer $c = \sum_{i=0}^{k-1} c_i 2^i$ if, for $i \notin I_d$, $c_i = 0$ and, for $i \in I_d$, c_i appears in the corresponding carry cell. We say that, at time t, the FCSR is in state $(m(t), c(t))$ if the main register contains the value $m(t)$ and the carry register contains $c(t)$. The state $(m(t+1), c(t+1))$ is computed from the state $(m(t), c(t))$ according to the following equations:

- For $0 \leq i \leq k-2$ and $i \notin I_d$
  $m_i(t+1) = m_{i+1}(t)$
- For $0 \leq i \leq k-2$ and $i \in I_d$
  $m_i(t+1) = m_{i+1}(t) \oplus c_i(t) \oplus m_0(t)$
  $c_i(t+1) = m_{i+1}(t)c_i(t) \oplus c_i(t)m_0(t) \oplus m_0(t)m_{i+1}(t)$
- For $i = k-1$
  $m_i(t+1) = m_0(t)$

A small example of this register representation with $q = -347$ appears in the Figure 1.

Let the initial state be given by $m(0) = p$ and $c(0) = 0$. The next state $(m(t+1), c(t+1))$ is computed from $(m(t), c(t))$ by looking at the least significant bit $m_0(t)$ of M. The main register is always right-shifted, which corresponds to a division by 2. The content of C is added to the result, as well as the number d when $m_0(t) = 1$. Thus,

$$\begin{aligned} m(t+1) + 2c(t+1) &= \frac{m(t) - m_0(t)}{2} + c(t) + dm_0(t) \\ &= \frac{m(t) + 2c(t) - qm_0(t)}{2} \end{aligned}$$

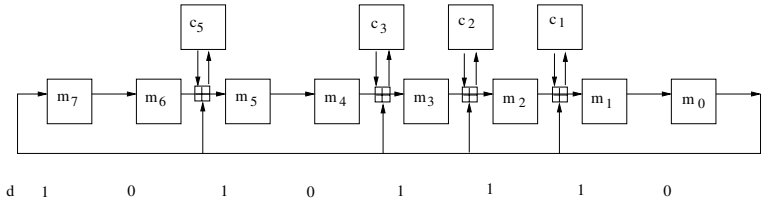


Fig. 1. Example of FCSR

Hence the following relations are always satisfied:

$$a_t = m_0(t) \quad \text{and} \quad p_t = m(t) + 2c(t).$$

So the register-oriented representation produces indeed the 2-adic representation of the fraction p/q through the bit $m_0(t)$ - generally called the **feedback bit**.

2.2 The F-FCSR Family

Stream ciphers based on FCSR were already proposed in the past [2, 16]. However it was shown that, despite the non-linearity, it is insecure to output directly bits from the FCSR sequence. Efficient algorithms [17] have been proposed to retrieve the initial state: when q is of size 128 bits, knowing 128 output bits is sufficient.

Arnault and Berger suggested [3] to apply a filtering function to the FCSR state, in order to produce output bits. They argued that a linear filter is sufficient, since the FCSR primitive is already non-linear by itself. At FSE 2005, they presented a new family of stream ciphers based on this general idea. The following parameters are chosen for all algorithms in the family: the main register has a size $k = 128$ bits and the feedback prime is

$$-q = 49387740064344360888838204820078394382 7.$$

This choice guarantees the maximal period, because the order of 2 modulo q is equal to $|q| - 1$. The Hamming weight of $d = \frac{1-q}{2}$ is $\ell = 69$ so the carry register has $\ell - 1 = 68$ memory cells. The secret key of the algorithm (of size 128 bits) is the initial value p introduced in the main register.

Differences between the four proposed algorithms lie in the nature of the output function. Two distinctions are made depending on

- the secrecy of the output taps. Two proposals use a fixed linear combination of the FCSR state. The other two use a dynamic output function (*i.e.* the taps are generated from the secret key, and are therefore unknown to an attacker).
- the number of output bits per advance. The basic construction produces 1 bit at each advance of the FCSR. To improve the encryption speed, it is possible to use 8 filtering functions in order to produce 8 output bits per advance.

2.3 Description of the Proposals

The first proposal is called **F-FCSR-SF1**. SF stands for "Static Filter". The stream cipher is based on the FCSR described above. It produces one single output bit at each advance of the FCSR. This output bit is computed through a known linear filter function, called F, of the form:

$$f(m_0, \ldots, m_{k-1}) = \bigoplus_{i=0}^{k-1} f_i m_i,$$

where $f_i \in \{0,1\}$. We call f the number $\sum_{i=0}^{k-1} f_i 2^i$. In their paper, the authors suggested to use the filter $f = d$. With this choice, the output bit of the stream cipher is the XOR of all the main register cells located just after a carry cell.

The second proposal is called **F-FCSR-SF8**. The same FCSR is used but eight output bits are produced at each advance. This is done quite naturally by using eight different filters $(F_i)_{1 \leq i \leq 8}$. The filters must be linearly independent and respect other conditions detailed in [3]. In order to have a simple extraction function for the 8 output bits, the authors suggest to use 8 filters with disjoint supports. More precisely, they recommend filters F_i such that

$$\forall i \in [1,8],\ \mathrm{Supp}(F_i) \subset \{j | j \equiv i \bmod 8\}. \tag{3}$$

The filters $(F_i)_{1 \leq i \leq 8}$ are public and part of the stream design.

The third proposal is called **F-FCSR-DF1**. DF stands for "Dynamic Filter". It works exactly as **F-FCSR-SF1**, except that the output filter F is kept secret and derived from the key through an invertible function g. This artificially increases the size of the cipher state from $196 = 128 + 68$ to 324 bits.

The fourth proposal is called **F-FCSR-DF8**. It works exactly as **F-FCSR-SF8**, except that the eight filters are kept secret and derived from the key. Here also the filters verify the condition 3. Since the filters have disjoint supports, a single invertible function g suffices to define the eight filters.

2.4 Initialization Vectors

Since stream ciphers produce bit sequences independently of the messages they encrypt, it is customary to add an initialization vector (IV), that allows the sequence to change from one encryption to the next. The IV is usually a public value, transmitted alongside the ciphertext.

Support for IV is often impossible to avoid: applications deal with relatively small messages (frames or data packets), which makes it highly inefficient to rekey the cipher for each message. Using one long keystream sequence raises important problems of synchronization. Therefore all new stream ciphers are expected to support IV's. For example, this is a requirement in the call for stream ciphers published recently by the european project ECRYPT [10].

In the F-FCSR family, the authors propose to **use the initial content of the carry register as the initialization vector**. Six advances are made before the encryption starts, to guarantee the diffusion of the IV in the initial state.

There is a slight problem of dimension, not solved in the FSE paper: the carry register has length 68 bits which is not a convenient IV size (64 bits would be better for instance). In the later, we assume that the IV has length 68 bits. We claim that variants of our attack could be envisaged even if another IV dimension (64 or 128 bits) was used and it was somehow mapped to the carry register state.

2.5 Resynchronization Attacks

Building a good initialization mechanism for a stream cipher is not an easy task. Indeed, the IV is a public value introduced inside the secret state of the cipher.

By looking at the link between the IV and the first keystream bits, some secret information may be leaked. This family of attacks has first been called "resynchronization attacks" [9], but its spectrum of applications has broaden [1, 13]. Practical applications have been shown for the 802.11 standard [18] or for the GSM standard [6].

Most attacks require only **known IV** (which is always the case since the IV is transmitted in clear). However, there are situations where we can envisage **chosen IV**. Firstly, an active attacker may modify the IV while it is transmitted on the communication channel. Secondly, the IV is often generated by an unspecified random generator, which might be partially controlled by the attacker.

Our attacks require pairs of related IV's which typically differ by only one bit. Since many implementations use a counter, such low-weight differences are likely to appear rapidly. Such chosen IV attacks often turn up to be more practical than expected (see the example of RC4 in the 802.11 standard [18]).

3 Resynchronization Attack with Static Filter

In this section, we describe an attack against the two proposals which use a static filter. We focus on **F-FCSR-SF1** but clearly the same attack applies to **F-FCSR-SF8**.

3.1 Principle of the Attack

Our basic observation is that the 6 initial advances are not sufficient for all cells of the main register to depend on the whole IV. Suppose we flip only one bit of the IV (*i.e.* one bit in the initial state of the carry register), then after 6 advances, only a few cells in the main register may be affected. Our idea is to predict the difference on the first keystream bit. To do that, only a small number of key bits need to be guessed.

The initial state of the main register is just the key $m(0) = K$. Similarly, the initial state of the carry register is denoted by $c(0)$ and is just equal to $c(0) = IV$. After 6 advances, the state of the carry register is $c(6)$ and the state of the main register is $m(6)$. The first keystream bit $z(0)$ is given by $z(0) = \bigoplus_{i=0}^{127} f_i m_i(6)$, where the output filter $f = \sum_i f_i 2^i$ is known.

Let $i \in I_d$ be a position where a carry cell is present. Suppose we initially replace $c_i(0)$ by $c_i(0) \oplus 1$. We are interested in how the first keystream bit $z(0)$ is affected by this modification[1]. Note that this difference propagates through the main register up to the end, one cell at a time, and will not disappear. Thus, after $n \leq i + 1$ advances of the FCSR, there will be a difference in the cell $i - n + 1$. Due to the carries, the difference may also linger on previous cells, but the probability of a difference staying for a long time is low.

If $i \geq 5$, then, only the bits $m_i(6), \ldots, m_{i-5}(6)$ of the main register may be affected, after the 6 initial advances, by the initial flip. Bit $m_{i-5}(6)$ is always flipped, and for the other bits, it depends on the initial state. This propagation

[1] We assume that $i \geq 5$, so the first 6 feedback bits are not affected.

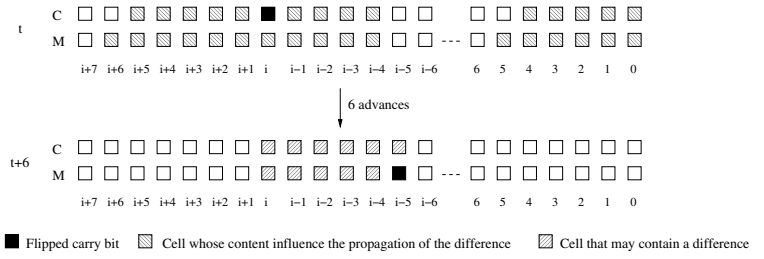


Fig. 2. Influence of state bits on the difference propagation

is illustrated in Figure 2. If $i < 5$, then the feedback bit is affected during the 6 initial advances and many bits in the main register may be affected. We want to avoid this case.

Therefore, in order to predict the difference on the first keystream bit $z(0)$, it is sufficient to know:

- the first 6 feedback bits, which are fully determined by key bits $m_0(0), \ldots, m_5(0)$ and IV bits $c_j(0), j < 5$ and $j \in I_d$,
- the "neighborhood" of cell i, *i.e.* key bits $m_{i-4}(0), \ldots, m_{i+6}(0)$ and IV bits $c_j(0), i-4 \leq j \leq i+5$ and $j \in I_d$.

The initial state of the carry register is known. So all we need to guess is $6+11 = 17$ key bits, in order to predict the propagation of the initial difference during 6 rounds, and therefore the difference on the first keystream bit $z(0)$.

If we guess correctly the 17 bits, we will observe the predicted difference. But observing the difference on one keystream bit do not allow us to deduce the correct guess among the 2^{17} possibilities. We are only able to eliminate roughly half of the candidates. So, this experiment provides roughly **one bit of information about the key**. However, we may hope to find the right candidate by iterating the process 17 times.

3.2 Limitations

In order to find the right guess on 17 key bits, we could think of using 17 independent experiments. This can be achieved by randomizing the IV. As mentioned above, the IV bits that are significant for the differential behavior are those located at positions 0 to 4 and $i-4$ to $i+5$. The exact number of bits that we can randomize depends on the set I_d, but we observed that between 6 and 10 bits are usually available for all choices of i. So we expect to get between 2^6 and 2^{10} possible experiments. In theory, this is enough to find the right guess.

Unfortunately, this does not work in practice. It is generally not possible to identify the correct guess among the 2^{17} candidates. As an illustration, suppose that the output taps satisfy $f_j = 0$ for $j = i-4, \ldots, i$. Then the only output tap in the "influence window" is at position $i-5$. The difference on the first keystream bit is therefore always 1, for all values of the 17 key bits. In this case, no information can be obtained by randomizing the IV.

Of course, this is the least favorable case. In most cases, we observed that many candidates for the 17 key bits can be discarded using the previous differential conditions. However, a small number of candidates may remain.

3.3 Key-Recovery Algorithm

To overcome these difficulties, we propose a "sliding window" algorithm. After considering a position i, a small number of candidates for 17 bits of the key remains. Next, we examine position $i-1$, and guess a few extra key bits in order to repeat the same attack[2]. From pairs of chosen IV's, we obtain new conditions which allow us to eliminate more candidates, and so on. Our goal is to keep the number of candidates as low as possible alongside the execution of the attack.

The resulting algorithm may be described as follows:

- Guess the 6 rightmost bits $(m_0(0), \ldots, m_5(0))$.
- Guess the 6 leftmost bits $(m_{122}(0), \ldots, m_{127}(0))$.
- For i from 120 down to 5 do:
  - Guess bit $m_{i+1}(0)$.
  - If $(i+5) \in I_d$ (a carry cell is present at the current position) do:
    - Flip the corresponding IV bit, and do as much experiments as possible (depending on the number of carry bits that are able to influence the output).
    - Discard guesses that are not compatible with the observed difference on the first keystream bit.
- Output the remaining correct guesses and test each of them to find the secret key.

When no carry bit is present at position i (case where $i \notin I_d$), we cannot eliminate any candidate, so the number of guesses to examine grows by a factor of 2. This can be quite inconvenient, so we propose a simple improvement. We flip the bit $c_{i+6}(0)$ (instead of $c_{i+5}(0)$) and look at the second keystream bit $z(1)$ (instead of $z(0)$). This is roughly the same idea as before with 7 advances instead of 6. Some technical details need to be fixed (for instance, we need to predict one more feedback bit), but the idea remains the same. And so on with more advances.

With this improvement, we can obtain conditions at every position (instead of only the positions $i \in I_d$) for little extra cost. This keeps the number of candidates low and makes the attack feasible.

3.4 Efficiency and Results

We implemented this attack on a standard PC, and observed the behavior of the previous algorithm on several randomly chosen keys. At each stage, the number of possible solutions never climbed above the starting point of 2^{13}. The set of

[2] Because of the overlapping between the two sets, only one new bit needs to be guessed.

solutions must be explored for each value of index i. Thus, the time complexity of the algorithm is about $128 \times 2^{13} \simeq 2^{20}$. This represents a few seconds on a PC.

With the selected value of q, there is an average of 2^8 experiments for each position, which means the number of possible experiments is about $128 \times 2^8 \simeq 2^{15}$. This means we need to process about 2^{15} pairs of chosen IVs in order to recover the full secret key. For each pair, we are only interested in learning the differential on one keystream bit. Thus the attack is very efficient and does not require a powerful adversary.

We also mention that some trade-offs are possible: if less than 2^{15} pairs of IV are available, we can stop the previous algorithm at any time, when only n bits from the key have been guessed. Then we can go through all the remaining correct guesses in the algorithm (at most 2^{13}) and also guess the remaining $128 - n$ key bits.

3.5 Adapting the Attack to F-FCSR-SF8

The eight filters chosen to produce the eight output bits in the **F-FCSR-SF8** version of the stream cipher have disjoint support. If we XOR the eight output bits, we obtain exactly the one bit of output we had in the previous attack. Thus, at worse, **F-FCSR-SF8** can be attacked with the same complexity as **F-FCSR-SF1**.

Moreover, it is also possible to use the extra information : 8 bits of information about the internal state are outputted at a time. This allows us to reject more candidates at each stage. We expect to discard everything but the correct guess at each step, which would decrease the time complexity of the attack to about 2^{16} as the total number of IV's to process.

4 Resynchronization Attacks with Dynamic Filter

In this section, we describe an attack against the two proposals using a dynamic filter. We will focus on **F-FCSR-DF1** for our description of the attack but it applies similarly on **F-FCSR-DF8**. As in Section 3, the attack is a chosen IV attack and recovers the secret key with only 2^{16} IV's.

4.1 Principle of the Attack

In the **F-FCSR-DF1** version of the stream cipher, the filter function is unknown of the adversary. It is derived from the secret key through an invertible function g. The function g, however, is public. This means that if the adversary is able to reconstruct the filter, he can immediately recover the corresponding secret key. Thus, our attack focuses on recovering the output filter.

As in Section 3, the principle of the attack is a differential cryptanalysis on the initial carry vector. We observe that when we introduce a difference on the bit i of the carry vector, after one clock of the register, this difference will be on the bit i of the main register, after two clocks, it will be on the bit $i - 1$ of the main register, and so on. The Figure 2 shows the cells that may contain a difference after 6 advances of the FCSR.

Since we no longer know the output taps, it is pointless to try to predict the state difference after 6 advances. However, we can try to predict **whether or not an output tap is present at each position**. As observed previously, if we flip the i-th carry bit, it is guaranteed to propagate to position $i-5$ of the main register after 6 advances. For positions $i-4 \leq j \leq i$, the difference may subsist depending on the carries, but these events are quite unlikely.

Thus, after six advances the output will be flipped with probability greater than $1/2$ if there is an output tap at position $i-5$, and not flipped with probability greater than $1/2$ otherwise. This observation opens a way of attack.

4.2 Details of the Attack

In order to predict the output tap number $i-5$, we flip the carry bit number i. Then we observe the difference on the first keystream bit $z(0)$. For positions $i \notin I_d$ where there is no carry cell, our trick consists in targeting the first $j \in I_d$ such that $j > i$ and observing the keystream bit $z(j-i)$.

With good probability, the presence of a difference in the output indicates that $f_i = 1$. Since we are interested in measuring a probability, we need several such experiments. Like we did in Section 3, we vary the experiments by changing the values of the carry bits in the influence zone. We obtain a ratio δ_i of the number of differences observed by the number of total experiments for the bit i.

We first perform a gross prediction of the filter F, quite simply by saying that $f_i = 1$ if $\delta_i > 0.5$, and $f_i = 0$ otherwise. From the different random keys we tried, we observed that this prediction already gives a quite accurate value of F. Indeed, only 20 to 30 errors remain among the 128 filter bits. Since we do not know the positions of these errors, it would be too long to enumerate all candidates at this point.

In order to decrease the number of errors, we propose an iterative algorithm to converge towards the correct filter. We modify locally some bits on the predicted filter and test whether the result fits better to the reality. This algorithm may be described as follows:

- Collect data on the stream cipher by doing the set of experiments described above. Store the values of the δ_i in an array Δ.
- Based on the data, obtain a gross prediction F' for the filter and deduce the corresponding key K' by inverting g.
- Reproduce the same set of experiments "offline", on a stream cipher using the current candidates for the key and the filter. Obtain an array Δ'.
- While $\Delta \neq \Delta'$, do:
  - Pick one byte from the filter and enumerate the 2^8 possible values. Keep the current values of F' for the other bytes. Do the set of experiments for each new guessed value. Keep the value that minimizes $d(\Delta, \Delta')$, where d is some distance function (for instance, the euclidian function).
  - If stuck on a wrong filter value, change the distance d.
- Output the filter and the corresponding key.

The underlying idea is that the **errors on the candidate filters are generally local**. So it is interesting to modify just one byte of the filter and test whether we obtain a better prediction. Random jumps are used in case we get stuck at some point.

We observed in practice that this algorithm was quite efficient and converged reasonably quickly towards the correct key and the correct filter.

4.3 Efficiency and Results

The first part of the algorithm consists in collecting data from the real stream cipher. This requires 2^{15} pairs of IV's. As in Section 3, we are only interested in learning the differential on one output bit. The rest of the attack is performed offline.

The time complexity of the algorithm is trickier to evaluate since it depends on the converging speed. Each step of the loop requires 2^{24} operations. Our experiments with random secret keys suggest that an average of 2^8 passes of the loop are needed to find the correct secret key. Thus the time complexity observed in practice is around 2^{32}. This attack runs in a few hours on a single PC.

4.4 Adapting the Attack to F-FCSR-SD8

The 8 filters are constructed as for **F-FCSR-SF8**. They verify equation 3. For a filter F_i, we know that two bits equal to 1 are separated at least by eight bits. Since the range of our experiments often affects only six or seven bits, we are able to directly guess the correct value of the filter F_i for most of the bits (those that correspond to an experiment with length less than 8). The remaining bits will also have a better prediction than in the previous case. Thus, the attack becomes more powerful and recovers the filter and the secret key much quicker.

5 Time/Memory/Data Trade-Off Attacks

5.1 Trade-Off Attacks Against Stream Ciphers

Since the internal state of a stream cipher changes during the encryption, a brute-force attack has several possible targets. Knowing any value of the internal state is often sufficient for an attacker. This is the main idea behind trade-off attacks: only a small portion of the possible states is enumerated, and we hope, using the birthday paradox, to find a match between the states encountered during the encryption and the states enumerated offline.

The first attack proposed in the literature is generally referred to as Babbage-Golic trade-off [5, 12]. Roughly, a stream cipher with internal state of n bits can be attacked with time, memory and data of the order of $2^{n/2}$. An alternative is to apply the famous Hellman time-memory trade-off [14] to the case of stream ciphers [7]. This second attack allows many trade-offs, but its complexity always remains above $2^{n/2}$ regarding the time complexity.

Because of these attacks, it is recommended for stream ciphers to use an internal state, which is larger than the expected strength (usually twice the size

of the key). Accordingly, Arnault and Berger [3] argued that F-FCSR has a state of $n = 196$ bits at least (it is even more when the output taps are secret), which brings the attack to a complexity larger or equal to 2^{98}, which is not practical. In this Section, we show that a better trade-off attack is possible against the F-FCSR family, due to the mathematical properties of the state-update function.

5.2 Real Entropy of the F-FCSR State

Let $s(t) = (m(t), c(t))$ denote the state of the cipher at time t. There are $2^{k+\ell-1} = 2^{196}$ possible states, and as explained in Section 2.1, each state is characterized by some integer $p_t < (-q)$ such that $p_t = m(t) + 2c(t)$.

A crucial observation is that **the state-update function of a FCSR is not invertible**. Take the example of Figure 1. Suppose that $m_7(t) = 0$ and $m_5(t) = c_5(t) = 1$. Then the previous state must satisfy two incompatible constraints:

$$0 = m_0(t-1)$$
$$1 = m_0(t-1) = m_6(t-1) = c_5(t-1)$$

Therefore, as the encryption proceeds, **the state looses entropy**. It is known that, provided the order of 2 modulo q is $|q| - 1$, the graph of the state-update function had a unique cycle of length $|q| - 1$. Moreover, each state $s = (m, c)$ in the cycle corresponds uniquely to some integer $x < (-q)$ such that $m + 2c = x$. Alongside the unique cycle, the integer x takes successively all values between 1 and $|q| - 1$.

The question is how fast does the state reach this unique cycle. Consider any pair of states (s, s'), that correspond to the same value x:

$$x = m + 2c = m' + 2c'.$$

In section 5.3, we show that these two states converge to the same internal state with very high probability, after 128 advances of the FCSR in average.

Now suppose that s was a state of the unique cycle. Since s' becomes synchronized with s in 128 advances in average, we know that **any state is mapped to a state of the unique cycle after** 128 **advances in average**. Provided we discard the first keystream bits, we can therefore assume that all internal states we encounter are part of the unique cycle. So, the real entropy of the internal state is only of $\log_2(q-1) \simeq 128$ bits.

5.3 Resynchronization in Probabilistic Time

We consider two states which satisfy $x = m + 2c = m' + 2c'$. We say that a cell in the main register is **synchronized** when its value is the same for these two states. We want to determine how many advances are necessary before all cells in the two states become synchronized.

First, we observe that the value of x remains identical for both states after any number of iterations. This follows immediately from the definition of a FCSR. Consequently, the feedback bit, *i.e.* the rightmost bit in the main register is always synchronized. Indeed, $x \bmod 2 = m \bmod 2 = m' \bmod 2$.

In addition, suppose that the i leftmost cells of the main register are synchronized at time t. Then, it is clear that they remain synchronized for all $t' > t$ (the only propagation from right to left in the FCSR comes from the feedback bit, which is synchronized). Moreover, if we are lucky the cell m_{127-i} may become synchronized at time $t+1$.

If $(127-i) \notin I_d$, there is no carry cell and $m_{127-i}(t+1) = m_{127-i+1}(t)$. Therefore m_{127-i} **becomes synchronized at time** $t+1$.

Otherwise, if $(127-i) \in I_d$, then there is a carry cell and

$$m_{127-i}(t+1) = m_{127-i+1}(t) \oplus c_{127-i}(t) \oplus m_0(t).$$

Remember that both bits m_0 and $m_{127-i+1}$ are already synchronized. Roughly, m_{127-i} **becomes synchronized with probability** 0.5 depending on the behavior of the carry register.

Therefore, the whole register is likely to be eventually synchronized. The average time for this to happen is

$$T = \sum_{i \notin I_d} 1 + \sum_{i \in I_d} 2 = 68 + 60 \times 2 = 188$$

advances. Actually, a similar property holds if we consider the rightmost cells, so the synchronization goes in both directions. Generally, it takes less than 128 advances. The phenomenon has been confirmed by our practical simulations.

5.4 Trade-Off Attack with Static Filters

Suppose that the output taps are known. Then the internal state has a "real" entropy of 128 bits only. This allows to apply trade-off attacks on this reduced space instead of the full space of internal states. We propose to apply a simple variant of Babbage-Golic's attack [5, 12]. It proceeds in two steps:

- **The online phase**
  Observe $D = 2^{64}$ keystream bits produced from an unknown secret key K. Discard the first 128 bits (hence it is very likely that we are not in the unique cycle of the state-update function, according to section 5.3), and store in a table each window of 128 keystream bits. There are $D - 128 - 127$ such windows. Each one should uniquely correspond to some value of the internal state[3].
- **The offline phase**
  We want to enumerate at random about 2^{64} internal states located on the unique cycle. For each state, we generate 128 keystream bits, and search for a match with the $D - 255$ windows of the online phase. As soon as a match is found, the attack stops.

[3] A chance remains that from two different states, the same window of 128 keystream bits is produced. It is customary to take an extra margin by considering windows slightly larger than 128 bits.

There are two possibilities to make this enumeration. First, we can just pick some internal states at random and apply 128 advances of the FCSR. Then it is very likely that we select states of the unique cycle. The complexity of the enumeration is 128×2^{64}.
An improved solution is to pick internal states which are located at regular interval on the unique cycle :

- First, pick a state s which is guaranteed to be on this cycle (for instance, $x = 1$ is good since it has a unique representation, $m = 1$ and $c = 0$).
- Next, compute the 2^{64}-th iterate of the state-update function, starting from s. We set the new value of s to this result. Computing such iterates can be done efficiently.
- Generate 128 keystream bits from the actual state s, and search for a match with the $D-255$ windows of the online phase. If a match is found, we stop. Otherwise, we repeat the previous step.

Hence we need to compute about 2^{64} times some 2^{64}-th iterates of the state-update. Since iterates can be computed efficiently, the complexity of this enumeration is about $T = 2^{64}$. Moreover it is guaranteed that all enumerated states are on the unique cycle.

When a match is found, we learn the value of the state at some time $t : s(t) = (m(t), c(t))$. The state-update is not invertible, however, we can compute $p_t = m(t) + 2c(t)$ and backtrack to the values of $p_{t'}$ for $t' < t$ using equation 2. Since $p_0 = m(0) + 2c(0) = K + 2IV$, it is easy to deduce the value of the secret key.

Other choices of D are possible. They result in different trade-offs between time, data and memory complexity. We propose to choose $D = 2^{64}$, therefore all complexities (time, memory and data) are of the order of 2^{64}. This attack is applicable to **F-FCSR-SF1** and **F-FCSR-SF8**.

5.5 Trade-Off Attack with Dynamic Filters

Now suppose that the output taps are unknown. We can no longer apply the previous attack, since the size of the internal state is artifically increased by 128 bits. We would need to guess the output taps in order to apply the trade-off attack.

In the case of **F-FCSR-DF8**, each keystream bit depends on only 16 output taps. We discard the keystream bits such that $i \neq 0 \bmod 8$. We need to guess only 16 taps, in order to apply the previous attack. The extra cost is a factor 8 in data and memory and 2^{16} in time. Therefore the resulting time complexity is about 2^{80}, while the data and memory complexity become about 2^{67}. Other trade-offs could be envisaged with more time and less data/memory.

6 Conclusion

We demonstrated several attacks against the new F-FCSR family of stream ciphers. The first result is a resynchronization attack which breaks all algorithms

in the family which support IV's. The complexity corresponds to about 2^{15} pairs of chosen IV's.

The second result concerns trade-off attacks which breaks several algorithms in the family with time, memory and data of the order of 2^{64}. These attacks are faster than generic attacks and demonstrate some undesirable properties of the underlying primitive, the Feedback with Carry Shift Register (FCSR).

While these attacks do not totally discard the use of FCSR in stream ciphers, they illustrate some important weaknesses in the F-FCSR family.

The European ECRYPT project [10] recently launched a call for primitives in the field of stream ciphers. As a result, 34 new algorithms were proposed and are currently under analysis [11]. Arnault and Berger proposed two algorithms called F-FCSR-8 and F-FCSR-H, which are new variants of the F-FCSR family [4]. In a paper posted on the ECRYPT server, we observed that these algorithms are vulnerable to similar attacks than those described here, and we also pointed out new weaknesses [15].

References

1. F. Armknecht, J. Lano, and B. Preneel. Extending the Resynchronization Attack. In H. Handschuh and A. Hasan, editors, *Selected Areas in Cryptography – 2004*, volume 3357 of *Lectures Notes in Computer Science*, pages 19–38. Springer, 2005.
2. F. Arnault and T. Berger. A new class of stream ciphers combining LFSR and FCSR architectures. In A. Menezes and P. Sarkar, editors, *Progress in Cryptology – INDOCRYPT'02*, volume 2551 of *Lectures Notes in Computer Science*, pages 22–33. Springer, 2002.
3. F. Arnault and T. Berger. F-FCSR: design of a new class of stream ciphers. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption – 2005*, volume 3557 of *Lectures Notes in Computer Science*, pages 83–97. Springer, 2005.
4. F. Arnault, T. Berger, and C. Lauradoux. Description of F-FCSR-8 and F-FCSR-H stream Ciphers. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/008, 2005. `http://www.ecrypt.eu.org/stream`.
5. S. Babbage. A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers. In *European Convention on Security and Detection*, volume 408. IEE Conference Publication, may 1995.
6. E. Barkan, E. Biham, and N. Keller. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. In D. Boneh, editor, *Advances in Cryptology – Crypto'03*, volume 2729 of *Lectures Notes in Computer Science*, pages 600–616. Springer, 2003.
7. A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In T. Okamoto, editor, *Advances in Cryptology – Asiacrypt'00*, volume 1976 of *Lectures Notes in Computer Science*, pages 1–13. Springer, 2000.
8. N. Courtois and W. Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In E. Biham, editor, *Advances in Cryptology – Eurocrypt'03*, volume 2656 of *Lectures Notes in Computer Science*, pages 345–359. Springer, 2003.
9. J. Daemen, R. Govaerts, and J. Vandewalle. Resynchronization Weaknesses in Synchronous Stream Ciphers. In T. Helleseth, editor, *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lectures Notes in Computer Science*, pages 159–167. Springer, 1994.

10. ECRYPT Network of Excellence in Cryptology
`http://www.ecrypt.eu.org/index.html`.
11. eSTREAM - The ECRYPT Stream Cipher Project
`http://www.ecrypt.eu.org/stream/`.
12. J. Golić. Cryptanalysis of Alleged A5 Stream Cipher. In W. Fumy, editor, *Advances in Cryptology – Eurocrypt'97*, volume 1233 of *Lectures Notes in Computer Science*, pages 239–255. Springer, 1997.
13. J. Golic and G. Morgari. On the Resynchronization Attack. In T. Johansson, editor, *Fast Software Encryption – 2003*, volume 2887 of *Lectures Notes in Computer Science*, pages 100–110. Springer, 2003.
14. M. Hellman. A Cryptanalytic Time-Memory Tradeoff. *IEEE Transactions on Information Theory*, 26(4):401–406, July 1980.
15. E. Jaulmes and F. Muller. Cryptanalysis of ECRYPT Candidates F-FCSR-8 and F-FCSR-H. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/046, 2005. `http://www.ecrypt.eu.org/stream`.
16. A. Klapper and M. Goresky. 2-adic shift registers. In R. Anderson, editor, *Fast Software Encryption – 2005*, volume 809 of *Lectures Notes in Computer Science*, pages 174–178. Springer, 1994.
17. A. Klapper and M. Goresky. Cryptanalysis based on 2-adic rational approximation. In D. Coppersmith, editor, *Advances in Cryptology – Crypto'95*, volume 963 of *Lectures Notes in Computer Science*, pages 262–274. Springer, 1995.
18. I. Mantin and A. Shamir. A Practical Attack on Broadcast RC4. In M. Matsui, editor, *Fast Software Encryption – 2001*, volume 2355 of *Lectures Notes in Computer Science*, pages 152–164. Springer, 2002.
19. W. Meier and O. Staffelbach. Fast Correlations Attacks on Certain Stream Ciphers. In *Journal of Cryptology*, pages 159–176. Springer, 1989.
20. T. Siegenthaler. Correlation-immunity of Nonlinear Combining Functions for Cryptographic Applications. In *IEEE Transactions on Information Theory*, volume 30, pages 776–780, 1984.
21. B. Zhang, H. Wu, D. Feng, and F. Bao. Chosen Ciphertext Attack on a New Class of Self-Synchronizing Stream Ciphers. In A. Canteaut and K. Viswanathan, editors, *Progress in Cryptology – INDOCRYPT'04*, volume 3348 of *Lectures Notes in Computer Science*, pages 73–83. Springer, 2004.

Fault Attacks on Combiners with Memory*

Frederik Armknecht[1] and Willi Meier[2]

[1] Universität Mannheim, 68131 Mannheim, Germany
armknecht@th.informatik.uni-mannheim.de
[2] FH Aargau, CH-5210 Windisch, Switzerland
meierw@fh-aargau.ch

Abstract. Fault attacks are powerful cryptanalytic tools that are applicable to many types of cryptosystems. Recently, general techniques have been developed which can be used to attack many standard constructions of stream ciphers based on LFSR's. Some more elaborated methods have been invented to attack RC4. These fault attacks are not applicable in general to combiners with memory.

In this paper, techniques are developed that specifically allow to attack this class of stream ciphers. These methods are expected to work against any LFSR-based construction that uses only a small memory and few input bits in its output function. In particular, efficient attacks are described against the stream cipher E0 used in Bluetooth, either by inducing faults in the memory or in one of its LFSR's. In both cases, the outputs derived from the faulty runs finally allow to describe the secret key by a system of linear equations. Computer simulations showed that inducing 12 faults sufficed in most cases if about 2500 output bits were available. Another specific fault attack is developed against the stream cipher SNOW 2.0, whose output function has a 64-bit memory. Similar to E_0, the secret key is finally the solution of a system of linear equations. We expect that one fault is enough if about 2^{12} output words are known.

Keywords: Stream cipher, combiner with memory, LFSR, fault attack, Bluetooth E0, SNOW 2.0.

1 Introduction

Fault analysis was first introduced in 1996 in [6] to attack number theoretic public key cryptosystems such as RSA, and later in [7] to attack product block ciphers such as DES. These attacks are practical, and various techniques have been described that induce faults during cryptographic computations (cf. [17]). More recently, fault analysis of stream ciphers has been introduced by Hoch and Shamir in [12]. As for other cryptosystems, fault analysis on stream ciphers is a powerful cryptanalytic tool which can work even if direct attacks are inefficient. In [12], general techniques are applied to attack standard constructions of stream ciphers based on LFSR's, and some specialized techniques are introduced that

* The first author has been supported by grant Kr 1521/7-2 of the DFG (German Research Foundation).

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 36–50, 2006.

work, e.g., against RC4. In [4], different and improved fault analysis methods on RC4 are derived. The methods developed in [12] apply for most schemes based on memoryless filters and combiners. However, as pointed out in [12], they don't work in general for combiners with memory. Well known stream ciphers that use a combiner with memory are E0, the keystream generator used in Bluetooth, and SNOW 2.0, that is proposed for standardization.

In this paper, fault attacks against general combiners with memory based on LFSR's are developed. It is assumed that faults may be induced either in the memory or in one of the LFSR's. These methods are expected to work against any LFSR-based stream cipher whose output function uses only a small amount of memory. In particular, our methods are adapted and refined to an efficient fault attack on E0, whose output function has a memory of only 4 bit. Our attack heavily exploits the particular structure of E0. Another specific attack is developed against the stream cipher SNOW 2.0 that uses a 64 bit memory.

2 Combiners with Memory

Keystream generators are used for the online encryption of bitstreams having arbitrary length, e.g. between two mobile phones. Initialized with a secret value, the key, they produce a bitstream $Z = z_1, z_2, \ldots$, called the keystream. A plaintext $p_1, p_2, \ldots$ is encrypted to $c_1, c_2, \ldots$ via $c_i := p_i \oplus z_i$ where $\oplus$ denotes the XOR-operation on two bits. A legal receiver who uses the same keystream generator and has knowledge of the key, can produce the keystream by himself and decrypt the received bitstream by computing $p_i = c_i \oplus z_i$.

Many keystream generators used in practice and discussed in theory are combiners with memory or, more formally, (k, ℓ)-combiners. A (k, ℓ) - combiner $\mathcal{C} = (f, \varphi)$ with k inputs and ℓ memory bits is a finite state machine (FSM) which is defined by an output function $f : \{0,1\}^\ell \times \{0,1\}^k \longrightarrow \{0,1\}$ and a memory update function $\varphi : \{0,1\}^\ell \times \{0,1\}^k \longrightarrow \{0,1\}^\ell$. Given a stream $(X_1, X_2, \cdots)$ of inputs, $X_i \in \{0,1\}^k$, and an initial assignment $Q_1 \in \{0,1\}^\ell$ to the memory bits, the corresponding output bitstream $(z_1, z_2, \cdots)$ is defined according to

$$z_t = f(Q_t, X_t) \quad \text{and} \quad Q_{t+1} = \varphi(Q_t, X_t), \tag{1}$$

for all $t \geq 1$. Using (1), we define $f_\varphi(Q, X_1, \ldots, X_r) := (z_1, \ldots, z_r)$ and $\varphi(Q, X_1, \ldots, X_r) := Q_{r+1}$ where $z_i = f(Q_i, X_i)$, $Q_{i+1} = \varphi(Q_i, X_i)$ and $Q_1 := Q$.

For keystream generation, the stream of inputs $(X_1, X_2, \ldots)$ is produced by the output of k driving devices where the initial states are determined by the secret key $K \in \{0,1\}^n$. We assume that these sequences are generated by LFSRs, i.e., each input is computed by $X_t = L_t(K)$ where L_t is a known linear function $\{0,1\}^n \to \{0,1\}^k$. Consequently, the initial state can be reconstructed, if enough linear equations relating input bits X_t are known.

An example for a combiner with memory is the summation generator which is based on the integer addition. In the case of $k = 2$ input bits $X_t = (a_t, b_t)$, the number of memory bits is $\ell = 1$ and the functions are defined by $z_t = f(Q_t, a_t, b_t) := a_t \oplus b_t \oplus Q_t$ and $Q_{t+1} = \varphi(q_t, a_t, b_t) := a_t b_t \oplus a_t Q_t \oplus b_t Q_t$ A

practical example for a combiner with memory using LFSRs is the keystream generator E_0 used in the Bluetooth standard. It is a $(4, 4)$ combiner with a 128-bit key.

It is commonly accepted to evaluate the security of combiners with memory in the following attack model.An adversary knows the functions f and φ and the driving devices. In the case that LFSRs are incorporated, he knows the linear functions L_t. Furtheron, he is able to observe the values of some keystream bits z_t. An attack is to use these informations to recover the secret key K.

It is clear from information theory that the knowledge of some keystream bits z_t must provide some information about the corresponding inputs X_t. This gives rise to the following definition:

Definition 1. *For a (k, ℓ)-combiner $\mathcal{C} = (f, \varphi)$, $r \geq 1$ and $(z_1, \ldots, z_r) \in \{0,1\}^r$, we define the set of possible inputs by*

$$X_{(z_1,\ldots,z_r)} := \{(X_1, \ldots, X_r) \,|\, \exists Q : \; f_\varphi(Q, X_1, \ldots, X_r) = (z_1, \ldots, z_r)\}. \qquad (2)$$

An important property of combiners with memory is the following result which has been also essential in algebraic attacks [1]:

Theorem 1. *Let $\mathcal{C} = (f, \varphi)$ be an arbitrary (k, ℓ)-combiner with k inputs and ℓ memory bits. Then there exists at least one output $(z_1, \ldots, z_{\ell+1}) \in \{0,1\}^{\ell+1}$ such that $|X_{(z_1,\ldots,z_r)}| \leq \frac{1}{2} \cdot 2^{k\cdot(\ell+1)}$. This means that this specific output can be generated by at most half of all possible inputs in $\{0,1\}^{k\cdot(\ell+1)}$. In particular this allows to rule out some values for $(X_1, \ldots, X_{\ell+1})$.*

Proof. Note that $(Q, X_1, \ldots, X_{\ell+1})$ uniquely determines the output $Z = (z_1, \ldots, z_{\ell+1})$ and therefore $\{0,1\}^{\ell+k\cdot(\ell+1)} = \dot{\bigcup}_Z f_\varphi^{-1}(Z)$. Furtheron, it is by definition $|f_\varphi^{-1}(Z)| \geq |X_Z|$.

Assume that the proposition is not true, i.e., $|X_Z| > 2^{k\cdot(\ell+1)-1}$ for each $Z \in \{0,1\}^{\ell+1}$. This leads to the contradiction

$$\begin{aligned} 2^{\ell+k(\ell+1)} &= \sum_Z |f_\varphi^{-1}(Z)| \geq \sum_Z |X_Z| \\ &> \sum_Z 2^{k(\ell+1)-1} = 2^{\ell+1} \cdot 2^{k(\ell+1)-1} = 2^{\ell+k(\ell+1)}. \end{aligned}$$

3 A General Fault Attack on Combiners with Memory

The model assumptions for fault attacks are that an adversary has access to the keystream generator as a physical device. Considering the proliferation of mobile phones or devices using Bluetooth, this is certainly a realistic scenario.

Furthermore, the attacker can re-set the device to the same unknown initial setting as often as he wants. As stream ciphers are descendants of the one-time pad, the normal mode of operation would disallow to re-set the stream cipher and to generate the same keystream more than once. On the other hand, an attacker

which has acquired the pysical device is certainly more powerful than a normal passive attacker. For example, consider the case of a stream cipher that uses a resynchronization mechanism where the frame key is generated from a master key and a public initial value IV (as for example it is the case in Bluetooth). The attacker could feed the same IV into the device several times which would practically re-set the stream cipher each time to the same unknown setting.

In addition, it is assumed that the attacker can cause an unknown fault into one of the registers during the computations. He knows the affected register, but can neither control the point in time of the disturbance nor the induced error. We refer in this context to [17] where a low-tech, low-cost method to induce faults at very specific locations were proposed.

A certainly important question is whether fault attacks on stream ciphers can be considered to be practical or not. For the reasons mentioned above, we would agree and believe that fault attacks on stream ciphers pose a potential threat being worth to be investigated further. On the other hand, no practical fault attacks on stream ciphers have been conducted so far and it might finally turn out that the attack assumptions are too powerful. We hope that the results in [12] and in this paper animate further research to finally settle this question (positive or negative).

In this section, we introduce a fault attack which works in principle for every (k, ℓ)-combiner based on LFSRs if k and ℓ are not too big. We assume the fault to be either induced into the memory register or into one of the LFSRs. In the first case our attack is directly applicable. This will be demonstrated by successful fault attacks on the memory of E0 and of SNOW 2.0. In the second case the additional problem to determine the induced fault in the LFSR is imposed. This may be practically infeasible for certain keystream generators. On the other hand, we will show in the next section that this problem can be solved for the keystream generators E_0.

Let $Q_1 \in \{0,1\}^\ell$, and $(X_1, \ldots, X_R) \in \{0,1\}^{k \cdot R}$, $R \geq 1$, be the output of the k LFSRs, depending on an initial setting K. The keystream is denoted by $(z_1, \ldots, z_R) := f_\varphi(Q_1, X_1, \ldots, X_R)$. The attacker is able to re-run the keystream generator an arbitrary number of times with the same unknown initial states Q_1 and K and to observe the produced keystream. During the run, he can induce a unknown fault either into the memory register or into one of the LFSRs, but has no control about the point in time t'. Let $(z'_1, \ldots, z'_R)$ be the corresponding keystream. The keystream bits z'_t and z_t are equal for $t < t'$ but may be different for $t \geq t'$. By comparing both keystreams, the first clock t with $z_t \neq z'_t$ indicates that the fault has already occurred.

Let $X'_t \in \{0,1\}^k$ be the output of the LFSRs at clock t of the faulty run and $\delta_t := X_t \oplus X'_t$ the unknown differences. The location of the induced fault and the kind of the keystream generator determine how to deal with the differences.

In the case where the memory register has been tampered, the LFSRs have been left unchanged. Hence, in this case $\delta_t = 0$ for all clocks t.

If the fault has been induced in one of the LFSRs, the situation is more complicated. Some of the δ_t are different to zero but the attacker does not know

which. For general (k, ℓ)-combiner, $f(Q_t, X_t) = z_t \neq z'_t = f(Q'_t, X'_t)$ does only implicate that $(X_t, Q_t) \neq (X'_t, Q'_t)$ but not that $X_t \neq X'_t$. Therefore, comparing both keystreams does not reveal immediately information about the induced fault. Nevertheless, the keystream generator may have additional properties, allowing the detection of the whole fault. One such example is E_0, as discussed in the next two sections.

A special case are simple combiners. Simple combiner means that the combiner is memoryless, e.g. $\ell = 0$, and that the k LFSRs are all distinct. As no memory bits are present, $z_t \neq z'_t$ implies $f(X_t) \neq f(X'_t)$ and therefore $X_t \neq X'_t$. Sampling enough such time instances permit to compute the whole fault. Therefore, simple combiners seem to be vulnerable to fault attacks in general.

Altogether, we assume that the adversary has the capability of finding out the induced fault.

The proposed fault attack is related to algebraic attacks. After observing the original keystream $z_1, \ldots, z_R$, an attacker knows (for a fixed value $\tau \geq 0$) that

$$X_t^{t+\tau} := (X_t, \ldots, X_{t+\tau}) \in X_{z_t,\ldots,z_{t+\tau}}, \quad 1 \leq t \leq R - \tau. \tag{3}$$

In the positive case, he may use this information to derive algebraic equations in $X_t^{t+\tau}$. Because of $X_t = L_t(K)$, this gives a system of equations in K. If the equations are linear, Gaussian elimination can be used to find K. In the case of non-linear equations, more sophisticated methods as relinearization or Gröbner bases might be helpful, in particular, if the number of equations exceeds the number of unknowns. It is known by [1] that for $\tau = \ell$, algebraic equations of degree at most $\lceil \frac{k \cdot (\ell+1)}{2} \rceil$ always do exist.

However, with the additional information at hand by several runs, an attacker can hope for equations of lower degree, or even linear equations. By observing the faulty keystream $z'_1, \ldots, z'_R$, the attacker can deduce that, additionally to (3), it holds that

$$X'^{t+\tau}_t \in X_{z'_t,\ldots,z'_{t+\tau}}, \quad t' \leq t \leq R - \tau. \tag{4}$$

If he can determine the differences $\delta_t := X_t \oplus X'_t$ for $t \geq t'$, combining (3) and (4) reduces the set of possible assignments of $X_t^{t+\tau}$ even further. It is

$$X_t^{t+\tau} \in X_{z_t,\ldots,z_{t+\tau}} \cap (X_{z'_t,\ldots,z'_{t+\tau}} \oplus (\delta_t, \ldots, \delta_{t+\tau})) \tag{5}$$

with

$$X_{z'_t,\ldots,z'_{t+\tau}} \oplus (\delta_t, \ldots, \delta_{t+\tau}) := \{(Y_0 \oplus \delta_t, \ldots, Y_\tau \oplus \delta_{t+\tau}) | (Y_0, \ldots, Y_\tau) \in X_{z'_t,\ldots,z'_{t+\tau}}\}.$$

As explained above, if the fault was in the memory bits, it is $\delta_t = 0$. This simplifies (5) to

$$X_t^{t+\tau} \in X_{z_t,\ldots,z_{t+\tau}} \cap X_{z'_t,\ldots,z'_{t+\tau}}. \tag{6}$$

After each run, the adversary checks if the remaining possible values of $X_t^{t+\tau}$ allow to set up equations of low degree in $X_t^{t+\tau}$. Algorithms to find such equations

have been used in several papers [8, 5, 1, 16], an explicit description is given in [3, Alg. 1]. An alternative is to use Gröbner bases.

Repeating these steps sufficiently often can result in a system of low degree (or even linear) functions. Thus, the final step consists in computing the solution to derive the LFSR's initial state. An example for this attack on the summation generator with $k = 2$ LFSRs is given in appendix A.

The whole attack is summarized in the following pseudo code:

General fault attack on (k, l)-combiners

Input: A (k, l)-combiner C, and integers $d, N \geq 1$ and $\tau \geq 0$
Task: Recover the initial state of the LFSRs
Algorithm:

- Derive for all possible outputs $(z_0, \ldots, z_\tau)$ the sets $X_{(z_0,\ldots,z_\tau)}$ of possible inputs
- Run the generator once and observe a keystream $z_1, \ldots, z_R$.
- For $t = 1, \ldots, R-\tau$, initialize the sets of possible inputs $\mathcal{X}_t \coloneqq X_{(z_t,\ldots,z_{t+\tau})}$
- While number of equations of degree $\leq d$ is less than N do
  - Induce an unknown fault in the memory register and observe the output $(z'_1, \ldots, z'_R)$.
  - Look for the first clock t' with $z_t \neq z'_{t'}$.
  - Determine the differences $\delta_t := X_t \oplus X'_t$ for $t \geq t'$.
  - For $t \geq t'$, reduce the set of possible inputs $X_t^{t+\tau}$ according to (5) or (6)
  - Derive the number of equations on $\mathcal{X}_t$ of degree $\leq d$ for $t = 0, \ldots, T-\tau$ and count them.
- end while;
- Recover the initial state by finding a solution to these equations.

4 Fault Attacks to the Bluetooth Keystream Generator

4.1 Preliminary Notes

In this section, we discuss fault attacks on the keystream generator E_0, being part of the Bluetooth encryption. E_0 employs four LFSRs, named A, B, C and D, of lengths 25, 31, 33 and 39, respectively. We denote their outputs at clock t by a_t, b_t, c_t and d_t, i.e., it is $X_t = (a_t, b_t, c_t, d_t)$. The memory bits Q_t are referred to as $Q_t = (p_t, q_t, p_{t-1}, q_{t-1})$ and the output z_t is computed via $z_t = q_t \oplus s_t$ where $s_t := a_t \oplus b_t \oplus c_t \oplus d_t$. We assume that the attacker is always able to produce and to observe R keystream bits $z_1, \ldots, z_R$. In the Bluetooth standard, the key changes after 2745 outputs. Hence, this value states a natural upper bound for R, at least in respect of practical attacks.

Referring to section 3, we consider two different scenarios: the fault does occur either in the memory register Q_t or in one of the four LFSRs. In [1], it was shown that for any four fixed successive outputs z_t,z_{t+2},z_{t+3},z_{t+4}, the set $X_{(z_t,z_{t+2},z_{t+3},z_{t+4})}$ of possible inputs X_t^{t+3} is a strict subset of $\{0,1\}^{4\cdot 4}$. Therefore, every quadruple of four known successive outputs excludes some values for X_t^{t+3}.

We will show how the keystreams $z'_1, \ldots, z'_R$ derived from faulty runs can be used to rule out as many inputs as possible. Computer tests showed that this does not allow to single out the actual value of X_t^{t+3} directly, but the information is nonetheless valuable for an attacker. It turned out that in every case, the remaining set of possible values of X_t^{t+3} satisfied a linear equation $L(X_t^{t+3}) = 0$. Implementing this for several clocks finally allows to describe the LFSRs' initial state by a system of linear equations.

4.2 Fault Attack on the Memory Register

In this section, we discuss the fault attack for the case that the memory register is disturbed. Let t' denote the point in time of the fault. As the LFSRs' output remain the same for all faulty runs, it is $\delta_t = 0$ for all t. The keystream bits $z'_{t'}, \ldots, z'_R$ allow directly to reduce the set of possible keys as formulated in (6).

We simulated this attack on a computer. For 1000 times, we have chosen a random initial setting and produced faulty keystreams until we were able to set up 128 different linear equations in the initial state. As the time instance t' differs from fault to fault, we made the moderate assumption of 500 generated faulted keystream bits per fault. The results are displayed in Table 1. It shows in how many test cases which number of faults were necessary to derive the system of linear equations. For example, in 17.3 percent of our test cases, 10 faults were required. We see that in more than 90 percent of all simulations, 12 faults were enough.

Table 1. Simulation results for the fault attack on the memory register

# induced faults	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Percentage	10.3	18.9	17.1	17.3	14.2	9.6	5.7	2.5	2.1	0.9	0.7	0.5	0.1	0	0	0.1

4.3 Fault Attack on One LFSR

Now we consider the second attack scenario, where the fault is induced into one of the LFSRs. We assume that the affected LFSR is A which has the shortest length. The state of A is altered at a certain uncontrollable point in time t' and afterwards, some of the inputs $X_t = (a_t, b_t, c_t, d_t)$, memory states Q_t and keystream bits z_t are changed to $X'_t = (a_t \oplus \tilde{\delta}_t, b_t, c_t, d_t)$, Q'_t and z'_t, respectively. This is depicted in Table 2. Let $s'_t := s_t \oplus \tilde{\delta}_t$ and $\tilde{\Delta}$ be the unknown fault $\tilde{\delta}_1, \ldots, \tilde{\delta}_R$. Observe that the first clock t with $z_t \neq z'_t$ implies immediately that $\tilde{\delta}_t = 1$. W.l.o.g., we assume that $\tilde{\delta}_1 = 1$. This is no restriction as we did not specify the value of R.

The general attack described in section 3 requires to identify the values of δ_t. We present an appropriate method, using correlations between linear combinations of the known z_t, z'_t and the correct δ_t. More precisely, for certain coefficient vectors $\Gamma = (\gamma_0, \ldots, \gamma_r)$, it holds that

$$\bigoplus_{i=0}^{r} \gamma_i \cdot (z_{t+i} \oplus z'_{t+i}) = \bigoplus_{i=0}^{r} \gamma_i \cdot \delta_{t+i} \tag{7}$$

Table 2. The fault attack on E_0 with faulty LFSR. The grey marked entries may change their values.

Clock	1	...	$t'-1$	t'	$t'+1$	...	R
Input	X_0	...	$X_{t'-1}$	$X'_{t'}$	$X'_{t'+1}$	...	X'_R
Memory	Q_0	...	$Q_{t'-1}$	$Q_{t'}$	$Q'_{t'+1}$	...	Q'_R
Keystream	z_0	...	$z_{t'-1}$	$z'_{t'}$	$z'_{t'+1}$	...	z'_R

is true with probability $> 1/2$ if the δ_t agree with the induced fault, i.e. $\delta_t = \tilde{\delta}_t$ for all t. On the other hand, if the $\delta_t \neq \tilde{\delta}_t$, the value on the right hand side is independent of the left side and (7) is true in only half of the cases. We use this as a distinguishing feature to separate the value $\tilde{\Delta}$ of the induced fault from all possible fault values Δ.

Some Γ together with their probabilities that (7) is true are displayed in Table 3. The probabilities were derived by simply counting the positive events for all possible cases. These probabilities cannot be derived directly from the correlations as determined in [11] for E0. However, Equation (7) is related to (conditional) correlated linear combinations of the the memory bits q_t which have proven to be useful for the cryptanalysis of combiners with memory before, especially in the case of E_0 (cf. [13, 14, 15]).

Let keystreams Z and Z' be given and coefficient vectors $\Gamma_1, \ldots, \Gamma_s$ be fixed. For a potential fault Δ, we define the value $\Sigma_\Gamma(\Delta) := \Sigma_\Gamma(Z, Z', \Delta)$ to be the number of clocks where (7) holds. An example for computing $\Sigma(\Delta)$ is given in appendix B.1. As explained above, we expect for the right Δ to have *on average* a higher value $\Sigma_\Gamma(\Delta)$ than for the wrong guesses.

Consequently, a logical approach is to use the values $\Sigma_\Gamma(\Delta)$ to set up a ranking list of all possible faults. This means that the value Δ with the highest value $\Sigma_\Gamma(\Delta)$ is the most probable fault an so on. Indeed, computer simulations showed that if the ratio of number of possible keys and number of outputs is too small, then the right fault was not amongst those values Δ at the top of the ranking lists.

Table 3. Some correlations together with their probabilities. * means that the probability is independent of the penultimate entry.

Γ	$Pr[\bigoplus_i \gamma_i \cdot (z_{t+i} \oplus z'_{t+i} \oplus \delta_{t+i}) = 0]$
$(1, 0, *, 1)$	0.508
$(1, 1, *, 1)$	0.523
$(1, 0, 0, *, 1)$	0.505
$(1, 0, 1, *, 1)$	0.505
$(1, 1, 0, *, 1)$	0.505
$(1, 1, 1, *, 1)$	0.524
$(1, 0, 0, 0, *, 1)$	0.530

Therefore, we refined this approach by using multiple ranking lists, defined by the s coefficient vectors Γ_i. The idea is now to look for values Δ which occur amongst the top elements of as many ranking lists as possible.

This is exactly our approach. For each Γ, we determine h different values Δ, having the highest values $\Sigma_\Gamma(\Delta)$. This gives so to speak a ranking list $L(\Gamma) := L(\Gamma, h)$ in terms of the values $\Sigma_\Gamma(\Delta)$, consisting of h different potential faults Δ. The next step is to regroup these according to their occurrences in the ranking lists. We define for each Δ its *index* by $\mathrm{Ind}(\Delta) := \mathrm{Ind}_{Z,Z'}(\Delta)$ to be the number of ranking lists $L(\Gamma_1), \ldots, L(\Gamma_s)$ containing Δ. An example for computing the index is given in appendix B.2. As explained above, if one candidate has a higher index than the others, we assume it to be the unknown induced fault.

Computer experiments using a data of 2745 keystream bits showed that often there was no unique Δ with maximum index. In this case, one can restart the keystream generator, induce another fault and hope for a better result this time.

A more refined alternative is to combine the results from two different runs. Let Z be the original keystream and Z' and Z'' be faulty keystreams, resulting from unknown faults $\tilde{\Delta}'$ and $\tilde{\Delta}''$, respectively. To keep the explanation simple, we assume that both faults $\tilde{\Delta}'$ and $\tilde{\Delta}''$ occurred at the same clock. The method does work without this condition too.

Now we bring a new viewpoint into play. Z'' can be seen also as the faulty keystream resulting from the "original" keystream Z' and the induced fault $\tilde{\Delta}' \oplus \tilde{\Delta}''$. This imposes additional conditions on the candidates for $\tilde{\Delta}'$ and $\tilde{\Delta}''$. Consequently, instead of looking for the most probable values for $\tilde{\Delta}'$ and $\tilde{\Delta}''$ independently, one can search for the most probable *pair* for $(\tilde{\Delta}', \tilde{\Delta}'')$. Candidates (Δ_1, Δ_2) which have high indices $\mathrm{Ind}_{Z,Z'}(\Delta_1)$, $\mathrm{Ind}_{Z,Z''}(\Delta_2)$ *and* $\mathrm{Ind}_{Z',Z''}(\Delta_1 \oplus \Delta_2)$ are therefore preferred. For our simulations, we simply considered the sum of these three values.

The computer experiments with a reduced length LFSR A proved the practicability of our method, at least in the considered test cases. For our tests, we assumed that the attacker is able to produce each time the whole 2745 keystream bits and that all faults occurred during the first 245 clocks. This is no severe condition. If after inducing the fault, the first clock t with $z_t \neq z'_t$ is ≤ 245, we can be sure that the condition is met. If this is not the case, one discards this fault and reattempts again. Therefore, the effort caused by useless runs is comparatively small. Furtheron, one can expect that in almost one out of ten attempts, the faults occurs within the required bounds.

The first tests showed that if all Δ had an index of 2 or less, the event of finding the right fault was very unlikely. Therefore, we restarted to induce faults for a fixed unknown initial setting until at least on index was 3 or higher.

We did 21 tests for the case that LFSR A has length 19 (using some random primitive polynomial for feedback). The correct pair $(\tilde{\Delta}', \tilde{\Delta}'')$ was uniquely reconstructed each time. In average, it was necessary to induce 13.86 faults whereas the maximum was 41. Furthermore, we conducted 3 experiments where A has length 20. Also here, the right pair was reconstructed every time. The average number of faults was 27.66 with the maximum being 59. Experiments with the

original length of 25 have been conducted but have always crashed, presumably because of memory shortage. It remains as an open question whether faults of length 25 could be recovered by a more efficient implementation.

The final step consists in reducing the set of possible inputs as described in section 3, once the faults are determined. We simulated this attack 1000 times and counted the number of faults needed to set up at least 128 linear equations in the last 500 clocks (i.e, in $X_{R-500}, \ldots, X_R$). Thereby, we always assumed that we figured out the right fault Δ. It turned out that in 763 cases, 11 faults were necessary, and it the remaining ones 12 faults. Similar to the attack where the memory register is tampered, 12 faults seem to be enough whereas 11 faults are always necessary.

4.4 Fault Attacks on Two-Level-E_0

The keystream generator is only one part of the Bluetooth stream cipher. Therefore, we shortly describe the whole system and point out the consequences of our attacks. During Bluetooth encryption, the secret key is re-initialized frequently, using a secret parameter $\mathcal{K}$, denoted as the master key, and publicly known values IV. The period using the same secret key is called a frame. The whole Bluetooth keystream generation consists of the following steps:

1. Derive the frame key $K = L(\mathcal{K}, IV)$ where L is a known linear Boolean function, $\mathcal{K}$ the master key and IV a publicly known parameter. This means that the Bluetooth cipher uses a linear key schedule. Use K to initialize the LFSRs in E_0.
2. Clock E_0 several times and compute 128 bit intermediate outputs $y_1, \ldots, y_{128}$.
3. Permute the 128 bits using a fixed permutation.
4. Use the 128 bit output to re-initialize the four LFSRs of E_0 and produce the keystream $z_1, \ldots, z_{2745}$

In [2], it was shown that if the intermediate outputs y_i are known for several frames, the master key $\mathcal{K}$ can be efficiently reconstructed. The proposed fault attacks provide efficient methods to attack one instantiation of E_0. Thus, our attacks can be applied to the second call of E_0 for several frames to determine the y_i's. Afterwards, the methods in [2] can be used to reconstruct $\mathcal{K}$. This shows that the whole Bluetooth encryption is affected by the fault attacks.

5 Fault Attack on SNOW 2.0

SNOW 2.0 has been introduced in [9] and is proposed for standardization. It consists of a length 16 LFSR with 32 bit words, feeding a finite state machine. The FSM consists of two 32 bit memory registers, called $R1$ and $R2$, as well as some operations to calculate the output and the next state (i.e., the next values of $R1$ and $R2$). A schematic illustration is given in Figure 1. For a precise description we refer to [9]. We only recall those facts that are needed in our attack.

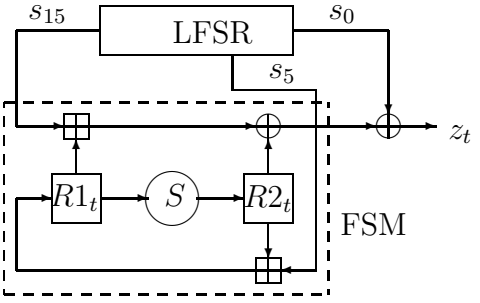


Fig. 1. A schematic picture of SNOW 2.0

Denote the 16 words in the LFSR at clock t by s_t,...,s_{t+15}. The output word z_t at clock t is defined by

$$z_t = (s_{t+15} + R1_t) \oplus R2_t \oplus s_t$$

where "+" means integer addition of 32 bit words. The rule to update registers $R1$ and $R2$ at clock t is given by

$$R2_{t+1} := \mathcal{S}(R1_t) \quad \text{and} \quad R1_{t+1} := R2_t + s_{t+5} \tag{8}$$

where $\mathcal{S}$ is a nonlinear 32 bit to 32 bit bijection, which is composed of the S-box and the `MixColumn` step in the AES block cipher. However, we don't make use of an explicit description of $\mathcal{S}$.

Our fault attack on SNOW 2.0 assumes that one can induce a fault in the memory register $R2$ (for a time instant t' over which there is no control). Now we point out a particular property of SNOW 2.0. Assume for one clock t that either $R2_t$ or $R1_t$ is faulty but not both registers at the same time. Then due

Fault was induced into $R2_{t'}$ before $z_{t'}$ is computed

Clock	...	$t'-1$	t'	$t'+1$	$t'+2$	$t'+1$	...
Register $R1$	...	$R1_{t'-1}$	$R1_{t'}$	$R1'_{t'+1}$	$R1_{t'+2}$	$R1'_{t'+3}$	...
Register $R2$	...	$R2_{t'-1}$	$R2'_{t'}$	$R2_{t'+1}$	$R2'_{t'+2}$	$R2_{t'+3}$	...
Keystream	...	$z_{t'-1}$	$z'_{t'}$	$z'_{t'+1}$	$z'_{t'+2}$	$z'_{t'+3}$	...

Fault was induced into $R2_{t'-1}$ after $z_{t'-1}$ has been computed

Clock	...	$t'-1$	t'	$t'+1$	$t'+2$	$t'+1$	...
Register $R1$	...	$R1_{t'-1}$	$R1'_{t'}$	$R1_{t'+1}$	$R1'_{t'+2}$	$R1_{t'+3}$	...
Register $R2$	...	$R2'_{t'-1}$	$R2_{t'}$	$R2'_{t'+1}$	$R2_{t'+2}$	$R2'_{t'+3}$	...
Keystream	...	$z_{t'-1}$	$z'_{t'}$	$z'_{t'+1}$	$z'_{t'+2}$	$z'_{t'+3}$	...

Fig. 2. The two different cases if inducing a fault into $R2$

to (8), one clock later the perturbed register is now undisturbed and vice versa. Therefore, for each clock $t \geq t'$, either $R2_t$ or $R1_t$ is faulty but not both together.

Assume that after inducing the fault, the output is changed starting from a clock t. Then, two different cases are possible. Either the value of $R2_{t'}$ has been changed *before* the computation of $z_{t'}$ or $R2_{t'-1}$ has been disturbed *after* the computation of $z_{t'-1}$. This is shown in Figure 2. The gray marked entries indicate the faulty ones.

In both cases several outputs z_t with faulty registers $R2_t$ are known. In the first case, it is $t \in \{t', t'+2, t'+4, \ldots\}$, in the second $t \in \{t'+1, t'+3, t'+5, \ldots\}$. As we do not know which case is true, we have to do the attack once for each case. Afterwards, the wrong solution can be easily discarded by computing additional keystream and comparing it with the observed data.

We describe the attack for the case that $R2_t$ has been disturbed before z_t has been computed. The attack for the other case is straightforward. In a basic version, the attack works as follows:

Consider a fault induced in $R2_t$, that complements the least significant bit, but does not complement the 2nd lowest bit in $R2_t$. The fault in the higher bits may be arbitrary. The faulty memory register is called $R2'_t$. That is $R2_t \oplus R2'_t = (*, \ldots, *, 0, 1)$. This fault is identically visible by comparing the output z'_t of the faulty generator by the original output z_t.

Now the memory registers are updated to

$$\begin{aligned} R2_{t+1} &= \mathcal{S}(R1_t) \\ R1_{t+1} &= R2_t + s_{t+5} \\ R1'_{t+1} &= R2'_t + s_{t+5} \end{aligned}$$

In $R1'_{t+1}$ a fault is induced that becomes visible by comparing z_{t+1} and z'_{t+1} (in $R2_{t+1}$ no difference has been induced). Thus we have

$z_{t+1} = (s_{t+16} + R2_t + s_{t+5}) \oplus R2_{t+1} \oplus s_{t+1}$ and $z'_{t+1} = (s_{t+16} + R2'_t + s_{t+5}) \oplus R2_{t+1} \oplus s_{t+1}$. Focus now on $S_1 = s_{t+16} + R2_t + s_{t+5} = s_{t+16} + s_{t+5} + R2_t$ and $S_2 = s_{t+16} + R2'_t + s_{t+5} = s_{t+16} + s_{t+5} + R2'_t$. Then the XOR of S_1 and S_2 agrees with that of z_{t+1} and z'_{t+1}. There are now two cases: Either the second last bit of S_1 agrees with that of S_2. Then $lsb(s_{t+16} + s_{t+5}) = 0$, otherwise $lsb(s_{t+16} + s_{t+5}) = 1$. In the first case there is thus no change of the carry, and in the second case there is one. Anyway, this gives a linear equation in the LFSR's state bits.

Carry out these steps for at least $512 = 32 \cdot 16$ different clock, until we have sufficiently many linear equations, so that we can solve for the contents of the LFSR. Observe that some linear equations might be linearly dependent, so that it might be necessary to consider more clocks. As in average every 4th fault induced in $R2_t$ has the required form and as we can use only every second output, we need at least $2 \cdot 4 \cdot 512 = 2^{12}$ output words produced by the correct and 2^{12} output words produced by the faulted generator.

One can modify and further relax the conditions on the fault induced, e.g., one could apply this attack on the 2nd lowest bit. Then one obtains quadratic instead of linear equations.

6 Conclusion

We have shown that fault attacks can be successfully applied to LFSR-based stream ciphers that use a combiner with memory. This is in particular demonstrated by fault attacks against the stream cipher E0 used in Bluetooth and against the stream cipher SNOW 2.0. As our general attack proposed in Section 3 relies on computing the set of possible inputs $X_{(z_1,\ldots,z_r)}$, an obvious countermeasure would be to choose the number k of inputs as well as the number ℓ of memory bits of the output function not too small (e.g., $k, \ell \geq 10$). Furtheron, the attack described in Section 4.3 would fail if LFSRs of greater length were used. Together with the results from [12], it seems to be reasonable to use few long LFSRs instead of many but short ones. Indeed, our attack on SNOW 2.0 shows that these countermeasures are not sufficient. As a result, it would be interesting to see if better countermeasures against fault attacks on stream ciphers exist. In particular, it seems open whether fault attacks apply in general against stream ciphers which are not based on LFSR's and whose driving devices are nonlinear.

Acknowledgement

The second author receives partial funding through GEBERT RÜF STIFTUNG. We would like to thank the anonymous reviewers which helped to improve the presentation and Li Yu and Stefan Lucks for helpful discussions.

References

1. Armknecht, Krause: *Algebraic Attacks on Combiners with Memory*, Crypto 2003, LNCS 2729, pp. 162-176, Springer, 2003.
2. Armknecht, Lano, Preneel: *Extending the resynchronization attack*, SAC 2004, LNCS 3357, pp. 19-38, Springer, 2004.
3. Armknecht: *On the existence of low-degree equations for algebraic attacks*, Cryptology ePrint Archive: Report 2004/185.
4. Biham, Granboulan, Nguyen: Impossible Fault Analysis of RC4 and Differential Fault Analysis of RC4, FSE 2005, Springer, 2005.
5. Biryukov, De Cannière: *Block Ciphers and Systems of Quadratic Equations*, FSE 2003, LNCS 2887, pp. 274-289, Springer, 2003.
6. Boneh, DeMillo, R.J. Lipton: *On the Importance of Checking Cryptographic Protocols for Faults*, Eurocrypt 1997, LNCS 1233, pp. 37-51, Springer, 1997.
7. Biham, Shamir: *Differential fault analysis of secret key cryptosystems*, Crypto 1997, LNCS 1294, pp. 513-525, Springer, 1997.
8. Courtois, Pieprzyk: *Cryptanalysis of block ciphers with overdefined systems of equations*, Asiacrypt 2002, LNCS 2501, pp. 267-287, Springer, 2002.
9. Ekdahl, Johansson: *A new version of the stream cipher SNOW*, SAC 2002, LNCS 2595, pp. 47-61, Springer, 2002.
10. Ekdahl: *On LFSR-based Stream Ciphers - Analysis and Design*, Ph.D. Thesis, Lund University, Sveden, November 21, 2003.
11. Golić, Bagini, Morgari: *Linear Cryptanalysis of Bluetooth Stream Cipher*, Eurocrypt 2002, LNCS 2332, pp. 238-255, Springer 2002.

12. Hoch, Shamir: *Fault Analysis of Stream Ciphers*, CHES 2004, LNCS 3156, pp. 240-253, Springer, 2004.
13. Lu, Vaudenay: *Faster Correlation Attack on the Bluetooth Keystream Generator*, Crypto 2004, LNCS 3152, pp. 407-425, Springer, 2004.
14. Lu, Vaudenay: *Cryptanalysis of Bluetooth Keystream Generator Two-Level E0*, Asiacrypt 2004, LNCS 3329, pp. 483-499, Springer, 2004.
15. Lu, Vaudenay, Meier: *The Conditional Correlation Attack: A Practical Attack on Bluetooth Encryption*, Crypto 2005, to appear.
16. Meier, Pasalic, Carlet: *Algebraic attacks and decomposition of Boolean functions*, Eurocrypt 2004, LNCS 3027, pp. 474-491, Springer, 2004.
17. Skorobogatov, Anderson, *Optical Fault Induction Attacks*, CHES 2002, LNCS 2523, pp. 2-12, Springer, 2002.

A An Example for Fault Attack on the Summation Generator with $k = 2$ Inputs

In this section we illustrate the general fault attack described in section 3 on the summation generator with $k = 2$ inputs $X_t = (a_t, b_t)$ and $\ell = 1$ memory bits from section 2. By theorem 1, not all outputs $(z_t, z_{t+1}) \in \{0,1\}^2$ can be produced by all 16 possible inputs $(X_t, X_{t+1}) \in \{0,1\}^4$. Due to the small values of k and ℓ, it is possible to exhaustively compute all possible input-output combinations over 2 clocks. From this, it is easy to derive for a given output (z_t, z_{t+1}) the set of possible inputs.

$$X_{0,0} = \{(0000), (0011), (1101), (1110), (0101), (0110), (1001), (1010)\}$$
$$X_{0,1} = \{(0001), (0010), (1100), (1111), (0100), (0111), (1000), (1011)\}$$
$$X_{1,0} = \{(0100), (0111), (1000), (1011), (0000), (0011), (1101), (1110)\}$$
$$X_{1,1} = \{(0101), (0110), (1001), (1010), (0001), (0010), (1100), (1111)\}$$

Note that each output (z_t, z_{t+1}) can be produced by only half of the 16 possible inputs $(a_t, b_t, a_{t+1}, b_{t+1})$, i.e., the property described in theorem 1 holds for each possible output. Any observed output segment (z_t, z_{t+1}) provides the information that $(X_t, X_{t+1}) \in X_{(z_t, z_{t+1})}$. It is easy to see that for each (z_t, z_{t+1}), no linear functions $g \not\equiv 0$ exists with $g(X_t, X_{t+1}) = 0$ for all $(X_t, X_{t+1}) \in X_{(z_t, z_{t+1})}$. More specifically, it is not possible for the adversary to derive a linear function $L : \{0,1\}^4 \to \{0,1\}$ such that $L(X_t, X_{t+1}) = 0$ holds.

In the fault attack scenario, the adversary can now re-run the keystream generator with the same initial setting and induce a unknown fault in the memory register during the run. Let $(z'_t, z'_{t+1}, \ldots)$ be the keystream of this second disturbed run, also observed by the adversary. He cannot choose the point in time t' of the disturbance, but estimate it by comparing the two keystreams $(z_1, \ldots)$ and $(z'_1, \ldots)$. Suppose that the fault is induced in $Q_{t'}$ with $t' \leq t$ and a different keystream segment $(z'_t, z'_{t+1}) \neq (z_t, z_{t+1})$ has occurred. Hence, it is $X_t \in X_{(z_t, z_{t+1})} \cap X_{(z'_t z'_{t+1})}$ and the number of possible assignments of X_t is further reduced. For example, if $(z_t, z_{t+1}) = (0,0)$ and $(z'_t, z'_{t+1}) = (1,0)$, then

$$X_t^{t+1} = (a_t, b_t, a_{t+1}, b_{t+1}) \in X_{(0,0)} \cap X_{(1,0)} = \{(0000), (0011), (1101), (1110)\}.$$

X_t^{t+1} is still not uniquely determined, but the adversary has gained the additional information that $a_t \oplus b_t = 0$ is true in any case. Repeating these steps for several clocks t, it is possible to derive a system of linear equations in X_t and hence in K. By computing the solution, it is easily possible to determine the unknown initial setting K.

B Examples Concerning Section 4.3

B.1 Example for Computing $\Sigma_\Gamma(Z, Z'\Delta)$

We demonstrate on an example how the value $\Sigma(\Delta)$ is computed. Let $\Gamma = (\gamma_0, \gamma_1, \gamma_2) := (1, 1, 0, 1)$. Now we consider the following situation:

t	1	2	3	4	5	6	7	8	9	10
z_t	1	1	0	0	0	1	1	1	0	0
z'_t	0	1	1	0	1	1	0	1	1	1
δ_t	0	0	1	1	1	0	1	0	0	1

By definition, $\Sigma_\Gamma(Z, Z', \Delta)$ is the number of clocks $1 \leq t \leq t$ with

$$(z_t \oplus z'_t) \oplus (z_{t+1} \oplus z'_{t+1}) \oplus (z_{t+3} \oplus z'_{t+3}) = \delta_t \oplus \delta_{t+1} \oplus \delta_{t+3}.$$

In this case, the values of $\bigoplus \gamma_i(z_t \oplus z'_t \oplus \delta_t)$ for $t = 1, \ldots, 7$ are 0, 0, 1, 1, 0, 1 and 0, respectively. Hence, it is $\Sigma_\Gamma(Z, Z', \Delta) = 4$.

B.2 Example for Computing the Index Ind(Δ)

In our example we assume that altogether 8 different possible faults Δ exist, denoted by $\Delta_0, \ldots, \Delta_7$. We use four different coefficient vectors $\Gamma_1, \ldots, \Gamma_4$ and consider for each of them the three Δ's having the highest values $\Sigma_{\Gamma_i}(Z, Z', \Delta)$ (see also previous section). We abbreviate $\Sigma_{\Gamma_i}(Z, Z', \Delta_j)$ by Σ_i^j.

Assume that computing the values Σ_i^j gives the following orderings:

$$\begin{array}{llll}
\Sigma_1^7 < \Sigma_1^0 < \Sigma_1^3 < \Sigma_1^4 < \Sigma_1^2 < & \Sigma_1^1 < \Sigma_1^6 < \Sigma_1^5 \\
\Sigma_2^1 < \Sigma_2^5 < \Sigma_2^0 < \Sigma_2^3 < \Sigma_2^7 < & \Sigma_2^6 < \Sigma_2^2 < \Sigma_2^4 \\
\Sigma_3^2 < \Sigma_3^0 < \Sigma_3^1 < \Sigma_3^4 < \Sigma_3^5 < & \Sigma_3^7 < \Sigma_3^5 < \Sigma_3^3 \\
\Sigma_4^7 < \Sigma_4^2 < \Sigma_4^5 < \Sigma_4^1 < \Sigma_4^3 < & \Sigma_4^0 < \Sigma_4^4 < \Sigma_4^6
\end{array}$$

The first Top-3-list $L(\Gamma_1)$ contains the three Δ_j having the highest values of Σ_1. In this case, these are Δ_1, Δ_6 and Δ_5. $L(\Gamma_2)$, $L(\Gamma_3)$ and $L(\Gamma_4)$ are defined analogously. It is $L(\Gamma_1) = [\Delta_1, \Delta_6, \Delta_5]$, $L(\Gamma_2) = [\Delta_6, \Delta_2, \Delta_4]$, $L(\Gamma_3) = [\Delta_7, \Delta_5, \Delta_3]$ and $L(\Gamma_4) = [\Delta_0, \Delta_4, \Delta_6]$.

The next step consists in counting for each Δ_j its number of occurrences in $L(\Gamma_1), \ldots, L(\Gamma_4)$, called its index. As Δ_6 has the highest index, namely 3, it is an apparent candidate for the unknown guess. But simulations showed that this strategy results in wrong guesses in significantly many cases. Hence, we imposed a stronger condition for a guess: the index of the guess must be at least two higher than the other indices.

New Observation on Camellia

Duo Lei[1], Li Chao[2], and Keqin Feng[3]

[1] Department of Science, National University of Defense Technology, Changsha, China
Duoduolei@163.com
[2] Department of Science, National University of Defense Technology, Changsha, China
[3] Department of Math, Tsinghua University, Beijing, China

Abstract. In this paper, some observations on Camellia are presented, by which the Square attack and the Collision attack are improved. 11-round 256-bit Camellia without FL function is breakable with complexity of 2^{250} encryptions. 9-round 128-bit Camellia without FL function is breakable with the complexity of 2^{90} encryptions. And 10-round 256-bit Camellia with FL function is breakable with the complexity of 2^{210} encryptions and 9-round 128-bit Camellia with FL function is breakable with the complexity of 2^{122} encryptions. These results are better than any other known results. It concludes that the most efficient attack on Camellia is Square attack.

1 Introduction

Camellia [1] is a 128-bit block cipher proposed by NTT and Mitsubishi in 2000. It has the modified Feistel structure with irregular rounds, which is called the FL/FL^{-1} function layers. Camellia had been submitted to the standardization and the evaluation projects such as ISO/IEC JTC 1/SC 27, CRYPTREC, and NESSIE.

The most efficient methods analyzing Camellia are truncated differential cryptanalysis[4][5][6] and higher order differential attack[7][9]. Camellia with more than 11 rounds is secure against truncated differential cryptanalysis. Square attack[11] is a most efficient attack on AES[11][12] . Y. He and S. Qing [2] showed that 6-round Camellia is breakable by it. Y.Yeom, S. Park, and I. Kim [3] improved the result to 8 rounds. Collision attack on Camellia was presented by WL Wu[10].

In this paper, some observations on Camellia are presented, by which the Square attack and the Collision attack are improved. Variant Square Attack can break 11-round 256-bit Camellia without FL function with complexity of 2^{250} encryptions. 9-round 128-bit Camellia without FL function is breakable with the complexity of 2^{90} encryptions. And 10-round 256-bit Camellia with FL function is breakable with the complexity of 2^{210} encryptions and 9-round 128-bit Camellia with FL function is breakable with the complexity of 2^{122} encryptions. These results are better than any other known results.

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 51–64, 2006.

Brief description of Camellia and some new structures equivalent to Camellia are presented in section2. In section 3, active bytes transformations on Camellias are illustrated and some new properties are demonstrated. Our attacking methods are described in section 4. Section 5 is some extension. The paper concludes with our most important results.

2 Equivalent Structures of Camellia

2.1 Description of the Camellia

Camellia has a 128-bit block size and supports 128-, 192- and 256-bit keys. Camellia with a 128-bit key and 256-bit key is written as 128-Camellia, 256-Camellia. The design of Camellia is based on the Feistel structure and its number of rounds is 18(128-bit key) or 24(192-, 256-bit key). The FL/FL^{-1} function layer is inserted in it every 6 rounds in order to thwart future unknown attacks. Before the first round and after the last round, there are pre- and post-whitening layers. F function contains key-addition, S-function and P-function. S-function contains 4 types of S-boxes s_1, s_2, s_3, and s_4. s_2,s_3,s_4 are variations of s_1. The P-function:$\{0,1\}^{64} \mapsto \{0,1\}^{64}$ maps $(z_1, ..., z_8)$ to $(z'_1, ..., z'_8)$, defined as:

$$
\begin{aligned}
z'_1 &= z_1 \oplus z_3 \oplus z_4 \oplus z_6 \oplus z_7 \oplus z_8 \\
z'_2 &= z_1 \oplus z_2 \oplus z_4 \oplus z_5 \oplus z_7 \oplus z_8 \\
z'_3 &= z_1 \oplus z_2 \oplus z_3 \oplus z_5 \oplus z_6 \oplus z_8 \\
z'_4 &= z_2 \oplus z_3 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7 \\
z'_5 &= z_1 \oplus z_2 \oplus z_6 \oplus z_7 \oplus z_8 \\
z'_6 &= z_2 \oplus z_3 \oplus z_5 \oplus z_7 \oplus z_8 \\
z'_7 &= z_3 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_8 \\
z'_8 &= z_1 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7
\end{aligned}
$$

We refer $X_{(r)}, K_{(r)}$ to the rth round input and subkey, refer $X^L_{(r)}$ and $X^R_{(r)}$ to the left, right half bytes of $X_{(r)}$, which implies $X_{(r)} = (X^L_{(r)}, X^R_{(r)})$. Let $X_{(ri)}$ is the ith byte of $X_{(r)}$, PL and CP be the Plaintext and Ciphertext, and $X_{(L)}$ be the last round output. The round function of Camellia is written as follows (named as Camellia-1) , which is shown in Fig. 1:

$$
\begin{gathered}
X^L_{(1)} = PL^L, X^R_{(1)} = PL^R, \\
X^L_{(r+1)} = X^R_{(r)} \oplus P(s(X^L_{(r)} \oplus K_{(r)})), \\
X^R_{(r+1)} = X^L_{(r)}, \\
CP^L = X^L_{(L)}, CP^R = X^R_{(L)}
\end{gathered}
$$

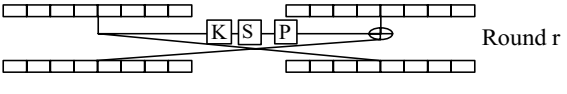


Fig. 1. Round function of Camellia-1

2.2 Three Equivalent Structures of Camellia

We can write Camellia in following form called Camellia-2, where P^{-1} is the inverse transformation of P-function and $\bar{X}_{(r)}$ is the rth round input of Camellia-2. Figure illustration of Camellia-2 is given in Fig. 2:

$$\bar{X}^L_{(1)} = P^{-1}(PL^L), \bar{X}^R_{(1)} = P^{-1}(PL^R),$$
$$\bar{X}^L_{(r+1)} = \bar{X}^R_{(r)} \oplus s(P(\bar{X}^L_{(r)}) \oplus K_{(r)}),$$
$$\bar{X}^R_{(r+1)} = \bar{X}^L_{(r)},$$
$$CP^L = P(\bar{X}^L_{(L)}), CP^R = P(\bar{X}^R_{(L)})$$

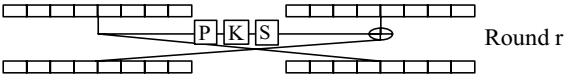


Fig. 2. Round function of Camellia-2

We can also write Camellia in the form of Camellia-3 where $\hat{X}_{(r)}$ is the rth round input. Figure illustration is given in Fig. 3:

$$\hat{X}^L_{(1)} = PL^L, \hat{X}^R_{(1)} = P^{-1}(PL^R),$$
$$\hat{X}^L_{(r+1)} = \hat{X}^R_{(r)} \oplus s(\hat{X}^L_{(r)} \oplus K_{(r)}), \text{ where r is odd}$$
$$\hat{X}^L_{(r+1)} = P(\hat{X}^R_{(r)} \oplus s(P(\hat{X}^L_{(r)}) \oplus K_{(r)})), \text{ where r is even}$$
$$\hat{X}^R_{(r+1)} = \hat{X}^L_{(r)},$$
$$CP^L = \hat{X}^L_{(L)}, CP^R = P(\hat{X}^R_{(L)})$$

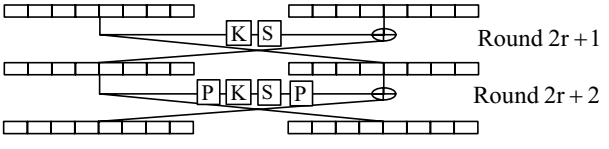


Fig. 3. Round function of Camellia-3

The structure of Camellia-4 is given as follows where $\tilde{X}_{(r)}$ is the rth round input of that. Figure illustration is given in Fig. 4:

$$\tilde{X}^L_{(1)} = P^{-1}(PL^L), \tilde{X}^R_{(1)} = PL^R,$$
$$\tilde{X}^L_{(r+1)} = P(\tilde{X}^R_{(r)} \oplus s(P(\tilde{X}^L_{(r)}) \oplus K_{(r)})), \text{ where r is odd}$$
$$\tilde{X}^L_{(r+1)} = \tilde{X}^R_{(r)} \oplus s(\tilde{X}^L_{(r)} \oplus K_{(r)}), \text{ where r is even}$$
$$\tilde{X}^R_{(r+1)} = \tilde{X}^L_{(r)},$$
$$CP^L = P(\tilde{X}^L_{(L)}), CP^R = \tilde{X}^R_{(L)}$$

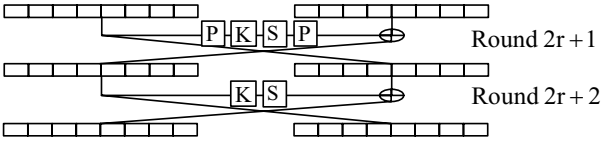


Fig. 4. Round function of Camellia-4

3 New Observations on Camellia

3.1 Preliminaries

Let a Λ-set be a set of 256 states that are all different in some of the state bytes (the active) and all equal in the other state bytes (the passive). We have:

$$\forall x, y \in \Lambda : \{ \begin{matrix} x_i \neq y_i & if\ x_i\ is\ active \\ x_i = y_i & else \end{matrix}$$

Let Γ-set be a set of 256 states that are all equal to zero in summation (the balanced).

$$\forall x \in \Gamma : \sum x_i = 0$$

Applying the S-function or Key-addition on a Λ-set results in a Λ-set with the positions of the active bytes unchanged. The result set of applying P-function to a Λ-set is not always a Λ-set but always a Γ-set.

Applying Key-addition or P-function on a Γ-set results in a Γ-set. Applying S-function on a Γ-set results in the active bytes and passive bytes are still balanced. Applying AND operation, OR operation or right shift operation on a Γ-set results in a Γ-set.

Here, we give some definitions that are used in following sections.

$\mathcal{F}$: A $\Lambda - set$ has the form of $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, \alpha_8, i, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \beta_8\}$, in which α_i, β_j are constant, $i \in \{0, .., 255\}$.

$\mathcal{F}_t$: A $\Lambda - set$ has the form of $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, \alpha_8, i, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \gamma_t\}_t$, in which α_i, β_j, γ_k are constant, $i \in \{0, .., 255\}$.

$\widetilde{\mathcal{F}}_t$: A $\Lambda - set$ has the form of $\{i, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \gamma_t, s_1(i \oplus k_1), \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, s_1(\gamma_t \oplus k_2)\}_t$, in which α_i, β_j, γ_k, k_1, k_2 are constant, $i \in \{0, .., 255\}$.

$\widehat{\mathcal{F}}_t$: A $\Lambda - set$ has the form of $\{\beta_1, i \oplus \beta_2, i \oplus \beta_3, i \oplus \beta_4, i \oplus \beta_5, \beta_6, \beta_7, i \oplus \gamma_t, s_1(i \oplus k_1), s_1(i \oplus k_1) \oplus \alpha_2, s_1(i \oplus k_1) \oplus \alpha_3, \alpha_4, s_1(i \oplus k_1) \oplus \alpha_5, \alpha_6, \alpha_7, s_1(i \oplus k_1) \oplus \alpha_8\}_t$, in which α_i, β_j, γ_k, k_1 are constant, $i \in \{0, .., 255\}$.

$\widetilde{\widetilde{\mathcal{F}}}_t$: A $\Lambda - set$ has the form of $\{s_1(i \oplus k_1), \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, s_1(\gamma_t \oplus k_2), \Delta_1 \oplus i, \Delta_2, \Delta_3, \Delta_4, \Delta_5, \Delta_6, \Delta_7, \Delta_8 \oplus \gamma_t\}$, in which $\{\Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5, \Delta_6, \Delta_7, \Delta_8\}$ satisfy Eq.(1), α_i, β_j, γ_k, η_1, η_2, η_3, η_4, η_5, η_6, η_7, η_8 k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9 are constant, $i \in \{0, .., 255\}$.

$$\begin{bmatrix} \Delta_1 \\ \Delta_2 \\ \Delta_3 \\ \Delta_4 \\ \Delta_5 \\ \Delta_6 \\ \Delta_7 \\ \Delta_8 \end{bmatrix} = \begin{bmatrix} 1\,0\,1\,1\,0\,1\,1\,1 \\ 1\,1\,0\,1\,1\,0\,1\,1 \\ 1\,1\,1\,0\,1\,1\,0\,1 \\ 0\,1\,1\,1\,1\,1\,1\,0 \\ 1\,1\,0\,0\,0\,1\,1\,1 \\ 0\,1\,1\,0\,1\,0\,1\,1 \\ 0\,0\,1\,1\,1\,1\,0\,1 \\ 1\,0\,0\,1\,1\,1\,1\,0 \end{bmatrix} \begin{bmatrix} s_1(s_1(i \oplus k_1) \oplus \eta_1 \oplus s_1(\gamma_t \oplus k_2)) \\ s_2(s_1(i \oplus k_1) \oplus \eta_2 \oplus s_1(\gamma_t \oplus k_2)) \\ s_3(s_1(i \oplus k_1) \oplus \eta_3 \oplus s_1(\gamma_t \oplus k_2)) \\ s_4(\eta_4) \\ s_2(s_1(i \oplus k_1) \oplus \eta_5 \oplus s_1(\gamma_t \oplus k_2)) \\ s_3(\eta_6 \oplus s_1(\gamma_t \oplus k_2)) \\ s_4(\eta_7 \oplus s_1(\gamma_t \oplus k_2)) \\ s_1(s_1(i \oplus k_1) \oplus \eta_8) \end{bmatrix} \quad (1)$$

$\hat{\hat{\mathcal{F}}}_t$: A $\Lambda - set$ has the form of $\{s_1(i \oplus k_1),\ s_1(i \oplus k_1) \oplus \alpha_2,\ s_1(i \oplus k_1) \oplus \alpha_3,\ \alpha_4,\ s_1(i \oplus k_1) \oplus \alpha_5,\ \alpha_6,\ \alpha_7,\ s_1(i \oplus k_1),\ s_1(s_1(i \oplus k_1) \oplus k_2),\ s_2(s_1(i \oplus k_1) \oplus k_3) \oplus i,\ s_3(s_1(i \oplus k_1) \oplus k_4) \oplus i,\ \alpha_4 \oplus i,\ s_2(s_1(i \oplus k_1) \oplus k_5) \oplus i,\ \alpha_6,\ \alpha_7,\ s_1(s_1(i \oplus k_1) \oplus k_6) \oplus i \oplus \gamma_t)\}_t$, in which α_i, β_j, γ_k, k_1, k_2, k_3, k_4, k_5, k_6 are constant, $i \in \{0, .., 255\}$.

In next section, we trace the position changes of the active bytes through 5 rounds transformations of Camellia-1∼4 and the plaintext set $\{PL\}$ is a $\Lambda - set$ $\mathcal{F}$. We just describe the evolution of left half bytes of round outputs, for the left half bytes of previous round pass to right half bytes of next round unchanged.

3.2 Active Bytes Changing Properties

In Camellia-1, $X^R_{(1,1)}$ is active byte, 1st round transformations convert the active byte to $X^L_{(2,1)}$, other bytes are passive. In 2nd round transformations, P-function converts the active byte to 5 active bytes which are $X^L_{(31)}$, $X^L_{(32)}$, $X^L_{(33)}$, $X^L_{(35)}$, $X^L_{(38)}$ and three passive bytes. In 3rd round transformation, P-function converts the active bytes to 8 balanced bytes. In 4th round transformation, S-function converts balanced bytes to unbalanced bytes. After 5th round transformation all bytes are unbalanced. Evolutions of active bytes are illustrated in Fig. 5.

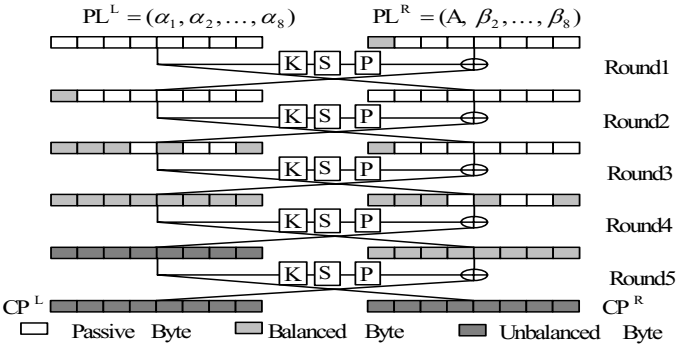


Fig. 5. Round function of Camellia-1

In Camellia-2, 1st byte of $\{PL\}$ is active, the pre-P^{-1}-function converts the active byte to 5 active bytes. 1st round transformations convert the 5 active bytes and 3 passive bytes to 5 active bytes and 3 passive bytes. 2nd round

transformations convert the 5 active bytes and 3 passive bytes to 1 active byte and 7 passive bytes. 3rd round transformation convert the 5 active bytes and 3 passive bytes to 2 active bytes, 4 balanced bytes and 2 passive bytes, where $\bar{X}^L_{(41)}, \bar{X}^L_{(42)}$ are active, $\bar{X}^L_{(42)}, \bar{X}^L_{(43)}, \bar{X}^L_{(45)}, \bar{X}^L_{(48)}$ are balanced and $\bar{X}^L_{(46)}, \bar{X}^L_{(47)}$ are passive. 4th round transformations convert those bytes to unbalanced bytes. Evolutions of active bytes are illustrated in Fig. 6.

Details of 2nd and 3rd transformation are given in Eq.(2)and Eq.(3).

$$\begin{aligned}
\bar{X}^L_{(3)} &= \bar{X}^R_{(2)} \oplus s(P(\bar{X}^L_{(2)} \oplus K_{(2)}) \\
&= \bar{X}^L_{(1)} \oplus s(P(\bar{X}^R_{(1)} \oplus s(P(\bar{X}^L_{(1)}) \oplus K_{(1)})) \oplus K_{(2)}) \\
&= \bar{X}^L_{(1)} \oplus s(P(P^{-1}(PL^R) \oplus s(P(\bar{X}^L_{(1)}) \oplus K_{(1)})) \oplus K_{(2)}) \\
&= \bar{X}^L_{(1)} \oplus s((PL^R \oplus s(P(\bar{X}^L_{(1)}) \oplus K_{(1)})) \oplus K_{(2)})
\end{aligned} \tag{2}$$

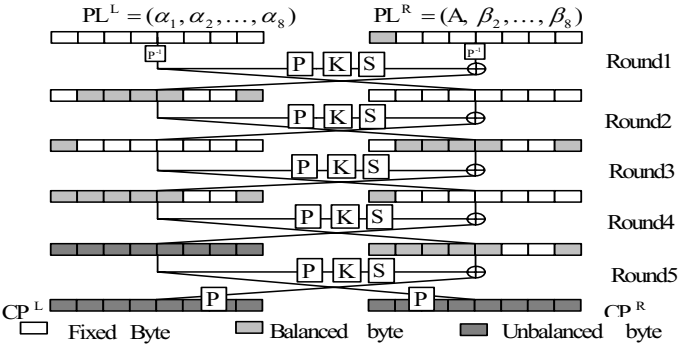


Fig. 6. Round function of Camellia-2

$\bar{X}^L_{(11)}$ is the only active byte in $\{\bar{X}^L_{(1)}\}$, since applying addition and S-function on it results in $\{\bar{X}^L_{(3)}\}$ with position of active byte unchanged, demonstrated in Eq.(2). Each byte of $\bar{X}^L_{(4)}$ can be written in the form of Eq.(3), in which $\bar{X}^L_{(41)}, \bar{X}^L_{(44)}$ are influenced by an active byte $\bar{X}^L_{(31)}$ so active, $\bar{X}^L_{(42)}$, $\bar{X}^L_{(43)}$, $\bar{X}^L_{(45)}$, $\bar{X}^L_{(48)}$ are influenced by 2 active bytes thus balanced and $\bar{X}^L_{(46)}, \bar{X}^L_{(47)}$ are derived from passive bytes, still passive.

$$\begin{aligned}
\bar{X}^L_{(41)} &= s(\bar{X}^L_{(31)} \oplus \bar{X}^L_{(33)} \oplus \bar{X}^L_{(34)} \oplus \bar{X}^L_{(36)} \oplus \bar{X}^L_{(37)} \oplus \bar{X}^L_{(38)} \oplus K_{(31)}) \oplus \bar{X}^R_{(31)} \\
\bar{X}^L_{(42)} &= s(\bar{X}^L_{(31)} \oplus \bar{X}^L_{(32)} \oplus \bar{X}^L_{(34)} \oplus \bar{X}^L_{(35)} \oplus \bar{X}^L_{(37)} \oplus \bar{X}^L_{(38)} \oplus K_{(32)}) \oplus \bar{X}^R_{(32)} \\
\bar{X}^L_{(43)} &= s(\bar{X}^L_{(31)} \oplus \bar{X}^L_{(32)} \oplus \bar{X}^L_{(33)} \oplus \bar{X}^L_{(35)} \oplus \bar{X}^L_{(36)} \oplus \bar{X}^L_{(38)} \oplus K_{(33)}) \oplus \bar{X}^R_{(33)} \\
\bar{X}^L_{(44)} &= s(\bar{X}^L_{(32)} \oplus \bar{X}^L_{(33)} \oplus \bar{X}^L_{(34)} \oplus \bar{X}^L_{(35)} \oplus \bar{X}^L_{(36)} \oplus \bar{X}^L_{(37)} \oplus K_{(34)}) \oplus \bar{X}^R_{(34)} \\
\bar{X}^L_{(45)} &= s(\bar{X}^L_{(31)} \oplus \bar{X}^L_{(32)} \oplus \bar{X}^L_{(36)} \oplus \bar{X}^L_{(37)} \oplus \bar{X}^L_{(38)} \oplus K_{(35)}) \oplus \bar{X}^R_{(35)} \\
\bar{X}^L_{(46)} &= s(\bar{X}^L_{(32)} \oplus \bar{X}^L_{(33)} \oplus \bar{X}^L_{(35)} \oplus \bar{X}^L_{(37)} \oplus \bar{X}^L_{(38)} \oplus K_{(36)}) \oplus \bar{X}^R_{(36)} \\
\bar{X}^L_{(47)} &= s(\bar{X}^L_{(33)} \oplus \bar{X}^L_{(34)} \oplus \bar{X}^L_{(35)} \oplus \bar{X}^L_{(36)} \oplus \bar{X}^L_{(38)} \oplus K_{(37)}) \oplus \bar{X}^R_{(37)} \\
\bar{X}^L_{(48)} &= s(\bar{X}^L_{(31)} \oplus \bar{X}^L_{(34)} \oplus \bar{X}^L_{(35)} \oplus \bar{X}^L_{(36)} \oplus \bar{X}^L_{(37)} \oplus K_{(38)}) \oplus \bar{X}^R_{(38)}
\end{aligned} \tag{3}$$

In Camellia-3, 1st byte of $\{PL\}$ is active, the pre-P-function converts the active byte to 5 active bytes. 1st round transformations convert the 5 active

bytes, 3 passive bytes to 5 active bytes,3 passive bytes. 2nd round transformations convert the 5 active bytes, 3 passive bytes to 5 active bytes, 3 active bytes. 3rd round transformation convert 5 active bytes, 3 passive bytes to 2 active bytes, 4 balanced bytes and 2 passive bytes, where $\hat{X}^L_{(41)}, \hat{X}^L_{(42)}$ are active, $\hat{X}^L_{(42)}, \hat{X}^L_{(43)}, \hat{X}^L_{(45)}, \hat{X}^L_{(48)}$ are balanced and $\hat{X}^L_{(46)}, \hat{X}^L_{(47)}$ are passive.And 4th round transformations convert those bytes to unbalanced bytes. The deducing procedure is similar to that of Camellia-2. Figure illustration is given in Fig.7.

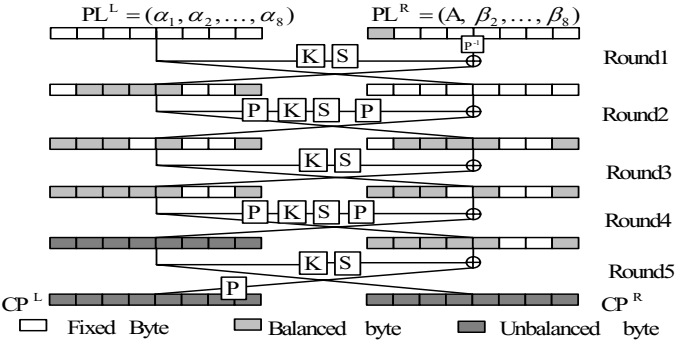


Fig. 7. Round function of Camellia-3

The outstanding properties of Camellia-3 are Eq.(4) and Eq.(5), that are used in Improved Square attack and Improved Collision attack in section 4.2 and section 4.3.

$$\sum_{PL\in\mathcal{F}} \hat{X}^R_{(5i)} = 0 \Rightarrow \sum_{PL\in\mathcal{F}} (s(\hat{X}^R_{(6i)} \oplus K_{(5i)}) \oplus \hat{X}^L_{(6i)} = 0, 6 \le i \le 7. \quad (4)$$

$$\hat{X}^R_{(5i)} \equiv C \Rightarrow s(\hat{X}^R_{(6i)} \oplus K_{(5i)}) \oplus \hat{X}^L_{(6i)} \equiv C, PL \in \mathcal{F}, 1 \le i \le 8. \quad (5)$$

Camellia-3 also have the property of Eq.(6) that is used in section 4.4.

$$\hat{X}^R_{(5i)} = B \Rightarrow s(\hat{X}^R_{(6i)} \oplus K_{(5i)}) \oplus \hat{X}^L_{(6i)} = B,\ PL \in \mathcal{F},\ B\ is\ active, i \in \{1, 4\}. \quad (6)$$

In Camellia-4, $X^R_{(11)}$ is active, 1st round transformations convert the active byte to an active byte, 2nd round transformations convert the active byte to an active byte and 3rd round transformations convert the active byte to 8 active bytes. Figure illustration is shown in Fig.8. The outstanding property of Camellia-4 is that seven 3rd round output bytes are passive, which will be used in variant Square attack in section 4.1.

$$\textstyle\sum_{PL\in\mathcal{F}} \tilde{X}^L_{(4i)} \oplus K_{(4i)} = 0, i \ne 1, \tilde{C}_i\ is\ a\ constant.$$

$$\Rightarrow \sum_{PL\in\mathcal{F}} (s^{-1}(\tilde{X}^L_{(5i)}) \oplus \tilde{C}_i) = 0, i \ne 1, \tilde{C}_i\ is\ a\ constant. \quad (7)$$

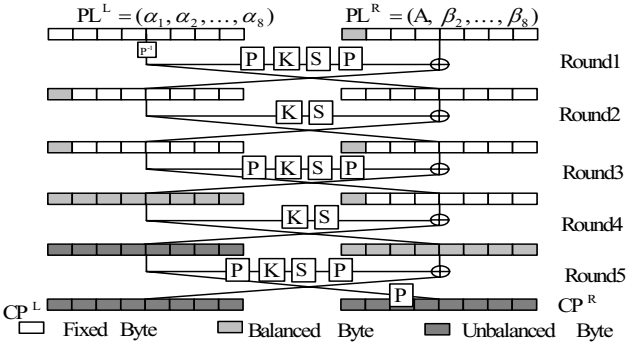


Fig. 8. Round function of Camellia-4

4 Some Attacks

In this section, we construct the attacks on Camellia without pre-, post- whitening and FL/FL^{-1} function. The influences of FL/FL^{-1} function are discussed in section 5.

4.1 Variant Square Attack

The 6-round variant Square attack use the property of Eq.(8), which is derives from Eq.(7), and use the structure of Camellia-4. This attack can be described by the following steps.

$$\sum_{PL \in \mathcal{F}_t} (s^{-1}(s(\tilde{X}^{(R)}_{(7i)} \oplus K_{(6i)}) \oplus \tilde{X}^{L}_{(7i)} \oplus \tilde{C}_i) = 0, \tilde{C}_i \textit{ is a constant}, i \neq 1. \quad (8)$$

- Step 1: Select the Plaintext sets $\{PL\}_t$ as $\{PL\}_t = \mathcal{F}_t, 1 \leq t \leq 3$, calculate the values of $(X^L_{(7)}, X^R_{(7)})$ and record them, which will be used in following steps.
- Step 2: Guess $k_{(65)}$ and $\tilde{C}_5$, then check whether Eq.(8) is satisfied or not, where the Plaintext set is $\mathcal{F}_1$. If Eq.(8) is satisfied, record the values of $k_{(65)}$ and $\tilde{C}_5$ that may be the correct pair. Step 2 is ended until all possible values are checked.
- Step 3: For all 'correct' pairs from step 2, checks whether Eq.(8) is satisfied or not, where the Plaintext set are $\mathcal{F}_t, 2 \leq t \leq 3$. ($\gamma_t$ does not influence the value of $\tilde{C}_5$).

In this 6-round attack, the time that step 1 takes is 3×2^8 6-round encryptions takes. Since Eq.(8) has $256 + 256 \times 3$ additions, 256×2 substitutions, and 6-round Camellia has 44×6 additions, 8×6 substitutions, the time Eq.(8) takes is nearly 6 times of that 6-round encryptions of Camellia take. In step 2, Eq.(8) repeats $2^8 \times 2^8$ times. The probability of wrong key passing the checking is 2^8, so there is 2^8 pairs passing the checking in step 2, that implies the time of step

3 and 4 take $2^8 \times 3$ and 3 times of that 6-round encryptions take, respectively. In this 6-round attack, we only do the step 1, step 2 and step 3 one time. So the 6-round attack's complexity is $2^8 \times 3 + 2^{16} \times 6 + 2^8 \times 6 \times 2 \approx 2^{18.6}$. The counts of selected Plaintexts are $2^8 \times 3$.

In 7-round attack, we add one round at the beginning, use the structure of Camellia-3 and select the input sets $\{\tilde{X}_{(1)}\}_t$ as $\widetilde{\mathcal{F}}_t$, where k_1 and k_2 are guessing key, for that: if $k_1 = k_{(11)}, k_2 = k_{(18)}$ *then* $\forall \tilde{X}_{(1)} \in \widetilde{\mathcal{F}}_t \Rightarrow \tilde{X}_{(2)} \in \mathcal{F}_t$.

This 7-round attack use Eq.(9) for checking and select the T as 7, for the probability of all key bytes pass the checking is $2^{-8\times T}$. The guessing step is as follows:

$$\sum_{\tilde{X}_{(1)} \in \widetilde{\mathcal{F}}_t} (s^{-1}(s(\tilde{X}^{(R)}_{(8i)} \oplus K_{(7i)}) \oplus \tilde{X}^{L}_{8i} \oplus \tilde{C}_i) = 0, i \neq 1, 1 \leq t \leq T\}. \tag{9}$$

- Step 1: Guess the values of $k_{(11)}$ and $k_{(18)}$.
- Step 2: For each $X_{(1)} \in \widetilde{\mathcal{F}}_t$ calculate the result of $\{X_{(8)}\}_t$.
- Step 3: Guess $k_{(75)}$ and $\tilde{C}_5$, and record the values that pass the checking Eq.(9),where the Plaintext set is $\widetilde{\mathcal{F}}_1$.
- Step 4: For all 'correct' pairs from step 3, checks whether Eq.(9) is satisfied or not, where $2 \leq t \leq 7$, if all the pairs can't pass the check, go to step 1.

The attacking complexity is $2^{16} \times (2^8 \times 7 + 2^{16} \times 6 + 2^8 \times 6 \times 6) \approx 2^{33.6}$. The complexity of 256-Camellia is $2^{25.6}$, since its 1st round key bits are the same as 7th round key bits.

The 8-round attack is similar to 7-round attack, the only difference is that $\tilde{X}^{(R)}_{(8i)}$ is unknown. Getting $\tilde{X}^{(R)}_{(8i)}$ from $\tilde{X}_{(9)}$ needs five 8th round key bytes, then complexity of this attack is $2^{16} \times (2^8 \times 12 + 2^{56} \times 6 + 2^{48} \times 6 \times 11) \simeq 2^{74.6}$, the complexity of 256-Camellia becomes $2^{66.6}$.

In 9-round attack, we add one round at the beginning and use the structure of Camellia-4, where the selected special plaintexts should satisfy the properties of that $\{\tilde{X}^{(2)}\}_t$ is a $\Gamma - set$ with the from of $\widetilde{\mathcal{F}}_t$. So we select $\{\tilde{X}_{(1)}\}_t$ as $\widetilde{\widetilde{\mathcal{F}}}_t$, where k_1,...,k_9 are guessing key. . The complexity of this attack is $2^{72} \times (2^8 \times 19 + 2^{56} \times 4 + 2^{48} \times 4 \times 18) \approx 2^{130}$. In 256-Camellia, the 2nd round key bytes are the same as 8th round key bytes, so the complexity of the attack is 2^{122}. In 128-Camellia 1st round key bytes are the same as 9th round key, so the complexity is 2^{90}. In 10-round attack, we add 1 round at the end and use the structure of Camellia-4, it will need to guess another 8 bytes key. The complexity is 2^{186}. And attacking on 11-round Camellia, the complexity is 2^{250}.

4.2 Improved Square Attack

The best result of Square attack against Camellia was given by Y.Yeom, S. Park, and I. Kim [3]. In this paper we improved the attacking result, since Camellia-3 satisfying Eq.(4), so called Improved Square attack.

The basic attack on 5-round camellia use the property of $s(\hat{X}^{R}_{(65)} \oplus K_{(55)}) \oplus \hat{X}^{L}_{(65)}$ is balanced byte if the plaintext set is $\mathcal{F}_t$, which is illustrated in Eq.(4),

and the probability of wrong key pass the checking is 2^{-8}. The attacking details are as follows.

- Step 1: Choose $\mathcal{F}_t, 1 \leq t \leq 2$ as plaintext sets, calculate the values of $\hat{X}^L_{(65)}$ and record them.
- Step 2: For each possible values of $K_{(55)}$, check whether Eq.(4) is satisfied or not, where the Plaintext set is $\mathcal{F}_1$, and record the passed key bytes, which may be the correct key byte. Go to next step until all possible values are checked.
- Step 3: For all 'correct' key bytes in step 2, checks whether Eq.(4) is satisfied or not, where the Plaintext set is $\mathcal{F}_2$.

For this 5-round attack, the time that step 1 takes equals the time that 2×2^8 5-round encryptions takes. Since Eq.(4) has $256 \times 2 + 256$ additions and 256 substitutions, and 5-round Camellia has 44×5 additions and 8×5 substitutions, so the time Eq.(4) takes is nearly 5 times of that 5-round Camellia encryptions take. In step 2, Eq.(4) repeats 2^8 times. The probability of wrong key passing the checking is 2^{-8}, so there will be few passing the checking, that means the time of step 3 takes 5 times of that 5-round encryptions take. So the complexity of 5-round attack is $2^8 \times 2 + 2^8 \times 5 + 5 \approx 2^{10.6}$.

6-round attack add one round at the begin and select the $\widehat{\mathcal{F}}_t, 1 \leq t \leq 5$ as plaintext sets and use the structure of Camellia-4. If k_1 in $\widehat{\mathcal{F}}_t, 1 \leq t \leq 5$ equals $k_{(11)}$, then $s(\hat{X}^R_{(75)} \oplus K_{(65)}) \oplus \hat{X}^L_{(75)}$ is balanced byte. So in this attack for each guessing k_1 we check wether the byte $s(\hat{X}^R_{(75)} \oplus K_{(65)}) \oplus \hat{X}^L_{(75)}$ is balanced or not, similar as 5-round attack. The complexity of the attack is $2^8 \times (2^8 \times 5 + 2^8 \times 4 + 4 \times 4) \approx 2^{18}$. We extend 6 round attacks to 7-round by adding one round at the end and select the plaintexs as 6-round attack. Then to get to know $\hat{X}^R_{(75)}$ we have to guess 5 7th round key bytes, so the complexity is $2^8 \times (2^8 \times 10 + 2^{48} \times 4 + 2^{40} \times 4) \approx 2^{58}$.

In 8-round attack, adding one round at the beginning, use the structure of Camellia-4. The input sets $\{\hat{X}_{(1}\}_t$ is selected as $\widehat{\widehat{\mathcal{F}}}_t, 1 \leq t \leq 15$. The key bytes are used in attack. The complexity of the attack is $2^{48} \times (2^8 \times 15 + 2^{48} \times 4 + 2^{40} \times 14) \approx 2^{98}$. The complexity of the attack on 256-Camellia is 2^{82}, since the 1st and 2nd round key bytes are the same as 7th and 8th round key bytes, respectively. The complexity of the attacks on 9 and 10 rounds are 2^{146} and 2^{212}, respectively.

4.3 Improved Collision Attack

Collision attack on Camellia is given by WL wu[10]. We improve the attacking results, called Improved Collision attack. In 5-round attack, Eq.(10) is used for checking, which is derived from Eq.(5).

$$s(\hat{X}^R_{(6i)} \oplus K_{(5i)}) \oplus \hat{X}^L_{(6i)} \equiv s(\hat{X}'^R_{(6i)} \oplus K_{(5i)}) \oplus \hat{X}'^L_{(6i)}, \hat{X}_{(1)}, \hat{X}'_{(1)} \in \mathcal{F}, i \in \{6, 7\}. \quad (10)$$

The procedure of this attack is similar to that of 5 round Improved Square attack. The time Eq.(10) takes is nearly 1/4 times that of 1-round Camellia

encryptions take, then the complexity of 5 round attack is $4 + 4 \times 2^8/(4 \times 5) \approx 2^{5.8}$. Similarly as section 4.2, The complexities of the attack on 6,7,8,9 rounds are $2^8 \times (5 + 5 \times 2^8/(6 \times 4)) \approx 2^{13.7}$, $2^8 \times (10 + 10 \times 2^{48}/(7 \times 4)) \approx 2^{54.5}$, $2^{48} \times (15 + 15 \times 2^{48}/(8 \times 4)) \approx 2^{94.9}$ and $2^{48} \times (23 + 23 \times 2^{112}/(9 \times 4)) \approx 2^{159.4}$. The complexity of 9-round attack on 128-Camellia is $2^{119.4}$. The complexities of the attack on 256-Camellia with 7,8,9 and 10 are $2^{46.5}$, $2^{78.9}$, $2^{143.4}$ and $2^{205.6}$.

4.4 Other Observations

We can build a new attack on Camellia based on Eq.(6), which implies the result of $s(\hat{X}^R_{(6i)} \oplus K_{(5i)}) \oplus \hat{(X)}^L_{(6i)}$ is a active byte. We select the $\Lambda - set\ \mathcal{F}$ as plaintext sets,then check whether $s(\hat{X}^R_{(6i)} \oplus K_{(5i)}) \oplus \hat{(X)}^L_{(6i)}$ is active byte or not for guessing key byte $K_{(5i)}$.

There is also an interesting property in Camellia-4. If we select the Plaintext $(\tilde{X}^L_1, \tilde{X}^R_1) \in \mathcal{F}$,then $\tilde{X}^R_{(58)}$ with the form of Eq.(11).

$$\tilde{X}^R_{(58)} = s_1(s_1(B \oplus \gamma) \oplus s_2(B \oplus \gamma) \oplus \delta) \oplus \epsilon,\ \gamma, \delta, \epsilon\ are\ passive,\ B\ is\ active \quad (11)$$

5 The Influence of FL/FL^{-1}

In this section, we construct the attacks on Camellia with FL/FL^{-1} function and without pre- and post-whitening.

5.1 Variant Square Attack

In 7-round variant Square attack, we use the structure of Camellia-4 and select the plaintexts $\{\tilde{X}^L_{(1)}, \tilde{X}^R_{(1)}\}_t$ as a series of $\Lambda - sets\ \bar{\bar{\mathcal{F}}}_t$. The equation used in this attack is Eq.(12).

$$\sum_{\tilde{X}^L_{(1)}, \tilde{X}^R_{(1)} \in \bar{\bar{\mathcal{F}}}_t} (s^{-1}(\tilde{X}^L_{(7i)} \oplus \tilde{C}_i) = 0, i \in \{1, 2, ..., 8\}, t \in \{1, 2, ..., T\}. \quad (12)$$

If there is a FL/FL^{-1} function in Camellia-4, we have $\tilde{X}^L_{(7i)} \neq \tilde{X}^R_{(8i)}$. Getting $\tilde{X}^L_{(7i)}$ from $\tilde{X}^R_{(8i)}$ needs to guess eight key bytes, which are used in $FL^{-}1$ function. Hence the attacking complexity is $2^{72} \times (2^8 \times 21 + 2^{72} \times 6 + 2^{64} \times 6 \times 20) \approx 2^{146.6}$, where the value of T is 21. In 128-Camellia,the complexity becomes $2^{90.6}$, since the key bytes used in FL^{-1} function are the same as 1st round key bytes.

In this 8-round attack, we add one round at the end. Then the attacking complexity is $2^{146.6+64}$. It becomes $2^{194.6}$ in 256-Camellia, since in 256-Camellia 2nd round key bytes are the same as 8th round key.

5.2 Improved Square Attack

In 7-round Improved Square attack, we use the structure of Camellia-3 and select the $\{\hat{X}^L_{(1)}, \hat{X}^R_{(1)}\}$ as $\bar{\bar{\mathcal{F}}}t$, and use Eq.(13) for checking .

$$\sum_{X_{(1)} \in \bar{\bar{\mathcal{F}}}} s(\hat{X}^R_{(8i)} \oplus K_{(7i)}) \oplus \hat{X}^L_{(8i)} = 0, i \in \{1, 2, ..., 8\}. \quad (13)$$

If there is a FL/FL^{-1} function in Camellia-3, we have to consider the property of $FL(\hat{X}^L_{(7)})$. Since the FL function results a $\Gamma - set$ in a $\Gamma - set$, we have a conclusion that whether there is a FL/FL^{-1} or not Eq.(13) always holds. Hence the complexity of that attack is $2^{48} \times (2^8 \times 10 + 2^8 \times 6 + 3 \times 9) \approx 2^{58.6}$, the key bytes required in this attack are $\{k_{(11)}, k_{(12)}, k_{(13)}, k_{(15)}, k_{(18)}, k_{(21)}, k_{(75)}\}$.

Table 1. The Summary of known attacks on Camellia

Rounds	FL/FL^{-1}	Methods	$Time$ $128-bit$	$Time$ $256-bit$	Notes
5	No	SA	2^{48}		[2]
5	No	SA	2^{16}		[3]
5	No	Improved SA	$2^{10.6}$		This Paper
5	No	Improved CA	$2^{5.8}$		This Paper
6	No	SA	2^{56}		[3]
6	No	Higher Order DC	2^{18}		[9]
6	No	Improved SA	2^{18}		This Paper
6	No	Improved CA	$2^{13.7}$		This Paper
6	No	Variant SA	$2^{18.6}$		This Paper
7	No	Truncated DC	192		[5]
7	No	Higher Order DC	2^{57}		[9]
7	Yes	SA	—	$2^{57.2}$	[3]
7	No	Improved SA	2^{58}	2^{50}	This Paper
7	No	Improved CA	$2^{54.7}$	$2^{46.7}$	This Paper
7	No	Variant SA	$2^{33.5}$	$2^{25.6}$	This Paper
7	Yes	Improved SA	$2^{58.6}$		This Paper
7	Yes	Variant SA	$2^{90.6}$	$2^{146.6}$	This Paper
8	No	Truncated DC	$2^{55.6}$		[5]
8	No	Higher Order DC	2^{120}		[9]
8	Yes	SA	—	2^{116}	[3]
8	Yes	Improved SA	2^{98}	2^{82}	This Paper
8	No	Improved CA	$2^{94.9}$	$2^{78.9}$	This Paper
8	No	Variant SA	$2^{74.6}$	$2^{66.6}$	This Paper
8	Yes	Improved CA	$2^{74.6}$	$2^{66,6}$	This Paper
8	Yes	Variant SA	—	$2^{194.6}$	This Paper
9	No	Higher Order DC	—	2^{188}	[9]
9	Yes	SA	—	$2^{181.4}$	[3]
9	Yes	Improved SA	2^{122}	2^{146}	This Paper
9	No	Improved CA	$2^{119.4}$	$2^{143.4}$	This Paper
9	No	Variant SA	2^{90}	2^{122}	This Paper
10	No	Higher Order DC	—	2^{252}	[9]
10	Yes	Improved SA	—	2^{210}	This Paper
10	No	Improved CA	—	$2^{207.4}$	This Paper
10	No	Variant SA	—	2^{186}	This Paper
11	No	Higher Order DC	—	$2^{259.6}$	[9]
11	No	Variant SA	—	2^{250}	This Paper

In 8-round attack, we add one round at the end and still use Eq.(13) for checking, than the attacking procedure becomes the same as that of section 4.2. The attacks on 9-and 10-round Camellia with FL function are also no difference from that of without FL function, which have been described in section 4.2.

6 Conclusions

Variant Square attack can break 9-round 128bit Camellia, 11-round 256 bit Camellia without FL function, further more, it is faster than exhaustive key search. The conclusions can be made that key schedule and P-function influence the security of Camellia and Square attack is still the best attack on Camellia. Table(1) give a summary of known attacks on Camellia.

Acknowledgement. We would like to thank Vincent Rijmen for his help for important comments and correctness of mistakes that improved the technical quality of the paper. The reviewer has kindly pointed out many technical problems and give suggestions in the submitted version of this draft. We would also like to thank the referees of the paper, for their constructive comments and suggestions.

References

1. K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima and T. Tokita.: Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. In: Proceedings of Selected Areas in Cryptography. Lecture Notes in Computer Science, Vol. 1281. Springer-Verlag, Berlin Heidelberg New York (2000) 39-56.
2. Y. He and S. Qing: Square Attack on Reduced Camellia Cipher. In: ICICS 2001, Lecture Notes in Computer Science, Vol. 2229. Springer-Verlag, Berlin Heidelberg New York (2001) 238-245.
3. Y.Yeom, S. Park, and I. Kim.: On the security of Camellia against the Square attack.: In: Proceed-ings of Fast Software Encryption. Lecture Notes in Computer Science, Vol. 2365. Springer-Verlag, Berlin Heidelberg New York (2002) 89-99.
4. M. Kanda and T. Matsumoto.: Security of Camellia against Truncated Differential Cryptanalysis. In: Proceedings of Fast Software Encryption Lecture Notes in Computer Science, Vol. 2355. Springer-Verlag, Berlin Heidelberg New York (2001) 286-299.
5. S. Lee, S. Hong, S. Lee, J. Lim and S. Yoon.: Truncated Differential Cryptanalysis of Camellia. In: ICISC 2001, Lecture Notes in Computer Science, Vol. 2288. Springer-Verlag, Berlin Heidel-berg New York (2001).32-38.
6. M. Sugita, K. Kobara and H. Imai.: Security of Reduced Version of the Block Cipher Camellia against Truncated and Impossible Differential Cryptanalysis. In: ASIACRYPT 2001, Lecture Notes in Computer Science, Vol. 2248. Springer-Verlag, Berlin Heidelberg New York (2001) 193-207.
7. T. Kawabata and T. Kaneko.: A Study on Higher Order Differential Attack of Camellia.: The 2nd open NESSIE workshop (2001).

8. J. Daemen, L. R. Knudsen and V. Rijmen.: The Block Cipher SQUARE. In Fast Software En-cryption, Lecture Notes in Computer Science, Vol. 1267. Springer-Verlag, Berlin Heidelberg New York (1997) 149-165.
9. Y.Hatano,H.Sekine, and T.Kaneko.: Higher order differential attack of Camellia(II). In: Proceed-ings of Selected Areas in Cryptography-SAC'02, Lecture Notes in Computer Science, Vol. 2595. Springer-Verlag, Berlin Heidelberg New York (2002) 39-56.
10. W.L.Wu,F. D.G. Feng: Collision attack on reduced-round Camellia, Science in China Series F-Information Sciences, 2005, Vol.48, No.1pp.78-90.
11. Joan Daemen and Vincent Rijmen, The Design of Rijndael, AES - The Advanced Encryption Standard, Springer-Verlag 2002, (238 pp.).
12. Niels Ferguson,John Kelsey,Stefan Lucks,Bruce Schneier,Mike Stay,David Wagner and Doug : Improved Cryptanalysis of Rijndael Whiting. In Proceedings of Fast Software Encryption-FSE'00, Vol 1978, Springer-Verlag, Berlin Heidelberg New York (2000) 213-230.

Proving the Security of AES Substitution-Permutation Network

Thomas Baignères* and Serge Vaudenay

EPFL
http://lasecwww.epfl.ch

Abstract. In this paper we study the substitution-permutation network (SPN) on which AES is based. We introduce AES*, a SPN identical to AES except that fixed S-boxes are replaced by random and independent permutations. We prove that this construction resists linear and differential cryptanalysis with 4 inner rounds only, despite the huge cumulative effect of multipath characteristics that is induced by the symmetries of AES. We show that the DP and LP terms both tend towards $1/(2^{128}-1)$ very fast when the number of round increases. This proves a conjecture by Keliher, Meijer, and Tavares. We further show that AES* is immune to any iterated attack of order 1 after 10 rounds only, which substantially improves a previous result by Moriai and Vaudenay.

Keywords: Differential Cryptanalysis, Linear Cryptanalysis, Differentials, Linear Hulls, Provable Security, AES.

1 Preamble

When we refer to "cryptanalysis", we usually think about its destructive side which consists in breaking cryptographic algorithms. Cryptanalysis however means "cryptographic analysis", which includes a constructive side that consists in proving the security of a system or the soundness of a construction. However, this last side has not received as much attention for block ciphers. Indeed, security proofs often rely on arguments derived from previous cryptanalytic attacks.

We can use linear and differential cryptanalysis [2, 3, 25, 24] (respectively denoted LC and DC) to illustrate this statement. If C denotes a block cipher, DC and LC have a complexity which is inversely proportional to the *differential probability*[1] (DP) [30] and to the *linear probability*[2] (LP) [4] terms respectively. When using an r-round Markov cipher [21], one can prove that the DP (resp. LP) is expressed as the sum of the product of the DP's (resp. LP's) in all possible inner chains of differences [37] (resp. masks). We thus usually refer to multipath characteristics or differentials [21] (resp. linear hull [31]). Typically, attacks make a heuristic approximation of the DP (resp. LP) by considering only one (single path) characteristic. If the LP or DP of such a characteristic is significant enough, then

* Supported by the Swiss National Science Foundation, 200021-107982/1.

[1] Given an input/output difference of (a, b), $\mathrm{DP}(a, b) = \Pr[\mathsf{C}(X) \oplus \mathsf{C}(X \oplus a) = b]$.

[2] Given input/output masks (a, b), $\mathrm{LP}(a, b) = (2\Pr[a \cdot X = b \cdot \mathsf{C}(X)] - 1)^2$.

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 65–81, 2006.

an attack can definitely be performed. In that situation, the cumulative effect of the differentials (resp. linear hull) can only make the attack work better than expected. Similar approximations are also made in *security proofs* of block ciphers. This could be acceptable only if one could make sure that, among the differentials (resp. linear hull), one single path characteristic is overwhelming (so that the rest can be neglected). Although this is actually the case for DES, this does not appear to be true for AES [6, 7]. Indeed, the argument saying that all DP and LP terms are at most 2^{-300} on 8 rounds [6, pp. 30–31] obviously cannot be true. Since for any a, the sum over all the 2^{128} values $\mathrm{DP}(a, b)$ (resp. $\mathrm{LP}(a, b)$) is equal to 1, at least one value of the DP is larger than 2^{-128}. Obviously, symmetries in AES are likely to lead to a considerable cumulative effect when considering many equivalent characteristics. Therefore, proving that there is no single path characteristic with a significant DP (resp. LP) is not sufficient to prove the resistance of a block cipher against DC (resp. LC).

In practice, however, differentials and linear hull are rarely taken into consideration in security proofs, as evaluating the true DP or LP is computationally not practical for a typical block cipher. One natural solution is to try to upper bound these terms. This approach was chosen by Keliher, Meijer, and Tavares [19, 18] who showed that the LP of AES is upper bounded by 2^{-75} for 7 or more rounds. Park et al. showed [33, 34] that the DP and LP for four rounds are respectively bounded by 1.144×2^{-111} and 1.075×2^{-106}. Finally, in a recent work [17], Keliher shows that the bound on the LP is 1.778×2^{-107} for 8 or more rounds.

Another solution is to adopt a Luby-Rackoff-like approach. In their seminal work [22], Luby and Rackoff showed how to construct a pseudo-random permutation from any pseudo-random function. They provided an example based on the Feistel scheme [8] (because it is the one on which DES is based). Since then, the security of Feistel ciphers with *random* and *independent* round functions received a considerable amount of attention (see [29, 35, 38, 27, 9], to name only a few). Although Substitution-Permutation Network (SPN) schemes security has already been widely studied (see for example [5, 13, 14, 19]), only a few papers adopted a Luby-Rackoff-like approach to study the one on which AES is based (see for example Moriai-Vaudenay [28] and Keliher-Meijer-Tavares [20]).

In this paper, we analyze the security of the SPN on which AES is based, where fixed S-boxes are replaced by *random* and *independent* permutations. This scheme, that we call AES^*, is introduced in Section 2, together with some of its properties with respect to the LP and DP terms. This includes a discussion about keyed operations, i.e., over the subkey addition and over the substitution boxes layer. In Section 3, we give an expression of the expected LP over AES^*, depending on the input/output masks and the number of rounds (see Theorem 6). Using this result, we prove a conjecture made by Keliher, Meijer, and Tavares in [20], namely that all DP's and LP's converge towards $1/(2^{128} - 1)$ as the number of rounds increases (see Theorem 8). This means that AES^* behaves exactly like the perfect cipher (as far as LC is concerned) when the number of rounds is high enough. The rest of Section 3 shows how to reduce the computational complexity of the expression given in Theorem 6 by exploiting some of the symmetries of AES^* (see Theorem 12). We

conclude the section by exhibiting results of practical experiments. We give the expected LP over AES* for several number of rounds and several S-box sizes, and deduce that AES* is protected against LC after four inner rounds only[3]. Section 4 shows how these results extend to differential cryptanalysis. In Section 5 we generalize the results on LC by considering *any* iterated attack of order 1 [39]. Recall that these kind of attacks are very similar to LC, except that the bit of information extracted from each plaintext/ciphertext pair is not necessarily computed by a linear masking of text bits, but can be derived using any type of *projection* (in the sense of [41,1]). Experimental results show that after 10 rounds, AES* is immune to iterated attacks of order 1. This substantially improves a previous result of Moriai and Vaudenay, who showed that 384 were sufficient [28]. Finally, we show in Section 6 by derandomization techniques that all security results on AES* remain valid when the random S-boxes are replaced by S-boxes with perfect pairwise decorrelation.

2 Preliminaries

2.1 Description of AES

AES [6,7] is a block cipher with a fixed block length of 128 bits, supporting 128, 192, and 256 bit keys. It is iterated, meaning that it is made of a succession of r identical rounds. It is byte oriented, meaning that each round takes as an input a 128 bit long state that is considered as a one-dimensional vector of 16 bytes. Each round also takes a round key as an input, issued from a key schedule that takes as an input the main 128 bit key. We do not detail the key schedule here, since we will assume that all round keys are randomly selected and independent. Each round is a succession of four operations (we use the notations of [7]): `SubBytes`, that applies a fixed S-box to each of the 16 input state bytes, `ShiftRows`, which shifts the rows of the state (considered as a 4×4 array) over four different offsets, `MixColumns`, that applies a linear transformation to each state columns, and `AddRoundKey`, which is a bitwise XOR between the subkey and the state. AES is made of 10 rounds (for a 128 bit key), where the last one does not include the `MixColumns` operation. The first round is preceded by a additional `AddRoundKey` operation.

2.2 Introducing AES*

In the subsequent, we will be considering a family of block ciphers based on AES. This family, that we call AES*, almost follows the same SPN as AES, except for the last round, which excludes *both* linear operations (that is, `MixColumns` and `ShiftRows`). Although this modification does not have any influence on the security results, it simplifies the notations. Moreover, it does not involve a fixed S-box. Following a Luby-Rackoff-like approach, each S-box will be considered as an independent permutation chosen uniformly at random. Consequently, we denote by `SubBytes`* the confusion step in AES*. In that sense, AES is a particular

[3] Note that this result does take hulls effect into consideration.

instance of AES* where all the S-boxes have been chosen to be the one defined in the specifications.

Clearly, a truly random S-box following the XOR of a random byte is equivalent to a truly random S-box. Hence, we can completely ignore the addition of round keys in AES*.

2.3 States, Activity Patterns, and Notations

We denote by $\mathrm{GF}(q)$ the finite field with q elements and by $\mathcal{S}$ the set of AES* states, so that $\mathcal{S} = \mathrm{GF}(q)^{16}$. In the case of AES, $q = 2^8$. AES* states (or equivalently, masks on AES* states) will generally be denoted by bold small letters such as $\boldsymbol{a}$,$\boldsymbol{b}$, etc. An arbitrary state $\boldsymbol{a}$ is a vector which can be viewed as a four by four array with elements in $\mathrm{GF}(q)$ denoted $(a_{i,j})_{1\leq i,j\leq 4}$. The four by four array of $\{0,1\}^{16}$ with 0's at the positions where the entry of $\boldsymbol{a}$ is 0, and with 1's where the entry of $\boldsymbol{a}$ non-zero, is called the *activity pattern* [6] or *support* corresponding to the state $\boldsymbol{a}$. Supports will either be denoted by Greek letters or by $\mathrm{SUPP}(\cdot)$. For example, the support corresponding to a state $\boldsymbol{a}$ will either be denoted $\boldsymbol{\alpha}$ or $\mathrm{SUPP}(\boldsymbol{a})$. The (i,j) entry in the array $\boldsymbol{\alpha}$ will be denoted $\alpha_{i,j}$. The Hamming weight of a support $\boldsymbol{\alpha}$, denoted $|\boldsymbol{\alpha}|$, is the number of 1's in this support (i.e., $|\boldsymbol{\alpha}| = \sum_{i,j} \alpha_{i,j}$), so that $0 \leq |\boldsymbol{\alpha}| \leq 16$. When $|\boldsymbol{\alpha}| = 0$ it means that $\boldsymbol{a}$ is zero, whereas when $|\boldsymbol{\alpha}| = 16$, it means that all entries of $\boldsymbol{a}$ are non-zero. In the latter case, we say that $\boldsymbol{a}$ is a state of *full-support*. The set of states limited to some specific support $\boldsymbol{\alpha}$ will be denoted $\mathcal{S}_{|\boldsymbol{\alpha}}$, and thus $\#\mathcal{S}_{|\boldsymbol{\alpha}} = \sigma^{|\boldsymbol{\alpha}|}$, with $\sigma = q - 1$. The set of states of full-support will be denoted $\mathcal{S}_{\mathrm{full}}$ so that $\#\mathcal{S}_{\mathrm{full}} = \sigma^{16}$.

2.4 The Scalar Product and the LP Coefficient

The scalar product of a state (plaintext) $\boldsymbol{x}$ and a state (mask) $\boldsymbol{a}$ is usually defined as the exclusive-or between several bits of $\boldsymbol{x}$, chosen according to a pattern specified by a mask $\boldsymbol{a}$, which depends on the way the elements of $\mathrm{GF}(q)$ are represented. We prefer here an equivalent definition in terms of the trace function[4] Tr, defined from $\mathrm{GF}(q)$ onto $\mathrm{GF}(2)$ by $\mathrm{Tr}(x) = x + x^2 + x^4 + x^8 + \cdots + x^{128}$. If $\boldsymbol{a}$ and $\boldsymbol{b}$ are two arbitrary states of AES, we define the scalar product of $\boldsymbol{a}$ and $\boldsymbol{b}$ as $\boldsymbol{a} \bullet \boldsymbol{b} = \sum_{i,j} \mathrm{Tr}(a_{i,j} b_{i,j})$. We use the following well known linear algebra property.

Lemma 1. *Let* M *denote an arbitrary* 16 *by* 16 *matrix of elements in* $\mathrm{GF}(q)$*, representing a linear transformation on* AES *states*[5]*. Let* $\boldsymbol{x}$ *be an input state to this linear transformation* M *and let* $\boldsymbol{b}$ *be a non-zero output mask. Then* $\boldsymbol{b} \bullet (\mathsf{M} \times \boldsymbol{x}) = (\mathsf{M}^T \times \boldsymbol{b}) \bullet \boldsymbol{x}$*.*

The efficiency of a linear cryptanalysis can be measured by means of the *linear probability* [4]. With our definition of the scalar product, this quantity is defined

[4] One advantage of this variant is that it does not depend on the way we represent $\mathrm{GF}(q)$. Namely, even if we represent the cells of AES states by the Zech logarithm, we can still define the scalar product in the same way.

[5] AES states are indeed considered as column vectors.

in the following way (here we use the notation introduced in [26]): if $\boldsymbol{a}$ and $\boldsymbol{b}$ are two states and C is some fixed permutation on $\mathcal{S}$, then $\mathrm{LP}^{\boldsymbol{C}}(\boldsymbol{a},\boldsymbol{b}) = (2\Pr_{\boldsymbol{X}\in\mathcal{S}}[\boldsymbol{a}\bullet\boldsymbol{X} = \boldsymbol{b}\bullet\mathsf{C}(\boldsymbol{X})]-1)^2$, where the probability holds over the uniform distribution of $\boldsymbol{X}$.

2.5 Expected LP over Keyed Operations in AES*

A round key, or simply a key, is an AES state. The only keyed operation in AES is `AddRoundKey`. As stated in Section 2.2, we can ignore this operation in AES*. We can thus consider the choice of the random S-boxes as the only keyed operation in AES*. The following lemma evaluates the average LP over all possible random S-boxes.

Lemma 2. *Let $a, b \in \mathrm{GF}(q)\setminus\{0\}$ be two non-zero input/output masks on the uniformly distributed random S-box S^* and let $\sigma = q-1$. The average* LP *value over all possible random S-boxes is independent of a and b, and is $\mathrm{E}_{\mathsf{S}^*}[\mathrm{LP}^{\mathsf{S}^*}(a,b)] = \sigma^{-1}$.*

Proof. See Lemma 14 in [39] for a direct proof. One can also use the explicit distribution of the LP of S^* [20], deduced from results available in [32]. □

Note that for any S-box S we have $\mathrm{LP}^S(a,0) = \mathrm{LP}^S(0,b) = 0$ (for non-zero a and b) and $\mathrm{LP}^S(0,0) = 1$. From this, we derive the expected LP over `SubBytes`*.

Lemma 3. *Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two non-zero masks in $\mathrm{GF}(q)^{16}$, and let $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ be their respective supports. Let $\sigma = q-1$. We have $\mathrm{E}[\mathrm{LP}^{\mathtt{SubBytes}^*}(\boldsymbol{a},\boldsymbol{b})] = \sigma^{-|\boldsymbol{\alpha}|}$ if $\boldsymbol{\alpha} = \boldsymbol{\beta}$ and 0 otherwise, where the mean is taken over all possible uniformly distributed and independent random S-boxes.*

3 Expected LP on AES*

3.1 From Sums over Masks to Sums over Supports

The complexity of computing the expected LP of AES is prohibitive for the reason that, once input/output masks are given, one has to sum over all possible intermediate masks in order to take into account every possible characteristic. We will see that AES* provides just enough randomization for this sum to be made over intermediate supports. Consequently, we will need to count the number of possible states succession corresponding to some given succession of supports.

Definition 4. *Let* LT *denote the linear transformation of* AES*, i.e., the operation corresponding to* `MixColumns` $\circ$ `ShiftRows`*. Let $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ be two supports and*

$$\mathrm{N}[\boldsymbol{\alpha},\boldsymbol{\beta}] = \#\{(\boldsymbol{a},\boldsymbol{b}) \in \mathcal{S}_{|\boldsymbol{\alpha}} \times \mathcal{S}_{|\boldsymbol{\beta}} \ :\ \mathsf{LT}^T \times \boldsymbol{b} = \boldsymbol{a}\},$$

where states $\boldsymbol{a}$ and $\boldsymbol{b}$ are considered as column vectors here. $\mathrm{N}[\boldsymbol{\alpha},\boldsymbol{\beta}]$ is the number of ways of connecting a support $\boldsymbol{\alpha}$ to a support $\boldsymbol{\beta}$ through LT*.*

From now on, we will consider an r-round version of AES^*, with $r > 1$. Round $i \in \{1, \ldots, r\}$ will be denoted by $\mathtt{Round}_i^*$, where the last round $\mathtt{Round}_r^*$ excludes the linear transformation LT. With these notations, $\mathsf{AES}^* = \mathtt{Round}_r^* \circ \cdots \circ \mathtt{Round}_1^*$. The input/output masks on $\mathtt{Round}_i^*$ will usually be denoted $\boldsymbol{c}_{i-1}$ and $\boldsymbol{c}_i$ respectively, while their corresponding supports will be denoted $\boldsymbol{\gamma}_{i-1}$ and $\boldsymbol{\gamma}_i$. Consequently, $\boldsymbol{c}_0$ and $\boldsymbol{c}_r$ will respectively denote the input and the output masks on a r-rounds version of AES^*. Using Lemma 3, we can derive the expected LP over one round and extend it to the full AES^*.

Lemma 5. *Let $\boldsymbol{c}_{i-1}$ and $\boldsymbol{c}_i$ be two non-zero masks in $\mathrm{GF}(q)^{16}$ of support $\boldsymbol{\gamma}_{i-1}$ and $\boldsymbol{\gamma}_i$ respectively. Let $\sigma = q - 1$. For $1 \leq i < r$, the expected linear probability over $\mathtt{Round}_i^*$ is given by $\mathrm{E}[\mathrm{LP}^{\mathtt{Round}_i^*}(\boldsymbol{c}_{i-1}, \boldsymbol{c}_i)] = \sigma^{-|\boldsymbol{\gamma}_{i-1}|}$ if $\boldsymbol{\gamma}_{i-1} = \mathrm{SUPP}(\mathsf{LT}^T \times \boldsymbol{c}_i)$ and 0 otherwise. Similarly, the expected* LP *over the last round is given by $\mathrm{E}[\mathrm{LP}^{\mathtt{Round}_r^*}(\boldsymbol{c}_{r-1}, \boldsymbol{c}_r)] = \sigma^{-|\boldsymbol{\gamma}_{r-1}|}$ if $\boldsymbol{\gamma}_{r-1} = \boldsymbol{\gamma}_r$ and 0 otherwise.*

Proof. We first consider the case where $1 \leq i < r$. Using Lemma 1, we have $\mathrm{E}[\mathrm{LP}^{\mathtt{Round}_i^*}(\boldsymbol{c}_{i-1}, \boldsymbol{c}_i)] = \mathrm{E}[\mathrm{LP}^{\mathtt{SubBytes}^*}(\boldsymbol{c}_{i-1}, \mathsf{LT}^T \times \boldsymbol{c}_i)]$. Lemma 3 allows to conclude. The proof for the $i = r$ case is similar, except that we don't make use of Lemma 1 as the last round excludes LT. □

Theorem 6. *Let $\boldsymbol{c}_0$ and $\boldsymbol{c}_r$ be two masks in $\mathrm{GF}(q)^{16}$ of support $\boldsymbol{\gamma}_0$ and $\boldsymbol{\gamma}_r$ respectively. Let $\sigma = q - 1$. The expected linear probability over $r > 1$ rounds of AES^*, when $\boldsymbol{c}_0$ is the input mask and $\boldsymbol{c}_r$ the output mask is*

$$\mathrm{E}[\mathrm{LP}^{\mathsf{AES}^*}(\boldsymbol{c}_0, \boldsymbol{c}_r)] = \sigma^{-|\boldsymbol{\gamma}_r|} \times \left(\mathcal{M}^{r-1}\right)_{\boldsymbol{\gamma}_0, \boldsymbol{\gamma}_r} ,$$

where $\mathcal{M}$ is a $2^{16} \times 2^{16}$ square matrix, indexed by pairs of masks $(\boldsymbol{\gamma}_{i-1}, \boldsymbol{\gamma}_i)$, such that $\mathcal{M}_{\boldsymbol{\gamma}_{i-1}, \boldsymbol{\gamma}_i} = \sigma^{-|\boldsymbol{\gamma}_{i-1}|}\mathrm{N}[\boldsymbol{\gamma}_{i-1}, \boldsymbol{\gamma}_i]$.

Proof. Following Nyberg [31], $\mathrm{E}[\mathrm{LP}^{\mathsf{AES}^*}(\boldsymbol{c}_0, \boldsymbol{c}_r)] = \sum \prod_{i=1}^{r} \mathrm{E}[\mathrm{LP}^{\mathtt{Round}_i^*}(\boldsymbol{c}_{i-1}, \boldsymbol{c}_i)]$, where the sum is taken over all intermediate masks $c_1, \ldots, c_{r-1}$. Using the results (and the notations) of Lemma 5 this gives

$$\mathrm{E}[\mathrm{LP}^{\mathsf{AES}^*}(\boldsymbol{c}_0, \boldsymbol{c}_r)] = \sum_{\substack{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{r-1} \\ \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_{r-1}}} \sigma^{-|\boldsymbol{\gamma}_{r-1}|} \mathbf{1}_{\boldsymbol{\gamma}_{r-1} = \boldsymbol{\gamma}_r} \prod_{i=1}^{r-1} \sigma^{-|\boldsymbol{\gamma}_{i-1}|} \mathbf{1}_{\substack{\boldsymbol{\gamma}_{i-1} = \mathrm{SUPP}(\mathsf{LT}^T \times \boldsymbol{c}_i) \\ \boldsymbol{\delta}_i = \boldsymbol{\gamma}_i}} ,$$

where the sum is also taken over all possible intermediate supports. Taking $\boldsymbol{\delta}_0 = \boldsymbol{\gamma}_0$ and $\boldsymbol{\delta}_r = \boldsymbol{\gamma}_r$ and including the sum over the $\boldsymbol{c}_i$'s in the product, we obtain $\mathrm{E}[\mathrm{LP}^{\mathsf{AES}^*}(\boldsymbol{c}_0, \boldsymbol{c}_r)] = \sigma^{-|\boldsymbol{\delta}_r|} \sum_{\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_{r-2}} \prod_{i=1}^{r-1} \sigma^{-|\boldsymbol{\delta}_{i-1}|}\mathrm{N}[\boldsymbol{\delta}_{i-1}, \boldsymbol{\delta}_i]$. The definition of the product of square matrices concludes the proof. □

Using supports drops the matrix size from 2^{128} down to 2^{16}. As one matrix multiplication roughly takes $(2^{16})^3$ field operations[6] and, using a square and multiply technique, $\log r$ such multiplications are needed, the overall number of operations needed to compute $\mathcal{M}^{r-1}$ is roughly equal to 2^{50} (for 8 rounds) by using 2×2^{32}

[6] Using Strassen's algorithm, the complexity drops to $(2^{16})^{\log 7}$ field operations [40].

multiple precision rational number registers. This is still pretty hard to implement using ordinary hardware. Nevertheless, from one computation of $\mathcal{M}^{r-1}$ we could deduce all expected linear probability over all possible input/output masks almost for free. In section 3.3, we show how to exploit symmetries of table $\mathrm{N}[\cdot,\cdot]$ in order to further reduce the matrix size.

3.2 Towards the True Random Cipher

For any non-zero mask $\boldsymbol{c}$, $\mathrm{LP}^{\mathsf{AES}^*}(\boldsymbol{c},0) = \mathrm{LP}^{\mathsf{AES}^*}(0,\boldsymbol{c}) = 0$ and $\mathrm{LP}^{\mathsf{AES}^*}(0,0) = 1$. Thus, the $2^{16}\times 2^{16}$ square matrix $\mathcal{M}$ of Theorem 6 has the following shape

$$\mathcal{M} = \left(\begin{array}{c|c} 1 & \mathbf{0} \\ \hline \mathbf{0} & \mathcal{M}' \end{array}\right) \tag{1}$$

where $\mathcal{M}'$ is a $(2^{16}-1)\times(2^{16}-1)$ square matrix, indexed by non-zero supports. We can now notice from Theorem 6 that $\mathrm{E}[\mathrm{LP}^{\mathsf{AES}^*}(\boldsymbol{c}_0,\boldsymbol{c}_2)] = \sigma^{-|\boldsymbol{\gamma}_2|}\mathcal{M}'_{\boldsymbol{\gamma}_0,\boldsymbol{\gamma}_2}$ for any non-zero supports $\boldsymbol{c}_0$ and $\boldsymbol{c}_2$. Recall that $\sum_{\boldsymbol{c}_2}\mathrm{E}[\mathrm{LP}^{\mathsf{AES}^*}(\boldsymbol{c}_0,\boldsymbol{c}_2)] = 1$. Hence

$$1 = \sum_{\boldsymbol{c}_2}\sigma^{-|\boldsymbol{\gamma}_2|}\mathcal{M}'_{\boldsymbol{\gamma}_0,\boldsymbol{\gamma}_2} = \sum_{\boldsymbol{\gamma}_2}\sigma^{|\boldsymbol{\gamma}_2|}\sigma^{-|\boldsymbol{\gamma}_2|}\mathcal{M}'_{\boldsymbol{\gamma}_0,\boldsymbol{\gamma}_2} = \sum_{\boldsymbol{\gamma}_2}\mathcal{M}'_{\boldsymbol{\gamma}_0,\boldsymbol{\gamma}_2}\,.$$

We also note that $\mathcal{M}'_{\boldsymbol{\gamma}_0,\boldsymbol{\gamma}_2} \geq 0$ for any $\boldsymbol{\gamma}_0$ and $\boldsymbol{\gamma}_2$.

Lemma 7. *The matrix $\mathcal{M}'$ defined by* (1) *is the transition matrix of a Markov chain, whose set of states is the set of non-zero supports and whose transition probability from a non-zero support $\boldsymbol{\gamma}$ to a non-zero support $\boldsymbol{\gamma}'$ is given by $\mathcal{M}_{\boldsymbol{\gamma},\boldsymbol{\gamma}'}$.*

The transition graph of the Markov chain is the directed graph whose vertices are the σ non-zero supports and such that there is an edge from $\boldsymbol{\gamma}$ to $\boldsymbol{\gamma}'$ when $\mathcal{M}_{\boldsymbol{\gamma},\boldsymbol{\gamma}'} > 0$. From the study of supports propagation [6] (which is based on the MDS criterion), it clearly appears that from any graph state, there is a path towards the graph state corresponding to the full support $\boldsymbol{\gamma}_{\mathrm{full}}$ (for example, two steps are required to go from a support of Hamming weight 1 to $\boldsymbol{\gamma}_{\mathrm{full}}$). Moreover, from the graph state corresponding to $\boldsymbol{\gamma}_{\mathrm{full}}$ one can reach any graph state. Hence, from each graph state there is a sequence of arrows leading to *any* other graph state. This means that the corresponding Markov chain is *irreducible* [12]. Since there is an arrow from $\boldsymbol{\gamma}_{\mathrm{full}}$ to itself, one can find a sequence of arrows leading from any graph state to any graph state, of *any* (yet long enough) length. This means the Markov chain is *aperiodic*. We can deduce that there exists exactly one stationary distribution (see for example chapter 5 in [12]), i.e., a $1\times(2^{16}-1)$ row vector $\boldsymbol{\pi} = (\pi_{\boldsymbol{\gamma}})_{\boldsymbol{\gamma}\neq\mathbf{0}}$ indexed by non-zero supports such that $\pi_{\boldsymbol{\gamma}} \geq 0$ for all non-zero $\boldsymbol{\gamma}$ with $\sum_{\boldsymbol{\gamma}\neq 0}\pi_{\boldsymbol{\gamma}} = 1$, and such that $\pi\mathcal{M}' = \pi$ (which is to say that $\pi_{\boldsymbol{\gamma}'} = \sum_{\boldsymbol{\gamma}\neq\mathbf{0}}\pi_{\boldsymbol{\gamma}}\mathcal{M}'_{\boldsymbol{\gamma},\boldsymbol{\gamma}'}$ for all non zero $\boldsymbol{\gamma}'$). It is easy to show that the row vector $\boldsymbol{\pi}$ indexed by non-zero supports such that $\pi_{\boldsymbol{\gamma}} = \sigma^{|\boldsymbol{\gamma}|}(q^{16}-1)^{-1}$ is a stationary distribution of the Markov chain described by the transition matrix $\mathcal{M}'$. Indeed,

$$\sum_{\boldsymbol{\gamma}\neq\mathbf{0}}\pi_{\boldsymbol{\gamma}} = \frac{1}{q^{16}-1}\sum_{\boldsymbol{\gamma}\neq\mathbf{0}}\left(\sum_{s=1}^{16}\mathbf{1}_{s=|\boldsymbol{\gamma}|}\right)\sigma^{|\boldsymbol{\gamma}|} = \frac{1}{q^{16}-1}\sum_{s=1}^{16}\binom{16}{s}\sigma^s = 1,$$

and therefore $\boldsymbol{\pi}$ is a probability distribution. Moreover, for any non-zero $\boldsymbol{\gamma}'$, $(\boldsymbol{\pi}\mathcal{M}')_{\boldsymbol{\gamma}'} = (q^{16}-1)^{-1}\sum_{\boldsymbol{\gamma}\neq\mathbf{0}} \mathrm{N}[\boldsymbol{\gamma},\boldsymbol{\gamma}'] = (q^{16}-1)^{-1}\sigma^{|\boldsymbol{\gamma}'|} = \pi_{\boldsymbol{\gamma}'}$, as the sum is simply the number of non-zero states that can be connected to some non-zero support $\boldsymbol{\gamma}'$ through LT, which is exactly the number of states of support equal to γ', as each state of support $\boldsymbol{\gamma}'$ has one and only one preimage through LT.

It is known [11] that $(\mathcal{M}'^r)_{\boldsymbol{\gamma},\boldsymbol{\gamma}'} \to \pi_{\boldsymbol{\gamma}'}$ when $r \to \infty$. As $\mathrm{E}[\mathrm{LP}^{\mathsf{AES}^*}(\boldsymbol{c_0},\boldsymbol{c_r})] = \sigma^{-|\boldsymbol{\gamma}_r|}(\mathcal{M}'^{r-1})_{\boldsymbol{\gamma}_0,\boldsymbol{\gamma}_r}$ for non-zero masks $\boldsymbol{c_0}$ and $\boldsymbol{c_r}$, we have proven the following theorem (which corresponds to the conjecture in [20]).

Theorem 8. *Let $\boldsymbol{c}_0$ and $\boldsymbol{c}_r$ be two non-zero masks in* $\mathrm{GF}(q)^{16}$*. Then*

$$\lim_{r\to\infty} \mathrm{E}[\mathrm{LP}^{\mathsf{AES}^*}(\boldsymbol{c}_0,\boldsymbol{c}_r)] = \frac{1}{q^{16}-1}\,. \tag{2}$$

We conclude this discussion by wondering *how fast* does the expected LP of AES^* tends towards $(q^{16}-1)^{-1}$. As $\mathcal{M}'$ is the transition matrix of a finite irreducible and aperiodic chain, the Perron-Frobenius Theorem [11] states that $\lambda_1 = 1$ is an eigenvalue of $\mathcal{M}'$, while the remaining eigenvalues $\lambda_2,\ldots,\lambda_m$ satisfy $|\lambda_j| < 1$. Assuming that $\lambda_1 > |\lambda_2| \geq \cdots \geq |\lambda_m|$, the rate of the convergence depends on $|\lambda_2|$. If we let λ be any real value such that $1 > \lambda > |\lambda_2|$, we deduce that for any non-zero masks $\boldsymbol{c}_0$ and $\boldsymbol{c}_r$, $\mathrm{E}[\mathrm{LP}^{\mathsf{AES}^*}(\boldsymbol{c}_0,\boldsymbol{c}_r)] = \frac{1}{q^{16}-1} + O(\lambda^r)$ when $r \to \infty$.

Note that the same results can be obtained on AES itself with independent round keys using almost the same proof. The only change is that one needs to prove that for any non-zero masks $\boldsymbol{a}$ and $\boldsymbol{b}$, there is a number of rounds r such that $\mathrm{LP}^{\mathtt{Round}_r\circ\cdots\circ\mathtt{Round}_1}(\boldsymbol{a},\boldsymbol{b}) \neq 0$. Equivalently, we can prove it with DP's by using results by Wernsdorf et al. [15, 42].

3.3 Combinatorial Tables on Supports

We will see that, thanks to the properties of LT, $\mathrm{N}[\boldsymbol{\gamma}_{i-1},\boldsymbol{\gamma}_i]$ only depends on the weights of the diagonals of $\boldsymbol{\gamma}_{i-1}$ and of the columns of $\boldsymbol{\gamma}_i$. We introduce notations to deal with Hamming weights of columns and diagonals. If $\boldsymbol{\gamma}_i$ is the ith support in a characteristic, we denote by $\mathbf{c}_i = (\mathsf{c}_{i,1},\mathsf{c}_{i,2},\mathsf{c}_{i,3},\mathsf{c}_{i,4})$ the vector of the four weights of $\boldsymbol{\gamma}_i$'s columns. Similarly, we denote by $\mathbf{d}_i = (\mathsf{d}_{i,1},\mathsf{d}_{i,2},\mathsf{d}_{i,3},\mathsf{d}_{i,4})$ the four weights of $\boldsymbol{\gamma}_i$'s diagonals. What we mean by columns and diagonals should be clear from Figure 1. Finally, we denote by $\mathbf{w}_j^i = (\mathbf{d}_i,\mathbf{c}_j)$ the *weight pattern* of a pair of supports $(\boldsymbol{\gamma}_i,\boldsymbol{\gamma}_j)$. Note that $|\mathbf{w}_j^i| = |\boldsymbol{\gamma}_i| + |\boldsymbol{\gamma}_j|$ and that this weight pattern only includes the weights of the diagonals of $\boldsymbol{\gamma}_i$ and of the columns of $\boldsymbol{\gamma}_j$. Consequently, if $\boldsymbol{\gamma}_{i-1}$ and $\boldsymbol{\gamma}_i$ are two successive masks in a characteristic, $\mathbf{w}_i^{i-1}$

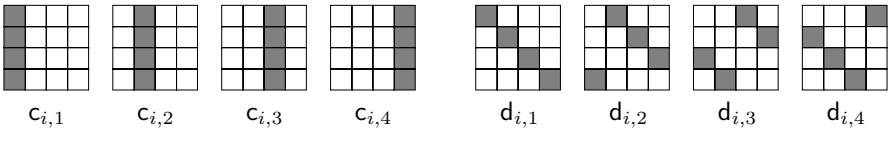


Fig. 1. The four column's and diagonal's weights of a state $\boldsymbol{\gamma}_i$

contains enough information to compute $\mathrm{N}[\boldsymbol{\gamma}_{i-1}, \boldsymbol{\gamma}_i]$ (as we will see in Corollary 10). We now recall a known fact about the weight distribution of MDS codes.

Theorem 9 (Theorem 7.4.1 in [16]). *Let $\mathcal{C}$ be an $[n, k, d]$ MDS code over $\mathrm{GF}(q)$. For $i = 0, \ldots, n$, the number A_i of codewords of weight i is given by $A_0 = 1$, $A_i = 0$ for $1 \leq i < d$ and $A_i = \binom{n}{i} \sum_{j=0}^{i-d} (-1)^j \binom{i}{j} (q^{i+1-d-j} - 1)$ for $d \leq i \leq n$, where $d = n - k + 1$.*

The `MixColumns` operation is a linear multipermutation [36], as the set of all codewords $(\boldsymbol{a}, \texttt{MixColumns}(\boldsymbol{a}))$ is a $[8, 4, 5]$ MDS code.

Corollary 10. *Let $\boldsymbol{\gamma}_{i-1}$ and $\boldsymbol{\gamma}_i$ be two successive supports of a characteristic and let $\mathbf{w}_i^{i-1} = (\mathbf{d}_{i-1}, \mathbf{c}_i)$ be their weight pattern. We have*

$$\mathrm{N}[\boldsymbol{\gamma}_{i-1}, \boldsymbol{\gamma}_i] = \prod_{s=1}^{4} \frac{A_{\mathsf{d}_{i-1,s}+\mathsf{c}_{i,s}}}{\binom{8}{\mathsf{d}_{i-1,s}+\mathsf{c}_{i,s}}}.$$

Thus, $\mathbf{w}_i^{i-1}$ is sufficient to compute $\mathrm{N}[\boldsymbol{\gamma}_{i-1}, \boldsymbol{\gamma}_i]$ so that we will denote this value by $\mathrm{N}[\mathbf{w}_i^{i-1}]$. By symmetry, it is clear that an arbitrary permutation applied on both the diagonal's and column's weights of $\mathbf{w}_i^{i-1}$ will not change the value of $\mathrm{N}[\mathbf{w}_i^{i-1}]$, i.e., if two weight patterns $\mathbf{w} = (\mathbf{d}, \mathbf{c})$ and $\mathbf{w}' = (\mathbf{d}', \mathbf{c}')$ are such that

$$(\mathsf{d}_1, \mathsf{d}_2, \mathsf{d}_3, \mathsf{d}_4, \mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \mathsf{c}_4) = (\mathsf{d}'_{\pi(1)}, \mathsf{d}'_{\pi(2)}, \mathsf{d}'_{\pi(3)}, \mathsf{d}'_{\pi(4)}, \mathsf{c}'_{\pi(1)}, \mathsf{c}'_{\pi(2)}, \mathsf{c}'_{\pi(3)}, \mathsf{c}'_{\pi(4)})$$

for some permutation π of $[1, 4]$, then $\mathrm{N}[\mathbf{w}] = \mathrm{N}[\mathbf{w}']$. It is natural to consider such weight patterns as equivalent and to choose a unique representative for each equivalence class. We arbitrarily choose to take the greatest element in the sense of the lexicographic order as the representative and denote it $\overline{\mathbf{w}}$. The number of elements in the equivalence class of $\overline{\mathbf{w}}$ will be denoted $\mathrm{C}[\overline{\mathbf{w}}]$. By the end of this section, we will be summing over weight patterns of supports surrounding the linear transformation LT (Theorem 12) instead of supports between individual rounds (Theorem 6). It will be natural to link both concepts. Given two successive weight patterns $\mathbf{w}_i^{i-1} = (\mathbf{d}_{i-1}, \mathbf{c}_i)$ and $\mathbf{w}_{i+1}^{i} = (\mathbf{d}_i, \mathbf{c}_{i+1})$, we denote by $\mathrm{P}[\mathbf{w}_i^{i-1}, \mathbf{w}_{i+1}^{i}]$ the number of supports $\boldsymbol{\gamma}$ (between rounds i and $i+1$) compatible with these weight patterns, i.e., the number of supports $\boldsymbol{\gamma}$ of weight pattern $(\mathbf{d}, \mathbf{c})$ such that $\mathbf{d} = \mathbf{d}_i$ and $\mathbf{c} = \mathbf{c}_i$ (see Figure 2). In other words, table $\mathrm{P}[\cdot, \cdot]$ gives the number of possible supports with given Hamming weights of the columns and of the diagonals. We note that by shifting columns, this is equivalent to counting 4×4 binary matrices with given weights for every row and column. Consequently, $\mathrm{P}[\mathbf{w}_i^{i-1}, \mathbf{w}_{i+1}^{i}]$ remains unchanged by permuting the weight of the diagonals given by $\mathbf{c}_i$ and/or the weight of the columns given by $\mathbf{d}_i$.

Lemma 11. *Let $(\boldsymbol{\gamma}_{i-1}, \boldsymbol{\gamma}_i, \boldsymbol{\gamma}_{i+1})$ be a characteristic of supports on two rounds, let $\mathbf{w}_i^{i-1} = (\mathbf{d}_{i-1}, \mathbf{c}_i)$ and $\mathbf{w}_{i+1}^{i} = (\mathbf{d}_i, \mathbf{c}_{i+1})$ be the weight pattern of $(\boldsymbol{\gamma}_{i-1}, \boldsymbol{\gamma}_i)$ and $(\boldsymbol{\gamma}_i, \boldsymbol{\gamma}_{i+1})$ respectively, and let $\overline{\mathbf{w}}_i^{i-1}$ and $\overline{\mathbf{w}}_{i+1}^{i}$ be their representatives. Then $\mathrm{N}[\mathbf{w}_i^{i-1}] = \mathrm{N}[\overline{\mathbf{w}}_i^{i-1}]$, $\mathrm{P}[\mathbf{w}_i^{i-1}, \mathbf{w}_{i+1}^{i}] = \mathrm{P}[\overline{\mathbf{w}}_i^{i-1}, \overline{\mathbf{w}}_{i+1}^{i}]$, and $|\mathbf{w}_i^{i-1}| = |\overline{\mathbf{w}}_i^{i-1}|$.*

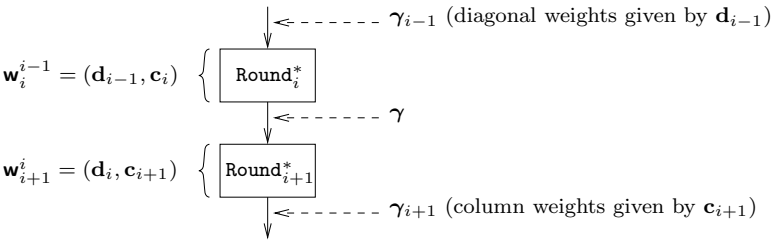


Fig. 2. Given $\mathbf{w}_i^{i-1}$ and $\mathbf{w}_{i+1}^{i}$, there are $\mathrm{P}[\mathbf{w}_i^{i-1}, \mathbf{w}_{i+1}^{i}]$ compatible supports $\boldsymbol{\gamma}$'s

3.4 From Sums over Supports to Sums over Weight Pattern Representatives

Theorem 12. *Let $\boldsymbol{c}_0$ and $\boldsymbol{c}_r$ be two masks in $\mathrm{GF}(q)^{16}$ of support $\boldsymbol{\gamma}_0$ and $\boldsymbol{\gamma}_r$ respectively. Let $\mathbf{d}_0$ denote the weight vector of $\boldsymbol{\gamma}_0$'s diagonals and let $\mathbf{c}_r$ denote the weight vector of $\boldsymbol{\gamma}_r$'s columns. Let $\sigma = q - 1$. Let $\mathcal{L}$ be the square matrix indexed by weight patterns representatives, defined by*

$$\mathcal{L}_{\overline{\mathbf{u}},\overline{\mathbf{v}}} = \mathrm{R}[\overline{\mathbf{u}}]\mathrm{P}[\overline{\mathbf{u}},\overline{\mathbf{v}}]\mathrm{R}[\overline{\mathbf{v}}] \quad \textit{where} \quad \mathrm{R}[\mathbf{u}] = \sqrt{\sigma^{\frac{1}{2}|\mathbf{u}|}\mathrm{C}[\mathbf{u}]\mathrm{N}[\mathbf{u}]}\ .$$

Finally, let $\mathcal{U}(\mathbf{d}_0)$ and $\mathcal{V}(\mathbf{c}_r)$ be the column vectors indexed by weight patterns representatives, defined by

$$\mathcal{U}(\mathbf{d}_0)_{\overline{\mathbf{v}}} = \sigma^{-\frac{1}{2}|\mathbf{d}_0|}\mathrm{R}[\overline{\mathbf{v}}]\mathrm{C}[\overline{\mathbf{v}}]^{-1} \sum_{\mathbf{u}=(\mathbf{d},\mathbf{c})} \mathbf{1}_{\overline{\mathbf{u}}=\overline{\mathbf{v}}}\mathbf{1}_{\mathbf{d}=\mathbf{d}_0} \quad \textit{and}$$

$$\mathcal{V}(\mathbf{c}_r)_{\overline{\mathbf{v}}} = \sigma^{-\frac{1}{2}|\mathbf{c}_r|}\mathrm{R}[\overline{\mathbf{v}}]\mathrm{C}[\overline{\mathbf{v}}]^{-1} \sum_{\mathbf{u}=(\mathbf{d},\mathbf{c})} \mathbf{1}_{\overline{\mathbf{u}}=\overline{\mathbf{v}}}\mathbf{1}_{\mathbf{c}=\mathbf{c}_r}\ .$$

Then the expected linear probability over $r > 1$ rounds of AES^* is

$$\mathrm{E}[\mathrm{LP}^{\mathsf{AES}^*}(\boldsymbol{c}_0, \boldsymbol{c}_r)] = \mathcal{U}(\mathbf{d}_0)^T \times \mathcal{L}^{r-2} \times \mathcal{V}(\mathbf{c}_r)\ .$$

Proof. For simplicity, $\mathrm{E}[\mathrm{LP}^{\mathsf{AES}^*}(\boldsymbol{c}_0, \boldsymbol{c}_r)]$ will simply be denoted $\mathrm{ELP}(\boldsymbol{c}_0, \boldsymbol{c}_r)$ and we will consider the case where $r > 2$. In Theorem 6, we had

$$\mathrm{ELP}(\boldsymbol{c}_0, \boldsymbol{c}_r) = \sigma^{-|\boldsymbol{\gamma}_r|} \sum_{\boldsymbol{\gamma}_1,\ldots,\boldsymbol{\gamma}_{r-2}} \prod_{i=1}^{r-1} \sigma^{-|\boldsymbol{\gamma}_{i-1}|}\mathrm{N}[\boldsymbol{\gamma}_{i-1}, \boldsymbol{\gamma}_i].$$

We notice that $2\sum_{i=1}^{r} |\boldsymbol{\gamma}_{i-1}| = |\mathbf{w}_0^r| + \sum_{i=1}^{r-1} |\mathbf{w}_i^{i-1}|$, where we used the fact that, as we do not need to take into account characteristics that give a zero linear probability, $\boldsymbol{\gamma}_{r-1} = \boldsymbol{\gamma}_r$ (see Lemma 5). From this and from Corollary 10, we deduce that $\mathrm{ELP}(\boldsymbol{c}_0, \boldsymbol{c}_r) = \sigma^{-\frac{1}{2}|\mathbf{w}_0^r|} \sum_{\boldsymbol{\gamma}_1,\ldots,\boldsymbol{\gamma}_{r-2}} \prod_{i=1}^{r-1} \mathrm{D}[\mathbf{w}_i^{i-1}]$, where $\mathrm{D}[\mathbf{w}] = \sigma^{\frac{1}{2}|\mathbf{w}|}\mathrm{N}[\mathbf{w}]$. As we want to consider weight patterns instead of supports, we introduce a new sum and permute both sums to obtain

$$\mathrm{ELP}(\boldsymbol{c}_0, \boldsymbol{c}_r) = \sigma^{-\frac{1}{2}|\mathbf{w}_0^r|} \sum_{\mathbf{u}_1^0,\ldots,\mathbf{u}_{r-1}^{r-2}} \left(\sum_{\boldsymbol{\gamma}_1,\ldots,\boldsymbol{\gamma}_{r-2}} \prod_{j=1}^{r-1} \mathbf{1}_{\mathbf{w}_j^{j-1}=\mathbf{u}_j^{j-1}} \right) \prod_{i=1}^{r-1} \mathrm{D}[\mathbf{u}_i^{i-1}]\ .$$

Denoting $\mathbf{u}_j^{j-1} = (\mathbf{d}'_{j-1}, \mathbf{c}'_j)$, it is easy to show that

$$\sum_{\gamma_1,\ldots,\gamma_{r-2}} \prod_{j=1}^{r-1} \mathbf{1}_{\mathbf{u}_j^{j-1}=\mathbf{w}_j^{j-1}} = \mathbf{1}_{\mathbf{d}'_0=\mathbf{d}_0} \mathbf{1}_{\mathbf{c}'_{r-1}=\mathbf{c}_{r-1}} \prod_{j=1}^{r-2} \sum_{\gamma_j} \mathbf{1}_{(\mathbf{d}_j,\mathbf{c}_j)=(\mathbf{d}'_j,\mathbf{c}'_j)} \cdot$$

As, by definition, $\mathrm{P}[\mathbf{u}_j^{j-1}, \mathbf{u}_{j+1}^{j}] = \sum_{\boldsymbol{\gamma}_j} \mathbf{1}_{(\mathbf{d}_j,\mathbf{c}_j)=(\mathbf{d}'_j,\mathbf{c}'_j)}$, this gives

$$\mathrm{ELP}(\boldsymbol{c}_0, \boldsymbol{c}_r) = \sigma^{-\frac{1}{2}|\mathbf{w}_0^r|} \sum_{\substack{\mathbf{u}_1^0,\ldots,\mathbf{u}_{r-1}^{r-2} \\ \mathbf{d}'_0=\mathbf{d}_0}} \mathbf{1}_{\mathbf{c}'_{r-1}=\mathbf{c}_{r-1}} \mathrm{D}[\mathbf{u}_{r-1}^{r-2}] \prod_{i=1}^{r-2} \mathrm{D}[\mathbf{u}_i^{i-1}] \mathrm{P}[\mathbf{u}_i^{i-1}, \mathbf{u}_{i+1}^{i}] \,.$$

We denote $\mathcal{L}_{\overline{\mathbf{u}},\overline{\mathbf{v}}} = \mathrm{C}[\overline{\mathbf{u}}]^{\frac{1}{2}} \mathrm{D}[\overline{\mathbf{u}}]^{\frac{1}{2}} \mathrm{P}[\overline{\mathbf{u}}, \overline{\mathbf{v}}] \mathrm{C}[\overline{\mathbf{v}}]^{\frac{1}{2}} \mathrm{D}[\overline{\mathbf{v}}]^{\frac{1}{2}}$ and $\mathrm{F}[\overline{\mathbf{u}}] = \mathrm{D}[\overline{\mathbf{u}}]^{\frac{1}{2}} \mathrm{C}[\overline{\mathbf{u}}]^{-\frac{1}{2}}$. Using Lemma 11, the last expression becomes

$$\mathrm{ELP}(\boldsymbol{c}_0, \boldsymbol{c}_r) = \sigma^{-\frac{1}{2}|\mathbf{w}_0^r|} \sum_{\mathbf{u}_1^0, \mathbf{u}_{r-1}^{r-2}} \mathbf{1}_{\mathbf{d}'_0=\mathbf{d}_0} \mathbf{1}_{\mathbf{c}'_{r-1}=\mathbf{c}_{r-1}} \mathrm{F}[\overline{\mathbf{u}}_1^0] \mathrm{F}[\overline{\mathbf{u}}_{r-1}^{r-2}] (\mathcal{L}^{r-2})_{\overline{\mathbf{u}}_1^0, \overline{\mathbf{u}}_{r-1}^{r-2}} \,.$$

Introducing $(\mathcal{U}(\mathbf{d}_0))_{\overline{\mathbf{u}}_1^0}$ and $(\mathcal{V}(\mathbf{c}_{r-1}))_{\overline{\mathbf{u}}_{r-1}^{r-2}}$ in the previous expression leads (as $\boldsymbol{c}_{r-1} = \boldsymbol{c}_r$) to the announced result. □

In order to evaluate the complexity of the matrix multiplication of Theorem 12, we need to evaluate the size of the matrices, i.e., the number of equivalence classes. There are $20475 \approx 2^{14.33}$ such classes. Yet, it is not necessary to consider those equivalence classes for which $\mathrm{N}[\cdot]$ is 0. It can be checked that the number of remaining equivalence classes is $1001 \approx 2^{10}$. The computation of $\mathcal{L}^{r-1}$ therefore roughly takes $2^{30} \cdot \log r$ operations, which is feasible on standard computers.

3.5 Experimental Linear Hull for Various S-Box Sizes

Theorems 6 and 12 remain valid with several sizes of S-boxes. We implemented the computation of Theorem 12 with various sizes, our experimental results were obtained using GMP [10]. They are shown in Table 1. It appears that 4 rounds

Table 1. $\max_{\boldsymbol{a},\boldsymbol{b}} \mathrm{E}[\mathrm{LP}^{\mathsf{AES}^*}(\boldsymbol{a}, \boldsymbol{b})]$ for various number of rounds r and S-box sizes

r	2	3	4	5	6	7	8	9
3 bits	$2^{-13.2294}$	$2^{-19.6515}$	$2^{-44.9177}$	$2^{-44.9177}$	$2^{-47.3861}$	$2^{-47.9966}$	$2^{-47.9999}$	$2^{-48.0}$
4 bits	$2^{-17.6276}$	$2^{-27.3482}$	$2^{-62.5102}$	$2^{-62.5102}$	$2^{-63.9852}$	$2^{-63.9999}$	$2^{-63.9999}$	$2^{-64.0}$
5 bits	$2^{-21.8168}$	$2^{-34.6793}$	$2^{-79.2671}$	$2^{-79.2671}$	$2^{-79.9999}$	$2^{-79.9999}$	$2^{-79.9999}$	$2^{-80.0}$
6 bits	$2^{-25.9091}$	$2^{-41.8409}$	$2^{-95.6364}$	$2^{-95.6364}$	$2^{-95.9999}$	$2^{-95.9999}$	$2^{-96.0}$	$2^{-96.0}$
7 bits	$2^{-29.9547}$	$2^{-48.9207}$	$2^{-111.8189}$	$2^{-111.8189}$	$2^{-111.9999}$	$2^{-111.9999}$	$2^{-112.0}$	$2^{-112.0}$
8 bits	$2^{-33.9774}$	$2^{-55.9605}$	$2^{-127.9096}$	$2^{-127.9096}$	$2^{-127.9999}$	$2^{-127.9999}$	$2^{-128.0}$	$2^{-128.0}$

are enough to provide security against LC. We do not provide any result for the case where the S-box acts on 2 bit elements as it is impossible to find a 4×4 matrix with elements in $\mathrm{GF}(2^2)$ such that `MixColumns` stays a multipermutation. A second independent implementation of the computation was implemented in Maple [23] in order to obtain perfect results instead of floating point numbers. It was used for the masks presenting the maximum expected LP in Table 1.

4 Expected DP on AES*

Just as the efficiency of LC can be measured by means of LP's, the efficiency of DC can be measured by means of DP's [30]. If C is some fixed permutation on $\mathcal{S}$ and if $\boldsymbol{a}$ and $\boldsymbol{b}$ are two masks, the differential probability is given by $\mathrm{DP}^{\mathsf{C}}(\boldsymbol{a}, \boldsymbol{b}) = \Pr_{\boldsymbol{X} \in \mathcal{S}}[\mathsf{C}(\boldsymbol{X} \oplus \boldsymbol{a}) = \mathsf{C}(\boldsymbol{X}) \oplus \boldsymbol{b}]$, where the probability holds over the uniform distribution of $\boldsymbol{X}$. Here, $\boldsymbol{a}$ (resp. $\boldsymbol{b}$) represents the input (resp. output) difference between the pair of plaintexts (resp. ciphertexts). The computations that we performed on the expected LP of AES^* can be applied, with almost no modification, in order to compute the expected DP. The major modification concerns Lemma 1. We provide here its version for the DP.

Lemma 13. *Let* M *denote an arbitrary* 16 *by* 16 *matrix of elements in* $\mathrm{GF}(q)$*, representing a linear transformation on* AES *states (considered as column vectors). If the difference between two inputs of this transformation is equal to* $\boldsymbol{a}$*, then the output difference is equal to* $\mathsf{M} \times \boldsymbol{a}$*.*

We now follow the steps that lead to the final result on the LP coefficient and see whether they apply to the DP coefficient. Lemma 2 applies to the DP coefficient, and therefore, it is also the case for Lemma 3 (where we use the independence of the the 16 inputs on the S-boxes in order to obtain a product of DP, instead of using Matsui's Piling-up Lemma). Because the relation between an input difference on the linear transformation of AES and its output difference is not the same as in the case where we considered input/output masks, it looks as if we must replace LT^T by LT^{-1} in Definition 4. But according to Theorem 9, the actual *values* of $\mathrm{N}[\cdot]$ do not depend on which multipermutation is used, it just needs to be one. In other words, replacing LT^T by LT^{-1} in the definition of $\mathrm{N}[\cdot]$ does not change its entries. The computations on the LP coefficient thus still apply for the DP coefficient. Theorems 6, 8, and 12 apply to the DP, the numerical results given in Table 1 being exactly the same.

5 Extension to Iterated Attacks of Order 1

In the Luby-Rackoff model [22], an adversary $\mathcal{A}$ has an unlimited computational power, but has limited access to an oracle $\mathcal{O}$. The oracle implements either an instance of a given cipher C (such as AES^*) or of the perfect cipher C^*, the objective of the adversary being to guess which is the case (see Figure 3). Eventually, the adversary will output 1 (resp. 0) if his guess is that the oracle implements C (resp. C^*). Denoting by $\Pr[\mathcal{A}^{\mathcal{O}} \to 1]$ the probability

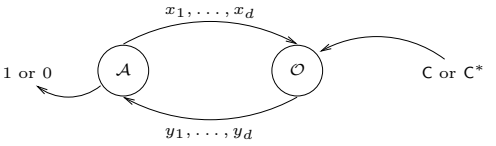


Fig. 3. An adversary $\mathcal{A}$ limited to d questions to an oracle $\mathcal{O}$

that the adversary outputs 1 depending on the oracle $\mathcal{O}$, his ability to distinguish C from C^* is measured by means of the advantage $\mathrm{Adv}_{\mathcal{A}} = |\Pr[\mathcal{A}^{\mathsf{C}} \rightarrow 1] - \Pr[\mathcal{A}^{\mathsf{C}^*} \rightarrow 1]|$. The most powerful adversary will select his d queries depending on the previous answers of the oracle. Such an adversary is called a *d-limited adaptative distinguisher* [39]. The advantage of the best distinguisher of this type is such that $\mathrm{Adv}_{\mathcal{A}} = \frac{1}{2} \parallel [\mathsf{C}]^d - [\mathsf{C}^*]^d \parallel_a$, where $[\mathsf{C}]^d$ is the d-wise distribution matrix[7] of the random permutation C over $\mathcal{S}$, and where $\parallel M \parallel_a = \max_{x_1} \sum_{y_1} \cdots \max_{x_d} \sum_{y_d} |M_{(x_1,\ldots,x_d),(y_1,\ldots,y_d)}|$ for any $\#\mathcal{S}^d \times \#\mathcal{S}^d$ matrix M (Theorem 11 in [39]). Proving the resistance of C against such a $2d$-limited distinguisher is sufficient to prove its resistance against any iterated attacks of order d (Theorem 18 in [39]). Using Theorem 14, we bound the advantage of the best 2-limited adaptative distinguisher and deduce the number rounds necessary to resist any iterated attacks of order 1.

Theorem 14. *Let C be a random permutation over $\{0,1\}^n$. If ϵ is the non-negative value such that $\epsilon = \max_{\boldsymbol{a} \neq 0, \boldsymbol{b}} \mathrm{E}[\mathrm{DP}^{\mathsf{C}}(\boldsymbol{a}, \boldsymbol{b})] - \frac{1}{2^n - 1}$, we have $\parallel [\mathsf{C}']^2 - [\mathsf{C}^*]^2 \parallel_a \leq 2^n \varepsilon$ where $\mathsf{C}'(x) = \mathsf{C}(x \oplus K_1) \oplus K_2$ with independent and uniformly distributed K_1 and K_2.*

Proof. Let $x_1, x_2, y_1, y_2 \in \{0,1\}^n$. Starting from the definition of $[\mathsf{C}']^2$, we have

$$[\mathsf{C}']^2_{(x_1,x_2),(y_1,y_2)} = \sum_{\mathsf{c}} \Pr_{K_1,K_2} \begin{bmatrix} \mathsf{c}(x_1 \oplus K_1) = y_1 \oplus K_2 \\ \mathsf{c}(x_2 \oplus K_1) = y_2 \oplus K_2 \end{bmatrix} \Pr[\mathsf{C} = \mathsf{c}],$$

as C is independent from (K_1, K_2). Furthermore, we have

$$\Pr_{K_1,K_2} \begin{bmatrix} \mathsf{c}(x_1 \oplus K_1) = y_1 \oplus K_2 \\ \mathsf{c}(x_2 \oplus K_1) = y_2 \oplus K_2 \end{bmatrix} = \sum_{u,v} \mathbf{1}_{\substack{x_1 \oplus x_2 = u \\ y_1 \oplus y_2 = v}} \Pr_{K_1,K_2} \begin{bmatrix} \mathsf{c}(K_1) = K_2 \\ \mathsf{c}(u \oplus K_1) = v \oplus K_2 \end{bmatrix}$$

where the probability in the sum is equal to

$$2^{-2n} \sum_{k_1,k_2} \mathbf{1}_{\substack{\mathsf{c}(k_1 \oplus u) = k_2 \oplus v \\ \mathsf{c}(k_1) = k_2}} = 2^{-n} \Pr_{K_1}[\mathsf{c}(K_1) \oplus \mathsf{c}(K_1 \oplus u) = v] = 2^{-n} \mathrm{DP}^{\mathsf{c}}(u, v) \; .$$

Therefore, $[\mathsf{C}']^2_{(x_1,x_2),(y_1,y_2)} = 2^{-n} \mathrm{E}_{\mathsf{C}}[\mathrm{DP}^{\mathsf{C}}(x_1 \oplus x_2, y_1 \oplus y_2)]$. As the sum of the $\mathrm{DP}^{\mathsf{C}^*}$ on the input mask is 1 (as C^* is a permutation), $\mathrm{E}_{\mathsf{C}^*}[\mathrm{DP}^{\mathsf{C}^*}(x_1 \oplus x_2, y_1 \oplus$

[7] Recall that the d-wise distribution matrix of a random function F is such that $[F]^d_{(x_1,\ldots,x_d),(y_1,\ldots,y_d)}$ is the probability that $F(x_i) = y_i$ for all $i = 1, \ldots, d$.

Table 2. Values of ϵ depending of the number of rounds r

r	2	3	4	5	6	7	8	9	10
ϵ	$2^{-33.98}$	$2^{-55.96}$	$2^{-131.95}$	$2^{-131.95}$	$2^{-152.17}$	$2^{-174.74}$	$2^{-200.39}$	$2^{-223.93}$	$2^{-270.82}$

$y_2)] = \frac{1}{2^n-1}$ when $x_1 \neq x_2$ (when $x_1 = x_2$, the DP value is always 0, except when $y_1 \oplus y_2$ is also 0, in which case DP is 1). From the last two equations we deduce $[\mathsf{C}']^2_{(x_1,x_2),(y_1,y_2)} - [\mathsf{C}^*]^2_{(x_1,x_2),(y_1,y_2)} = 2^{-n}\left(\mathrm{E}_{\mathsf{C}}[\mathrm{DP}^{\mathsf{C}}(x_1 \oplus x_2, y_1 \oplus y_2)] - \frac{1}{2^n-1}\right)$, and thus, by definition of the $||\cdot||_a$ norm, $||[\mathsf{C}']^2 - [\mathsf{C}^*]^2||_a$ is upper bounded by $2^{-n}\sum_{y_1,y_2} \max_{x_1 \neq x_2} |\mathrm{E}_{\mathsf{C}}[\mathrm{DP}^{\mathsf{C}}(x_1 \oplus x_2, y_1 \oplus y_2)] - (2^n-1)^{-1}| = 2^n\varepsilon$. □

Such an ϵ always exists, as the maximum DP (or LP) value is always larger or equal to $1/(2^n-1)$. Experimental results on ϵ (obtained both with our GMP and Maple implementations) are given in Table 2 for several number of rounds. We conclude that provable security is achieved for 10 rounds of AES^* (which substantially improves [28], where it is shown that 384 rounds are enough).

6 Derandomizing the S-Boxes

We note that all results presented so far hold if replace the uniformly distributed random S-box S^* by *any* random S-box S, provided that it satisfies $\mathrm{E}_{\mathsf{S}}[\mathrm{LP}^{\mathsf{S}}(a,b)] = \sigma^{-1}$ (which is proved for S^* in Lemma 2). According to Lemma 14 in [39],

$$\mathrm{E}_{\mathsf{S}}[\mathrm{LP}^{\mathsf{S}}(a,b)] = q^{-2}\sum_{\substack{x_1,x_2\\y_1,y_2}} (-1)^{(x_1\oplus x_2)\bullet a+(y_1\oplus y_2)\bullet b}\Pr[\mathsf{S}(x_1)=y_1,\mathsf{S}(x_2)=y_2].$$

Hence, $\mathrm{E}_{\mathsf{S}}[\mathrm{LP}^{\mathsf{S}}(a,b)]$ only depends on the pairwise distribution. If S has a perfect pairwise decorrelation, we deduce $\mathrm{E}_{\mathsf{S}}[\mathrm{LP}^{\mathsf{S}}(a,b)] = \sigma^{-1}$. In order to construct such a variant of AES, one can just insert a `MulRoundKey` operation before each `AddRoundKey` of AES, with independent subkeys, where `MulRoundKey` is the component-wise product in $\mathrm{GF}(q)$ of an input state and a subkey, i.e., considering the three states $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{k}$ as a one-dimensional vectors of 16 bytes,

$$\boldsymbol{b} = \texttt{MulRoundKey}(\boldsymbol{a}, \boldsymbol{k}) \quad \Leftrightarrow \quad b_i = a_i \times k_i \quad \text{for } i = 1, \ldots, 16.$$

Note that all the component of a subkey $\boldsymbol{k}$ used in a `MulRoundKey` operation have to be non-zero to preserve bijectivity.

7 Discussion and Conclusion

We studied the SPN on which AES is based using a Luby-Rackoff-like approach. Following [20] and [28], we considered that the only "round function" that can reasonably be replaced by a random one is the S-box. We chose to replace the S-boxes by random and *independent* permutations. In this model, we computed the

exact (i.e., using neither heuristic approximations nor bounds) hull and differential average probabilities. Clearly, a better model (i.e., intuitively closer to the real AES) would be to choose *one* permutation at random and use it throughout the whole cipher, although it is not clear to us that one can easily prove similar security results in that case. Obviously, we cannot draw direct consequences on the security of AES. At least we get some increased confidence in its high-level structure and claim that AES with independent keys has no useful linear hull nor differentials, unless the S-box structure selection is really unfortunate. We also pushed the analysis further by studying iterated attacks of order 1. We showed that ten inner rounds are sufficient to ensure the security of AES^* against any attack of this kind. Finally, we proved the (non-surprising) convergence of AES^* towards the perfect cipher (as far as LC and DC are concerned) as the number of rounds increases, which was only conjectured so far.

Acknowledgments. We would like to thank the anonymous referees, Pascal Junod, and Matthieu Finiasz for helpful comments, as well as Ralph Wernsdorf for quite useful references.

References

[1] T. Baignères, P. Junod, and S. Vaudenay. How far can we go beyond linear cryptanalysis? In P.J. Lee, editor, *Advances in Cryptology* - ASIACRYPT *'04*, volume 3329 of *LNCS*, pages 432–450. Springer-Verlag, 2004.

[2] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4:3–72, 1991.

[3] E. Biham and A. Shamir. Differential cryptanalysis of the full 16-round DES. In E.F. Brickell, editor, *Advances in Cryptology* - CRYPTO *'92*, volume 740 of *LNCS*, pages 487–496. Springer-Verlag, 1993.

[4] F. Chabaud and S. Vaudenay. Links between differential and linear cryptanalysis. In A. De Santis, editor, *Advances in Cryptology* - EUROCRYPT *'94*, volume 950 of *LNCS*, pages 356–365. Springer-Verlag, 1995.

[5] Z.G. Chen and S.E. Tavares. Towards provable security of substitution-permutation encryption networks. In S.E. Tavares and H. Meijer, editors, *Selected Areas in Cryptography,* SAC *'98*, volume 1556 of *LNCS*, pages 43–56. Springer-Verlag, 1999.

[6] J. Daemen and V. Rijmen. AES proposal: Rijndael. NIST AES Proposal, 1998.

[7] J. Daemen and V. Rijmen. *The Design of Rijndael.* Information Security and Cryptography. Springer-Verlag, 2002.

[8] H. Feistel. Cryptography and computer privacy. *Scientific American*, 228:15–23, 1973.

[9] H. Gilbert and M. Minier. New results on the pseudorandomness of some block-cipher constructions. In M. Matsui, editor, *Fast Software Encryption* - FSE *'01*, volume 2355 of *LNCS*, pages 248–266. Springer-Verlag, 2002.

[10] GMP. GNU Multiple Precision arithmetic library. `http://www.swox.com/gmp`.

[11] G. Grimmett and D. Stirzaker. *Probability and Random Processes.* Oxford University Press, 3d edition, 2001.

[12] O. Häggström. *Finite Markov Chains and Algorithmic Applications.* London Mathematical Society Student Texts. Cambridge University Press, 2002.

[13] H.M. Heys and S.E. Tavares. Substitution-permutation networks resistant to differential and linear cryptanalysis. *Journal of Cryptology*, 9(1):1–19, 1996.
[14] S. Hong, S. Lee, J. Lim, J. Sung, D. Cheon, and I. Cho. Provable security against differential and linear cryptanalysis for the SPN structure. In B. Schneier, editor, *Fast Software Encryption* - FSE*'00*, volume 1978 of *LNCS*, pages 273–283. Springer-Verlag, 2001.
[15] G. Hornauer, W. Stephan, and R. Wernsdorf. Markov ciphers and alternating groups. In T. Helleseth, editor, *Advances in Cryptology* - EUROCRYPT*'93*, volume 765 of *LNCS*, pages 453–460. Springer-Verlag, 1994.
[16] W.C. Huffman and V.S. Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003.
[17] L. Keliher. Refined analysis of bounds related to linear and differential cryptanalysis for the AES. In H. Dobbertin, V. Rijmen, and A. Sowa, editors, *Fourth Conference on the Advanced Encryption Standard* - AES*4*, volume 3373 of *LNCS*, pages 42–57. Springer-Verlag, 2005.
[18] L. Keliher, H. Meijer, and S.E. Tavares. Improving the upper bound on the maximum average linear hull probability for Rijndael. In S. Vaudenay and A.M. Youssef, editors, *Selected Areas in Cryptography,* SAC*'01*, volume 2259 of *LNCS*, pages 112–128. Springer-Verlag, 2001.
[19] L. Keliher, H. Meijer, and S.E. Tavares. New method for upper bounding the maximum average linear hull probability for SPNs. In B. Pfitzmann, editor, *Advances in Cryptology* - EUROCRYPT*'01*, volume 2045 of *LNCS*, pages 420–436. Springer-Verlag, 2001.
[20] L. Keliher, H. Meijer, and S.E. Tavares. Toward the true random cipher: On expected linear probability values for SPNs with randomly selected S-boxes. In V. Bhargava, H.V. Poor, V. Tarokh, and S. Yoon, editors, *Communication, Information and Network Security*, pages 123–146. Kluwer Academic Publishers, 2003.
[21] X. Lai, J. Massey, and S. Murphy. Markov ciphers and differential cryptanalysis. In D.W. Davies, editor, *Advances in Cryptology* - EUROCRYPT*'91*, volume 547 of *LNCS*, pages 17–38. Springer-Verlag, 1991.
[22] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.
[23] Maplesoft. Maple 9. `http://www.maplesoft.com/`.
[24] M. Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In Y.G. Desmedt, editor, *Advances in Cryptology* - CRYPTO*'94*, volume 839 of *LNCS*, pages 1–11. Springer-Verlag, 1994.
[25] M. Matsui. Linear cryptanalysis method for DES cipher. In T. Helleseth, editor, *Advances in Cryptology* - EUROCRYPT*'93*, volume 765 of *LNCS*, pages 386–397. Springer-Verlag, 1994.
[26] M. Matsui. New structure of block ciphers with provable security against differential and linear cryptanalysis. In D. Gollmann, editor, *Fast Software Encryption* - FSE*'96*, volume 1039 of *LNCS*, pages 205–218. Springer-Verlag, 1996.
[27] U. Maurer and K. Pietrzak. The security of many-round Luby-Rackoff pseudorandom permutations. In E. Biham, editor, *Advances in Cryptology* - EUROCRYPT*'03*, volume 2656 of *LNCS*, pages 544–561. Springer-Verlag, 2003.
[28] S. Moriai and S. Vaudenay. On the pseudorandomness of top-level schemes of block ciphers. In T. Okamoto, editor, *Advances in Cryptology* - ASIACRYPT*'00*, volume 1976 of *LNCS*, pages 289–302. Springer-Verlag, 2000.
[29] M. Naor and O. Reingold. On the construction of pseudorandom permutations: Luby-Rackoff revisited. *Journal of Cryptology*, 12(1):29–66, 1999.

[30] K. Nyberg. Perfect nonlinear S-boxes. In D.W. Davies, editor, *Advances in Cryptology* - EUROCRYPT *'91*, volume 547 of *LNCS*, pages 378–386. Springer-Verlag, 1991.

[31] K. Nyberg. Linear approximation of block ciphers. In A. De Santis, editor, *Advances in Cryptology* - EUROCRYPT *'94*, volume 950 of *LNCS*, pages 439–444. Springer-Verlag, 1995.

[32] L. O'Connor. Properties of linear approximation tables. In B. Preneel, editor, *Fast Software Encryption* - FSE *'94*, volume 1008 of *LNCS*, pages 131–136. Springer-Verlag, 1995.

[33] S. Park, S.H. Sung, S. Chee, E-J. Yoon, and J. Lim. On the security of Rijndael-like structures against differential and linear cryptanalysis. In Y. Zheng, editor, *Advances in Cryptology* - ASIACRYPT *'02*, volume 2501 of *LNCS*, pages 176–191. Springer-Verlag, 2002.

[34] S. Park, S.H. Sung, S. Lee, and J. Lim. Improving the upper bound on the maximum differential and the maximum linear hull probability for SPN structures and AES. In T. Johansson, editor, *Fast Software Encryption* - FSE *'03*, volume 2887 of *LNCS*, pages 247–260. Springer-Verlag, 2003.

[35] J. Patarin. Security of random Feistel schemes with 5 or more rounds. In M. Franklin, editor, *Advances in Cryptology* - CRYPTO *'04*, volume 3152 of *LNCS*, pages 106–122. Springer-Verlag, 2004.

[36] S. Vaudenay. On the need for multipermutations: Cryptanalysis of MD4 and SAFER. In B. Preneel, editor, *Fast Software Encryption* - FSE *'94*, volume 1008 of *LNCS*, pages 286–297. Springer-Verlag, 1995.

[37] S. Vaudenay. On the security of CS-cipher. In L. Knudsen, editor, *Fast Software Encryption* - FSE *'99*, volume 1636 of *LNCS*, pages 260–274. Springer-Verlag, 1999.

[38] S. Vaudenay. On the Lai-Massey scheme. In L. Kwok Yan, O. Eiji, and X. Chaoping, editors, *Advances in Cryptology* - ASIACRYPT *'99*, volume 1716 of *LNCS*, pages 8–19. Springer-Verlag, 2000.

[39] S. Vaudenay. Decorrelation: a theory for block cipher security. *Journal of Cryptology*, 16(4):249–286, 2003.

[40] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2nd edition, 2003. First published 1999.

[41] D. Wagner. Towards a unifying view of block cipher cryptanalysis. In B. Roy and W. Meier, editors, *Fast Software Encryption* - FSE *'04*, volume 3017 of *LNCS*, pages 16–33. Springer-Verlag, 2004.

[42] R. Wernsdorf. The round functions of Rijndael generate the alternating group. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption* - FSE *'02*, volume 2365 of *LNCS*, pages 143–148. Springer-Verlag, 2002.

An Attack on CFB Mode Encryption as Used by OpenPGP

Serge Mister and Robert Zuccherato

Entrust, Inc., 1000 Innovation Drive, Ottawa, Ontario, Canada K2K 3E7
{serge.mister, robert.zuccherato}@entrust.com

Abstract. This paper describes an adaptive chosen-ciphertext attack on the Cipher Feedback (CFB) mode of encryption as used in OpenPGP. In most circumstances it will allow an attacker to determine 16 bits of any block of plaintext with about 2^{15} oracle queries for the initial setup work and 2^{15} oracle queries for each block. Standard CFB mode encryption does not appear to be affected by this attack. It applies to a particular variation of CFB used by OpenPGP. In particular it exploits an ad-hoc integrity check feature in OpenPGP which was meant as a "quick check" to determine the correctness of the decrypting symmetric key.

Keywords: OpenPGP, Cipher-Feedback Mode, chosen-ciphertext attacks, encryption.

1 Introduction

The OpenPGP Message Format is described in RFC 2440 [4]. It is a very popular and commonly used format for signing and encrypting data files, particularly for signing and encrypting email. The formats described in the OpenPGP RFC have been implemented in a wide variety of popular freeware and commercial encryption products. Symmetric encryption in OpenPGP is performed using a variant of the standard Cipher-Feedback (CFB) Mode for block ciphers.

Adaptive chosen-ciphertext attacks on cryptographic protocols allow an attacker to decrypt a ciphertext C, getting the plaintext M, by submitting a series of chosen-ciphertexts $C' \neq C$ to an oracle which returns information on the decryption. The ciphertexts can be adaptively chosen so that information on previous decryptions is available before the next chosen ciphertext is submitted. These attacks have been used in the past to attack the RSA PKCS #1 v1.5 [12] encryption scheme [3], the Cipher-Block-Chaining (CBC) Mode of encryption when used with certain exploitable redundancies (e.g. padding schemes) [2, 5, 15, 16, 17] and the OpenPGP CFB mode [13, 11, 14] itself. The attack on the OpenPGP CFB mode in [13, 11] was able to obtain the entire plaintext using one oracle query which returned to the attacker the entire decryption of C' and the attacks in [14] were able to extend the previous attack to adaptive scenarios.

This paper describes an adaptive chosen-ciphertext attack on the OpenPGP CFB mode of encryption. However, the oracle required is much weaker than

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 82–94, 2006.

in traditional chosen-ciphertext attacks. Access to the complete decryption of the oracle queries is not required. Only access to the validity of the ciphertext is required. Since the oracle is weaker, we suggest calling this a chosen-cipher validation attack. The attack requires an oracle that returns information on an ad-hoc integrity check in the OpenPGP CFB mode. We show that this oracle is likely instantiated in most applications that include OpenPGP. With about 2^{15} oracle queries for an initial setup and about 2^{15} queries for each block, the attacker can determine the first two bytes of plaintext in each block. The attack does require that the attacker also know the first two bytes of plaintext of any one block to bootstrap the process, but we show how to likely obtain these bytes in the majority of circumstances.

2 Cipher-Feedback (CFB) Mode

This section will first describe the standard Cipher Feedback (CFB) mode of operation for block ciphers. The particular variant of CFB mode that is used in OpenPGP will then be described.

The standard CFB mode, by itself, does not appear to be affected by the results in this paper. However, if the data that has been encrypted using standard CFB mode has also been padded to produce an integer number of blocks of plaintext and there exists an oracle for determining when an encrypted message has been correctly padded, the techniques introduced in this paper along with the ideas in [2, 15, 16, 17] can be used to decrypt part or all of any ciphertext block.

2.1 Standard CFB Mode

We describe the standard CFB mode of operation as described in ANSI X9.52 [1] and NIST Special Publication 800-38A [8]. We will assume that the block size of the underlying block cipher, the block size of the CFB mode and the size of the feedback variable are all b bytes, since this is the case for the variant used by OpenPGP. We are doing this simply for ease of explanation and note that nothing in this paper depends upon this fact.

Let $E_K(\cdot)$ be encryption with the symmetric key K by the underlying block cipher. Let $\oplus$ be bitwise exclusive-or. The plaintext message to be encrypted will be $M = (M_1, M_2, \ldots, M_n)$ where each M_i is b bytes long. A random b-byte initialization vector IV is required in order to produce the ciphertext $C = (C_0, C_1, C_2, \ldots, C_n)$ as

$$\begin{aligned} C_0 &= IV \\ C_1 &= E_K(IV) \oplus M_1 \\ C_2 &= E_K(C_1) \oplus M_2 \\ &\vdots \\ C_i &= E_K(C_{i-1}) \oplus M_i \\ &\vdots \\ C_n &= E_K(C_{n-1}) \oplus M_n. \end{aligned}$$

2.2 OpenPGP CFB Mode

The OpenPGP Message Format [4] uses a variant on the standard CFB mode. The main difference with the OpenPGP variant is in the construction of the initialization vector. A random block R is encrypted (described below) as the first block of ciphertext, which serves as an IV for feedback. Two bytes of R are then repeated in the second block in order to quickly check whether the session key K is incorrect upon decryption. We note that this "quick check" is really an integrity check on the key and it is this ad-hoc integrity mechanism, used in a mode of operation that wasn't designed to accommodate it, that allows the attack.

Let $\overline{0}$ be the b-byte all zero block. By $X_{i,j}$ or $[X]_{i,j}$ we will mean the ith and jth bytes of the block X and by X_{i-j} or $[X]_{i-j}$ we will mean the ith through jth bytes of X. The concatenation operation will be represented by $||$. Then, using notation as introduced in the previous section, the ciphertext $C = (C_1, C_2, \ldots, C_{n+2})$ is computed as[1]

$$\begin{aligned}
C_1 &= E_K(\overline{0}) \oplus R \\
C_2 &= E_K(C_1)_{1,2} \oplus R_{b-1,b} \\
C_3 &= E_K([C_1]_{3-b}||C_2) \oplus M_1 \\
C_4 &= E_K(C_3) \oplus M_2 \\
&\vdots \\
C_i &= E_K(C_{i-1}) \oplus M_{i-2} \\
&\vdots \\
C_{n+2} &= E_K(C_{n+1}) \oplus M_n.
\end{aligned}$$

We note here that C_2 is not a full b-byte block, but instead consists of just 2 bytes. We will leave this slight abuse of notation as it will be useful later on.

The random number R is not sent to the receiver to use directly. Its purpose is to provide a random initial value on which to apply the block cipher and encrypt the first plaintext block M_1. Note though that repetition of the two bytes of R in the computation of C_1 and C_2 allows the recipient to quickly check, after decrypting only blocks C_1 and C_2 whether or not the session key is likely correct. This is done by comparing the $b+1$st and $b+2$nd blocks with the $b-1$st and bth blocks. If they match, then the current key was likely the one used to encrypt the data and the remaining blocks can be decrypted. If they don't match, then the key is likely in error, and decryption can be halted. The OpenPGP specification (RFC 2440) describes this "quick check" and most implementations of OpenPGP include it. However, this is really an ad-hoc integrity check that leaks crucial information, as we shall see.

[1] We note that the description of the OpenPGP CFB mode is incorrect in [13] where they incorrectly describe how to compute C_3.

3 Attacking the OpenPGP CFB Mode

This section will describe the attack in detail. First we will describe the oracle required and the information that can be obtained from a successful oracle query. Then we will look at the format of the OpenPGP messages that are being decrypted and show that certain bits of M_1 can be predicted. We will then use the oracle and the known plaintext bits to determine 16 bits from any block of ciphertext.

3.1 The Oracle

This attack requires the presence of an oracle $\mathcal{O}$ that, when given a purported ciphertext C' encrypted using the OpenPGP CFB mode of operation with a given key, will correctly determine whether or not the integrity check described in Section 2.2 was successful. We note that this oracle is likely to be implemented in practical implementations of OpenPGP. RFC 2440 requires that implementations implement this check to prevent obviously incorrect ciphertexts from being decrypted. Further details on the practical aspects of implementing this oracle will be discussed in Section 3.3.

Let's assume that such an oracle does, in fact, exist. Then if the oracle query is successful we know that for the purported ciphertext $C' = (C'_1, C'_2, C'_3, \ldots)$,

$$[C'_1]_{b-1,b} \oplus [E_K(\overline{0})]_{b-1,b} = C'_2 \oplus E_K(C'_1)_{1,2}. \tag{1}$$

We note that C'_1 and C'_2 are known since they are part of the ciphertext C'. If we knew $E_K(C'_1)_{1,2}$, then we could determine $[E_K(\overline{0})]_{b-1,b}$ and similarly if we knew $[E_K(\overline{0})]_{b-1,b}$, then we could determine $E_K(C'_1)_{1,2}$. The method for the attack is now clear. We need to construct a message so that we know $E_K(C'_1)_{1,2}$, that will allow us to obtain $[E_K(\overline{0})]_{b-1,b}$. This value is the same for all messages encrypted using K. Then, we can use that information to determine $E_K(C'_1)_{1,2}$ in specially constructed messages, from which we will get the first two bytes of any plaintext block.

3.2 Obtaining Some Known Plaintext

In order for the attack described in this paper to work, the first two bytes of any one message block M_i must be known by the attacker. In this section, we will describe how an attacker can plausibly determine the first two bytes of M_1 in the majority of circumstances.

According to RFC 2440, the message M that is encrypted using the OpenPGP CFB mode consists entirely of a complete OpenPGP message. Both GnuPG (available at `http://www.gnupg.org`) and PGP Freeware (available at `http://www.pgp.com`) compress data by default before encrypting it. Thus, in the vast majority of circumstances, the encrypted message will consist of a compressed data packet. We will examine this situation first.

When the data being encrypted is a compressed data packet, the first two bytes of this packet are very predictable. The first byte consists of a "packet tag", which indicates the format of the packet contents. If the packet is compressed, then this

packet tag will typically take the value `0xA3`. The second byte indicates the compression algorithm used. Typically this will be `0x01` to represent the ZIP compression algorithm. Other common values for the second byte are `0x00` for uncompressed data, `0x02` for ZLIB compression and `0x03` for BZip2 compression. Thus, if the attacker knows that the encrypted data is compressed, then the first two bytes will either be known or can be determined by trying a small number of possible values. If it is not known that the data is a compressed data packet, then the attacker can reasonably guess that it has been and thus guess the first two bytes.

If it is known that the encrypted data is not a compressed data packet, then there are a small number of choices for the first two bytes of M_1. The first byte will again be a "packet tag" to indicate the format of the packet contents. There are only a small number of possible packet tags likely to be used (14 are defined in the OpenPGP RFC). The second byte will either be the length of the encrypted message, or a portion of the length, depending upon whether or not more than one byte is required to represent the length. It is not unreasonable to assume that the attacker may know, or be able to guess, the packet type of the encrypted message and will likely be able to deduce the length from the message itself. Thus, it is not unreasonable to assume that the attacker will know, or be able to deduce the first two bytes of M_1.

Once the attacker knows $[M_1]_{1,2}$ then, from the definition of the OpenPGP CFB mode, it immediately also knows $[E_K([C_1]_{3-b}||C_2)]_{1,2}$. In the general case, if the attacker knows $[M_{i+1}]_{1,2}$ for $i \geq 1$ then it also knows $[E_K(C_{i+2})]_{1,2}$. We will assume that the attacker knows one of these values.

3.3 The Initial Setup Work

In this section we will describe how the attacker can determine $[E_K(\overline{0})]_{b-1,b}$. We will assume that the attacker has intercepted $C = E_K(M)$ and knows the first two bytes of some plaintext block. As we saw in the last section, this is not an unreasonable assumption. We will first consider the situation where the attacker knows the first two bytes of M_1, next we will consider the situation where the attacker knows the first two bytes of M_{i+1} for $i \geq 1$.

When the attacker knows the first two bytes of M_1, the attacker will construct a ciphertext $C' = ([C_1]_{3-b}||C_2, D, C_3, C_4, \ldots)$ for particular values of D and submit it to the oracle to determine whether or not it is a properly constructed ciphertext. In other words, the oracle will determine whether or not the integrity check in the OpenPGP CFB mode was successful. When it is successful, we can use Equation (1) to determine $[E_K(\overline{0})]_{b-1,b}$.

In this situation, the attacker should use the following algorithm to determine $[E_K(\overline{0})]_{b-1,b}$.

1. Let D be a two byte integer representing the value 0.
2. Construct $C' = ([C_1]_{3-b}||C_2, D, C_3, C_4, \ldots)$.
3. Submit C' to the oracle $\mathcal{O}$. If the oracle returns "success" then

$$[E_K(\overline{0})]_{b-1,b} = C_2 \oplus D \oplus [E_K([C_1]_{3-b}||C_2)]_{1,2}.$$

Otherwise, let $D = D + 1$ and goto Step 1.

The correctness of this result follows immediately from Equation (1), the construction of C' and the fact that we are exhausting over all possible values of D. We note that from the previous section, the attacker knows $[E_K([C_1]_{3-b}||C_2)]_{1,2}$ and thus can, in fact compute $[E_K(\overline{0})]_{b-1,b}$.

We will now consider the more general case when the attacker knows the first two bytes of M_{i+1} for $i \geq 1$. This time the attacker will construct a ciphertext $C' = (C_{i+2}, D, C_3, C_4, \ldots)$ and proceed as in the previous case. The attacker would use the following algorithm to determine $[E_K(\overline{0})]_{b-1,b}$.

1. Let D be a two byte integer representing the value 0.
2. Construct $C' = (C_{i+2}, D, C_3, C_4, \ldots)$.
3. Submit C' to the oracle $\mathcal{O}$. If the oracle returns "success" then

$$[E_K(\overline{0})]_{b-1,b} = [C_{i+2}]_{b-1,b} \oplus D \oplus [E_K(C_{i+2})]_{1,2}.$$

   Otherwise, let $D = D + 1$ and goto Step 1.

Here we note that the attacker knows $[E_K(C_{i+2})]_{1,2}$ from the results in the previous section and thus can also compute $[E_K(\overline{0})]_{b-1,b}$.

It is clear that in either of these situations the oracle will return "success" for some value of D less than 2^{16}. Thus, we would expect that, on average, our attacker would require about $2^{15} = 32,768$ oracle queries in order to determine $[E_K(\overline{0})]_{b-1,b}$. Alternatively, all 2^{16} oracle queries (corresponding to all possible values of D) could be precomputed and submitted in parallel, thereby making the attack non-adaptive.

3.4 Determining 16 Bits of Any Plaintext Block

Once our attacker has determined $[E_K(\overline{0})]_{b-1,b}$, the first two bytes of any plaintext block can be determined with about 2^{15} queries to the oracle. It is a simple variation on the algorithms in the previous section that provides it to the attacker.

In order to determine $[M_{i+1}]_{1,2}$ for any $i \geq 1$ the attacker should use the following algorithm.

1. Let D be a two byte integer representing the value 0.
2. Construct $C' = (C_{i+2}, D, C_3, C_4, \ldots)$.
3. Submit C' to the oracle $\mathcal{O}$. If the oracle returns "success" then

$$[E_K(C_{i+2})]_{1,2} = [C_{i+2}]_{b-1,b} \oplus D \oplus [E_K(\overline{0})]_{b-1,b}.$$

   Otherwise, let $D = D + 1$ and goto Step 1.
4. Then $[M_{i+1}]_{1,2} = [E_K(C_{i+2})]_{1,2} \oplus [C_{i+3}]_{1,2}$.

Again, the correctness of this result follows immediately from Equation (1), the construction of C' and the fact that we are exhausting over all possible values of D. As in the previous section we would expect that the attacker would require on average about 2^{15} oracle queries to determine the first two bytes of any plaintext block.

Also as in the previous section the attack can be made non-adaptive by computing all 2^{16} possible oracle queries and submitting them in parallel.

In the $i = 0$ case, where the first two bytes of M_1 are not already known, they can be obtained by a simple modification to the above algorithm by setting $i = 0$ and replacing C_{i+2} by $[C_1]_{3-b}||C_2$.

Thus, we see that under the reasonable assumption that an attacker can determine the first two bytes of any one message block, with one-time initial work of about 2^{15} oracle queries, the first two bytes of any other message block can be determined with about 2^{15} oracle queries per message block.

3.5 The Attack Without Plaintext

We note that the attack can be implemented even if it is not possible to know the first two bytes of some plaintext block. In this situation, we can simply replace the assumed known value $[M_1]_{1,2}$ with an indeterminate, say Z and implement the algorithms in Sections 3.3 and 3.4. Note that all of the formulae still carry through, including the "$\oplus Z$" term. Now instead of actually determining $[M_{i+1}]_{1,2}$ for any $i \geq 1$, we determine $[M_{i+1}]_{1,2} \oplus Z$. For example, from the definition of the OpenPGP CFB mode we get

$$[E_k([C_1]_{3-b}||C_2)]_{1,2} = [C_3]_{1,2} \oplus Z.$$

Then, in Step 3 of the first algorithm in Section 3.3 we get

$$[E_K(\overline{0})]_{b-1,b} = C_2 \oplus D \oplus [C_3]_{1,2} \oplus Z.$$

Let $A = C_2 \oplus D \oplus [C_3]_{1,2}$, which is a known value. In Step 3 of the algorithm in Section 3.4 we get

$$[E_K(C_{i+2})]_{1,2} = [C_{i+2}]_{b-1,b} \oplus D \oplus A \oplus Z.$$

Let $B = [C_{i+2}]_{b-1,b} \oplus D \oplus A$, which is also now a known value. So,

$$[M_{i+1}]_{1,2} = B \oplus Z \oplus [C_{i+3}]_{1,2},$$

from which we can calculate $[M_{i+1}]_{1,2} \oplus Z$ for many values of i.

If enough of these values are recovered and if the values of the $[M_{i+1}]_{1,2}$ can be bounded, then Z can be determined, thus revealing the plaintext.

For example, if it is known that all of the M_i are ASCII text, then it wouldn't take very many values of $[M_{i+1}]_{1,2} \oplus Z$ to be recovered before Z could be determined.

3.6 Extending the Attack to Other Modes

We note that this attack is not really an attack on CFB mode encryption, but an attack on the two repeated bytes in the first two blocks of an OpenPGP encrypted message. It is likely that similar attacks would be possible with any non-authenticated encryption mode whenever the decryptor checks for repeated

bytes. For example, Hal Finney has pointed out that a similar attack is possible against CBC mode if the decryptor checks for such a plaintext stutter [9].

As with the attack on padding in CBC mode [16], we note that in all of these situations the decryptor is checking for a specific redundancy when using a non-authenticated mode. This practice leaks too much information to the attacker. Such checks should be disabled or an authenticated mode of operation should be used whenever possible. (See also [15].)

4 The Attack in Practice

In previous sections we showed that if a certain oracle exists, then under reasonable assumptions it is possible for an attacker to determine the first two bytes of any plaintext block. This section will examine how likely it is that the required oracle will exist in practice.

We first note that the required oracle $\mathcal{O}$ simply implements the integrity check required in the OpenPGP standard. Thus it is not unreasonable to expect that most implementations would leak this kind of information. This is not a very "powerful" oracle in the sense that it is not leaking a great deal of information. We contrast this with the oracle required in the attack on the OpenPGP CFB mode described in [13, 11]. In that attack only a single oracle query is required to determine the entire plaintext, however, the oracle must return the decryption of the chosen ciphertext. In most environments it is not likely that the attacker will actually have access to the decryption of the chosen ciphertext. It is not unreasonable though to assume that error information is obtained either directly, or through side-channels.

4.1 Non-server-Based OpenPGP Users

By a "non-server-based OpenPGP user" we refer to a human user interacting with an OpenPGP-enabled application. This is, by far, the most common scenario of OpenPGP-based applications. In this scenario it is not unreasonable to assume that some error information regarding the decryption of any ciphertext will be leaked to an attacker. However, it is not likely at all that a human user will attempt to decrypt over 32,000 messages whose decryptions actually fail without realizing that there is a problem and discontinuing.

Thus, we view an attack in this situation as unlikely and will not consider it any further.

4.2 Server-Based OpenPGP Users

A "server-based OpenPGP user" is an automated process that responds to requests that have been encrypted for it using OpenPGP. It attempts to decrypt the request and respond to it appropriately. Few OpenPGP users are server-based as compared with those in the previous section. In this situation, however, it is more likely that information on errors (including decryption errors) will be returned to the requester, which in this case could be an attacker.

There are at least two ways in which an attacker could gain information that would instantiate the oracle. The server could return an error directly to the attacker that the integrity check in the OpenPGP CFB mode failed. As we will see in the next section, some common OpenPGP toolkits do, in fact return error codes which could allow server-based OpenPGP users to, unwittingly, instantiate the oracle. Even if this error code is not returned, information on whether or not the integrity check was successful can likely be obtained through timing or other information. (See [5] for a similar attack that uses this kind of timing information.) RFC 2440 says that the integrity check "allows the receiver to immediately check whether the session key is incorrect". Thus, most implementations would abandon the decryption process immediately if the check failed thereby allowing timing attacks to determine the result of the check. As we will see in the next section, this is what happens.

In fact, timing attacks can be made more feasible by constructing the ciphertext C' (in step 2 of Sections 3.3 and 3.4) so that the decryption process will necessarily take a large amount of time. For example, after the fourth block of C' the value of the ciphertext does not affect the values required for the attack and the ciphertext can be as long as possible. Thus, if the attacker lets $C_5, C_6, \ldots$ be an extremely large number of random blocks, then decrypting C', in the event of a succesful oracle query, will take a relatively large amount of time. This would make detecting a successful oracle query more feasible in some applications.

We also need the oracle to use the same symmetric key K each time that it is invoked. This is not difficult to do. After constructing the ciphertext C' as described in previous sections, this ciphertext should simply be packaged in a standard OpenPGP message. The Public-Key Encrypted Session Key Packet in this message should always contain the same value as in the original OpenPGP message that is being attacked. This packet will contain the key K encrypted for the victim. Each time that the chosen ciphertext is sent to the victim, he/she will decrypt and thus use the same key K each time.

4.3 Common Toolkits May Instantiate the Oracle

To determine the likelihood of this oracle being instantiated we looked at two common toolkits that implement the OpenPGP RFC. We considered GnuPG 1.2.6 [10] and Cryptix OpenPGP [7].[2]

In GnuPG 1.2.6, the integrity check is performed in the `decrypt_data()` function call. If the integrity check is not successful, then the error `G10ERR_BAD_KEY` is returned and decryption is abandoned. Thus, it is not unreasonable to expect that some server-based applications based upon this toolkit would leak this error information either directly, or based upon timing information.

In the Cryptix OpenPGP toolkit, the `PGPEncryptedDataPacket.decrypt()` method performs the integrity check. If the integrity check is not successful then

[2] We note that later versions of these toolkits than the ones examined here have since incorporated countermeasures mentioned in the next section and thus no longer instantiate the oracle. In fact, most implementations of OpenPGP have now incorporated these countermeasures.

an exception is thrown with "No good session key found" and decryption doesn't proceed. Thus, again, it is not unreasonable to expect that some server-based applications based upon this toolkit would leak this error information either directly, or based upon timing or other information.

4.4 Implementing the Attack

We implemented the attack on GnuPG 1.2.4. As it turns out, GnuPG is very helpful in that it appears to display the error "`decryption failed: bad key`" if and only if our oracle does not return success.

We encrypted data with compression turned off and without MDC (see next Section). This was only for ease of implementation, as we have seen (and shall see) this is not required. We then implemented the algorithms in Sections 3.3 and 3.4 as batch scripts cycling through all possible values of D. When we did not get a "`decryption failed: bad key`" we knew that the integrity check was successful and could utilize the given formulae to produce the plaintext. We note that in all of our experiments we only received one value of D that did not give a "`decryption failed: bad key`" error.

Implemented on a 1.8 GHz Pentium M processor running Windows XP Professional, it took under 2 hours to exhaust all values of D. Thus, with less than 4 hours of work an attacker could obtain the first two bytes of any plaintext block. The first two bytes of additional plaintext blocks could be obtained with an additional 2 hours each.

4.5 The Effect of Compression

When 64-bit blocks are used an attacker can obtain at most 25% of the plaintext and when 128-bit blocks are being used at most 12.5%. If the plaintext is uncompressed data this would be devastating. However, typically plaintext OpenPGP data is compressed. In this situation it is not clear if obtaining even 25% of the compressed data will compromise any of the uncompressed data. This is a big mitigating factor against the attack in practice.

5 Attack Prevention

In this section we examine two potential methods for avoiding this attack. One method does not work, the other does.

5.1 Integrity Protected Data Packet Doesn't Work

The OpenPGP RFC is currently up for revision [6] and a new packet type will be included that provides additional integrity protection for encrypted data. GnuPG also implements this additional packet type, called the Symmetrically Encrypted Integrity Protected Data Packet. Encryption using this packet type differs from the non-integrity protected version in two ways. First, the OpenPGP CFB mode of encryption as described is not used. Instead $b + 2$ random bytes,

with the $b+1$st and $b+2$nd bytes matching the $b-1$st and bth bytes, are prefixed to the message and then the standard CFB mode is used to encrypt the prefixed message. As previously, the repeated bytes should be checked and decryption abandoned if they do not match. Second, the entire plaintext message, including the prefix is hashed. The hash value, known as an MDC, is appended to the message and encrypted using CFB mode as well.

Let us first consider the modified CFB mode of operation. We note that, in general, the attack described still works with slight modifications (e.g. replace C_2 with $[C_2]_{1,2}$). However, it will likely now become more difficult for an attacker to obtain the first two bytes of a plaintext message block in order to bootstrap the attack. Notice that now the suggested known plaintext will be in bytes 3 and 4 of the plaintext corresponding to C_2. If the first two bytes of any plaintext message block is known, however, the attack will still be valid.

The purpose of the hash is to detect modifications of the message by the attacker. The attack described in this paper involves modifications to the message and the hash will, in fact, detect it. However, since the check of the hash occurs **after** the decryption of the entire plaintext and the ad-hoc integrity check of the bytes in C_1 and C_2 occurs **before** the decryption of the entire plaintext, it is still likely that information that instantiates the oracle will be leaked. In fact, since a hash will now need to be computed before determining whether or not the plaintext is valid in addition to decrypting the message, it is likely that timing attacks to determine the information will be more effective. We note that GnuPG implements this new packet type and still returns different error codes for the two errors and abandons decryption if the repeated bytes do not match.

Thus, this packet type, by itself, will not prevent this attack, although it may make it more difficult to start.

5.2 The Solution

One obvious solution to prevent this attack is to switch to a true authenticated mode of operation instead of CFB. However, assuming that PGP designers wish to continue using CFB mode, the only method that appears to always work in thwarting this attack is to not instantiate the required oracle. Thus, implementations should not do the check that the repeated bytes in the first two blocks match. If the non-integrity protected packet type is being used, then the data should all be decrypted and an attempt should be made at parsing it. If the integrity protected packet type is being used, then the entire ciphertext should, again, be decrypted and the hash calculated and checked. If it doesn't match, then an error can be thrown.

Unfortunately, for backwards compatibility with the substantial installed user-base it is not possible to remove these random repeated bytes from the encrypted data format. However, future versions should simply ignore these bytes.

If a similar "quick check" that would allow OpenPGP users to quickly determine whether or not the given symmetric key is correct is required, then one possible solution is to include a cryptographic hash of the symmetric key with the ciphertext. The message recipient could then compute the hash of the purported

symmetric key and compare it with the given value before decrypting. Note that this solution would not provide an integrity check on the entire message and would require changes to the OpenPGP RFC.

6 Conclusion

We have described an attack on the OpenPGP CFB mode of operation. This attack works under reasonable assumptions about the knowledge of certain plaintext bytes and requires an oracle which is likely instantiated in most applications using OpenPGP. However, since the attack requires 2^{15} oracle queries, on average, for the initial setup and 2^{15} oracle queries to determine the first two bytes of any plaintext block, it likely won't effect applications with human end users. Server-based applications would be more vulnerable to attack though. In order to thwart this attack, future implementations should not perform the ad-hoc integrity check in the OpenPGP CFB mode.

Acknowledgements. We would like to thank Jon Callas, Hal Finney, Don Johnson and the anonymous reviewers for their helpful comments.

References

1. ANSI X9.52 – 1998, "Triple Data Encryption Algorithm Modes Of Operation", American National Standards Institute, July 29, 1998.
2. J. Black and H. Urtubia, "Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption," In *Proceedings of the 11th USENIX Security Symposium*, pp. 327-338, 2002. Available at `http://www.usenix.org/events/sec02/full_papers/black/black_html/`
3. D. Bleichenbacher. "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1." In H. Krawczyk, editor, *Advances in Cryptology – Crypto '98*, volume 1462 of *Lecture Notes in Computer Science*, pp. 1 - 12. Springer Verlag, 1998.
4. J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, "OpenPGP Message Format," RFC 2440, Nov 1998.
5. B. Canvel, A. Hiltgen, S. Vaudenay and M. Vuagnoux, "Password Interception in a SSL/TLS Channel," In Dan Boneh, editor *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pp. 583-599, Springer-Verlag, 2003.
6. J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, "OpenPGP Message Format," draft-ietf-openpgp-rfc2440bis-XX.txt, *work in progress.*
7. Cryptix OpenPGP, 20041006 snapshot. Available at `http://www.cryptix.org/`
8. M. Dworkin, "Recommendation for Block Cipher Modes of Operation," US Department of Commerce, *NIST Special Publication 800-38A*, 2001. Available at `http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf`
9. H. Finney, *personal communications.*
10. The GNU Privacy Guard, version 1.2.6. Available at `http://www.gnupg.org/`
11. K. Jallad, J. Katz, and B. Schneier, "Implementation of Chosen-Ciphertext Attacks against PGP and GnuPG," In *Proceedings of the 5th International Conference on Information Security*, pp. 90-101, 2002.

12. B. Kaliski, "PKCS #7: RSA Encryption, Version 1.5," RFC 2313, Mar 1998.
13. J. Katz and B. Schneier, "A Chosen Ciphertext Attack against Several E-Mail Encryption Protocols," In *Proceedings of the 9th USENIX Security Symposium* pp. 241-246, 2000. Available at `http://www.usenix.org/publications/library/proceedings/sec2000/katz.html`
14. H.C. Lin, S.M. Yen and G.T. Chen, "Adaptive-CCA on OpenPGP Revisited," In *Information and Communications Security: 6th International Conference – ICICS 2004*, volume 3269 of *Lecture Notes in Computer Science*, pp. 452-464, Springer-Verlag, 2004.
15. C.J. Mitchell, "Error Oracle Attacks on CBC Mode: Is There a Future for CBC Mode Encryption?" to be presented at *ISC 05, The 8th Information Security Conference*, Singapore, September 2005.
    See also: Royal Holloway, University of London, Mathematics Department Technical Report RHUL-MA-2005-7, April 2005, 18 pages. Available at `http://www.ma.rhul.ac.uk/techreports/2005/RHUL-MA-2005-7.pdf`
16. S. Vaudenay, "Security Flaws Induced by CBC Padding-Applications to SSL, IPSEC, WTLS ...," In Lars Knudsen, editor *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pp. 534-545, Springer-Verlag, 2002.
17. A.K.L. Yau, K.G. Paterson and C.J. Mitchell, "Padding oracle attacks on CBC-mode encryption with random and secret IVs," In H. Gilbert and H. Handschuh (eds.) *Fast Software Encryption: 12th International Workshop, FSE 2005*, volume 3557 of *Lecture Notes in Computer Science*, pp.299-319, Springer-Verlag 2005.

Parallelizable Authentication Trees

W. Eric Hall and Charanjit S. Jutla

IBM T.J. Watson Research Center, Yorktown Heights, NY 10598-704

Abstract. We define a new authentication tree in the symmetric key setting, which has the same computational time, storage and security parameters as the well known Merkle authentication tree, but which unlike the latter, allows for all the cryptographic operations required for an update to be performed in parallel. As in Merkle authentication trees, the cryptographic operations required for verification can also be parallelized. In particular, we show a provably secure scheme for incremental MAC with partial authentication secure against substitution and replay attacks, which on total data of size 2^n blocks, and given n cryptographic engines, can compute incremental MACs and perform individual block authentication with a critical path of only one cryptographic operation

1 Introduction

We design a novel incremental MAC (message authentication code) with partial authentication secure against substitution and replay attacks. Before we give detailed definitions, to motivate the definitions and as an application, consider the following problem of checking correctness of memory [4].

In this application, a tamper proof processor uses an insecure storage device (e.g. RAM), open to attack from an adversary who can read and modify the RAM. However, the tamper proof processor may also have a small amount of *internal* memory which can store a MAC of the whole unprotected RAM. The tamper proof processor is required to generate incremental MACs and authenticate individual blocks of RAM (without computing the MAC on the whole RAM). Of course, these computations should not substantially deteriorate the performance of the overall system. A similar situation arises in NAS (network attached storage) systems.

Clearly, there are two extreme (though impractical) solutions to this problem. One is to store the whole RAM (say 2^n blocks) *inside* the tamper proof device. The other is to have 2^n cryptographic engines (e.g. AES or SHA-1) inside the tamper proof device which can compute/authenticate a MAC of the whole unprotected memory using a parallel MAC scheme like XOR-MAC ([1], also see section 6), with a critical path of one cryptographic operation. We stress here that although XOR-MAC can compute incremental MACs with only one engine, to verify an individual block of RAM, it must compute the MAC on the whole RAM (i.e. XOR-MAC is *not* incremental with respect to verification). The ability to verify only a single block (without having to compute the whole MAC) is a crucial requirement of our problem.

One could try a memory/processor tradeoff, by dividing the unprotected memory into super-blocks (say of size 2^m blocks each), and storing an

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 95–109, 2006.

XOR-MAC (one block) of each super-block inside the tamper proof device, and computing/authenticating an XOR-MAC in parallel using 2^m cryptographic engines. Now the tamper proof memory required to store the MACs has been reduced to 2^{n-m} blocks[1]. Note however, that the number of engines plus the tamper proof memory required remains at least $2^{n/2}$.

The main contribution of this paper is a provably secure scheme for this problem, which with only n cryptographic engines, and 1 block of tamper proof memory, can compute incremental MACs (and do individual block authentication) with a critical path of one cryptographic operation. The *only* overhead is an increase in size of the unprotected memory by a factor of two.

Before we describe our scheme, lets describe the other existing solution to this problem. This solution of [4] uses Merkle authentication trees [13]. However, Merkle trees are not fully parallelizable (i.e. although a Merkle tree based solution would only require n cryptographic engines, and 1 block of tamper proof memory, the critical path of an incremental MAC computation would be n cryptographic operations). Not surprisingly though, as we will soon see, ours is also a tree based solution. In section 7 we contrast our scheme with Merkle Trees in more practical settings, and also elaborate on other systems issues.

Main Result. As for XOR-MAC, for every finite secure PRF (pseudorandom function) family F, we construct an *incremental MAC with partial authentication secure against substitution and replay attacks*. The key difference from XOR-MAC is that our scheme does efficient parallel partial authentication. To achieve our goal, the scheme we describe generates auxiliary data which can be stored in unprotected memory. In other words, when provided with correct auxiliary data a single block can be authenticated, whereas no adversary can get a wrong block authenticated even with auxiliary data of its choice.

Surprisingly, the MAC is just a nonce value chosen independently of the data to be authenticated! It is the auxiliary data which provides complete authentication, and in a way we are trying to make the naive solution of "storing MACs outside in unprotected memory" work (see footnote). Informally, the scheme works as follows. Each pair of data blocks is locally MACed along with a new nonce value (chosen randomly or deterministically), and each pair of these nonce values are locally MACed along with yet another new nonce value at a higher level, and so on, till we reach the root of this tree. The new nonce value at the root is the MAC of the scheme. The local MACs are all stored outside in unprotected memory as auxiliary data. We stress that we are not locally MACing two child nodes with the value at the parent as key, but rather MACing all three values together using a global secret key. The former would lead to an insecure solution, because the parent value (if used as the MAC key) would be available to the adversary. The latter can be seen as a tweakable MAC (cf. tweakable block ciphers [11], [10]), i.e. we use the parent value as a tweak.

[1] Note that storing MACs outside in unprotected memory provides only integrity, and not protection from replay attacks.

Note that all the local computations can be done in parallel once the nonces have been chosen. Efficient incrementality follows as an update in a data block only requires updates on the path from this leaf to the root. Efficient parallel partial authentication, i.e. authentication of data at a leaf, follows as it requires checking the local MACs on the path from this leaf to the root.

The only thing that remains to be seen is that we do not reuse nonce values, and that the adversary cannot move around the auxiliary data including replaying old auxiliary data, which is the crux of the proof of security.

We finally describe how an XOR-MAC like scheme PMAC [3] (which uses XOR universal hash function families [8], [5]) can be extended to provide efficient partial authentication.

The rest of the paper is organized as follows. In section 2, we define a novel notion of incremental MAC with partial authentication, and give its definition of secutiy. In section 3, we define our scheme PAT, while the proofs of security are given in section 4. Further optimizations are given in Section 5. In section 6, we describe PMAC Trees, an extension of PMAC which allows partial authentication. In section 7, we discuss various practical issues like cache-based and pipelined systems, and contrast our scheme with both Merkle Trees and PMAC Trees.

2 Definitions

As is often conveniently done, for a function F with many parameters, if the first parameter is a key then F_x will denote the function with the first parameter x. The symbol $||$ will denote concatenation of strings. For a message M consisting of m blocks $M_1, ..., M_m$, $M\langle i, a\rangle$ will denote the modified message with the ith block M_i replaced by a.

We now define simple Message Authentication Codes(MAC)and their security.

Definition 2.1. A simple MAC scheme consists of a function F which takes a secret key x of k bits, a plaintext M of m bits and produces a value τ of h bits.

Security of the simple MAC scheme is defined using the following experiment. An oracle adversary A is given access to the oracle $F_x(\cdot)$. After A queries $F_x(\cdot)$ on plaintexts $M_1, M_2, ..., M_q$ (adaptively), it outputs a pair M', τ', where M' is not one of the queried plaintexts. The adversary's success probability is given by

$$\Pr_x[\mathrm{F}_x(M') = \tau']$$

Let Sec-MAC$_F(q, t)$ be the maximum success probability of any adversary running in time at most t, and making at most q queries.

The above security is known as security under the *impersonation* attack. There is another notion of security called the *substitution attack*. In this model, the adversary A works as follows. After A queries $F_x(\cdot)$ on plaintexts $M_1, M_2, ..., M_q$ (adaptively), and the algorithm returns $\tau_1, \tau_2, ..., \tau_q$, the adversary then outputs an M'. The adversary's success probability is given by

$$\Pr[\exists\, j\ 1 \leq j\ \leq\ q :\ \mathrm{F}_x(M') = \tau_j\ \wedge\ M' \neq M_j]$$

Finally, there is a third notion of security called the *replay attack*. In this model, the adversary works as follows. After A queries $F_x(\cdot)$ on plaintexts $M_1, M_2, ..., M_q$ (adaptively), and the algorithm returns $\tau_1, \tau_2, ..., \tau_q$, the adversary then outputs a pair M', τ', such that M' is just required to be different from M_q. The adversary's success probability is given by

$$\Pr[\mathrm{F}_x(M') = \tau']$$

Of course, there is an adversary which always manages a replay attack. However, the notion can be strengthened by requiring that τ' must agree with some portion of τ_q. As we will see, this is an important notion for incremental MACs.

To the best of our knowledge the next two definitions are novel.

Definition 2.2. An *incremental MAC with partial authentication and with auxiliary data* (**IMACAUX**) consists of the following:

- MAC-AUX: MAC-AUX is a probabilistic function with arguments a key x of k bits, and a plaintext M of size at most 2^n blocks, each block being of size m bits, and which produces a tuple $< \sigma, \tau >$, where σ can be an arbitrarily long string, and τ is of size h bits. The string σ will be called the *auxiliary data* and τ will be called the *authentication tag.*
  We will write [MAC-AUX$_x(M)$] for all tuples $< \sigma, \tau >$ which have non-zero probability of occurring as MAC-AUX$_x(M)$.
- Verify: Verify is a boolean function which takes a key x of k bits, an index $i \in [0..2^n - 1]$, an m bit block a, auxiliary data σ and tag τ, with the following property: Verify$_x(i, a, \sigma, \tau)$ must return 1 if there exists an M, such that the ith block of M is a, and $< \sigma, \tau >$ is in [MAC-AUX$_x(M)$].
- INC-MAC-AUX (update): INC-MAC-AUX is a probabilistic function which takes (apart from the key x) an index i, a block of plaintext a, auxiliary data σ, and tag τ, and produces either a tuple $< \sigma', \tau' >$ or $\perp$. If there exists an M such that $< \sigma, \tau >$ is in [MAC-AUX$_x(M)$] then it must return a $< \sigma', \tau' >$ such that $< \sigma', \tau' >$ is in [MAC-AUX$_x(M\langle i, a\rangle)$].

We note that one way to implement the above is to embed M in the auxiliary data σ. We have said nothing about the security of IMACAUX, which we address next.

Definition 2.3 (*Security under substitution and replay attacks*)**.** The security of an IMACAUX scheme $\langle$ MAC-AUX, Verify, INC-MAC-AUX$\rangle$ is defined using the following experiment. A three oracle adversary A is given access to the oracles MAC-AUX$_x(\cdot)$, Verify$_x(\cdot, \cdot, \cdot, \cdot)$, and INC-MAC-AUX$_x(\cdot, \cdot, \cdot, \cdot)$. The adversary first requests an initial MAC-AUX$_x$ to be computed on an initial plaintext M^0 of 2^n blocks. Let MAC-AUX$_x(M^0)$ return $< \sigma_0, \tau_0 >$.

Subsequently, the adversary requests a sequence of q adaptive update operations, each specifying a block number and a block of replacement text, along with auxiliary data of its choice. However, τ supplied on each request must be same as that produced in the previous request.

More precisely, for each j, $0 < j \leq q$, let i_j be the block number for the jth incremental update, with text a_j. Let $I_j = \text{INC-MAC-AUX}_x(i_j, a_j, \sigma'_{j-1}, \tau_{j-1})$, where i_j, a_j, and σ'_{j-1} are adaptively chosen by the adversary. As required in definition 2.2, the return value I_j is either $\perp$ or $< \sigma_j, \tau_j >$. If the return value is $\perp$, then we let $M^j = M^{j-1}$ and $\tau_j = \tau_{j-1}$. If the return value is $< \sigma_j, \tau_j >$ then we let $M^j = M^{j-1}\langle i_j, a_j \rangle$. In the latter case we also say the update was *valid*.

Finally, the adversary requests a verification on a block at index i, with text a, such that a is different from M_i^q. The adversary's *success probability* is given by

$$\Pr_x[\text{Verify}_x(i, a, \sigma', \tau_q) = 1]$$

Again, σ' is adaptively chosen by the adversary. However, τ_q remains the same as produced by the last valid update. We stress that a is *only required to be different from the last plaintext* block at index i, and the adversary is allowed to choose an a (along with a related σ') which may have occurred at an earlier point of time at index i.

Let $\text{Sec-IMACAUX}_{\langle \text{MAC-AUX,Verify, INC-MAC-AUX} \rangle}(q, t)$ be the maximum success probability of any adversary running in time at most t, and making at most q INC-MAC-AUX requests.

3 Parallelizable Authentication Tree

We now describe an IMACAUX scheme called PAT with a description of each of its component functions, i.e. MAC-AUX, Verify and INC-MAC-AUX. The functions will employ a simple MAC scheme F (see definition 2.1) with the same secret key x as chosen for PAT. We will describe the various size parameters later (before Theorem 1). All F computations (which will be the only cryptographic operations) in the computation of these functions can be done in parallel.

MAC-AUX. Given a 2^n block plaintext M, we now describe a valid MAC-AUX on it, i.e. all pairs $< \sigma, \tau >$ which are in $[\text{MAC-AUX}_x(M)]$. $\text{MAC-AUX}_x(M)$ will be an r-ary labeled tree (see Figure 1). For simplicity, we only consider r=2. The tree will be balanced and will have 2^n leaves. For a non-leaf node u, *left*(u) will denote its left child and *right*(u) will denote its right child.

- Leaf nodes u will have as label a data value *data*(u) which will be the corresponding block from M.
- the labels at each non-leaf node u will be a nonce value $V(u)$, and a local mac value $C(u)$ such that $C(u) = \text{F}_x(V(\textit{left}(u)) \,||\, V(u) \,||\, V(\textit{right}(u)))$, where $||$ is the concatenation operator (see Fig 1).

The auxiliary data σ is just the whole labeled tree except the root V label. The authentication tag τ is just the V label of the root node. Thus, $< \sigma, \tau >$ is the whole labeled tree. Any such $< \sigma, \tau >$, which satisfies the local F constraints, is in $[\text{MAC-AUX}(M)]$.

Verify. Let the input to the function be (i, a, σ, τ).

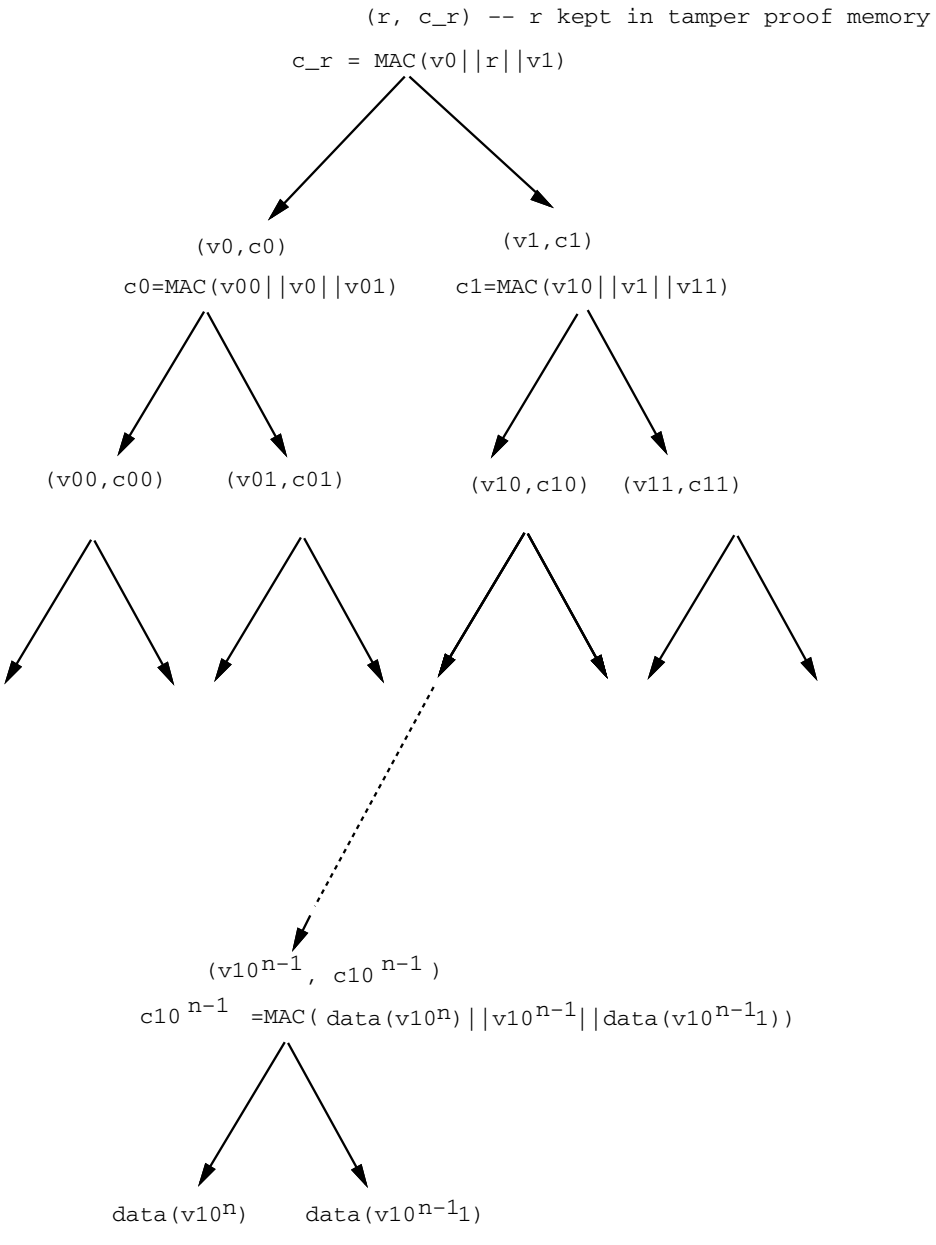


Fig. 1. Parallelizable Authentication Tree (PAT)

Since the boolean function Verify takes the leaf node index as an argument (i.e. i), it just checks that the nonce values are consistent with C along the path from the root to this leaf. We will give an algorithmic description of this function.

More precisely, let the path from the root to the specified leaf z (the ith leaf) be $u_0 = r,\ u_1, ..., u_{n-1},\ u_n = z$. Recall that $V(u_0)$ is τ.
If for all levels $y = 0$ to $n-2$, $C(u_y)$ equals
$\mathrm{F}_x(V(\mathit{left}(u_y)) \,||\, V(u_y) \,||\, V(\mathit{right}(u_y)))$, and
if u_n is the left child of u_{n-1} then $C(u_{n-1})$ equals
$\mathrm{F}_x(a \,||\, V(u_{n-1}) \,||\, \mathrm{data}(\mathit{right}(u_{n-1})))$, and
if u_n is the right child of u_{n-1} then $C(u_{n-1})$ equals
$\mathrm{F}_x(\mathrm{data}(\mathit{left}(u_{n-1})) \,||\, V(u_{n-1}) \,||\, a)$
then return 1, else return 0.

Note that the nonce values V and the data values in the above verification are given as part of σ.

INC-MAC-AUX (update)**.** Let the input to the function be (i, a, σ, τ). We will give an algorithmic description of this function.

Since the tree has 2^n leaves, the root can be considered to be at the 0th **level**, whereas the leaves are at the nth level. A path from root r to leaf z (the ith leaf) can then be written as $u_0 = r, u_1, ..., u_{n-1}, u_n = z$. This will be called the *update path*. Let $v_1, ...v_n$ be the *sibling path*, i.e. for each $y \in [1..n]$, v_y is the sibling of u_y. The update algorithm first checks that the nonce values V provided as part of σ of the update and sibling path nodes are correct. More precisely,

Step 1 : If for all $y = 0$ to $n-2$,
$C(u_y)$ equals $\mathrm{F}_x(V(\mathit{left}(u_y)) \,||\, V(u_y) \,||\, V(\mathit{right}(u_y)))$, and
$C(u_{n-1})$ equals $\mathrm{F}_x(\mathit{data}(\mathit{left}(u_{n-1})) \,||\, V(u_{n-1}) \,||\, \mathit{data}(\mathit{right}(u_{n-1})))$,
then perform the update below, else return $\perp$.
Step 2 : For $y = 0$ to $n-1$, update $V(u_y) = \tau + n - y$.
For $y = 0$ to $n-2$, update $C(u_y) = \mathrm{F}_x(V(\mathit{left}(u_y))||V(u_y)||V(\mathit{right}(u_y)))$.
Step 3 : At the leaf node $z = u_n$, update $\mathit{data}(u_n) = a$, and
update $C(u_{n-1}) = \mathrm{F}_x(\mathit{data}(\mathit{left}(u_{n-1})) \,||\, V(u_{n-1}) \,||\, \mathit{data}(\mathit{right}(u_{n-1})))$.

The newly re-labeled tree is returned as σ, τ. Note that τ is just the new $V(u_0)$, i.e. old τ plus n. Note that all the F operations in steps 1 to 3 combined can be done in parallel. If these steps are indeed done in parallel in some implementation, there could be a possibility that Step 1 fails, in which case step 2 and 3 are abandoned, and the old σ, τ is reverted to. Since the chances of Step 1 failing are hopefully small, this does not incur much cost.

That finishes the description of INC-MAC-AUX.

4 Security of Parallelizable Authentication Tree

4.1 Optimized Initialization of the Authentication Tree

Since, in the definition of security which described the adversarial model (see definition 2.3), the adversary only makes one initial call to MAC-AUX, our scheme (i.e. MAC-AUX) will return a σ_0, with all V values set to zero, and similarly the τ_0 set to zero. This can be seen as initializing the data structure.

Moreover, with this simple initialization, all the internal node MAC values C are the same, and hence need not be computed multiple times. Further still, we will assume that the data values are also initialized to zero, in which case the MAC values at the leaves will also be same.

This does not change the adversarial model, since if the adversary requested a different initial plaintext M^0, our algorithm could return the requested data structure by simulating several updates.

For example, before the first update request, as mentioned all nonce values are zero. After the first update request, the nonce value of the node closest to the leaf, i.e. u_{n-1} will be 1, and the nonce value of the node closest to the root, i.e. u_1 will be $n-1$, and the nonce value of the root will just be n.

As we will see, this assures that in each incremental request, the nonce values are chosen afresh, i.e. are never repeated.

Let d be the number of bits in the nonce labels V above.
Let h be the number of bits in the C label.
Let m be the number of bits in each block of data stored at a leaf.
Let k be the number of bits in the key.
Let 2^n be the number of leaves in the balanced binary tree.
Let F be a function $F : \{0,1\}^k \times \{0,1\}^{\max\{d,m\}+d+1} \rightarrow \{0,1\}^h$

The above three algorithms together describe an **IMACAUX** scheme and will be called $PAT^F(k, m, n, h, d)$ (parallelizable authentication tree) when using F as its local MAC algorithm.

In the following theorem, Sec-MAC$_F$ refers to the security under the impersonation attack in definition 2.1.

Theorem 1: For any positive integers k, m, n, h, d, q, t and any function $F : \{0,1\}^k \times \{0,1\}^{\max\{d,m\}+d+1} \rightarrow \{0,1\}^h$

$$\text{Sec-MAC}_F(2qn, t) \;\geq\; \text{Sec-IMACAUX}_{PAT^F(k,m,n,h,d)}(q, t)$$

Proof: Let A be a three oracle adversary as in the experiment of Theorem 2.3. Let B be an oracle adversary which participates in the experiment of Definition 2.1 (simple MAC scheme), and is given access to the oracle $F_x(\cdot)$, with x chosen uniformly at random.

Adversary B will simulate the three oracles for A, i.e. MAC-AUX, Verify, and INC-MAC-AUX of PAT using its own oracle $F_x(\cdot)$. B will then just imitate A. During the simulation, B will make several oracle calls to $F_x(\cdot)$. It will also maintain a List of pairs. Let List$_j$ denote all pairs (a, b) till the end of the jth INC-MAC-AUX (update) query made by A, such that B during its simulation made a call to F_x with input a and $F_x(a)$ returned b. Ultimately, while simulating the final Verify query of A (or even during the INC-MAC-AUX queries' step 1, which is essentially a verify query), we show that for some a and b determined by A, such that a has not been a query to $F_x(\cdot)$, the verify query returns 1 iff $F_x(a)$ equals b. This claim proves the theorem.

We will follow notation from definition 2.3 for all the queries of adversary A.

We say that a node u was *assigned* a value by the algorithm PAT in the jth update query if the jth update is valid and this node is in the update path of the jth query. Clearly, the root is assigned a value in each valid update query, and a leaf is assigned a value in an update query only if it was the leaf being updated and the update was valid. For each node v, let $last(v, j)$ be the largest $j' \leq j$ such that v was assigned a value in the j'th query. If it was never assigned a value in an update query ($\leq j$) then $last(v, j) = 0$. Let $latest(v, j)$ be the V value assigned to v by the algorithm INC-MAC-AUX (of PAT) in query $last(v, j)$; if v is a leaf node then $latest(v, j)$ is just the data assigned to that leaf in the $last(v, j)$ query. The initial MAC-AUX query will be considered as the 0th query. *Without loss of generality*, assume that all updates retuen non-$\perp$ values (i.e. are valid updates), as updates which return $\perp$ do not cause any change in state. Fact 1(a) below follows from the optimized initialization of the authentication tree.

Fact 1: (a) For all u, $latest(u, 0) = 0$.
(b) For $l \geq 1$, for all u if $last(u, l) = l$, then $latest(u, l) = l * n-\text{level}(u)$, where level($u$) is the distance of u from the root.
(c) For all u, j and l, if $last(u, j) \leq l$, then $latest(u, j) = latest(u, l)$.

Claim 2: for any non-leaf nodes u, v, $u \neq v$, for all t, t',
$latest(u, t) \neq latest(v, t')$, or
$latest(u, t) = latest(v, t') = 0$.

Proof: Suppose both values are non-zero. Then by fact 1(a) and 1(c), let $last(u, t) = l \geq 1$, and $last(v, t') = l' \geq 1$. Then by Fact 1(b), $latest(u, t) = l * n-\text{level}(u)$, and $latest(v, t') = l' * n-\text{level}(v)$. It follows that if these two values are same then level(u) = level(v), and $l = l'$ (as $|level(u) - level(v)| < n$). But that is impossible, as in the lth update query only one node at each level gets a new V value. □

In the algorithm INC-MAX-AUX, each node in the update path and the sibling path is first verified. Define S_j ($1 \leq j \leq q$) to be the union of nodes in the update and sibling path in the jth update query. We will also call the final Verify query of adversary A, the $(q+1)$th query. We define S_{q+1} to be the collection of nodes in the path from the leaf being verified to the root.

Unless otherwise mentioned, whenever we refer to $V(u)$ for some node, we will mean the V value supplied for node u by adversary in the jth query. The same will hold for data(u).

Claim 3: Either (a) for every j, $1 \leq j \leq q+1$, for all nodes $u \in S_j$, $V(u)$ (or data(u)) = $latest(u, j-1)$, or
(b) there exists a j, $1 \leq j \leq q+1$, and a non-leaf node $u \in S_j$ such that ($V(left(u)) \,||\, V(u) \,||\, V(right(u))$) is not in List_{j-1} (i.e. is not equal to the first entry of any pair in List_{j-1}).

Proof: Suppose (b) does not hold. Then we prove (a) by induction on j.

Base case ($j = 1$). By Fact 1(a), for all u, $latest(u, j-1) = 0$. Hence, the only entry in List_0 is the one corresponding to argument $(0 \,||\, 0 \,||\, 0)$. Now, suppose (a) does not hold for some $u \in S_j$ (i.e. $\text{V}(u) \neq 0$). If u is a non-leaf node then (b) does not hold for that u (with $j = 1$). If u is a leaf node then (b) does not hold for parent of u.
Suppose that the induction hypothesis holds for $j - 1$.

We do a nested induction on the nodes of the update path. If $j < q + 1$, consider a non-leaf node u in S_j, but restricted to the update path. If $j = q+1$, then consider a non-leaf node u in S_{q+1}. Suppose its supplied $V(u)$ is indeed same as $latest(u, j-1)$. We will show that for v being either child of u, $V(v)$ (or $\text{data}(v)$) supplied in the jth query is indeed $latest(v, j-1)$.

Let $last(u, j-1)$ be $l \leq j-1$. If $l = 0$, then both its children v have $V(v) = 0 = latest(v, j-1)$. Otherwise u is in S_l and the lth update was valid. Now, since $last(u, j-1) = l$, neither of u's children v have $last(v, j-1) > l$. In fact one of them, say $v1$, has $last(v1, j-1) = l$, and the other, say $v2$, has $last(v2, j-1) < l$ (i.e. $v2$ is in the sibling path in the lth query). Moreover, $v2$ is also in S_l. Thus, by outer induction hypothesis, V value supplied for $v2$ in the lth query is $latest(v2, l-1)$, which by Fact 1(c) is same as $latest(v2, j-1)$.

On the other hand, V value assigned to $v1$ in lth query is $latest(v1, j-1)$. Without loss of generality, assume $v1$ is the left child. Then, $(latest(v1, j-1) \,||\, latest(u, j-1) \,||\, latest(v2, j-1))$ was inserted in $List_l$. Moreover, by Claim 2, and Fact 1(b), these are the only values in List_{j-1}, with middle value $latest(u, j-1)$. Now, suppose for one of these v (i.e. $v1$ or $v2$), $V(v)$ (i.e. supplied in the jth query) is not the same as $latest(v, j-1)$. But, by the claim that (b) does not hold we have that $(V(v1) \,||\, latest(u, j-1) \,||\, V(v2))$ is in List_{j-1}, which leads to a contradiction.

Thus, for v being either child of u, $V(v)$ supplied in the jth query is indeed $latest(v, j-1)$. That completes the nested induction step.

But since, $V(r)$ supplied in the jth query is indeed same as $latest(r, j-1)$ (as the τ values are not altered by the adversary), the induction step is proven. □

We are now ready to complete the proof of Theorem 1. We point out again that we use notation from Definition 2.3. Also, recall that we assume, w.l.o.g that all updates returned non-$\perp$ values. Let **Verified** be the event $\text{Verify}_x(j, a, \sigma', \tau_q) = 1$.

Let u correspond to the leaf at index i as specified in the final Verify query of A. Then $M_i^q = latest(u, q)$. This follows from the fact that either the ith leaf was never validly updated, in which case $M_i^q = M_i^0 = latest(u, 0) = latest(u, q)$, or it was updated at $last(u, q) = l$ to be block a_l, in which case $M_i^q = a_l = latest(u, l) = latest(u, q)$.

Since, data a to be verified at leaf node u corresponding to block number j is different from M_j^q, and hence is different from $latest(u, q)$, Claim 3(a) does not hold. Hence, Claim 3(b) must hold. Let j be the query and v be the non-leaf node in S_j, such that $(V(left(v)) \,||\, V(v) \,||\, V(right(v)))$ is not in List_{j-1}. Let $M' = (V(left(v)) \,||\, V(v) \,||\, V(right(v)))$, and $\tau' = C(v)$, where $C(v)$ is the value supplied by the adversary A in the jth query. Now $F_x(M') = \tau'$, for otherwise PAT would return $\perp$ (see step 1 of INC-MAC-AUX), which is not possible as we

had assumed w.l.o.g. that no update returns $\perp$. Thus if **Verified** happens with probability p, then in the experiment of Definition 2.1, $F_x(M') = \tau'$ happens with probability at least p as well. □

5 Optimizations

The PAT scheme used a function $F : \{0,1\}^k \times \{0,1\}^{\max\{d,m\}+d+1} \rightarrow \{0,1\}^h$, and the security of PAT required this function to be a secure MAC. It is well know that if F is a secure pseudorandom function family, then it is also a secure MAC (as in definition 2.1) [7]. The question then boils down to building an efficient PRF from $\max\{d, m\} + d + 1$ bits to l bits.

First note that, a simple MAC scheme obtained from a PRF is susceptible to birthday attacks, and hence it is secure only up to at most $2^{h/2}$ queries. This implies, that d need only be $h/2$. Thus, one needs to implement a PRF from $3h/2$ bits to h bits, with a key of size k. In fact, instead of considering a binary tree, one could consider a 4-ary tree, in which case one needs a PRF from $5/2h$ to h. HMAC [9] with SHA-1 is a reasonable option for such a PRF (with $h = 160$) where the key is fed through the IV of SHA (see [9] for a discussion of keying through the IV).

In such a 4-ary tree implementation, the number of leaf nodes is 4^n. Thus, the amount of data can be $4^n * (h/2)$, assuming each block of data is 80 bits, i.e. m=80. The number of non-leaf nodes is $(4^n - 1)/3$, and each internal node has memory requirement of $d + h = 3h/2$ bits. Thus, the memory overhead for auxiliary data is 100%.

In an 8 or 16 N-ary tree implementation, the storage overhead is reduced for the intermediate nodes to a more practical level of less than 43% or 20% respectively.

6 Alternative Schemes

Definition 6.1 (Universal Hash function family [5])**.** A set $\mathcal{H}$ of functions with signature $\{0,1\}^t \rightarrow \{0,1\}^n$ is said to be a Universal Hash Function family if $\Pr_H[H(i) = H(j)] \leq 2^{-n}$ for all i, j, where the probability is taken over H chosen uniformly at random from $\mathcal{H}$.

Definition 6.2 (XOR-universal function family [8])**.** A set $\mathcal{H}$ of functions with signature $\{0,1\}^t \rightarrow \{0,1\}^n$ is said to be an XOR-universal family if $\Pr_H[H(i) \oplus H(j) = k] \leq 2^{-n}$ for all i, j, k, where the probability is taken over H chosen uniformly at random from $\mathcal{H}$.

In general, if $h1$ is an XOR-universal hash function family with key $k1$, from c bits to c bits, then $a + h1_{k1}(b)$, is a universal hash function family from $2c$ bits ($\langle a, b \rangle$) to c bits.

Instead of describing XOR-MAC [1], we now briefly describe the essential idea behind the PMAC [3] scheme, which is a more advanced version of XOR-MAC.

Let M be a plaintext of length 2^n blocks, on which PMAC is to be computed. Each block is of length m. For each block of M identify a node u with a unique

address, namely $address(u)$. The data at node u will be called $M(u)$. The PMAC scheme employs a pseudorandom permutation (PRP) E from m bits to m bits with key x. The PMAC algorithm also picks a function H randomly and uniformly from a universal hash function family. Thus, the secret key of the PMAC algorithm is x and the function H. The PMAC value of M is then just

$$E_x(\bigoplus_{i=1}^{2^n} E_x(H(M(u_i) \mid\mid address(u_i))))$$

In other words, the function H is used to hash the data value at each node, along with its address, to be the input of the PRP E_x. The output of all the E_x operations is xored and sent through E_x once again to obtain the mac value.

Note that this scheme has a critical path of length two (E operations) in a parallel implementation. Moreover, it allows for fast incremental updates. For example, if the PMAC value of M is τ, then the PMAC value of $M\langle i, a\rangle$ is obtained by first employing E_x^{-1} on τ to get a temporary value s. Then, s is updated by xoring it with $E_x(H(M(u_i) \mid\mid address(u_i)))$ and $E_x(H(a \mid\mid address(u_i)))$, i.e.

$$s = s \oplus E_x(H(M(u_i) \mid\mid address(u_i))) \oplus E_x(H(a \mid\mid address(u_i)))$$

Finally, the new PMAC value is computed as $E_x(s)$.

In this section we point out that XOR-MAC can also be extended to be an efficient and parallel IMACAUX scheme. Recall that XOR-MAC was deficient in the sense that it did not allow efficient partial authentication. So to add that feature, we first must use a version of XOR-MAC which uses universal hash functions. The schemes PMAC ([3]) and XECB-MAC([6]) are such schemes.

We now show how to extend PMAC to allow efficient partial authentication.

This scheme will also have a binary tree structure as the auxiliary data, just as in PAT. The leaves will just be the 2^n nodes described above in PMAC. As in PAT, each internal node has a C value and a V value. The V value is only computed during updates or verification (i.e. it is secret), and is not stored as a part of the tree (only C values are stored or returned as part of σ). The local MAC is computed using PRP E. In particular, the V value at an internal node is computed by an xor-sum of the V values of all its children, and then the C label of this internal node is computed by encrypting the V value with E.

The V value at leafs is computed slightly differently. The function H is used to hash the data value at each leaf, along with its address, to be the input of the PRP E. The result of this application of E is the V value of the leaf.

We observe that the V value of any internal node u (including the root) is the XOR-sum of the V values of all the leaf nodes under this internal node u. By keeping the C labels of the internal node, we assure integrity of a leaf by just checking the path from the leaf to the root. The "address sensitivity" is built into the leaves using the function H.

This scheme however has the drawback of having a critical path of a decryption followed by encryption in a parallel implementation. Essentially, all the C values on an update path and the sibling path must first be decrypted, and

then the V values are updated by the changed value at the leaf, and finally the new C values are obtained by an encryption. Contrast this with INC-MAC-AUX (update) of PAT where all the F operations in step 1 to 3 can be done in parallel.

7 Systems Issues

In this section we address several systems issues, which also highlights the motivation behind PAT. Naturally, the comparison of this scheme with Merkle trees and PMAC trees arises.

In any practical tamper proof system, we do not expect to fully parallelize the operations involved in an update or memory authentication. Most likely, the tamper proof system will have a reasonably large cache (i.e. inside the tamper proof device). This would allow the updates to be done in a lazy fashion.

Thus, the PAT tree is never completely updated in the insecure memory after the initialization (unless completely flushed from the cache – an atypical event). When a new data node is brought into cache, all the nodes in the update path between the leaf (i.e. data node) and the lowest ancestor which is already present in the cache, must be brought in (or filled). When a modified (or dirty) node is cast-out (cache cast-out), not only is it updated in the PAT representation in the insecure memory, its parent even if it still remains in cache is assigned a new V label. The nodes above this parent node are not required to get a new label unless their children are themselves being cast out. Essentially, all nodes which are in the insecure memory, must be consistent with the lowest ancestor which is in cache – as all nodes in the cache are considered authentic.

We point out that entries at higher levels of the tree cover larger areas of memory, eventually covering whole "working sets" and benefiting from "locality-of-reference" effects in the caching. The highest levels of the tree will entirely fit in the cache, and so after the initial verify for the first misses, will never be cast-out (and so will never be updated in memory). This will appear as multiple smaller trees with the "root" V's being updated and keep in the internal cache.

Having discussed the benefits of cache, practical high performance solutions will still have to "pipeline" memory and buses. Pipelined memories and buses tend to reach their peak bandwidth when bursting blocks of data. This will imply authentication trees with higher N-ary (8 or 16) structures to take advantage of the burst bandwidth. Given the large block size of the nodes, it will be less costly (less hardware buffering in the data path) if a block can be processed in a "pipelined" (higher bandwidth) verification engine as soon as it arrives from the bus.

For PAT this implies pipelining the intermediate level node blocks as a sequence of data bursts from the root to the leaf (top-down), thus having the higher level V available for the processing of the next lower level. To reduce the hardware buffering in the update (cache cast-out) path, the processing will need to proceed in the same top-down manner phased slightly ahead of the top-down verification (cache fill) so that the previous cache line contents have been saved (update) to external memory before being overwritten by the new contents from the verification path (cache fill).

In contrast, Merkle Tree updates have a computational dependency from the leaf nodes to the root node (bottom-up) and a top-down dependency only for verifications. This again implies pipelining the intermediate level node blocks as a sequence of data bursts, but this time from the leaf to the root (bottom-up) for both verifications and updates. Again, with the bottom-up updates (cache cast-out) phased slightly ahead of the bottom-up verifications (cache fill) to avoid additional hardware buffering. The hash operations for verify can be done in a pipelined or parallel manner, and the updates which were caused by cast-outs at various levels in the tree cache will typically be unrelated and so may also be pipelined. But additional complexity will be involved in the handling of cases where a higher level parent node in the cast-out sequence will require the result from a next lower level child node's update (cast-out) hash computation before being able to begin it's own store and hash operations. This additional delay would then also be reflected in the verification processing to maintain the update to verify or cache block save to fill phase alignment. Observe that this effect is only reduced by hash engines with lower individual block computation latencies; higher bandwidth engines will not reduce the delay for an individual update, only the average delay for multiple parallel updates.

Finally, PMAC Tree updates have a bottom-up computational dependency that is similar to Merkle Trees (leaf to root), but this time only the leaf node to the first intermediate level has the full latency of the crypto computation (hash followed by encryption), the higher levels are simple XORs with very small latency effects. The PMAC tree verification is a comparison between levels and so is fully parallelizable. As in the Merkle Tree case, the pipelining of the intermediate level nodes would be from the leaf to the root (bottom-up) for both verify and update, again with the same phase requirements. Similarly, the verify and the typically unrelated updates may be pipelined. In contrast to Merkle Trees however, the additional complexity will only occur when handling cases where the first intermediate level parent node in the cast-out sequence requires the hash and encrypt result from a data level child's update (cast-out) before being able to begin it's own encrypt and store operation. This additional delay would then need to be reflected in the verify processing to meet the update to verify phase requirement, but in the PMAC case, this will only occur at most once during a cache miss sequence. Unfortunately, the data level processing involved in computing the encrypted hash values is the same for both update (cast-out) and verify (fill), while the processing of the intermediate levels involve encryption for update (cast-out) and decryption for verify (fill). This means that the verify (fill) path will require a crypto engine capable of both the forward and inverse version of the chosen cryptographic algorithm, and this will typically require more gates and cause more potential timing problems.

Acknowledgments. The authors would like to thank several anonymous referees for their helpful comments.

References

1. M. Bellare, R. Guerin and P. Rogaway, XOR MACs: New methods for message authentication using finite pseudorandom functions, Advances in Cryptology - Crypto 95 Proceedings, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed, Springer-Verlag, 1995.
2. M. Bellare, O. Goldreich, S. Goldwasser, "Incremental Cryptography with Applications to Virus Protection", Proc. STOC 1995
3. John Black and Phillip Rogaway, "A Block-Cipher Mode of Operation for Parallelizable Message Authentication", Advances in Cryptology - EUROCRYPT '02, Lecture Notes in Computer Science, Springer-Verlag, 2002
4. M. Blum, W. Evans, P. Gemmell, S. Kannan, M. Naor, " Checking the Correctness of Memories", Algorithmica, Vol 12, pp 223-244, 1994.
5. J. Carter, M. Wegman, "Universal Classes of Hash Functions", *JCSS*, Vol. 18, 1979, pp 143-154.
6. V.D. Gligor, P.Donescu, "eXtended Electronic Code Book MAC", `http://csrc.nist.gov/encryption/modes/proposedmodes`
7. O. Goldreich, S. Goldwasser, and S. Micali, " How to construct random functions", J. ACM, vol. 33, no. 4, 1986.
8. Hugo Krawczyk, "LFSR-based Hashing and Authentication", *Proc. Crypto 94*, LNCS 839, 1994
9. M. Bellare, R. Canetti, and H. Krawczyk, " Keying hash functions for message authentication", Advances in Cryptology—Crypto '96, 1996.
10. C. S. Jutla, " Encryption Modes with Almost Free Message Integrity", Eurocrypt 2001, LNCS 2045.
11. Moses Liskov, Ronald L. Rivest, David Wagner: Tweakable Block Ciphers, CRYPTO 2002: 31-46
12. M. Luby, "Pseudorandomness and Cryptographic Applications", Princeton Computer Science Notes, Princeton Univ. Press, 1996
13. R. Merkle, "A certified digital signature", Crypto 89, LNCS 435, 1989.

Improved Time-Memory Trade-Offs with Multiple Data

Alex Biryukov[1], Sourav Mukhopadhyay[2], and Palash Sarkar[2]

[1] Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC,
Kasteelpark Arenberg 10, B–3001 Heverlee, Belgium
http://www.esat.kuleuven.ac.be/~abiryuko/
[2] Cryptology Research Group, Applied Statistics Unit,
Indian Statistical Institute, 203, B.T. Road, Kolkata 700108, India

Abstract. In this paper we study time/memory/data trade-off attacks from two points of view. We show that Time-Memory trade-off (TMTO) by Hellman may be extended to Time/Memory/Key trade-off. For example, AES with 128-bit key has only 85-bit security if 2^{43} encryptions of an arbitrary fixed text under different keys are available to the attacker. Such attacks are generic and are more practical than some recent high complexity chosen related-key attacks on round-reduced versions of AES. They constitute a practical threat for any cipher with 80-bit or shorter keys and are marginally practical for 128-bit key ciphers. We show that UNIX password scheme even with carefully generated passwords is vulnerable to practical trade-off attacks. Our second contribution is to present a unifying framework for the analysis of multiple data trade-offs. Both Babbage-Golic (BG) and Biryukov-Shamir (BS) formulas can be obtained as special cases of this framework. Moreover we identify a new class of *single table* multiple data trade-offs which cannot be obtained either as BG or BS trade-off. Finally we consider the analysis of the rainbow method of Oechslin and show that for multiple data, the TMTO curve of the rainbow method is inferior to the TMTO curve of the Hellman method.

Keywords: time/memory/data trade-off, block-cipher, key sizes.

1 Introduction

In 1980, Hellman [8] introduced the technique of time-memory trade-off (TMTO) attack on block ciphers. In its more general form, it can be viewed as a general one-way function inverter. The original work by Hellman considered inverting a one-way function f at a single data point. Babbage [1] and Golic [7] (BG) have shown that in the context of stream ciphers multiple data points can be used by another trade-off attack relying on birthday paradox. Independently, Biham [3, 2] has shown that birthday-paradox trade-off applies to block-ciphers in a frequently changing key scenario. Both BG and Biham's results show that theoretical strength of a block or stream cipher without an IV (nonce) can not exceed the square root of the size of the key space. However birthday trade-offs suffer from a weakness: they lack flexibility due to strong binding of memory

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 110–127, 2006.

complexity with the data complexity which typically results in unrealistic data or memory requirements. Later Biryukov and Shamir [5] (BS) have shown that multiple data can be combined with Hellman's tradeoff, resulting in a flexible time/memory/data tradeoff formula.

In the context of block ciphers with reasonably long keys, the original Hellman attack is typically not considered to be of a threat since its precomputation time is the same as the exhaustive search of the key. Moreover, the attack works for a single chosen plaintext encryption and cannot benefit if more plaintext-ciphertext pairs are available to the attacker since the precomputed tables are "wired" to a fixed plaintext. This is contrary to what happens in Babbage-Golic-Biham or Biryukov-Shamir's trade-off where precomputation time is way below the exhaustive key search complexity.

At the rump session of ASIACRYPT 2004, Hong and Sarkar [9] have demonstrated that stream ciphers with short IVs can be attacked via the Biryukov-Shamir time/memory/data trade-off [5] in a frequent re-synchronization scenario. More recently in [10] they also provide a careful study of time/memory/data trade-off attack in the context of various modes of operation of block-ciphers noticing that these essentially constitute a stream cipher. However, we believe that [10] does not consider in sufficient details the important cases of ECB (a mode typically assumed in theoretical cryptanalysis) or CBC with known IV's or counter and OFB modes in chosen IV scenarios, which directly lead to very powerful attacks. They also describe attacks which have preprocessing times higher than the complexity of exhaustive search and thus seem to be less relevant.

Our Contributions: Our contribution is two-fold. We present new applications as well as a new analysis of time/memory/data trade-off attacks.

We describe three applications.

- The usual time/memory/data trade-off attack on stream ciphers can be considered to be a time/memory/key trade-off attack on block ciphers. This attack applies to situations where the goal of the attacker is to obtain one out of many possible keys. See Table 1 for a picture how various block and stream cipher tradeoffs are related.
- We carefully consider the key size of block ciphers and conclude that in the view of trade-off attacks one may no longer assume k-bit security even for a good k-bit cipher. This is true for ECB mode but unfortunately also true for CBC mode even with proper IV choice. Counter and OFB modes are vulnerable to this attack in chosen IV scenarios.
- The last application is to Unix password scheme, where the short size of the "salt" is insufficient to stop trade-off attacks. We describe practical attacks on this scheme when a password file with reasonably many password hashes is available to the attacker.

In the second part of the paper, we perform a new unified analysis of Hellman's attack in the presence of multiple data. The BG and the BS attacks are obtained as special cases of the general attack. So far, it has been believed that any multiple data TMTO is either BG or BS. Our work reveals that there are other possible

Table 1. Relation between block and stream cipher trade-off attacks

	Block ciphers (varying keys or IV's)	Stream ciphers (varying keys or IV's)
Type of	Biham's collision [2]	Babbage-Golic birthday [1, 7]
trade-off	this paper and [10]	Biryukov-Shamir TMD [5, 10]

desirable trade-offs for *single table, multiple column* attacks which are not obtainable from either the BG or the BS attack. The time required for table look-ups can be reduced by using Rivest's idea of distinguished point (DP) method. We analyse this method in a general setting. Finally, we consider the rainbow method and show that in the presence of multiple data, the TMTO curve of the rainbow method is inferior to the TMTO curve of the Hellman+DP method.

2 Time/Memory/Data Trade-Off Methodology

In this section we give a quick introduction for the methodology behind Hellman's [8] time/memory trade-off as well as for the closely related time/memory/data trade-off attacks by Biryukov-Shamir [5].

Suppose that $f : \{0,1\}^n \rightarrow \{0,1\}^n$ is a one-way function, i.e. a function which can be efficiently evaluated in forward direction, but which is hard to invert. The goal of the attacker is to invert this function, i.e. given $f(x)$ to find x, while keeping complexity of the inversion algorithm (time T, data D, memory M and preprocessing time P) as low as possible. Throughout this paper we will denote by $N = 2^n$ the size of the domain of the function.

In [8] Hellman has proposed a general algorithm for inversion of an arbitrary one-way function which obeyed the formula: $N^2 = TM^2, D = 1, P = N$. In [5] Hellman's algorithm has been generalized for the case of multiple data, which resulted in a formula $N^2 = TM^2D^2, 1 \leq D^2 < T, P = N/D$. Here we will directly describe this generalized trade-off.

The algorithm consists of two stages: a one-time offline stage followed by an online stage. In the online stage, we will be given D points $y_1, \ldots, y_D$ in the range of f and our goal is to find the pre-image of any one of these points. In the offline stage, a set of tables are prepared covering N/D of the domain points. Since N/D domain points are covered and the online stage involves D domain points, by the birthday bound, we are assured of constant probability of success in finding the pre-image of one of the y's.

Let $f_1, \ldots, f_r$ be simple output modifications of f ($f_i = g_i \circ f$, where g_i can be a simple bit permutation of the output bits), such that the f_i's can be assumed to be pairwise independent. In the offline stage r tables (one per f_i) are prepared. The ith table is prepared in the following manner. A total of m random domain points are chosen. For each domain point, the function f_i is iteratively applied t times to reach an end point. The pairs (start-point, end-point) are stored as part of the ith table sorted by the end points. The total storage requirement is rm pairs of points, while the total coverage is rmt points.

In the online stage, for each data point y_j we look for a pre-image in the set of points covered by the tables. For searching in the ith table, we first apply g_i

to y_j to obtain y'_j, then we iteratively apply f_i a total of t times to y'_j. After each application of f_i, we look in the end points of the ith table for a match. If a match is found, we go to the corresponding start point and iterate f_i until we reach y'_j. The preceding point is a possible pre-image of y, which is verified by applying f to it and checking whether we obtain y. This requires a total of t applications of f and t table look-ups per table per data item. In order to minimize the waste of table coverage due to birthday collisions the proper choice of parameters m and t would typically satisfy $N = mt^2$ (in Sec. 8 it will be shown how to obtain a new trade-off formula by using sub-optimal choices of m and t). Since a single matrix covers only mt points in order to cover the full space one will need $r = N/mt = t$ tables corresponding to different functions $f_i, i = 1, \ldots, r$. However in generalized case we need to cover only a fraction N/D of space, and thus $r = t/D$ tables would suffice. By eliminating parameters r, m, t one gets a tradeoff formula $N^2 = TM^2D^2, 1 \leq D^2 < T, P = N/D$. For more details regarding the method see [8, 5].

We discuss some general issues about TMTO. In Hellman's original scheme, $D = 1$; the table preparation time is disregarded and only the online time and memory requirements are considered. The assumption is that the tables would be prepared once for all in an offline phase. Once the tables are prepared, they will not change and can be used to find different pre-images. In this scenario, the table preparation time can be huge and even larger than exhaustive search. Thus, the security of a cryptographic algorithm with respect to this kind of TMTO has a hidden cost of offline (and one time) exhaustive search.

If multiple data is available, the actual table preparation time will be less than exhaustive search. Since this is an offline activity, it might be reasonable to expect the table preparation time to be more than the online time but less than exhaustive search time.

The precomputation time will be in general more than the memory requirement. In the table preparation stage, the entire table will have to be computed and only a fraction of it stored. This shows that the offline time will be at least as large as the memory requirement. Hellman in his original paper [8], considered the condition where the online time is equal to the memory requirement. In the presence of multiple data, it is perhaps more practical to require the data and memory requirement to be less than the online and offline time requirements. This has been considered in [5].

3 Time/Memory/Key Trade-Offs

It is easy to see that all the reasoning from the Time/Memory/Data trade-off in the case of stream ciphers [5] can be applied to the block-cipher "Time-Memory-Key" case. Namely we no longer need a full coverage of the space N, but rather can cover a fraction N/D_k, where we denote by D_k the number of possible keys at the online stage. Thus, we will use t/D_k tables instead of t, which means that memory requirements go down to $M = mt/D_k$ (here m is the number of Hellman's tables). Our time requirements are $T = t/D_k \cdot t \cdot D_k = t^2$ (less tables to check but for more data points), which is the same as in the original Hellman's

trade-off. Finally, the matrix stopping rule is again: $N = mt^2$. Using the matrix stopping rule and eliminating the parameters m and t we get a trade-off formula:

$$N^2 = T(MD_k)^2.$$

This is exactly the same formula as the one derived in [5] for the case of stream ciphers. For example, for the case of AES with 128-bit key, assuming that one is given 2^{32} encryptions of a plaintext "all zeroes" (or any other fixed text, like 16 spaces, "Hello Joe, " etc.) under different unknown keys, one can recover one of these keys after a single preprocessing of 2^{96} steps, and using 2^{56} memory for table storage and 2^{80} time for the actual key-search[1]. It is important to note that unlike in Hellman's original trade-off, the preprocessing time is much lower than the exhaustive search and thus technically this is a break of cipher. Though even better theoretical attacks for block-ciphers exist in this setting [2] they are in direct correspondence to Babbage-Golic "birthday" trade-off attacks and thus suffer from the same lack of flexibility due to $T = D$. Such attack will require impractical amount of 2^{64} fixed text encryptions as well as high storage complexity of 2^{64}. We believe that if one would try to implement these attacks he would prefer to use less data and less memory at the expense of more preprocessing and longer attack time. In Table 2, we summarize complexities of TMD attacks for various schemes. For example we believe that the attack on full Skipjack with 2^{32} fixed plaintexts and 2^{48} preprocessing complexity, 2^{32} memory and time is tempting to implement and to try in practice. Another important observation is that the attack is not exactly a chosen plaintext attack – since the specific value of the fixed plaintext is irrelevant. Thus, in order to obtain the attack faster than exhaustive search the attacker will first check which plaintext is the most frequently used in the specific application, collect the data for various keys and then perform the attack. The attack is technically faster than the exhaustive search even if the attacker obtains a relatively small number of arbitrary fixed text encryptions. For example if the attacker obtains only 2^8 128-bit key AES encryptions, then after preprocessing of 2^{120} steps and using 2^{60} memory and 2^{120} analysis steps, one of the keys would be recovered. In practical applications, it might be a rather non-trivial task to ensure that the attacker never obtains encryptions of 2^8 fixed known plaintexts. This attack is much better than the existing state of the art attacks on 128-bit AES, which barely break 7-rounds of this cipher. Note that Biham's attack for the same amount of fixed text would formally have the same 2^{120} total complexity but would require unrealistic amount of memory 2^{120} which is probably the reason why such trade-off attacks have not been viewed as a threat by the crypto community. In addition to all said above note that intentionally malicious protocol design may ensure that some fixed plaintext is always included into the encrypted stream (for example by fixing a header in communication, using communication to a fixed address or using fixed file header as is common in many applications).

[1] At the moment of this writing 2^{85} computations is approximately the power of all computers on the internet during 1 year.

Table 2. Comparison of TMD attacks on various ciphers

Cipher	Key size	Keys (Data)	Time	Memory	Preprocessing
DES	56	2^{14}	2^{28}	2^{28}	2^{42}
Triple-DES	168	2^{42}	2^{84}	2^{84}	2^{126}
Skipjack	80	2^{32}	2^{32}	2^{32}	2^{48}
AES	128	2^{32}	2^{80}	2^{56}	2^{96}
AES	192	2^{48}	2^{96}	2^{96}	2^{144}
AES	256	2^{85}	2^{170}	2^{85}	2^{170}
Any cipher	k	$2^{k/4}$	$2^{k/2}$	$2^{k/2}$	$2^{3k/4}$
Any cipher	k	$2^{k/3}$	$2^{2k/3}$	$2^{k/3}$	$2^{2k/3}$
Any cipher[2]	k	$2^{k/2}$	$2^{k/2}$	$2^{k/2}$	$2^{k/2}$

Table 3. Trade-off attacks on Skipjack (and any other 80-bit cipher)

Attack	Data Type	Keys (Data)	Time	Memory	Preprocessing
BS TMD	FKP	2^{8}	2^{60}	2^{42}	2^{72}
BS TMD	FKP	2^{20}	2^{40}	2^{40}	2^{60}
BS TMD	FKP	2^{32}	2^{32}	2^{32}	2^{48}
Biham[2]	FKP	2^{40}	2^{40}	2^{40}	2^{40}
BBS Imp.Diff*[4]	CP	2^{34}	2^{78}	2^{64}	2^{64}

* — the attack breaks 31 out of 32 rounds of Skipjack, the data is encrypted under a single key. FKP – fixed known plaintext, CP – chosen plaintext.

Table 4. Trade-off attacks on 128-bit key AES (and any other 128-bit key cipher)

Attack	Data Type	Keys (Data)	Time	Memory	Preprocessing
BS TMD	FKP	2^{8}	2^{120}	2^{60}	2^{120}
BS TMD	FKP	2^{20}	2^{100}	2^{58}	2^{108}
BS TMD	FKP	2^{32}	2^{80}	2^{56}	2^{96}
BS TMD	FKP	2^{43}	2^{84}	2^{43}	2^{85}
Biham[2]	FKP	2^{64}	2^{64}	2^{64}	2^{64}
GM collision*	CP	2^{32}	2^{128}	2^{80}	?
FSW partial sum*	CP	2^{128}–2^{119}	2^{120}	2^{64}	?

* — only 7 out of 10 rounds. FKP – fixed known plaintext, CP – chosen plaintext.

Table 5. Trade-off attacks on 192-bit key AES (and any other 192-bit key cipher)

Attack	Data Type	Keys (Data)	Time	Memory	Preprocessing
BS TMD	FKP	2^{48}	2^{96}	2^{96}	2^{144}
BS TMD	FKP	2^{64}	2^{128}	2^{64}	2^{128}
Biham[2]	FKP	2^{96}	2^{96}	2^{96}	2^{96}

FKP – fixed known plaintext.

Results shown in Table 2 compare favorably to the best attacks on such ciphers as DES, Triple-DES, Skipjack and AES. Moreover, the scenario of TMD attacks is much more practical than that of related key attacks as is discussed in more

Table 6. Tradeoff attacks on 256-bit key AES (and any other 256-bit key cipher)

Attack	Data	Type	Keys (Data)	Time	Memory	Preprocessing
BS TMD	FKP		2^{64}	2^{128}	2^{128}	2^{192}
BS TMD	FKP		2^{85}	2^{170}	2^{85}	2^{170}
Biham[2]	FKP		2^{128}	2^{128}	2^{128}	2^{128}

FKP – fixed known plaintext.

detail in Section 4. We believe that complexities of future cryptanalytic attacks should be benchmarked against the time-memory-key attacks.

Due to the importance of some trade-off points we provide Tables 3–6 for several important ciphers (key lengths) and compare them with best attacks known so far.

4 Types of Cryptanalytic Attacks and Key-Size Considerations

Cryptanalytic attacks may be divided into three main classes by the type of access to an encryption/device. In the first class of attacks, which we call *fixed key* attacks, we assume that a black box with the encryption/decryption device is given to the attacker. The attacker is then able to make arbitrary number of queries (with unknown, known or chosen inputs) to the device. The goal of the attacker is to find the secret key of the box, which remains unchanged during the attack. Note that the queries could be performed adaptively (i.e. based on the results of previous queries). For example: differential, linear, boomerang or multiset attacks are of this type. Moreover linear cryptanalysis *requires* a fixed key scenario, while differential, boomerang or multiset attacks may tolerate key changes which are not too frequent during the attack.

The second type of attack which we call *variable key* attacks, assumes that the attacker is given both the black box with the encryption/decryption device as well as a black box of the key-schedule device. The attacker can then perform both the fixed key attacks as well as re-keying the cipher to a new secret key value at any given moment. The goal of the attacker is to find one of the keys of the device. This scenario is strictly more powerful than the fixed key scenario and can be efficiently exploited for example in the "weak key" attacks or in time/memory/key trade-off attacks.

The third type of attacks is what is called *related key* scenario. In this case the attacker is not only allowed to change the keys of the device. He is essentially given access to two or more encryption/decryption devices and he knows or even chooses the relations between the keys used in these devices. This scenario is highly unrealistic in practice but may identify undesirable certificational weaknesses in the key-schedule of a cipher.

Applicability of the attack scenarios described above in practice may be hindered by the use of certain *mode of operation* (which for example may preclude the use of chosen plaintext queries) or by the *key-change provision*, which may

enforce a key-change every 1000 encryptions thus rendering statistical attacks which assume fixed key scenario — impractical.

4.1 Key-Size Consideration

Modern symmetric ciphers typically have keys larger or equal to 128 bits and they assume that exhaustive search is the best way one could recover a secret key[2].

As this note shows however in a *variable key* scenario no k-bit cipher can offer a k-bit security against some quite practical attacks. One may assume that this problem can be cured by introducing the IV which has to be of the same size as the key. However in such popular block-cipher modes of operation like CBC due to a simple XOR of the *known* IV with the first plaintext block the attacker capable of mounting chosen plaintext attack can easily obtain encryptions of arbitrary fixed text under different keys. In the less likely but still not impossible case of a chosen IV attack other modes of operation like CFB, OFB or counter mode become vulnerable as well. A careful study of what should be the IV size in order to avoid trade-off attacks is given in [10], however a simple rule of a thumb is that the IV size should be at least equal to the key-size, since the state of the cipher at any given moment has to be twice the key-size in order to avoid birthday time-data attacks [2, 1, 7]. XORing of the IV into the plaintext/ciphertext should be avoided.

Following these simple observations it is clear that 80-bit (or less) key ciphers should not be used since they allow for practical attacks in real-life scenarios, while 128-bit ciphers (which in practice provide security of about 80-bits) should not be used when full 128-bit security is required. At least 192-bit keys should be used for this security level.

One may argue that generic trade-off attacks do not exploit weaknesses of specific designs and thus should be considered separately from other attacks. There are two counter-arguments to this point: first of all we have at the moment no proof that existing trade-off attacks (such as Hellman's attack) are the best possible and thus a popular maxim "The attacks only become better, they do not get worse" may still apply. Moreover trade-off attacks may be sped up by specific properties of the design, for example by what is called in a stream cipher case — cipher's sampling resistance [5]. In the case of stream cipher LILI-128 low sampling resistance was used to obtain trade-off attack [13] with a complexity much lower than a naive application of a trade-off technique would suggest.

It seems that we will have to give up the convenient world in which we assumed a k-bit security for a good k-bit cipher.

5 Application to the Unix Password Scheme

The attacks described in this paper are not limited to block or stream ciphers, they are applicable to other one-way constructions, for example to hash functions.

[2] Depending on the mode of operation used, there are also distinguishing attacks which may require about 2^{64} fixed key data, and do not lead to key-recovery. Those attacks are not considered to be of a threat by the community and are typically taken care of by key-change provisions.

Table 7. Trade-off attacks on UNIX password scheme

Passwords attacked	State Size (bits)	Data	Time	Memory	Preprocessing
Alphanumeric	60	2^8	2^{34}	2^{34} (128 Gb)	2^{52}
Alphanumeric	60	2^{10}	2^{32}	2^{34}	2^{50}
Full keyboard	63	2^{10}	2^{36}	2^{35} (256 Gb)	2^{53}
Alphanumeric[a][11]	60	1	2^{40}	2^{40}	2^{60}

[a] The paper provides analysis for a single fixed salt value.

Time/memory/data trade-off [5] ($N = TM^2D^2$) could be used to analyze Unix password scheme for example, if the attacker obtains access to a file storing password hashes of a large organization ($D = 1000$ password hashes). Indeed the trade-off space consists of 56-bits of the unknown key (i.e. password) and 12-bits of known salt. Since the salt size is much shorter than the key-size its effect on making the trade-off harder is not very significant. Suppose that the attacker knows that passwords are selected from a set of arbitrary 8-character alphanumeric passwords, including capital letters and two additional symbols like dot and comma which in total can be encoded in 48-bits. Thus together with a 12-bit salt the state is $N = 2^{60}$ bits. For example the following attack parameters seem quite practical: preprocessing time done once: $P = N/D = 2^{50}$ Unix hash computations, parallelizable. A memory of $M = 2^{34}$ 8-byte entries (12+48 bits) which takes one 128 Gbyte hard disk. This way we store 2^{34} start-end pointers. Attack time is then $T = 2^{32}$ Unix hash evaluations — about an hour on a fast PC or about 8 seconds on a BEE2 FPGA [11]. The attack will recover one password from about every 1000 new password hashes supplied. This is two – three orders of magnitude faster than the results described in [11]. The relatively lengthy preprocessing step may be performed in parallel on a network of PC's (hundred PC's may take less than a month) or it may take about 1.5 months for a single BEE2 FPGA. The number of tables computed in parallel may be as high as $t/D = 2^{17}/1000 = 2^7$. In order to reduce the number of hard disk accesses the attack will need to use distinguished points with 16-bit prefixes. This will allow to make only 2^{16} disk accesses (which is less than 6 minutes).

In fact it is clear that such trade-off can analyze all passwords typable on a keyboard. The space is $N = 84^8 \cdot 2^{12} = 2^{63}$. Assuming again $D = 2^{10}$, we get precomputation time $P = 2^{53}$, $M = 2^{35}$ 8-byte entries or one 256 Gb hard disk, $T = 2^{36}$ hash evaluations.

6 A New Analysis

In this section, our goal is different from that of the previous sections. Here we present a new analysis of the general TMTO methodology described in Section 2. We show that the BG and the BS methods are special cases of the general analysis. In particular, we show that there are certain interesting trade-offs which were not known earlier. Some of these are summarized in Table 8.

Table 8. Some of the new trade-offs

N	Precomputing time (P)	Memory (M)	Data (D)	Run time (T)
2^{80}	2^{50}	2^{30}	2^{30}	2^{50}
	$2^{46.6}$	$2^{33.3}$	$2^{33.3}$	$2^{46.6}$
2^{100}	$2^{62.5}$	$2^{37.5}$	$2^{37.5}$	$2^{62.5}$
	$2^{58.3}$	$2^{41.6}$	$2^{41.6}$	$2^{58.3}$
* N	$N^{\frac{1+d}{3}}$	$N^{\frac{2-d}{3}}$	$N^{\frac{2-d}{3}}$	$N^{\frac{1+d}{3}}$

* Here d is a constant such that $\frac{1}{2} < d < 1$.

Recall from Section 2 that the general description of the TMTO methodology uses r tables each of size $m \times t$. In the BG attack, $r = t = 1$ and hence is not very flexible. The BS attack is more flexible but assumes $mt^2 = N$ and $r = t/D$ leading to the constraint $D^2 \leq T$. The last restriction can prove to be a drawback of the BS method when the amount of available data is high. From the Hellman attack we have the following relations.

$$\left.\begin{array}{lll} T_f & = r(t-1)D & (\#\ f \text{ invocations in the online phase}) \\ T_t & = rtD & (\#\ \text{table look-ups in the online phase}) \\ P & = rmt & (\#\ f \text{ invocations in the pre-computation phase}) \\ & = \frac{N}{D} & (\text{coverage}) \\ M & = rm & (\text{memory}) \\ mt^2 & \leq N & (\text{birthday bound}) \end{array}\right\} \quad (1)$$

If $t \gg 1$, we can assume $t - 1 \approx t$ and $T_f \approx rtD = T_t$. We will usually make this assumption, except for the analysis of the BG attack, where $t = 1$. Let γ be the ratio of the time required for performing one table look-up to the time required for one invocation of f, i.e.,

$$\gamma = \frac{\text{time for one table look-up}}{\text{time for one invocation of } f}. \quad (2)$$

We assume that one unit of time corresponds to one invocation of f (i.e., one unit of time = time required for completing one invocation of f) and also $\gamma \geq 1$. The time required for the table look-ups is then γrtD. Define $T = \max(T_f, \gamma T_t) = \gamma rtD$. The parameter T is a measure of the time required during the online phase. The actual online time is proportional to $T_f + \gamma T_t$. However, this is only at most twice the value of T. Thus, we will perform the analysis with T instead of $T_f + \gamma T_t$.

For the present, we will assume that $\gamma = 1$ (and $T = T_t \approx T_f$), i.e., the cost of one invocation of f is equal to the cost of one table look-up. The value of γ need not actually be one; even if it is a small constant (or a negligible fraction of N), we can assume it to be one and that will not affect the asymptotic analysis. On the other hand, [5] mentions that γ may be as large as one million ($\approx 2^{20}$). If N is only moderately large (like 2^{64} for A5/1), then γ can be a significant proportion of N. In such a situation, we cannot assume $\gamma = 1$ and the cost of table look-up will dominate the total online cost. This case will be considered later.

Using (1), we can solve for r, m and t as follows.

$$\left.\begin{array}{lll} t & = \frac{N}{MD} \geq 1 & \text{(number of columns)} \\ m & = \frac{N}{T} & \text{(number of rows)} \\ r & = \frac{MT}{N} \geq 1 & \text{(number of tables)} \\ mt^2 & = \frac{N^3}{TM^2D^2} \leq N & \text{(birthday bound)} \end{array}\right\} \quad (3)$$

Note that all three of r, m and t must be at least 1. Since $m = N/T$ and for a valid attack we must have $N > T$, the condition on m is trivially satisfied. The advantage of writing in the form of (3) is that given values for T, M and D satisfying the proper constraints, we can immediately design a table structure which achieves these values.

6.1 TMTO Curve

From Equations (3), we know $MT \geq N$ and $mt^2 \leq N$. Further, for a feasible attack we must have $1 \leq D < N$ and $M, T < N$. We capture these in the following manner:

$$D = N^a; \quad MT = N^b; \quad M = N^c; \quad mt^2 = N^d; \quad (4)$$

with $0 \leq a, c < 1$, $0 \leq d \leq 1$ and $b \geq 1$. Consequently, we have $P = N/D = N^{1-a}$; $T = N^b/M = N^{b-c}$, with $0 \leq b - c < 1$. Further, $t \geq 1$ implies $MD \leq N$ and hence $a + c \leq 1$. Substituting in the last equation of (3), we obtain $2a + b + c + d = 3$. Thus, any set of values for a, b, c and d which satisfy the following constraints constitute a valid attack.

$$\left.\begin{array}{l} \textbf{C1: } 2a + b + c + d = 3 \\ \textbf{C2: } 0 \leq a < 1 \\ \textbf{C3: } 0 \leq c, b - c < 1 \leq b \\ \textbf{C4: } a + c \leq 1 \\ \textbf{C5: } 0 \leq d \leq 1 \end{array}\right\} \quad (5)$$

The so-called TMTO curve can be obtained as the following relations.

$$\left.\begin{array}{lll} TM^2D^2 & = N^{3-d} \\ PD & = N \\ MD \leq & N \leq MT \\ M, D, T & < N. \end{array}\right\} \quad (6)$$

Also, we have the following values of r, m and t.

$$r = N^{b-1}; \quad m = N^{1-(b-c)}; \quad t = N^{1-a-c}. \quad (7)$$

Since $MT = N^b \geq N$, we have $r = 1$ if and only if $MT = N$. With $r = 1$, we have only one table and hence if there are more than one tables, then MT is strictly greater than N.

BG Attack [1, 7]: In this case, we have $r = t = 1$. This implies $T_f = 0$, i.e., the online phase does not require invocation of f. The cost in the online phase is $T = T_t$ and we have $MD = N = MT$ and hence $T = D$; $M = N/D$. This corresponds to the conditions $a + c = 1; b = 1; d = 1 - a$.

BS Attack [5]: In [5], $r = t/D$ and $d = 1$ is used. Then $T = t^2$, $M = mt/D$ and hence $r = N^{-a+(b-c)/2}$. Since $r \geq 1$, we have the restriction $0 \leq 2a \leq b - c$ (i.e., $1 \leq D^2 \leq T$) in addition to (5).

The conditions $d = 1$ and $r = t/D$ are related (e.g., if $r = 1$, then $t = D$ and $T = t^2 = D^2$). In the following analysis, we will proceed without these two conditions. Later, we show the situation under which making these two assumptions is useful.

7 Distinguished Point Method

We now consider the case where $\gamma \gg 1$. In this case, a direct application of the Hellman method leads to $T = \gamma rtD$, i.e., the time required for the table look-ups dominate the online time. It is useful to consider the distinguished point method of Rivest to reduce the number of table look-ups. See [5] for a description of the DP method.

Using the distinguished point method results in reducing the number of table look-ups from rtD to rD, i.e., one table look-up per table per data. Then $T_t = rD = N^{a+b-1}$. (Note $T_t = N^a = D$, i.e., only one table look-up is required per data item if and only if $b = 1 = r$, i.e., $MT = N$.)

The total cost of the table look-ups is γrD whereas the cost of invoking the one-way function is rtD. In this case, the ratio of the two costs is γ/t. If $t \geq \gamma$, then the ratio is at most one. Hence, we can again ignore the cost of table look-up and perform the analysis by considering simply the cost of invoking the one-way function. The actual runtime will be at most twice the runtime obtained by such an analysis.

Suppose $t < \gamma$. Then the analysis performed above does not hold. We now investigate the situation under which $t < \gamma$ holds. This certainly holds for $t = 1$ (the BG attack), but in the BG attack the entire online computation consists of table look-ups and hence the general analysis is not required. Recall that $t = N^{1-(a+c)} = 2^{n(1-(a+c))}$, $D = N^a$ and $M = N^c$. Suppose $\gamma = 2^e$. Then $t \geq \gamma$ if and only if $a + c \leq 1 - (e/n)$. The value of e is a constant whereas n increases. Hence, $(1 - e/n) \to 1$ as n grows. Thus, we can have $a + c > 1 - e/n$ only for small values of n. The smallest value of n for which we can expect to have a secure cryptographic algorithm is 64. Further, as mentioned in [5], e can be at most around 20 and so $1 - e/n \geq 2/3$ for $n \geq 64$.

Consider $a = c = 1/3$, as in the solution $(a, b, c, d) = (1/3, 1, 1/3, 1)$ corresponding to $P = T = N^{2/3}$; $M = D = N^{1/3}$; $r = 1$ of [5]. If $n \geq 64$, then $a + c = 2/3 \leq 1 - e/n$ and the time analysis assuming $T = rtD = tD$ holds. On the other hand, for the solution $(a, b, c, d) = (3/8, 1, 3/8, 7/8)$ corresponding to $P = T = N^{5/8}$; $M = D = N^{3/8}$; $r = 1$ considered in Section 8, we have $a + c = 3/4$. For $n = 64$, $a + c > 1 - e/n$ and we have to assume $T = \gamma rD = \gamma D$,

whereas for $n = 100$, $a + c \leq 1 - e/n$ and we can assume $T = rtD = tD$. Thus, for relatively small n, we should solve (5) with the constraint $a + c \leq 1 - e/n$ instead of $a + c \leq 1$. This disallows some of the otherwise possible trade-offs.

There is another issue that needs to be considered. We have to ensure that t is large enough to ensure the occurrence of a DP in a chain. Let 2^{-p} be the probability of a point being a DP. Hence, we can expect one DP in a random collection of 2^p points. Thus, if $t \geq 2^p$, we can expect a DP in a chain of length t. This implies $p \leq \log_2 t$. Any attempt to design the tables with $t < 2^p$, will mean that several trials will be required to obtain a chain terminating in a DP. This will increase the pre-computation time. In fact, [5] has shown that use of the DP method in the BG attack divides into two different trade-offs leading to unrealistic requirements on data and memory.

Using (7), we have $p/n \leq 1-(a+c)$. This leads to the condition $a+c \leq 1-p/n$ ($MD \leq N^{1-p}$) instead of the condition $a + c \leq 1$ (resp. $MD \leq N$) in (5) (resp. (6)). For small n, this condition has to be combined with $a + c \leq 1 - e/n$ and we should solve (5) with the constraint $a + c \leq 1 - 1/n \times \max(p, e)$ instead of the constraint $a + c \leq 1$. This puts further restrictions on otherwise allowed trade-offs.

BSW Sampling. There is an elegant application of TMTO in [6], which uses a special type of sampling technique called the BSW sampling. This technique uses only part of the available online data and also reduces the search space. The trade-off curve does not change, but the number of table look-ups reduces significantly. Use of this technique allowed particularly efficient attacks on A5/1.

Use of the BSW technique reduces the amount of available online data. This makes it difficult to use a single table to carry out the TMTO. In such a situation, our analysis does not lead to any new insight into the BSW technique. On the other hand, if the available online data (even after sampling) is large enough to allow the use of a single table, then our analysis applies and one can consider a wider variety of trade-offs.

8 Single Table Trade-Offs

The case $N = 2^{100}$ has been considered in [5]. It has been mentioned in [5] that the Hellman attack with $D = 1$; $T = M = N^{2/3} = 2^{66}$ requires unrealistic amount of disk space and the BG attack with $T = D = N^{2/3} = 2^{66}$; $M = N^{1/3} = 2^{33}$ requires unrealistic amount of data. (Note $T = M = D = N^{1/2} = 2^{50}$ also gives a BG attack. However, as mentioned in [5] in a different context, data and memory requirement of more than 2^{40} is unrealistic.) Further, [5] mentions $P = T = 2^{66}$ and $D = M = 2^{33}$ to be a (barely) feasible attack. This corresponds to the parameters $(a, b, c, d) = (1/3, 1, 1/3, 1)$ and $(r, m, t) = (1, N^{1/3}, N^{1/3})$.

From Proposition 2, if we choose $d = 7/8$, then we obtain $M = D = N^{3/8} = 2^{37.5}$ and $P = T = N^{5/8} = 2^{62.5}$. The corresponding parameters are $(a, b, c, d) = (3/8, 1, 3/8, 7/8)$ and $(r, m, t) = (1, N^{3/8}, N^{1/4})$. This brings down the attack time while keeping the data and memory within feasible limits. Since $t > 1$, this

cannot be obtained from the BG attack. Further, choosing $d = 7/8$ and $D^2 > T$ ensures that this attack cannot also be obtained from the BS attack. We would like to point out that [5] mentions that choosing $d < 1$ is "wasteful". The above example shows that this is not necessarily the case and choosing $d < 1$ can lead to more flexible trade-offs. We show below the condition under which choosing $d < 1$ is indeed "wasteful".

As mentioned earlier, we have one table (i.e., $r = 1$) if and only if $MT = N$. The reason for moving to more than one tables is when $mt^2 > N$ and we begin to have more and more repetitions within a table.

Proposition 1. *There is a solution to (6) with $r = 1 = b$ (and hence $MT = N = PD$) if and only if $2a + c \geq 1$.*

Proof : Suppose $r = 1$. Then $b = 1$ and $2a + c + d = 2$. Hence $d = 2 - (2a + c)$. Since $d \leq 1$, this shows $2a + c \geq 1$.

On the other hand assume that $2a + c \geq 1$. Choose $b = 1$ and set $d = 2 - (2a + c) \leq 1$. This choice satisfies the conditions of (6). Further, since $b = 1$, we have $r = 1$. □

Suppose $2a+c < 1$. Then $b+d > 2$ and $b > 2-d$. Since $MT = N^b$, we would like to minimize b and hence we choose $d = 1$. We can now modify the suggestion of [5] and say that it is "wasteful" to choose $mt^2 < N$ *if there are more than one tables.* Since $b > 1$, we have $2a + c < 1 < b$ and hence $2a < b - c$ which gives $D^2 < T$ and we are back to the situation described in [5].

Thus, the analysis of [5] actually applies to the situation where the data is small enough to require more than one tables. On the other hand, for the case of one table, the restrictions of [5] are not required and removing these restrictions provide more flexible trade-offs. We would like to point out that there are interesting situations where a single table can be used. Apart from the examples $D = M = N^{1/3}$ and $D = M = N^{3/8}$ already considered, other possible examples are ($D = N^{0.3}$, $M = N^{0.4}$); ($D = N^{0.25}$, $M = N^{0.5}$), etcetera.

8.1 Small N

Consider $N = 2^{64}$, as in A5/1. It is mentioned in [6] that $D \approx 2^{22}$ is a reasonable choice. Further, $M \approx 2^{40}$ is also feasible. We consider possible values of P and T satisfying these values of M and D.

Trade-Off 1: $(P, D, M, T) = (2^{48}, 2^{16}, 2^{40}, 2^{24})$*:* The table parameters are $(r, m, t) = (1, 2^{40}, 2^8)$ and $d = 7/8$.
Trade-Off 2: $(P, D, M, T) = (2^{42}, 2^{22}, 2^{40}, 2^{24})$*:* The table parameters are $(r, m, t) = (1, 2^{40}, 2^4)$ and $d = 11/16$.

None of the above two trade-offs are obtainable as BG attacks, since in both cases $t > 1$. Further, neither can any of them be obtained as BS trade-offs since in both cases $d < 1$ and hence $D^2 > T$. For both trade-offs, the data and memory are within reasonable limits and the online times are the same. The offline time is lower for the second trade-off and is within doable limits (especially as an offline one-time activity), while for the first attack it is probably just outside the

doable limit. Hence, both the attacks are feasible for any 64-bit function and hence also for A5/1. However, as mentioned before, using special properties of A5/1, it is possible to obtain faster attacks as in [6].

9 Certain Special Cases

Here we consider in detail two special cases. These should be considered to be mainly of theoretical interest. The first condition of $T = M$ was originally considered by Hellman in the situation where $D = 1$, while the second condition was briefly considered in [5] for the case $N = 2^{100}$.

9.1 Condition $T = M$

The condition $T = M$ was considered by Hellman [8]. We perform an analysis of (6) with $T = M$. Then $c = b - c$, whence $c = b/2$. Condition **C1** becomes $2a + 3c + d = 3$ and so $\frac{b}{2} = c = 1 - \frac{2a+d}{3}$. Using $a + c \leq 1$, we obtain $a \leq d$. Also since $b \geq 1$, we have $c = b/2 \geq 1/2$. This gives $d \leq 3/2 - 2a$. Since, we already know $d \leq 1$, we obtain $a \leq d \leq \min(1, \frac{3}{2} - 2a)$. Thus, any non-negative solution in a and d to this gives a valid attack with $T = M = N^c$.

We are interested in minimizing the value of c. We see that the value of c is minimized by maximizing the value of d. In fact, we can choose $d = 1$ as long as $1 \leq \frac{3}{2} - 2a$, i.e., $2 - (1/2a) \leq 0$ or $a \leq 1/4$. Thus, for $a \leq 1/4$, we obtain $T = M = N^{b/2} = N^{(2-2a)/3}$.

In the case $3/2 - 2a \leq 1$, we have $a \leq d \leq 3/2 - 2a$. For the gap to be non-empty we must have $a \leq 1/2$. For minimizing c, we use the upper bound, i.e., $d = 3/2 - 2a \leq 1$. Thus, for $1/4 \leq a \leq 1/2$, we have $c = 1/2$ and $T = M = N^{1/2}$. Finally, we obtain the following result.

Theorem 1. *If $T = M$, then $D \leq N^{1/2}$ and the following conditions hold.*

1. $N^{1/2} \leq T = M = N^{(2-2a)/3} \leq N^{2/3}$, *for* $1/4 \geq a \geq 0$.
2. $T = M = N^{1/2}$, *for* $1/4 \leq a \leq 1/2$.

For the first case we have, $(a, b, c, d) = (a, 2(2 - 2a)/3, (2 - 2a)/3, 1)$ and for the second case we have $(a, b, c, d) = (a, 1, 1/2, 3/2 - 2a)$. The corresponding values of (r, m, t) are $(N^{(1-4a)/3}, N^{(1+2a)/3}, N^{(1-a)/3})$ and $(1, N^{1/2}, N^{1/2-a})$ respectively.

In the second case of Theorem 1, exactly one table is required. However, it is not the BG attack, since the number of columns can be more than one. Also, we have $T \leq P \leq N$. The situation with $T < P < N$ is interesting, since the pre-computation time is less than exhaustive search. Even though P is more than T, since it is an offline activity, we might wish to spend more time in the pre-computation part than in the online attack.

In the second case of Theorem 1, we have $r = 1$ and $M = T = N^{1/2}$. The allowed range of a for this case is $1/4 \leq a \leq 1/2$. The case $a = 1/4$ can be obtained from the BS analysis and the case $a = 1/2$ can be obtained from the BG analysis. However, the range $1/4 < a < 1/2$ for which $T = M = N^{1/2}$ can be attained, cannot be obtained from either the BG or the BS analysis and provide

previously unknown trade-offs. The advantage is that the data can be increased (thus lowering offline time) without increasing either time or memory.

9.2 Condition $P = T$

Since both P and T represent time, the case $P = T$ puts equal emphasis on both the offline and the online times. The condition $P = T$ implies $P = N^{1-a} = T = N^{b-c}$ and so $m = N^{1-(b-c)} = N^a = D$. (On the other hand, $P = M$ is possible only if $t = 1$.) Since $PD = N$, we have $T = N/D$ and so the curve becomes $M^2D = N^{2-d}$. If $P = T$, then $r = M/D$. If further $M = D$, then $M = D = N^{(2-d)/3}$ and $P = T = N^{(1+d)/3}$.

Proposition 2. *If $P = T$ and $M = D$ in (6), then $M = D = N^{(2-d)/3}$ and $P = T = N^{(1+d)/3}$. Further, $r = 1$, i.e., exactly one table is required.*

Proposition 2 gives us a nice way to control the trade-off between time and data/memory requirement by varying d. Choosing $d = 1$, corresponds to choosing $(P, D, M, T) = (N^{2/3}, N^{1/3}, N^{1/3}, N^{2/3})$ and has been observed in [5]; choosing $d = 1/2$ corresponds to $(P, D, M, T) = (N^{1/2}, N^{1/2}, N^{1/2}, N^{1/2})$ which is the square root birthday (BG) attack.

From Proposition 2, we have $r = 1$, i.e., all trade-offs attaining this condition use a single table. In the plausible situation, $M = D \leq P = T$, we have $1/2 \leq d \leq 1$. The case $d = 1$ can be obtained from the BS analysis. In the BG analysis, we have $d = 1 - a$. Since $a - (2 - d)/3$, this condition leads to $d = 1/2$. Thus, the range $1/2 < d < 1$ for which the condition $P = T = N^{(1+d)/3}$; $M = D = N^{(2-d)/3}$ can be attained was not known earlier.

10 The Rainbow Attack

The rainbow attack was introduced in [12]. The number of table look-ups of the rainbow method is comparable to that of the Hellman+DP method. See [12] for a discussion of the relative advantages of the rainbow method with respect to the DP method.

In the rainbow attack, we use a table of size $m \times t$ and suppose there are D online data points. Then the total number of invocations of the one-way function is $t^2D/2$ while the cost of the table look-ups is tD. Again, we will ignore the factor of two in the runtime since it does not significantly affect the analysis. Then, the total number of invocations of f is t^2D and the total number of table look-ups is tD. Also, we have $mt = N/D$.

If we assume $\gamma \approx 1$, then the cost of invoking f dominates the online cost and we have $M = m$ and $T = t^2D$. Assume $D = N^a$ and $M = N^c$ as in the case of Hellman analysis. Then since $mt = N/D = N^{1-a}$, we have $t = N^{1-a-c}$ and $T = t^2D = N^{2-a-2c}$. Also, since $t \geq 1$, we must have $a + c \leq 1$. The TMTO curve for rainbow in the presence of multiple data is $TM^2D = N^2$ which is inferior to the Hellman TMTO curve when $D > 1$.

The rainbow parameters are $(P, D, M, T) = (N^{1-a}, N^a, N^c, N^{2-a-2c})$. We now compare the rainbow parameters with the Hellman parameters for same data

and memory. For multiple table Hellman, we choose $d = 1$ and hence the corresponding Hellman parameters are $(P, D, M, T) = (N^{1-a}, N^a, N^c, N^{2-2a-2c})$. If $a > 0$, i.e., if multiple data is available, then clearly Hellman time is less than rainbow time.

If γ is a significant fraction of N, then the cost of table look-ups is γtD while the cost of invoking f is still t^2D. In the case $\gamma > t$, which happens for relatively small N (around 2^{64} or so), the cost of table look-up dominates the online cost. To compare to the Hellman method we have to consider the Hellman+DP algorithm. For the case $\gamma > t$, the online cost of the Hellman method is also γtD. Hence, for this case, the costs of online time for the rainbow and the Hellman+DP methods are equal. In this situation, one might prefer to use the rainbow method for the possibly lower rate of false alarms compared to the DP method [12].

Thus, we conclude that in the presence of multiple data, in general the Hellman attack is better than the rainbow attack. For the case of small N, the online times of both attacks can be comparable and one might prefer rainbow for obtaining other possible advantages.

11 Conclusion

In this paper, we have considered two aspects of time/memory/data trade-offs:3 new application and new analysis.

We show several applications to block-ciphers. By applying Biham-Biryukov [5] multiple data trade-off to block ciphers we show that 80-bit ciphers allow practical attacks in real world scenarios (2^{32} data, memory and time, with 2^{48} steps for preprocessing), while 128-bit ciphers provide only about 80-bits of security against attacks with practical amounts of data and memory. We further show realistic attacks on Unix password hashing scheme even if strong random passwords are chosen.

In the second part of the paper we provide general analysis which shows that Hellman's attack, Babbage-Golic attack and the Biryukov-Shamir all fit into a single unifying general framework. Our new contribution is the identification of a new class of single table trade-offs which are not obtainable as either the BG or the BS attacks. Finally, we consider the rainbow attack of Oechslin and show that with the utilization of multiple data, the TMTO curve of the rainbow attack is inferior to the TMTO curve of the Hellman attack.

References

[1] S. Babbage, "Improved "exhaustive search" attacks on stream ciphers," in *ECOS 95 (European Convention on Security and Detection)*, no. 408 in IEE Conference Publication, May 1995.

[2] E. Biham, "How to decrypt or even substitute DES-encrypted messages in 2^{28} steps," *Information Processing Letters*, vol. 84, pp. 117–124, 2002.

[3] Eli Biham, "How to Forge DES-Encrypted Messages in 2^{28} Steps", Technical report CS 884, August 1996.

[4] E. Biham, A. Biryukov, and A. Shamir, "Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials," in *Proceedings of Eurocrypt'99* (J. Stern, ed.), no. 1592 in Lecture Notes in Computer Science, pp. 12–23, Springer-Verlag, 1999. To appear in the Journal of Cryptology.

[5] A. Biryukov and A. Shamir, "Cryptanalytic time/memory/data trade-offs for stream ciphers," in *Proceedings of Asiacrypt'00* (T. Okamoto, ed.), no. 1976 in Lecture Notes in Computer Science, pp. 1–13, Springer-Verlag, 2000.

[6] A. Biryukov, A. Shamir and D. Wagner. Real Time Cryptanalysis of A5/1 on a PC, Proceedings of Fast Software Encryption 2000.

[7] J. D. Golic, "Cryptanalysis of alleged A5 stream cipher," in *Advances in Cryptology – EUROCRYPT'97* (W. Fumy, ed.), vol. 1233 of *Lecture Notes in Computer Science*, pp. 239–255, Springer-Verlag, 1997.

[8] M. E. Hellman, "A cryptanalytic time-memory tradeoff," *IEEE Transactions on Information Theory*, vol. 26, pp. 401–406, 1980.

[9] J. Hong and P. Sarkar, "Time memory tradeoff attacks on streamciphers," 2004. Rump session talk at ASIACRYPT'04.

[10] J. Hong and P. Sarkar, "Rediscovery of time memory tradeoffs," 2005. `http://eprint.iacr.org/2005/090`.

[11] N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede, "Cracking Unix passwords using FPGA platforms," 2005. Presented at SHARCS'05, in submission.

[12] P. Oechslin, "Making a faster cryptanalytic time-memory trade-off," in *Advances in Cryptology – CRYPTO 2003* (D. Boneh, ed.), vol. 2729 of *Lecture Notes in Computer Science*, pp. 617–630, Springer-Verlag, 2003.

[13] M.-J. O. Saarinen, "A time-memory trade-off attack against LILI-128," in *Proceedings of Fast Software Encryption – FSE'02* (J. Daemen and V. Rijmen, eds.), no. 2365 in Lecture Notes in Computer Science, pp. 231–236, Springer-Verlag, 2002.

A Space Efficient Backdoor in RSA and Its Applications

Adam Young[1] and Moti Yung[2]

[1] LECG LLC*
ayoung@mitre.org
[2] RSA Labs and Columbia University
moti@cs.columbia.edu

Abstract. In this paper we present an RSA backdoor that, for example, can be used for a hardware-based RSA key recovery system. The system is robust in the sense that a successful reverse-engineer is not able to obtain previous nor future RSA private keys that have been/will be generated within the key generation device. The construction employs the notion of two elliptic curves in which one is the "twist" of the other. We present a proof in the random oracle model that the generated RSA key pairs that are produced by the cryptographic black-box are computationally indistinguishable (under ECDDH) from "normal" RSA key pairs, thus ensuring the integrity of the outputs. Furthermore, the security level of the key recovery mechanism is nearly identical to that of the key pair being generated. Thus, the solution provides an "equitable" level of security for the end user. This solution also gives a number of new kleptographic applications.

Keywords: Key recovery, Diffie-Hellman, Decision Diffie-Hellman, SETUP, tamper-resistance, RSA, black-box ciphers, elliptic curve cryptography, twist on elliptic curves.

1 Introduction

The ability to be able to perform recovery is a necessity for large organizations that need timely access to encrypted information assets. Conventional solutions to the problem often involve the use of PKCS #12 files to store private keys for the long-term in encrypted envelopes. For the RSA cryptosystem, it has been shown that the transmission channel that exists in composites can be used to implement a key recovery system that is transparent with respect to the end user [28], thereby elimating the need for numerous PKCS#12 files and similar storage methods.

In this work we present a new and space efficient RSA [24] key recovery system/backdoor in this transparent model that has a running time that is faster than all previous approaches. Recall that a secretly embedded trapdoor with universal protection (SETUP) in RSA key generation utilizes the public key of the designer to "display" an asymmetric ciphertext in a channel[1] in composites

* Author is now at MITRE Corporation.
[1] About $|n|/2$ bits can be displayed in the bit representation of n.

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 128–143, 2006.

[9], thereby allowing an authorized escrow authority to access the RSA private decryption key of the key owner. Furthermore, the reverse-engineer who breaches the black-box and learns its internals is unable to factor the public keys of those who used the key generation black-box.

The primary technical motivation of this work is the following. First, it is a way to assess the consequences of elliptic curve (EC) technology in regards to constructing hardware-based key recovery for RSA and related technologies. The reason why current RSA SETUPs are deficient is the following. The first RSA SETUP [28] in RSA-1024 that was presented in 1996 is no longer secure since it requires that the designer's embedded public key be 512-bits in size. The security parameter of the designer's public key is half that of the RSA key being generated. So, the user's 1024-bit key ends up being no more secure than RSA-512 with respect to the reverse-engineer. This problem results from the fact that a subexponential time algorithm is known that solves the integer factorization problem and this leads to "bulky" embedded RSA ciphertexts (decryptable only by the designer). Recall that in 1996 the 430-bit RSA composite for the RSA-130 factoring challenge was solved [6] while the tougher challenges remained unsolved.[2] In December 2003, the 576-bit RSA composite RSA-576 was factored [23]. So, whereas it was conceivable in 1996 that a 512-bit modulus provided some security, this is certainly not the case now.

What this means is that there is currently no known way to implement a secure SETUP in RSA-1024 key generation. In this paper we solve this practical problem.[3] In fact, the use of a pair of twisted elliptic curves leads to a solution that is so *space efficient* that it can be used to build a SETUP in RSA-768 key generation as well, provided that a sound cryptographic hash function is available.

Another technical motivation relates to run-time efficiency. It has been noted that care must be taken to define the ensemble from which each of the RSA primes p and q is chosen and ensure that the SETUP conforms to this ensemble [27, 8]. An approach to doing so was presented [27] and we follow this approach. However, the expected number of primality tests in [27] is about $O((\log p)^2)$ (due to the Prime Number Theorem). So, a second technical motivation is to develop a recovery system that produces primes drawn from the correct distribution while achieving an expected number of primality tests that is about $O(\log p)$ as in normal key generation. Our technical contributions are as follows:

1. We present the first *strong* SETUP that is secure for RSA keys as small as 768 bits (given the current strengths of factoring and ECC and given a suitable hash function) and that has $O(\log p)$ expected primality tests.
2. We present the first SETUP in RSASSA-PSS that permits a 20-bit message to be securely transmitted within the 160-bit padding field. This is more robust than a straightforward channel [25], since previously transmitted messages remain confidential (they are asymmetrically encrypted) even after reverse-engineering.

[2] At that time RSA Inc. used decimal to title their challenges.

[3] To eliminate any possible confusion: "the problem" is one that the designer faces.

A strong SETUP in RSA key generation [29] permits the key generation "devices" (either hardware or software) to be manufactured identically (there is no need for unique identifier strings). Consider the setting in which some fraction of the deployed RSA key generators have the SETUP in them. A strong SETUP makes the following possible: even if one of the devices with a SETUP is reverse-engineered it is still not possible, given only oracle access, to distinguish the remaining devices that have been SETUP from the "good" ones.[4]

The SETUP is made possible by the use of the elliptic curve Diffie-Hellman (ECDH) key exchange algorithm. To date there is no publicly known subexponential algorithm for solving the elliptic curve discrete logarithm problem (ECDLP) [14]. As a result an ECDH key exchange value is very small, particularly when point-compression is used. This allows us to overcome the bulky ciphertext that results from "displaying" an RSA ciphertext in the channel in RSA composites, thereby allowing us to build a secure SETUP in RSA-1024 key generation. In a nutshell our SETUP carries out a ECDH key exchange between the device and the designer to permit the designer to factor the RSA modulus that is produced. To achieve the indistinguishability requirements of a strong SETUP two elliptic curves are used, one which is a "twist" of the other.

2 Background, Definitions, and Notation

A number of SETUPs in RSA key generation have been presented [28, 29, 27]. Also, approaches have been presented that emphasize speed [8]. This latter approach is intended to work even when Lenstra's composite generation method is used [20] whereas the former three will not. However, all of these approaches fail when half of the bits of the composite are chosen pseudorandomly using a seed [28] (this drives the need for improved public key standards, and forms a major motivation for the present work). It should be noted that [8] does not constitute a SETUP since it assumes that a secret key remains hidden even after reverse-engineering.

We adapt the notion of a strong SETUP [29] to two games. For clarity this definition is tailored after RSA key generation (as opposed to being more general). The threat model involves: a designer, an eavesdropper, and an inquirer.

The designer builds the SETUP into some subset of all of the black-box key generation devices that are deployed. The goal of the designer is to learn the RSA private key of a user who generates a key pair using a device contained in this subset when the designer only has access to the RSA public keys. Before the games start, the eavesdropper and inquirer are given access to the SETUP algorithm in its entirety.[5] However, in the games they play they are not given access to the internals of the particular devices that are used (they cannot reverse-engineer them).

[4] Timing analysis, power analysis, etc. are not considered here, but should be considered in future work. Our goal is to lay the foundation for building a SETUP in RSA keys wherein the security parameter of the user RSA key is at the lower end of being secure.

[5] e.g., found in practice via the costly process of reverse-engineering one of the devices.

Assumptions: It is assumed that the eavesdropper and inquirer are probabilistic poly-time algorithms and that the RSA key generation algorithm is deployed in tamper-proof black-boxes. It is traditional to supply an RSA key generator with 1^k where k is the security parameter (for theoretically meaningful runtimes). However, for simplicity we assume that the generator takes no input and that the security parameter is fixed. It is straightforward to relax this assumption. Let D be a device containing the SETUP mechanism.

Game 1: Let A and B be two key generation devices. A has a SETUP in it and B does not (B is "normal"). One of these is selected uniformly at random and then the inquirer is given oracle access to the selected machine. The inquirer wins if he correctly determines whether or not the selected device has a SETUP in it with probability significantly greater than 1/2.

Property 1: (indistinguishability) The inquirer fails Game 1 with overwhelming probability.

Game 2: The eavesdropper may query D but is only given the public keys that result, not the corresponding private keys. He wins if he can learn one of the corresponding private keys.

Property 2: (confidentiality) The eavesdropper fails Game 2 with overwhelming probability.

Property 3: (completeness) Let (y, x) be a public/private key generated using D. With overwhelming probability the designer computes x on input y.

In a SETUP, the designer uses his or her own private key in conjunction with y to recover x. In practice the designer may learn y by obtaining it from a Certificate Authority.

Property 4: (uniformity) The SETUP is the same in every black-box cryptographic device.

Property 4 implies that there are no unique identifiers in each device. The *importance* of a strong SETUP then, is that it permits the distribution of a compiled binary program in which all of the instances of the "device" will necessarily be identical without diminishing the security of the SETUP. In hardware implementations it simplifies the manufacturing process.

Definition 1. *If an RSA key generation algorithm satisfies properties 1, 2, 3, and 4 then it is a* **strong SETUP**.

A method for displaying asymmetric ciphertexts in a fashion that is indistinguishable from random bit strings was put forth in [29]. This is accomplished using the *probabilistic bias removal method* which was also employed in [1]. Other recent work in this area includes [21].

The present work utilizes the notion of a "twist" on an elliptic curve over $\mathbb{F}_q$. For typical elliptic curves used in cryptography (e.g., the curves in FIPS 186-2) only about half of $\mathbb{F}_q$ corresponds to x-coordinates on a given curve. However, by

utilizing two curves—one which is a twist of the other, it is possible to implement a trapdoor one-way permutation from $\mathbb{F}_q$ onto itself. The notion of a twist has been used to implement these types of trapdoor one-way permutations which have the (conjectured) property that inversion is exponential in the security parameter [19]. For the RSA permutation inversion is subexponential in the security parameter.

The notion of a twist has also been used to implement strong pseudorandom bit generators and to achieve a simple embedding of plaintexts in the EC version [17, 18] of ElGamal [12]. Recently, twists have been shown to be useful in the problem of implementing a PKCS in which the ciphertexts appear to be uniformly distributed bit strings [21]. In a related fashion, we use the notion of a twist to produce Diffie-Hellman (DH) [10] key exchange values that appear to be uniformly distributed bit strings.

The following is some notation that is used. Let $A \bigoplus B$ denote the bitwise exclusive-or of bit string A with bit string B where $|A| = |B|$. Let $x \approx y$ denote that the integer x is approximately equal to y. Let $x \in_R S$ denote the selection of an element x uniformly at random from set S. Uppercase is used to denote a point on an elliptic curve and lowercase is used to denote the multiplicand. So, $P = kG$ denotes the EC point P that results from adding the point G to itself k times. Let $\#E_{a,b}(\mathbb{F}_q)$ denote the number of points on the elliptic curve $E_{a,b}$ that is defined over $\mathbb{F}_q$. In the pseudocode that is provided, logical indentation will be used to show flow-control (e.g., the body of an "if" statement is indented to the right).

3 System Setup

The key generation algorithm utilizes a pair of binary curves. Each curve is described by the Weierstrass equation $E_{a,b}$ given by $y^2 + xy = x^3 + ax^2 + b$. Here the coefficients a and b are in $\mathbb{F}_{2^m}$ and $b \neq 0$. We use the standard group operations for binary elliptic curve cryptosystems. The value m should be an odd prime to avoid the possibility of the GHS attack [13]. Also, these curves must provide a suitable setting for the elliptic curve decision Diffie-Hellman problem (ECDDH). We mention this since for certain elliptic curves, DDH is tractable [16].

It is well known that Hasse's inequality implies that $|\#E_{a,b}(\mathbb{F}_{2^m}) - 2^m - 1| < 2 * 2^{m/2}$. Recall that if the trace $\mathrm{Tr}_{\mathbb{F}_{2^m}/\mathbb{F}_2}(a) \neq \mathrm{Tr}_{\mathbb{F}_{2^m}/\mathbb{F}_2}(a')$ then $E_{a,b}$ and $E_{a',b}$ are "twists" of one another. When two such curves are twists of one another then for every $x \in \mathbb{F}_{2^m}$ there exists a $y \in \mathbb{F}_{2^m}$ such that (x, y) is a point on $E_{a,b}$ or $E_{a',b}$. The two possibilities are as follows. Either (x, y) and $(x, x + y)$ are points on the same curve or $(x, y) = (0, \sqrt{b})$ is on both curves.

The sum of the number of points on both curves is given by $\#E_{a,b}(\mathbb{F}_{2^m}) + \#E_{a',b}(\mathbb{F}_{2^m})$ which is equal to $2^{m+1} + 2$. It follows from Hasse's inequality that $\#E_{a,b}(\mathbb{F}_{2^m}) \approx \#E_{a',b}(\mathbb{F}_{2^m}) \approx 2^m$.

Since m is odd $\mathrm{Tr}_{\mathbb{F}_{2^m}/\mathbb{F}_2}(0) = 0$ and $\mathrm{Tr}_{\mathbb{F}_{2^m}/\mathbb{F}_2}(1) = 1$. So, we use $E_{0,b}$ and $E_{1,b}$ as a pair of twisted curves. It remains to choose suitable curves that resist known cryptanalytic attacks (e.g., satisfying the MOV condition). Using point-counting techniques it is known how to efficiently generate two curves $E_{0,b}$ and

$E_{1,b}$ with orders $4q_0$ and $2q_1$, respectively, where q_0 and q_1 are prime. $E_{0,b}$ will have a cofactor of 4 and $E_{1,b}$ will have a cofactor of 2.

Once two such curves are found, a base point G_0 having order q_0 that is on $E_{0,b}(\mathbb{F}_{2^m})$ is found as well as a base point G_1 having order q_1 that is on $E_{1,b}(\mathbb{F}_{2^m})$. Using these base points, the designer generates the EC private key $x_0 \in_R \{1, 2, ..., q_0 - 1\}$ and corresponding public key $Y_0 = x_0 G_0$. The designer also generates $x_1 \in_R \{1, 2, .., q_1 - 1\}$ and corresponding public key $Y_1 = x_1 G_1$. The values (G_0, G_1, Y_0, Y_1) are included in the RSA key generation device.

4 Building Blocks

By using point compression, the SETUP is able to make efficient use of space. The ECC point will be embedded in the upper order bits of the RSA modulus that is being SETUP using a well known channel in composites (see [28]). A point (x, y) on the binary curve over $\mathbb{F}_{2^m}$ can be uniquely expressed using $m+1$ bits [26]. The compressed point is $(x, ybit)$ where $ybit \in \{0, 1\}$.

We define $\mathsf{PointCompress}(E_{v,b}, P)$ to be a function that compresses the point $P = (x, y)$ on curve $E_{v,b}$ and outputs $(x \mid\mid ybit)$ which is the compressed representation of (x, y). Also, we define $\mathsf{PointDecompress}(E_{v,b}, x \mid\mid ybit)$ to be a function that decompresses $(x \mid\mid ybit)$ and outputs (P, w). If $w = 1$ then P is the decompressed point on the curve $E_{v,b}$. If $w = 0$ then $(x, ybit)$ is not a point on the curve $E_{v,b}$ and P is undefined.

The following algorithm is used to generate the DH key exchange parameter and the DH shared secret. The algorithm effectively conducts an ECDH key exchange between the device and the designer wherein: the shared secret is used to generate one of the RSA primes, and the public DH paramter is *displayed* in the upper order bits of the published RSA modulus. The function below returns the public and private strings that the device uses for this "key exchange."

GenDHParamAndDHSecret():
Input: none
Output: $s_{pub}, s_{priv} \in \{0, 1\}^{m+1}$
1. with probability $\frac{4q_0 - 1}{2^{m+1}}$ set $a = 0$ and with probability $\frac{2q_1 - 1}{2^{m+1}}$ set $a = 1$
2. choose k uniformly at random such that $0 < k < q_a$
3. choose $\mu \in_R \{0, 1, 2, 3\}$
4. set $P = kG_a$
5. solve for Q such that $P = 2Q$ and Q has order $2q_a$
6. if $a = 0$ then
7. if $\mu = 1$ then set $P = Q$
8. if $\mu \in \{2, 3\}$ then choose Q_1 randomly such that $Q = 2Q_1$ and set $P = Q_1$
9. if $a = 1$ then
10. if $\mu \in \{2, 3\}$ then set $P = Q$
11. set $s_{pub} = \mathsf{PointCompress}(E_{a,b}, P)$
12. set $s_{priv} = \mathsf{PointCompress}(E_{a,b}, kY_a)$ and return (s_{pub}, s_{priv})

The "public" DH key exchange value is s_{pub}. The shared secret is s_{priv}. The following is used to recover the shared secret.

RecoverDHSecret(s_{pub}, x_0, x_1):
Input: $s_{pub} \in \{0,1\}^{m+1}$ and EC private keys x_0, x_1
Output: $s_{priv} \in \{0,1\}^{m+1}$
1. set $v = 0$
2. $(P_1, w) =$ PointDecompress$(E_{0,b}, s_{pub})$
3. if $(w = 0)$ then
4. compute $(P_1, w) =$ PointDecompress$(E_{1,b}, s_{pub})$
5. set $v = 1$
6. set $P_1 = 2^i P_1$ where $i \in \{0,1,2\}$ is the smallest value making P_1 have prime order
7. compute $P_2 = x_v P_1$
8. return $s_{priv} =$ PointCompress$(E_{v,b}, P_2)$

Let Π_θ be the set of all permutations from $\{0,1\}^\theta$ onto itself. We assume that θ is even. The SETUP utilizes the family of permutations $\pi_\theta : \{0,1\}^\theta \rightarrow \{0,1\}^\theta$ for $i = 1, 2, 3, ...$ where π_θ is chosen randomly from Π_θ. We assume that π_θ and π_θ^{-1} are efficiently computable *public* functions (e.g., they are oracles). Given a random oracle $\mathcal{H}$ this family can be constructed. (We assume the Random Oracle model).

In practice this family can be heuristically implemented using strong cryptographic hash functions. For instance, let $H : \{0,1\}^* \rightarrow \{0,1\}^{160}$ and $F : \{0,1\}^* \rightarrow \{0,1\}^{160}$ be distinct cryptographic hash functions.

$\pi_{320}(x)$:
Input: $x \in \{0,1\}^{320}$
Output: $y \in \{0,1\}^{320}$
1. let x_u and x_ℓ be bit strings such that $x = x_u \mid\mid x_\ell$ and $|x_u| = 160$
2. return $y = (x_u \bigoplus F(x_\ell \bigoplus H(x_u))) \mid\mid (x_\ell \bigoplus H(x_u))$

$\pi_{320}^{-1}(x)$:
Input: $y \in \{0,1\}^{320}$
Output: $x \in \{0,1\}^{320}$
1. let y_u and y_ℓ be bit strings such that $y = y_u \mid\mid y_\ell$ and $|y_u| = 160$
2. return $x = (y_u \bigoplus F(y_\ell)) \mid\mid (y_\ell \bigoplus H(y_u \bigoplus F(y_\ell)))$

Note that the transformations $(\pi_\theta, \pi_\theta^{-1})$ are similar to the padding scheme used in RSA-OAEP [3].

The following defines the "public" specification of allowable RSA primes. This definition requires that each of the two most significant bits be set to 1.

IsAcceptablePrime(e, len, p_1):
Input: RSA exponent e, required bit length len of p_1, candidate prime p_1
Output: true iff p_1 is an acceptable prime, false otherwise
1. if $(|p_1| \neq len)$ then halt with false
2. if the two uppermost bits of p_1 are not 11_2 then halt with false

3. if p_1 is composite then halt with false
4. if $(\gcd(p_1 - 1, e) \neq 1)$ return false else return true

Let $\mathcal{R} : \{0,1\}^* \rightarrow \{0,1\}^\infty$ be a random oracle as defined in [2]. The function $\mathsf{GetBlock}(str, i, \ell)$ returns ℓ consecutive bits of bit string str. The first such bit is the bit at bit position i of str. The bits are ordered from left to right starting with 0. For example, if $str = \mathcal{R}(s) = 01101001...$ then $\mathsf{GetBlock}(str, 0, 4) = 0110$, $\mathsf{GetBlock}(str, 1, 4) = 1101$, and $\mathsf{GetBlock}(str, 4, 4) = 1001$.

$\mathsf{GenPrimeWithOracle}(s, len, e)$:
Input: $s \in \{0,1\}^{m+1}$, required bit length len, RSA exponent e.
Output: Acceptable RSA prime p_1 such that $|p_1| = len$
1. set $j = 0$
2. let $u = \mathsf{GetBlock}(\mathcal{R}(s), j * T, T)$
3. choose $\ell \in_R \{0,1\}^{len-T}$
4. let p_1 be the integer corresonding to the bit string $u \mid\mid \ell$
5. if ($\mathsf{IsAcceptablePrime}(e, len, p_1) =$ true) then output p_1 and halt
6. set $j = j + 1$ and goto step 2

The RSA modulus of the user is $n = p_1 q_1$. We require that $|n|/4 \leq T \leq len$. A selection for T is given in Section 5.

5 Application 1: Elliptic Curve SETUP in RSA Key Generation

$N/2$ is the size in bits of each prime in n and e is the RSA exponent (this variable q_1 is different from the elliptic curve value q_1 and should be clear from the context). For simplicity we assume that N is even.

$\mathsf{GetPrimes}_{\mathsf{N,e}}(s_{pub}, s_{priv})$:
Input: $s_{pub}, s_{priv} \in \{0,1\}^{m+1}$
Output: A pair of acceptable RSA primes (p_1, q_1)
1. set $p_1 = \mathsf{GenPrimeWithOracle}(s_{priv}, N/2, e)$
2. choose $s_0 \in_R \{0,1\}^{\theta-(m+1)}$
3. compute $t = \pi_\theta(s_0 \mid\mid s_{pub})$
4. choose $r_2 \in_R \{0,1\}^{N-\theta}$
5. set $n_c = (t \mid\mid r_2)$ /* $\theta + |r_2|$ bits long */
6. solve for (q_1, r_c) in $n_c = q_1 p_1 + r_c$ /* find quotient q_1 */
7. if ($\mathsf{IsAcceptablePrime}(e, N/2, q_1) =$ false) then goto step 2
8. output (p_1, q_1) and halt

Since algorithm $\mathsf{KleptoKeyGen}_{\mathsf{N,e}}$ is deployed in this exact form in all black-box devices, Property 4 of a strong SETUP holds.

$\mathsf{KleptoKeyGen}_{\mathsf{N,e}}()$:
Input: none
Output: A pair of acceptable RSA primes (p_1, q_1)

1. $(s_{pub}, s_{priv}) = \mathsf{GenDHParamAndDHSecret}()$
2. output $(p_1, q_1) = \mathsf{GetPrimes}_{\mathsf{N,e}}(s_{pub}, s_{priv})$ and halt

Once p_1 is found in step 1 it does not change. From the Prime Number Theorem it follows that the expected number of primality tests is $O(\log n)$, the same as in normal RSA key generation. Coppersmith showed that if the $|n|/4$ significant bits of p_1 are known then $n = p_1 q_1$ can be efficiently factored [7]. For typical RSA key generation we can use $T = |n|/4$ in the SETUP. So, we use Coppersmith's method to factor n given the resulting upper order bits.

The value MAX is used to prevent the recovery algorithm from running for too long when it is supplied with an RSA private key that was not generated using the SETUP (e.g., a normal RSA key). By taking $MAX = \lceil \frac{N}{2} * 160 * \ln 2 \rceil$ it follows that if the SETUP was used then the factorization will be found with overwhelming probability.

$\mathsf{KleptoRecoveryKey}_{\mathsf{N,e}}(n, x_0, x_1)$:
Input: the user's RSA modulus n and EC private keys (x_0, x_1)
Output: Factorization of $n = p_1 q_1$ or "failure"
1. let n_s be n represented as a binary string
2. let $t = \mathsf{GetBlock}(n_s, 0, \theta)$
3. let t_0 be the integer corresponding to t
4. for $\beta = 0$ to 1 do:
5. set $t_2 = t_0 + \beta \bmod 2^\theta$
6. let t_3 be the bit string corresponding to t_2
7. set $s_{pub} = \mathsf{GetBlock}(\pi_\theta^{-1}(t_3), \theta - (m+1), m+1)$
8. $s = \mathsf{RecoverDHSecret}(s_{pub}, x_0, x_1)$
9. set $j = 0$
10. let $u = \mathsf{GetBlock}(\mathcal{R}(s), j * T, T)$
11. Attempt to factor $n = p_1 q_1$ by supplying (u, n) to Coppersmith's algorithm [7] and halt with the factorization if it is found
12. set $j = j + 1$
13. if $j < MAX$ then goto step 10
14. output "failure" and halt

The reseason that β is used is to account for a potontial borrow bit being taken from the upper order bits in n_c during the computation of $n = n_c - r_c = q_1 p_1$. A possible configuration of the attack is $N = 768$, $m = 191$, and $\theta = 320$. Observe that $768/2 - 320 = 64$. So, it is not likely that t will have a borrow bit taken from it. It is not hard to see that Property 3 holds. The security of the SETUP is proven in Appendix A.

6 Application 2: Hardware Based Key Escrow

The strong SETUP from Section 5 can be used to implement a lightweight hardware-based key escrow system. The EC private keys (x_0, x_1) are retained by a key recovery authority or may be shared using threshold cryptography among a set of trusted key recovery authorities.

The SETUP is implemented in the hardware devices of the users. The presence and details of the SETUP is publicly disclosed. To recover a given user's RSA private key, the escrow authority or authorities need only obtain the public key of the user to derive the corresponding private key.

7 Application 3: SETUP in RSASSA-PSS

We now present a SETUP in RSASSA-PSS [22] when RSASSA-PSS utilizes SHA-1. Recall that the RSASSA-PSS signature scheme is a Provably Secure Signature (PSS) scheme [4] that constitutes a Signature Scheme with an Appendix (SSA). This scheme is compatible with the IFSSA scheme as amended in the IEEE P1363a draft [15].

For concreteness we set $m = 139$. The use of $\mathbb{F}_{2^{139}}$ is based on the fact that the most recently solved binary EC discrete-logarithm challenge was ECC2-109 [11] and currently one of the easiest open challenges from Certicom is ECC2-131.[6] So, we are admittedly cutting it close.

The encryption and decryption algorithms utilize the cryptographic hash function $H : \{0,1\}^* \rightarrow \{0,1\}^{20}$. The plaintext space is $\{0,1\}^{20}$ and the ciphertext space is $\{0,1\}^{160}$. The asymmetric encryption algorithm $E_{160}(m)$ operates as follows. Let $m \in \{0,1\}^{20}$ be the plaintext message. First, E_{160} computes the value $(s_{pub}, s_{priv}) = \mathsf{GenDHParamAndDHSecret}()$. It then hashes s_{priv} by computing $pad = H(s_{priv})$. The asymmetric ciphertext is $c_1 = (s_{pub} \mid\mid (pad \bigoplus m))$.

The decryption algorithm $D_{160}(c_1)$ operates as follows. It extracts s_{pub} from the asymmetric ciphertext c_1. Algorithm D_{160} then computes the value $s_{priv} = \mathsf{RecoverDHSecret}(s_{pub}, x_0, x_1)$. It then computes $pad = H(s_{priv})$. This pad is then XORed with the 20 least significant bits of c_1 to reveal m.

The following is the SETUP in RSASSA-PSS. The signing algorithm can be used to transmit any 20 bits of information m (e.g., bits of the RSA private key, bits of a symmetric key, etc.) through RSASSA-PSS. It does so by computing $c_1 = E_{160}(m)$ and using c_1 as the random 160-bit "salt" in RSASSA-PSS. The salt/ciphertext is pseudorandom and can be recovered by anyone that is able to perform digital signature verification. However, only the designer who knows (x_0, x_1) can decrypt the salt and recover m.

Note that (E_{160}, D_{160}) is malleable and so does not achieve rigorous notions of security for a PKCS. To see this note that an active adversary can flip plaintext bit i where $0 \leq i \leq 19$ by XORing "1" with the corresponding ciphertext bit.

However, for many applications this asymmetric cryptosystem may provide sufficient security. In the SETUP in RSASSA-PSS, an active adversary that changes a bit as such will with overwhelming probability invalidate the signature. So, in this application of E_{160} non-malleability is achieved.

This SETUP differs in a fundamental way from most channels since confidentiality of m holds even if the cryptographic black-box is opened and scrutinized. Also, the approach of [21] cannot be used to implement this since it involves a

[6] There is an open Koblitz curve challenge called ECC2K-130 as well.

hash field. This hash makes security against chosen ciphertext attacks possible, but causes the minimum-length ciphertext to exceed 160 bits.

This SETUP applies equally well to the padding field in RSA-OAEP. In that scenario the designer must solicit an encrypted message from the user (since in general OAEP padding is not publicly obtainable). In this scenario, when a message is signed and encrypted using PKCS #1, it is possible to transmit $20 + 20 = 40$ bits in a SETUP. This is enough transmit a 64-bit key used by the user to secure other communications. Also, if the channel is repeated a constant number of times many cryptographic secrets can be transmitted (while being protected against reverse-engineering).

8 Conclusion

We presented a key recovery system for factoring based cryptosystems that uses elliptic curve technology. Specifically, we updated SETUP algorithms for use with RSA-1024. The SETUP achieves the notion of a strong SETUP and employs the notion of twisted elliptic curves in a fundamental way. Finally, we showed a SETUP in RSASSA-PSS and pointed out that the RSA digital signatures that result have the added advantage of providing non-malleability of the SETUP ciphertexts.

References

1. L. von Ahn, N. J. Hopper. Public-Key Steganography. In *Advances in Cryptology—Eurocrypt '04*, pages 323–341, 2004.
2. M. Bellare, P. Rogaway. Random Random oracles are practical: A paradigm for designing efficient protocols. In *1st Annual ACM CCCS*, pages 62–73, 1993.
3. M. Bellare, P. Rogaway. Optimal Asymmetric Encryption. In *Advances in Cryptology—Eurocrypt '94*, pages 92–111, 1995.
4. M. Bellare and P. Rogaway. PSS: Provably Secure Encoding Method for Digital Signatures. Submission to IEEE P1363 working group, August 1998.
5. D. Boneh. The Decision Diffie-Hellman Problem. In *Third Algorithmic Number Theory Symposium*, LNCS 1423, pages 48–63, 1998.
6. J. Cowie, B. Dodson, R.M. Elkenbracht-Huizing, A. K. Lenstra, P. L. Montgomery, J. Zayer. A world wide number field sieve factoring record: On to 512 bits. In *Advances in Cryptology—Asiacrypt '96*, pages 382–394, 1996.
7. D. Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In *Eurocrypt '96*, pages 178–189, 1996.
8. C. Crépeau, A. Slakmon. Simple Backdoors for RSA Key Generation. In *The Cryptographers' Track at the RSA Conference*, pages 403–416, 2003.
9. Y. Desmedt. Abuses in Cryptography and How to Fight Them. In *Advances in Cryptology—Crypto '88*, pages 375–389, 1988.
10. W. Diffie, M. Hellman. New Directions in Cryptography. In volume IT-22, n. 6 of *IEEE Transactions on Information Theory*, pages 644–654, Nov. 1976.
11. eCompute ECC2-109 Project. ECC2-109 solved April, 2004. Details downloaded from http://www.ecompute.org/ecc2.

12. T. ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Crypto '84*, pages 10–18, 1985.
13. P. Gaudry, F. Hess, N. Smart. Constructive and Destructive Facets of Weil Descent on Elliptic Curves. In *J. of Cryptology*, v. 15, pages 19–46, 2002.
14. D. Hankerson, A. J. Menezes, S. Vanstone. Guide to Elliptic Curve Cryptography. Preface, Springer-Verlag, Jan. 2004.
15. IEEE P1363 working group. IEEE P1363a D10: Standard Specifications for Public Key Cryptography: Additional Techniques. Nov. 1, 2001.
16. A. Joux, K. Nguyen. Separating DDH from CDH in Cryptographic Groups. In *Journal of Cryptology*, v. 16, n. 4, pages 239–247, 2003.
17. B. S. Kaliski. A Pseudo-Random Bit Generator Based on Elliptic Logarithms. In *Advances in Cryptology—Crypto '86*, pages 84–103, 1986.
18. B. S. Kaliski. Elliptic Curves and Cryptography: A Pseudorandom Bit Generator and Other Tools. PhD Thesis, MIT, Feb. 1988.
19. B. S. Kaliski. One-Way Permutations on Elliptic Curves. In *Journal of Cryptology*, v. 3, n. 3, pages 187–199, 1991.
20. A. K. Lenstra. Generating RSA Moduli with a Predetermined Portion. In *Advances in Cryptology—Asiacrypt '98*, pages 1–10, 1998.
21. B. Möller. A Public-Key Encryption Scheme with Pseudo-Random Ciphertexts. In *ESORICS '04*, pages 335–351, 2004
22. PKCS #1 v2.1: RSA Cryptography Standard. RSA Labs, Jun. 14, 2002.
23. E. Weisstein. RSA-576 Factored. MathWorld Headline News, Dec. 5, 2003. Factored by: J. Franke, T. Kleinjung, P. Montgomery, H. te Riele, F. Bahr and NFSNET (that consisted of D. Lecliar, P. Leyland, R. Wackerbarth).
24. R. Rivest, A. Shamir, L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. CACM, v. 21, n. 2, pages 120–126, Feb. 1978.
25. G. J. Simmons. Subliminal Channels: past and present. In *European Trans. on Telecommunications*, v. 5, pages 459–473, 1994.
26. S. A. Vanstone, R. C. Mullin, G. B. Agnew. Elliptic curve encryption systems. US Patent 6,141,420, Filed: Jan. 29, 1997.
27. A. Young. Kleptography: Using Cryptography Against Cryptography. PhD Thesis, Columbia University, 2002.
28. A. Young, M. Yung. The Dark Side of Black-Box Cryptography, or: Should we trust Capstone? In *Advances in Cryptology—Crypto '96*, pages 89–103, 1996.
29. A. Young, M. Yung. Kleptography: Using Cryptography Against Cryptography. In *Advances in Cryptology—Eurocrypt '97*, pages 62–74, 1997.

A Security

In this section we prove that indistinguishability and confidentiality holds. Not surprisingly, indistinguishability holds under the ECDDH assumption. The reduction algorithms utilize *point halving* (where we are interested in halving point B to obtain a point C having composite order where $B = 2C$).

A.1 Indistinguishability

The proof below randomizes 3-tuples (see [5]).

Claim 1: (Random Oracle Model) If ECDDH is hard then the SETUP produces prime pairs that are computationally indistinguishable from RSA prime pairs.

Proof: (Sketch) It will be shown that if the SETUP produces primes that are computationally distinguishable from pairs of RSA primes then ECDDH is easy. So, let $\mathcal{D}$ be a distinguisher that distinguishes pairs of primes from the SETUP from pairs of normal RSA primes.

Let $t = \mathsf{GetBlock}(n_s, 0, \theta)$ where n_s is $p_1 q_1$ represented as a binary string. Consider $s = \mathsf{GetBlock}(\pi_\theta^{-1}(t), \theta - (m+1), m+1)$. Note that s is on exactly one of the two twisted curves. There are 3 possibilities. Either,

1. $\mathcal{D}$ distinguishes with non-negligible advantage for such points s on $E_{0,b}(\mathbb{F}_{2^m})$ but with negligible advantage when such points s are on $E_{1,b}(\mathbb{F}_{2^m})$, or
2. $\mathcal{D}$ distinguishes with negligible advantage for such points s on $E_{0,b}(\mathbb{F}_{2^m})$ but with non-negligible advantage when such points s are on $E_{1,b}(\mathbb{F}_{2^m})$, or
3. $\mathcal{D}$ distinguishes with non-negligible advantage for such points s that are on $E_{0,b}(\mathbb{F}_{2^m})$ or $E_{1,b}(\mathbb{F}_{2^m})$.

Without loss of generality suppose that case (1) holds (a similar reduction argument holds for case (2)). For case (1) it follows that $\mathcal{D}$ distinguishes with probability greater than $\frac{1}{2} + \frac{1}{t_1^{\alpha_1}}$ for some fixed $\alpha_1 > 0$ and sufficiently large t_1.

Consider machine $\mathcal{M}_1$ that takes as input an ECCDH problem instance given by $(A_1, A_2, G_0, m, b) = (a_1 G_0, a_2 G_0, G_0, m, b)$ where G_0 has order q_0 and $a_1, a_2 \in \{1, 2, ..., q_0 - 1\}$. The problem is to compute $a_1 a_2 G_0$.

$\mathcal{M}_1(A_1, A_2, G_0, m, b)$:
Input: points A_1, A_2 each having order q_0 that are on curve $E_{0,b}(\mathbb{F}_{2^m})$,
base point G_0, EC parameters m and b
Output: point P having order q_0 on curve $E_{0,b}(\mathbb{F}_{2^m})$
1. choose $u_1, u_2 \in_R \{1, 2, ..., q_0 - 1\}$ and $\mu \in_R \{0, 1, 2, 3\}$
2. set $(B_1, B_2) = (u_1 A_1, u_2 A_2)$
3. set $C_1 = B_1$
4. solve for C_2 in $B_1 = 2C_2$ such that C_2 has order $2q_0$
5. choose C_3 in $C_2 = 2C_3$ randomly (C_3 has order $4q_0$)
6. if $\mu = 1$ then set $C_1 = C_2$
7. if $\mu \in \{2, 3\}$ then set $C_1 = C_3$
8. set $Y_0 = B_2$
9. set $s_{pub} = \mathsf{PointCompress}(E_{0,b}, C_1)$
10. choose s_{priv} to be a random compressed point on $E_{0,b}(\mathbb{F}_{2^m})$ having order q_0
11. randomly choose a base point G_1 having order q_1 that is on $E_{1,b}(\mathbb{F}_{2^m})$
12. choose $x_1 \in_R \{1, 2, ..., q_1 - 1\}$ and set $Y_1 = x_1 G_1$
13. compute $(p_1, q_1) = \mathsf{GetPrimes}_{\mathsf{N},\mathsf{e}}(s_{pub}, s_{priv})$
14. set L to be the empty list
15. step through the operation of $\mathcal{D}(p_1, q_1, G_0, G_1, Y_0, Y_1, m, b)$ while trapping all calls to $\mathcal{R}$, and for each call to $\mathcal{R}$, add the argument to $\mathcal{R}$ to L
16. if L is non-empty then
17. choose $s \in_R L$ and compute $(P, w) = \mathsf{PointDecompress}(E_{0,b}, s)$
18. if $(w = 1)$ then

19. if P has order q_0 then output $(u_1u_2)^{-1}P$ and halt
20. output a random point with order q_0 on $E_{0,b}(\mathbb{F}_{2^m})$ and then halt

Clearly the running time of $\mathcal{M}_1$ is efficient. Note that $\mathcal{D}$ makes at most a polynomial number of calls to random oracle $\mathcal{R}$. So, the number of elements in L is at most $p_2(m)$ where p_2 is polynomial in m.

Consider the algorithm $\mathcal{M}_2$ that takes as input an ECDDH problem instance $(A_1, A_2, A_3, G_0, m, b) = (a_1G_0, a_2G_0, a_3G_0, G_0, m, b)$ where G_0 has order q_0 and $a_1, a_2, a_3 \in \{1, 2, ..., q_0 - 1\}$.

$\mathcal{M}_2(A_1, A_2, A_3, G_0, m, b)$:
1. choose $u_1, u_2, v \in_R \{1, 2, ..., q_0 - 1\}$ and $\mu \in_R \{0, 1, 2, 3\}$
2. set $(B_1, B_2, B_3) =$
$$(vA_1 + u_1G_0, A_2 + u_2G_0, vA_3 + u_1A_2 + vu_2A_1 + u_1u_2G_0)$$
3. set $C_1 = B_1$
4. solve for C_2 in $B_1 = 2C_2$ such that C_2 has order $2q_0$
5. choose C_3 in $C_2 = 2C_3$ randomly (C_3 has order $4q_0$)
6. if $\mu = 1$ then set $C_1 = C_2$
7. if $\mu \in \{2, 3\}$ then set $C_1 = C_3$
8. set $Y_0 = B_2$
9. set $s_{pub} = \mathsf{PointCompress}(E_{0,b}, C_1)$
10. set $s_{priv} = \mathsf{PointCompress}(E_{0,b}, B_3)$
11. randomly choose a base point G_1 having order q_1 on curve $E_{1,b}(\mathbb{F}_{2^m})$
12. choose $x_1 \in_R \{1, 2, ..., q_1 - 1\}$
13. set $Y_1 = x_1G_1$
14. compute $(p_1, q_1) = \mathsf{GetPrimes}_{\mathsf{N},\mathsf{e}}(s_{pub}, s_{priv})$
15. return $\mathcal{D}(p_1, q_1, G_0, G_1, Y_0, Y_1, m, b)$ and halt

Clearly the running time of $\mathcal{M}_2$ is efficient. Observe that if (A_1, A_2, A_3) is an EC Diffie-Hellman triple then (B_1, B_2, B_3) is an EC Diffie-Hellman triple. If (A_1, A_2, A_3) is not an EC Diffie-Hellman triple then (B_1, B_2, B_3) is a random 3-tuple. If the input is not an EC Diffie-Hellman triple then with probability $(1 - \gamma_1(m))$ the tuple (B_1, B_2, B_3) will not be an EC Diffie-Hellman triple. Here γ_1 is a negligible function of m. Thus, with overwhelming probability (B_1, B_2, B_3) matches the input 3-tuple in regards to being a DH triple or not.

Let $1 - \gamma_0(k_0)$ denote the probability that s (recall that we are considering case (1)) corresponds to the EC Diffie-Hellman key exchange value.[7] Here γ_0 is a negligible function of k_0.

Let p_{trap} denote the probability that $\mathcal{D}$ calls $\mathcal{R}$ with the DH shared secret corresponding (B_1, B_2). There are two possible cases. Either, (1) $p_{trap} > \frac{1}{t_2^{\alpha_2}}$ for some fixed $\alpha_2 > 0$ and sufficiently large t_2, or (2) $p_{trap} \leq \gamma(m)$ where γ is a negligible function of m. If it is case (1) then $\mathcal{M}_1$ solves the elliptic curve Diffie-Hellman problem with probability at least $\left(\frac{1}{2} + \frac{1}{t_1^{\alpha_1}}\right) \frac{1}{p_2(m)}(1 - \gamma_0(k_0))\frac{1}{t_2^{\alpha_2}}$. If it is case (2) then $\mathcal{M}_2$ solves the ECDDH problem with probability at least

[7] A borrow bit can be taken and prevent s from being the correct point.

$(1-\gamma(m))\,(1-\gamma_0(k_0))\,(1-\gamma_1(m))\left(\frac{1}{2}+\frac{1}{t_1^{\alpha_1}}\right)$. So, at least one of the machines in the set $\{\mathcal{M}_1,\mathcal{M}_2\}$ can be used to solve ECDDH efficiently. ◇

Claim 1 proves that Property 1 holds.

A.2 Confidentiality

Claim 2: (Random Oracle Model) If the factorization and the EC Computational Diffie-Hellman (ECCDH) problems are hard then confidentiality of the SETUP holds.

Proof: (Sketch) It will be shown that if the confidentiality of the SETUP does not hold then factoring or ECCDH is easy. Let $\mathcal{A}$ be an algorithm that foils the confidentiality of the SETUP with non-negligible probability. More specifically, with probability greater than $\frac{1}{t_1^{\alpha_1}}$ for some fixed $\alpha_1 > 0$ and sufficiently large t_1, $\mathcal{A}(n, G_0, G_1, Y_0, Y_1, m, b)$ returns a prime factor p_1 that divides n.

Consider the following efficient algorithm.

$\mathcal{M}_{1,0}(A_1, A_2, G, m, b)$:
Input: points A_1, A_2 with order q_0 on curve $E_{0,b}(\mathbb{F}_{2^m})$, base point G_0 having order q_0 on curve $E_{0,b}(\mathbb{F}_{2^m})$, EC parameters m and b
Output: A point P on $E_{0,b}(\mathbb{F}_{2^m})$ having order q_0
1. choose $u_1, u_2 \in_R \{1, 2, ..., q_0 - 1\}$ and $\mu \in_R \{0, 1, 2, 3\}$
2. set $(B_1, B_2) = (u_1 A_1, u_2 A_2)$
3. set $C_1 = B_1$
4. solve for C_2 in $B_1 = 2C_2$ such that C_2 has order $2q_0$
5. choose C_3 in $C_2 = 2C_3$ randomly (C_3 has order $4q_0$)
6. if $\mu = 1$ then set $C_1 = C_2$
7. if $\mu \in \{2, 3\}$ then set $C_1 = C_3$
8. set $Y_0 = B_2$
9. set $s_{pub} = \mathsf{PointCompress}(E_{0,b}, C_1)$
10. choose s_{priv} to be a random compressed point on $E_{0,b}(\mathbb{F}_{2^m})$ having order q_0
11. randomly choose a base point G_1 having order q_1 on curve $E_{1,b}(\mathbb{F}_{2^m})$
12. choose $x_1 \in_R \{1, 2, ..., q_1 - 1\}$
13. set $Y_1 = x_1 G_1$
14. compute $(p_1, q_1) = \mathsf{GetPrimes}_{\mathsf{N,e}}(s_{pub}, s_{priv})$
15. set $n = p_1 q_1$ and let L be the empty list
16. step through the operation of $\mathcal{A}(n, G_0, G_1, Y_0, Y_1, m, b)$ while trapping all calls to $\mathcal{R}$, and for each call to $\mathcal{R}$, add the argument to $\mathcal{R}$ to L
17. if L is non-empty then
18. choose $s \in_R L$ and compute $(P, w) = \mathsf{PointDecompress}(E_{0,b}, s)$
19. if $(w = 1)$ then
20. if P has order q_0 then output $(u_1 u_2)^{-1} P$ and halt
21. output a random point on $E_{0,b}(\mathbb{F}_{2^m})$ having order q_0 and then halt

The size of list L is at most $p_2(m)$ where p_2 is polynomial in m. A similar reduction algorithm $\mathcal{M}_{1,1}$ can be constructed for the elliptic curve in which

the value $a = 1$. The remainder of this proof considers EC Diffie-Hellman over $E_{0,b}(\mathbb{F}_{2^m})$ unless otherwise specified. Now consider the following algorithm.

$\mathcal{M}_2(n)$:
1. randomly choose a base point G_0 having order q_0 on curve $E_{0,b}(\mathbb{F}_{2^m})$
2. randomly choose a base point G_1 having order q_1 on curve $E_{1,b}(\mathbb{F}_{2^m})$
3. choose $x_0 \in_R \{1, 2, ...q_0 - 1\}$ and choose $x_1 \in_R \{1, 2, ..., q_1 - 1\}$
4. compute $Y_0 = x_0 G_0$ and $Y_1 = x_1 G_1$
5. output $\mathcal{A}(n, G_0, G_1, Y_0, Y_1, m, b)$

Clearly the running time of $\mathcal{M}_2$ is efficient.

Let $t = \mathsf{GetBlock}(n_s, 0, \theta)$ where n_s is $p_1 q_1$ represented as a binary string. Consider $s = \mathsf{GetBlock}(\pi_\theta^{-1}(t), \theta - (m+1), m+1)$. Let $1 - \gamma_0(k_0)$ denote the probability that s corresponds to the EC Diffie-Hellman key exchange value. Here γ_0 is a negligible function of k_0.

Let p_{trap} denote the probability that $\mathcal{A}$ calls $\mathcal{R}$ with the DH shared secret corresponding to (B_1, B_2). One of the following must occur: (1) $p_{trap} > \frac{1}{t_2^{\alpha_2}}$ for some fixed $\alpha_2 > 0$ and sufficiently large t_2, or (2) $p_{trap} \leq \gamma(m)$ where γ is a negligible function of m. If it is case (1) then $\mathcal{M}_{1,0}$ (or $\mathcal{M}_{1,1}$) solves ECCDH with probability at least $\frac{1}{t_1^{\alpha_1}} \frac{1}{t_2^{\alpha_2}} (1 - \gamma_0(k_0)) \frac{1}{p_2(m)}$. If it is case (2) then $\mathcal{M}_2$ factors with probability at least $(1 - \gamma(m)) \frac{1}{t_1^{\alpha_1}}$ that is equal to $\frac{1}{t_1^{\alpha_1}} - \frac{\gamma(m)}{t_1^{\alpha_1}}$. It follows that ECCDH or factoring is efficiently solvable. $\diamond$

Claim 2 proves that Property 2 of a strong SETUP holds. So, we have:

Theorem 1. $(\mathsf{KleptoKeyGen_{N,e}}, \mathsf{KleptoRecoverKey_{N,e}})$ *is a* **strong SETUP**.

An Efficient Public Key Cryptosystem with a Privacy Enhanced Double Decryption Mechanism

Taek-Young Youn[1,⋆], Young-Ho Park[2], Chang Han Kim[3], and Jongin Lim[1]

Graduate School of Information Security, Korea University, Seoul, Korea
{taekyoung, jilim}@cist.korea.ac.kr
Dept. of Information Security, Sejong Cyber University, Seoul, Korea
youngho@cybersejong.ac.kr
Dept. of Information Security, Semyung University, Jecheon, Korea
chkim@semyung.ac.kr

Abstract. A clue of double decryption mechanism was introduced at Eurocrypt '02 by Cramer and Shoup, and it was revisited at Asiacrypt '03 by Bresson, Catalano and Pointcheval. Previous double decryption schemes are designed based on $\mathbb{Z}_{n^2}$ where $n = pq$ for two primes, p and q. Note that, they use the Paillier's scheme as a primitive scheme to design a double decryption mechanism. In this paper, we propose an efficient public key scheme with double decryption mechanism based on $\mathbb{Z}_{p^2q}$. Our scheme is more efficient than the previous schemes. Moreover, we review the previous schemes in a privacy point of view and propose a privacy enhanced double decryption scheme.

Keywords: public key cryptosystem, double trapdoor decryption mechanism, semantic security, privacy.

1 Introduction

Public key cryptosystem (PKC) is regarded as a useful tool for secure communication, therefore designing a good PKC is an essential task for not only theoretic purpose but also practical purpose. When designing a PKC, we have to consider two conditions, security and efficiency.

In security point of view, two conditions are considered, one-wayness and semantic security. One-wayness is regarded as a basic condition for secure scheme, but it is not a sufficient condition to gain real security. In these days, the semantic security is also required as a fundamental condition for the security of PKC. In [2], some examples which illustrate the need of semantic security are listed. For example, to design an authenticated key exchange protocol in the public-key setting, the scheme has to meet IND-CCA2 security. However, even though a scheme is secure, we have to check whether the scheme is efficient or not. In general, the efficiency is regarded as an important condition as the security.

⋆ This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Assessment).

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 144–158, 2006.

In [4], Cramer and Shoup introduced a clue of double decryption mechanism. After that, Bresson *et al.* revisited the mechanism and proposed a double decryption scheme [1]. The previous double decryption mechanisms are derived from Paillier's scheme [13], which is designed based on $\mathbb{Z}_{n^2}$ where $n = pq$ for two primes. Since the size of modulo of Paillier's scheme is twice than that of standard RSA or ElGamal in same security level, the efficiency of Paillier's scheme is not favorable. Hence the efficiency of the previous double decryption scheme is also not good. So, it is worth designing an efficient double decryption scheme.

In [1], Bresson *et al.* stated the necessity of double decryption scheme and proposed a scheme that solves the following two scenarios.

1. The head of a group may want to be able to read any message sent to the members of the group.
2. People may want to be able to recover the plaintexts even if they loose their private key.

To solve above two scenarios simultaneously, we need a kind of super-key. In [1], factoring information is given to authority as a super-key. But, in this scenario, we have some apprehensions about an invasion of privacy, since authority can decrypt any ciphertext without any consent of corresponding user. To prevent the abuse of super-key, i.e., to enhance the privacy of users, we need a way to restrict the excessive ability of authority.

In this paper, we propose an efficient double decryption scheme based on $\mathbb{Z}_{p^2q}$. Our scheme is efficient than the previous schemes [4, 1], since our scheme executes the cryptographic operations, such as modulo multiplication and exponentiation, on a smaller modulo than the previous schemes. Our basic scheme provides about 3 times faster encryption and decryption for ordinary user than the previous scheme. Moreover, the authority can decrypt a ciphertext about 4 times faster than the previous scheme. By modifying our basic scheme, we propose a scheme which can get rid of the apprehension of the invasion of privacy without losing the double decryption mechanism.

2 Preliminaries

From now, we review two previous schemes and describe the number-theoretic framework underlying our scheme.

2.1 Previous Schemes

In [1], Bresson, Catalano and Pointcheval proposed a modification of Paillier's scheme, which provides double trapdoor decryption mechanism. Let $n = pq$ be a safe-prime modulo, i.e. p and q are primes of the form $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are also primes. Let $\mathbb{G}$ be the cyclic group of quadratic residues modulo n^2. Then $ord(\mathbb{G}) = \lambda(n^2)/2 = n\lambda(n)/2$. Note that every element of order n is of the form $\alpha = 1 + kn$ for some $k \in \mathbb{Z}_n$.

Key generation. Choose two primes p and q ($|p| = |q|$), and let $n = pq$. Choose a random $\alpha \in \mathbb{Z}_{n^2}$, a random number $a \in [1, ord(\mathbb{G})]$ and set $g = \alpha^2 \mod n^2$

and $h = g^a \mod n^2$. The public key is (n, g, h) while the corresponding secret key is a. Two prime factors, p and q, are superkey-like secret key.

Encryption. Given a message $m \in \mathbb{Z}_n$, choose a random $r \in \mathbb{Z}_{n^2}$. Then the ciphertext is computed as follows: $C = (A, B)$ where $A = g^r \mod n^2$ and $B = h^r(1 + mn) \mod n^2$.

Decryption1. First trapdoor is similar to that of ElGamal scheme. Knowing a, decryption operation is executed as follows: $m = (B/A^a - 1 \mod n^2)/n$.

Decryption2. Second decryption mechanism uses the factoring information of n as a secret key. If the prime factors are provided, $a \mod n$ and $r \mod n$ are easily recovered (see [1] for details). If we write $ar \mod ord(\mathbb{G})$ by $\gamma_1 + \gamma_2 n$ then $\gamma_1 = ar \mod n$. Firstly, compute

$$D = (\frac{B}{g^{\gamma_1}})^{\lambda(n)} = \frac{(g^{ar}(1 + mn))^{\lambda(n)}}{g^{\gamma_1 \lambda(n)}} = 1 + m\lambda(n)n \mod n^2.$$

Let π be the inverse of $\lambda(n)$ in $\mathbb{Z}_n^*$, then m is recovered by the following simple computation: $m = ((D - 1 \mod n^2)/n)\pi \mod n$.

By analyzing the scheme in [1], we can see that if a group has a mechanism that solves the discrete logarithm problem, a double decryption scheme can be designed based on the group.

In [12], a novel scheme proposed by Okamoto and Uchiyama as OU scheme in short, is based on $\mathbb{Z}_{p^2q}$. The scheme uses such a mechanism that solves the discrete logarithm problem. Therefore we can design a double decryption scheme based on $\mathbb{Z}_{p^2q}$. So, it is meaningful to review the scheme [12].

OU scheme is based on a logarithmic function, L, defined over p-Sylow subgroup of $\mathbb{Z}_{p^2}^*$. Let $\Gamma = \{x \in \mathbb{Z}_{p^2}^* | x \equiv 1 \mod p\}$. Then, for $x \in \Gamma$, L is defined as $L(x) = \frac{x-1}{p}$. The function L is well-defined and has a homomorphic property from multiplication to addition (see [12] for details). Let $x_p = x^{p-1} \mod p^2$ for $x \in \mathbb{Z}_n$.

Key generation. Choose two primes p and q ($|p| = |q| = k$), and let $n = p^2q$. Choose a random g in $\mathbb{Z}_n$ such that the order of $g^{p-1} \mod p^2$ is p. Let $h = g^n \mod n$. The public key is (n, g, h, k) while the corresponding secret key is (p, q).

Encryption. Given a message $m \in [1, 2^{k-1}]$, choose a random $r \in \mathbb{Z}_n$. Then the ciphertext is computed as following: $C = g^m h^r \mod n$.

Decryption. Compute $C_p = C^{p-1} \mod p^2$ and $g_p = g^{p-1} \mod p^2$. Then the plaintext is computed as following: $m = L(C_p)/L(g_p) \mod p$.

2.2 Discrete Logarithm and Diffie-Hellman Problem over $\mathbb{Z}_{p^2q}$

The Diffie-Hellman problem [6] is a well-known cryptographic primitive. Until now, the Diffie-Hellman problem remains the most widely used cryptographic technique. Our scheme is also designed based on a kinds of Diffie-Hellman problem, denoted by p-DHP.

Let $\mathcal{P}(k)$ be the set of prime numbers of length k. Choose two primes p and q in $\mathcal{P}(k)$. From now, let $n = p^2q$. Let $\mathbb{G}_p = \{x \in \mathbb{Z}_n \mid \text{order of } x^{p-1} \mod p^2 \text{ is } p\}$. Formal definition of p-DHP is described below.

Definition 1. *(p-DHP) The p-DHP is defined as follows: Given a set* $\mathbb{G}_p$*, an element* g *of* $\mathbb{G}_p$ *and* $(g^a \mod n,\ g^b \mod n)$ *for* $a, b \in_R [1, p-1]$*, find* $g^{ab} \mod n$*.*

Although it is not known whether DHP over $\mathbb{Z}^*_{p^2q}$ is more tractable than DHP over $\mathbb{Z}^*_{rs}$ (r and s are prime numbers such that $|r| = |s|$) or vice versa, the security of a prime order subgroup of $\mathbb{Z}^*_{rs}$ is studied in [10]. The attack described in [10] is valid only on the prime order subgroup of $\mathbb{Z}^*_{rs}$ rather than composite order subgroup. Note that, our scheme use a generator g whose order is composite number. Moreover, there is no known attack for breaking the DHP over $\mathbb{Z}^*_{p^2q}$. So, the hardness of p-DHP is based on the size of modulo. The size of exponent, k bit, is not too small to be broken, since 160 bit of exponent is sufficient to gain the desired security on DHP in these days.

Conjecture 1. For every probabilistic polynomial time algorithm $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for sufficiently large k,

$$\Pr\left[\mathcal{A}(n, A, B) = C \;\middle|\; \begin{array}{cc} p, q \leftarrow \mathcal{P}(k); & n = p^2q; \\ g \leftarrow \mathbb{G}_p; & a, b \leftarrow_R [1, p-1]; \\ A = g^a \mod n; & B = g^b \mod n; \\ C = g^{ab} \mod n; & \end{array}\right] \leq negl(k).$$

From now, we define a kind of DLP over $\mathbb{Z}^*_n$, denoted by p-DLP. After that, we will prove that the hardness of p-DLP is equivalent to factoring n.

Definition 2. *(p-DLP) The p-DLP is defined as follows: Given a set* $\mathbb{G}_p$*, an element* g *of* $\mathbb{G}_p$ *and* $g^a \mod n$ *for* $a \in_R \mathbb{Z}_n$*, find* $a \mod p$*.*

Theorem 1. *p-DLP over* $\mathbb{Z}^*_n$ *is hard to solve if and only if the factoring assumption holds.*

Proof. ($\Rightarrow$) Suppose that the factoring assumption does not hold. Let $A = g^a \mod n$ for some $a \in \mathbb{Z}_n$. Since we can find the factoring of n, $a \mod p$ is recovered as following: $a' = a \mod p = L(A_p)/L(g_p) \mod p$.

($\Leftarrow$) Suppose that there exist an algorithm $\mathcal{A}$ which solves p-DLP over $\mathbb{Z}_n$. Choose a random $k \in [2^{k+1}, n]$ and compute g^k. Then, for given g^k, $\mathcal{A}$ outputs $k' = k \mod p$. Since $k > p$, we have $k' \neq k$. So, we get $gcd(n, k - k') = p$, a factor of n. □

Remark 1. We proved that p-DLP over $\mathbb{Z}_n$ is hard to solve if and only if the factoring assumption holds by using the idea in [12]. In [12], it is proved that the one-wayness of OU scheme is intractable if and only if the factoring assumption holds. The hardness of p-DLP is equivalent to the one-wayness of OU scheme and so we have the following relations: p-DLP $\Leftrightarrow$ factoring assumption $\Leftrightarrow$ one-wayness of OU scheme.

2.3 Semantic Security

The notion of securities are firstly considered in [8, 5]. After the concept of semantic securities are announced, many general conversion methods that make a semantically secure scheme from a naive scheme are proposed in [7, 15, 3, 9].

From now, we describe one of the previous general conversion methods. By using the method, we can make a semantically secure double decryption scheme from our naive double decryption scheme. Notice that, this is just a summary of the general conversion method of Kiltz and Lee [9], so, any understanding reader who knows the method need not see this section.

General Conversion Method of Kiltz and Lee. Kiltz and Lee proposed a general construction for public key encryption schemes that are IND-CCA2 secure in the random oracle model [9]. The conversion method based on a general hard problem called as Y-computational problem (YCP). They point out that many of the most widely used cryptographic primitives, such as RSA and Diffie-Hellman, are YCP.

Definition 3. *An instance generator $\mathcal{I}_{YC}(1^k)$ for YC outputs a description of (S_1, S_2, f_1, f_2, t). Here S_1 and S_2 are sets with $|S_1| = k$, $f_1, f_2 : S_1 \to S_2$ are functions and $t : S_2 \to S_2$ is a (trapdoor) function such that for all $x \in S_1$, $t(f_1(x)) = f_2(x)$. The functions f_1, f_2 and t should be easy to evaluate and it should be possible to sample efficiently from S_1. Let $\mathcal{A}$ be an adversary and define*

$$Adv_{\mathcal{A},\mathcal{I}_{YC}}(1^k) = Pr\left[\begin{array}{ll}(S_1, S_2, f_1, f_2, t) \leftarrow \mathcal{I}_{YC}(1^k); & x \in S_1; \\ f_2(x) \leftarrow \mathcal{A}(S_1, S_2, f_1, f_2, f_1(x)); & \end{array}\right].$$

We define the advantage function $Adv_{\mathcal{I}_{YC}}(1^k, t) = max\{Adv_{\mathcal{A},\mathcal{I}_{YC}}(1^k)\}$ where the maximum is taken over all adversaries that run for time t. We say that YCP is hard for $\mathcal{I}_{YC}(1^k)$ if t being polynomial in k implies that the advantage function $Adv_{\mathcal{I}_{YC}}(1^k, t)$ is negligible in k.

Under YCP in the random oracle model, Kiltz and Lee propose a general construction of an IND-CPA secure cryptosystem. The conversion model is composed as following: $\mathcal{E}_{pk}(m, r) = (f_1(r), E_\kappa(m))$ where E is symmetric encryption function and $\kappa = G(f_2(r))$ where G is an hash function. Let the converted encryption scheme as Π_0.

By applying the conversion method in [7], they convert the cryptosystem Π_0 to a cryptosystem Π_1 that is IND-CCA2 secure in the random oracle model. The converted IND-CCA2 secure scheme is composed as following: $\mathcal{E}_{pk}(m, r) = (f_1(H(m||r)), E_\kappa(m||r))$. where E is a symmetric encryption function, H is an hash function and $\kappa = G(f_2(H(m||r)))$ where G is an hash function.

2.4 User's Privacy Against Authority

In general, a malicious authority of a system is not distinguished from other adversaries. However, in some cases, the authority has more information than other players. Hence, it needs to distinguish the malicious authority from other adversaries. The authority in the system of the previous double decryption scheme also has such an information, the factoring. So, the authority can decrypt any ciphertext without the consent of an user by using the factoring information. As a result, any user can not expect a privacy against the authority. For this reason, we define the privacy of an encryption scheme against a malicious authority as the one-wayness and semantic security against the authority.

Definition 4. *(One-Wayness against Authority) Suppose that there is a system with an authority. Let s_{sp} be a secret system parameter only known to the authority and p_{sp} be a public system parameter known to all users in the system. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme of an user. Let $\mathcal{A}$ be a malicious authority that breaks the one-wayness of the scheme, then the advantage of $\mathcal{A}$ is defined as following:*

$$Adv^{ow-atk}_{\mathcal{A},\Pi,s_{sp}}(1^k) = Pr\left[\begin{array}{c|l} \mathcal{A}(C, p_k, s_{sp}) & (s_k, p_k) \leftarrow \mathcal{K}(p_{sp});\ M \leftarrow \{0,1\}^n; \\ = M & C \leftarrow \mathcal{E}_{sk}(M); \end{array}\right].$$

Then the scheme Π is one-way against the authority if and only if there exists a negligible function $negl(\cdot)$ such that for sufficiently large k,

$$Adv^{ow-atk}_{\mathcal{A},\Pi,s_{sp}}(1^k) \leq negl(k).$$

Definition 5. *(Semantic Security against Authority) Under the same condition of Definition 4, let $\mathcal{A}$ be the authority that breaks the semantic security of the scheme, then the advantage of $\mathcal{A}$ is defined as following:*

$$Adv^{ind-atk}_{\mathcal{A},\Pi,s_{sp}}(1^k) = 2Pr\left[\begin{array}{c|l} \mathcal{A}(c, p_k, s_{sp}) & (s_k, p_k) \leftarrow \mathcal{K}(p_{sp});\ M_0, M_1 \leftarrow \{0,1\}^n; \\ = b & b \leftarrow \{0,1\}; \qquad C \leftarrow \mathcal{E}_{sk}(M_b); \end{array}\right] - 1.$$

Then the scheme Π is semantically secure against the authority if and only if there exists a negligible function $negl(\cdot)$ such that for sufficiently large k,

$$Adv^{ind-atk}_{\mathcal{A},\Pi,s_{sp}}(1^k) \leq negl(k).$$

It is easy to grasp the notions, since the notions are defined based on the existing notions of one-wayness and semantic security. A difference of these notions and the existing notions is the information given to the adversary. If we think the secret system parameters s_{sp} as a public information, one-wayness and semantic security against the authority are same as the previous notion of securities against the ordinary adversary. So, in this case, only the secret key of a user s_k is secret information which does not given to the authority. Since the notion of securities against the authority can be seen as the previous notions, the relation of notion of securities [2] are holds equivalently.

In the previous scheme [1], the modulo n and the generator g is public system parameters while the prime factors p and q are the secret system parameters. Sometimes, the public system parameters are duplicated with the public key information of an user. For example, the generator g is public system parameter and also the public key information of all users. However, the secret system parameters are not duplicated and remains secret to the public, so the duplication is not a matter for our definition.

3 A New Double Decryption Scheme

3.1 Description of the Proposed Scheme

Key generation. Choose two primes p and q ($|p| = |q| = k$), and let $n = p^2q$. Choose a random g in $\mathbb{Z}^*_n$ such that the order of $g^{p-1} \mod p^2$ is p. Choose a

random $k-1$ bit a and compute $h = g^a \mod n$. The public key is (n, g, h, k) while the corresponding secret key is a. Two prime factors, p and q, are superkey-like secret key. Only the authority knows the factoring of n.

Encryption. Given a message $m \in \mathbb{Z}_n$, choose a random $k-1$ bit r. Then the ciphertext is computed as $C = (A, B)$ where $A = g^r \mod n$ and $B = h^r m \mod n$.

Decryption 1. First trapdoor is similar to that of ElGamal scheme. With the knowledge of a, one can decrypt m as following: $m = B/A^a \mod n$.

Decryption 2. Second decryption mechanism depends on the factoring information of n. Firstly, compute $h_p = h^{p-1} \mod p^2$. Then the secret value a is computed as following: $a = L(h_p)/L(g_p) \mod p$. Compute $A^a \mod n$ with a, then m is recovered by the following computation: $m = B/A^a \mod n$.

3.2 Security Analysis of the Proposed Scheme

One-Wayness. Our scheme is broken if one can solves p-DHP or p-DLP. In general, DLP is hard to solve than DHP. Therefore, it is sufficient to show that the one-wayness of our scheme is equivalent to the hardness of p-DHP.

Theorem 2. *Our double decryption scheme is one-way if and only if the p-DHP is hard.*

Proof. Assume that the p-DHP is not hard. Then there exists a polynomial time algorithm $\mathcal{B}$ which can solve p-DHP with non-negligible probability. We will construct a polynomial time algorithm $\mathcal{A}$, with help of $\mathcal{B}$, which can break the one-wayness of our scheme. Let the challenge ciphertext and public key be $(A = g^r \mod n, B = g^{ar} m \mod n)$ and (n, g, g^a), respectively. Since $\mathcal{B}$ can compute $g^{ar} \mod n$ from $(g^r \mod n, g^a \mod n)$, the corresponding plaintext is computed as $m = B/g^{ar} \mod n$. So, the scheme is not one-way.

Conversely, suppose that the proposed scheme is not one-way. Then for given ciphertext, an adversary $\mathcal{A}$ can recover the plaintext with non-negligible probability. We can make a polynomial time adversary $\mathcal{B}$, with help of $\mathcal{A}$, which can solve p-DHP. Let $(g^a \mod n, g^b \mod n)$ be a challenge pair to compute $g^{ab} \mod n$. Set (n, g, g^a) and $(A = g^b \mod n, B = g^k \mod n)$ for some $k \in \mathbb{Z}_n$ as public key data and ciphertext, respectively. Note that $k = ab + k'$ for some k'. So $g^k = g^{ab+k'} = g^{ab} g^{k'} = g^{ab} m \mod n$. Note that $m = g^{k'} \mod n$. Since the proposed scheme is not one-way, $\mathcal{A}$ can recover the corresponding plaintext m from $(A = g^b \mod n, B = g^k \mod n)$. With help of $\mathcal{A}$, $\mathcal{B}$ can compute $g^{ab} = g^k/m \mod n$ from $(g^a \mod n, g^b \mod n)$. □

Remark 2. When m is an element of $\mathbb{Z}_n$ of order $q-1$, m has no component in $\mathbb{G}_p$. In this case, the problem of inverting the encryption function for such message is not reduced to the p-DHP. However, the probability that a randomly chosen massage m has of order $q-1$ is about $\frac{1}{2^k}$. So, the problem of inverting the encryption function is completely reduced to the p-DHP except for negligible probability $\frac{1}{2^k}$.

Semantic Security. To apply the general conversion method proposed in [9], the security of a scheme has to be based on a kind of YCP. As commented in [9], DHP is a YCP. So, we can apply the conversion method to our scheme and then our scheme is semantically secure against CCA2 adversary.

Let H, G be two hash functions, then the converted encryption function is given as following: $\mathcal{E}_{pk}(m, r) = (g^{H(m||r)} \mod n, E_\kappa(m||r))$ where E is symmetric encryption function and $\kappa = G(h^{H(m||r)} \mod n)$. As commented in [9], we can enhance the efficiency by using the one-time pad as the symmetric function E (i.e., $E_\kappa(m||r) = \kappa \oplus (m||r)$).

In [9], the enhanced security of ElGamal encryption scheme is proved. According to their proof, ElGamal encryption scheme is secure against CCA2 attacker if the corresponding computational DHP is intractable and the symmetric encryption scheme is OTE[1] secure.

Since our scheme is a ElGamal type encryption scheme which is based its security on the computational p-DHP. We omit the proof of the semantic security against CCA2 attack, since the security proof for ElGamal type is given in [9].

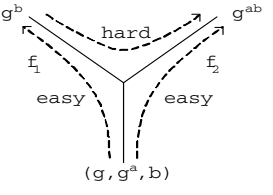


Fig. 1. Y-computational problem: the case of Diffie-Hellman problem

Theorem 3. *In the random oracle model, the converted our scheme is IND-CCA2 secure if the computational p-DHP is intractable and the symmetric encryption scheme E is OTE secure.*

3.3 Efficiency

We denote a modular exponentiation under modulo M with exponent e by $ME(|M|, |e|)$ where $|M|$ and $|e|$ are the bit length of modulo M and exponent e, respectively. Note that, the computational complexity of $ME(a, b)$ and $ME(\alpha a, \beta b)$ are in the ratio of 1 to $\alpha^2\beta$. For example, the calculation of $ME(a, b)$ is $12 = 2^2 3$ times faster than that of $ME(2a, 3b)$.

In [14], Rene Peralta claims that the hardness of factoring of 1600 bit integer of the form p^2q is equivalent to 1000 bit RSA integer. So, to compare the efficiency in same security level, we have to compare 1600 bit integer of the form p^2q with 1000 bit RSA modulo.

We compare our scheme with the previous double decryption scheme [1]. The previous scheme is semantically secure in the standard model. On the other

[1] Here, OTE means *one time encryption*. Since we use the term OTE to use the previous conversion model [9], we did not explain the detailed explanation about the OTE.

Table 1. Efficiency Comparison

Scheme	The scheme in [1]	Proposed Scheme
Plaintext	1000 bit	1600 bit
Ciphertext	4000 bit	3200 bit
Encryption	$2ME(2000, 1000)$	$2ME(1600, 533)$
Decrytion 1	$ME(2000, 1000)$	$ME(1600, 533)$
Decrytion 2	$2ME(2000, 1000)$	$ME(1066, 533) + ME(1600, 533)$

hand, our scheme is semantically secure in the random oracle model. So, we compare the efficiency of two double decryption schemes in the plain scheme point of view. Here, the term *plain scheme* means the basic model that is not transformed to CCA2 secure scheme.

Our scheme is more efficient than the previous double decryption schemes. The scheme in [1] needs 4000 bit ciphertext to guarantee the same security as 1000 bit RSA. However, our scheme needs only 3200 bit ciphertext. Moreover, the length of plaintext is larger than that of the scheme in [1]. From a computational complexity point of view, our scheme is more efficient that that of [1]. The encryption and decryption of our scheme is faster than the previous scheme about 3 times. Moreover, the cost of authority's decryption is about 4 times cheaper than the previous scheme.

4 Privacy Enhanced Double Decryption Scheme

Our double decryption scheme, proposed in the section 3, involves the same apprehension about the invasion of privacy as the previous double decryption schemes, i.e., authority can decrypt any ciphertext without a consent of an user. To solve the problem, we need some trick to restrict the ability of authority. In this section, we propose a double decryption scheme which provides a way to restrain the unlimited ability of authority.

From now, we give a detailed description of the scheme for a reader to gain a better understanding of our idea, though there are many repetition.

4.1 Description of the Privacy Enhanced Double Decryption Scheme

Key generation. Choose two primes p and q ($|p| = |q| = k$), and let $n = p^2q$. Choose a random $g \in \mathbb{Z}_n^*$ such that the order of $g^{p-1} \mod p^2$ is p. If an user allows the authority to decrypt his ciphertext, he chooses a random $k-1$ bit integer a and compute $h = g^a \mod n$. Otherwise, he choose a random t bit a where $t > k$, and compute $h = g^a \mod n$. Then the public key is (n, g, h, k) or (n, g, h, t) while the corresponding secret key is a. Two prime factors, p and q, are superkey-like secret key. Only the authority knows the factoring of n.

Encryption. Given a message $m \in \mathbb{Z}_n$, choose a random t bit r. Then the ciphertext is computed as follows: $C = (A, B)$ where $A = (g^r \mod n$ and $B = h^r m \mod n)$.

Decryption 1. With the knowledge of the secret value a, one can decrypt given a ciphertext as following: $m = B/A^a \mod n$.

Decryption 2. Second decryption mechanism depends on the factoring information of n. The authority can decrypt a ciphertext with a corresponding user's consent. If the user permits the authority to decrypt, the authority can recover the plaintext. Firstly, he computes $h_p = h^{p-1} \mod p^2$. Then the random $k-1$ bit a is computed as following: $a = L(h_p)/L(g_p) \mod p$. The authority computes $A^a \mod n$ by using a. Then m is recovered by the following simple computation: $m = B/A^a \mod n$.

Remark 3. If the user does not consent, the authority can not recovers the secret exponent a. The authority can find $a' = a \mod p$ since he knows the factoring of n. However, for sufficiently large k and t, it is hard to finding out a from a'. So the authority can not recover the plaintext for given ciphertext. The hardness of this problem will be discussed and proved in the next section.

Remark 4. When a sender want to permit the authority's ability of decryption, he choose $k-1$ bit r and compute $g^r \mod n$. Then the authority can decrypt the ciphertext generated by the sender by computing the secret exponent r though the corresponding receiver dose not consent the authority's decryption. At first, the authority computes $(g^r)_p = (g^r)^{p-1} \mod p^2$. Then the random $k-1$ bit r is computed as following: $r = L((g^r)_p)/L(g_p) \mod p$.

Remark 5. In the case of the encrypted information is not important to an user's privacy, the user will consent the authority to decrypt his ciphertext to enjoy the properties of double decryption mechanism.

4.2 Security Analysis of the Privacy Enhanced Double Decryption Scheme

Since the privacy enhanced double decryption scheme is not based its security on p-DHP, we introduce a variant of p-DHP, denoted by t-DHP. Formal definition of t-DHP is given below.

Definition 6. *(t-DHP) The t-DHP is defined as follows: Given a set $\mathbb{G}_p$, an element g of $\mathbb{G}_p$ and ($g^a \mod n$, $g^b \mod n$) for $a, b \in [1, 2^t - 1]$ where $t > k$, find $g^{ab} \mod n$ where $n = p^2q$.*

Conjecture 2. For every probabilistic polynomial time algorithm $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for sufficiently large t and k,

$$\Pr\left[\mathcal{A}(n, A, B) = C \;\middle|\; \begin{array}{ccc} p, q \leftarrow \mathcal{P}(k); & & n = p^2q; \\ g \leftarrow \mathbb{G}_p; & & a, b \leftarrow_R [1, 2^t - 1]; \\ A = g^a \mod n; & B = g^b \mod n; & \\ C = g^{ab} \mod n; & & \end{array}\right] \leq negl(t, k).$$

Intuitively, we can say that the t-DHP is harder than p-DHP, since the exponent of t-DHP is larger than that of p-DHP. Under the hardness of t-DHP, we can prove that the privacy enhanced double decryption scheme is intractable.

Theorem 4. *The privacy enhanced double decryption scheme is one-way if and only if the t-DHP is hard to solve.*

Proof. If the t-DHP is not hard, there exists a polynomial time algorithm $\mathcal{B}$ which solves the t-DHP. By using $\mathcal{B}$, we can construct a polynomial time algorithm $\mathcal{A}$ that breaks the one-wayness of the privacy enhanced scheme. Let (n, g, g^a) be the public key parameters and let $(A = g^r \mod n, B = g^{ar}m \mod n)$ be a challenge ciphertext. Then, $\mathcal{B}$ can compute $g^{ar} \mod n$ from the pair $(g^a \mod n, g^r \mod n)$, so $\mathcal{A}$ can compute the corresponding plaintext as following: $m = B/g^{ar} \mod n$.

Suppose that the scheme is not one-way. Then there exists an adversary $\mathcal{A}$ that recovers the plaintext for given ciphertext. By using $\mathcal{A}$, we can solve the t-DHP in polynomial time. Let $(g^a \mod n, g^b \mod n)$ be a challenge. Set (n, g, g^a) as the public key parameters. Then, compute $(A = g^b \mod n, B = g^k \mod n)$ for some $k \in \mathbb{Z}_n$ and set the pair as the ciphertext which corresponds to the public key parameters. In this case, $k = ab + k'$ for some k'. Then, the algorithm $\mathcal{A}$ returns $m = g^{k'} \mod n$ as the plaintext of the ciphertext $(A = g^b \mod n, B = g^k \mod n)$ since $g^k = g^{ab+k'} = g^{ab}g^{k'} = g^{ab}m \mod n$. We can solve the t-DHP by computing $g^{ab} = g^k/m \mod n$ with help of $\mathcal{A}$. □

We have to consider the security of the privacy enhanced scheme against the authority since a malicious authority has more information than other adversary, the factoring of n. So, we define a variant of *t-DHP* named as *t_p-DHP* to formalize the security of privacy enhanced scheme against the authority.

Definition 7. *(t_p-DHP) Given a set $\mathbb{G}_p$, an element g of $\mathbb{G}_p$ and $(g^a \mod n, g^b \mod n)$ for $a, b \in [1, 2^t - 1]$ where $t > k$, compute $g^{ab} \mod n$ where $n = p^2q$ and the prime factors are known.*

We can simplify the t_p-DHP to show that the problem is sufficiently hard to solve against the authority. Consider a pair (g^a, g^b) where $g \in \mathbb{G}_p$ and $a, b \in [1, 2^t - 1]$. Let $a' = a \mod p$ and $b' = b \mod p$, then $a = a' + a''p$ and $b = b' + b''p$ for some integers a'', b''. Then, we can rewrite g^{ab} as following: $g^{ab} = g^{(a'+a''p)(b'+b''p)} = g^{a'b'+(a'b''+a''b')p+a''b''p^2} \mod n$. Note that the authority knows the factoring of n and so he can compute a' and b'. Then three values $g^{a'}$, $g^{b'}$ and $g^{a'b'}$ are easily computed. By using the values, $g^{a''p}$ and $g^{b''p}$ are computed as following: $g^a/g^{a'} = g^{a'+a''p}/g^{a'} = g^{a''p} \mod n$ and $g^b/g^{b'} = g^{b'+b''p}/g^{b'} = g^{b''p} \mod n$. Then $g^{(a'b''+a''b')p}$ is computed as following: $(g^{b''p})^{a'}(g^{a''p})^{b'} = g^{a'b''p}g^{a''b'p} = g^{(a'b''+a''b')p} \mod n$. The authority can computes $g^{a'b'}$ and $g^{(a'b''+a''b')p}$, so the following equation shows that the t_p-DHP is equal to the problem of solving the DHP for $(g^{a''p}, g^{b''p})$: $g^{ab}/g^{a'b'}g^{(a'b''+a''b')p} = g^{a''b''p^2} \mod n$. The factoring information permits the authority to compute partial information about the secret exponent when the modulo has the form of $n = p^2q$. However, if the exponent is a multiple of p, the authority can not find any information about the secret exponent. So, for the pair $(g^{a''p}, g^{b''p})$, he can not recover any information about the secret exponent, i.e., the authority can not solve the DHP for given (g^a, g^b) by computing the secret exponent, a and b.

Since the authority knows the factoring, he can reduce the problem on $\mathbb{Z}_n$ to the problem on a subgroup of $\mathbb{Z}_n$. As commented in [11], the hardness of DLP over $\mathbb{Z}_n$ is equals to the hardness of DLP over the subgroup of $\mathbb{Z}_n$. Similarly, the DHP over $\mathbb{Z}_n$ is equals to the problem of the DHP over $\mathbb{Z}_{p^2}$ and $\mathbb{Z}_q$ and we prove it in Theorem 5.

Lemma 1. *Let $n = p^2q$ where p, q are primes, then the following equation holds for some integer a: $(\alpha p^2 + \beta q)^a = (\alpha p^2)^a + (\beta q)^a \mod n$.*

Proof. Recall that, $(\alpha p^2 + \beta q)^a = \sum_{i=0}^{a} C_i (\alpha p^2)^i (\beta q)^{a-i} \mod n$. If $i \neq 0, a$ then $p^2 | (\alpha p^2)^i$ and $q | (\beta q)^{a-i}$, and so $(\alpha p^2)^i (\beta q)^{a-i} = 0 \mod n$. So, we have $(\alpha p^2 + \beta q)^a = {}_aC_a (\alpha p^2)^a (\beta q)^0 + {}_aC_0 (\alpha p^2)^0 (\beta q)^a = (\alpha p^2)^a + (\beta q)^a \mod n$. □

Lemma 2. *Let $n = p^2q$ where p, q are primes, then the following equations hold for some integer a: $(p^2(p^{-2} \mod q))^a = p^2(p^{-2} \mod q) \mod n$ and $(q(q^{-1} \mod p^2))^a = q(q^{-1} \mod p^2) \mod n$.*

Proof. Let $l = p^2(p^{-2} \mod q) \mod n$. It suffices to show that $l^2 = l \mod n$. Note that, $l^2 = l \Leftrightarrow l^2 - l = 0 \Leftrightarrow l(l-1) = 0 \mod n$. Since $l = p^2(p^{-2} \mod q) \mod n$, $l(l-1)$ can be expressed as following; $l(l-1) = (p^2(p^{-2} \mod q))(p^2(p^{-2} \mod q) - 1) \mod n$. Then $l(l-1) = 0 \mod n$ holds since $p^2(p^{-2} \mod q) = 0 \mod p^2$ and $p^2(p^{-2} \mod q) - 1 = 0 \mod q$. □

Theorem 5. *Suppose that the factoring of $n = p^2q$ is known. Then the DHP over $\mathbb{Z}_n$ is intractable if and only if the DHP over $\mathbb{Z}_{p^2}$ and $\mathbb{Z}_q$ are intractable.*

Proof. If there exists an algorithm $\mathcal{A}$ that solves the DHP over both $\mathbb{Z}_{p^2}$ and $\mathbb{Z}_q$, then we can solve the DHP over $\mathbb{Z}_n$ by using the algorithm. Let $(g^a \mod n, g^b \mod n)$ be a challenge. Since the factoring of n is known, we can compute $(g^a \mod p^2, g^b \mod p^2)$ and $(g^a \mod q, g^b \mod q)$ from given challenge. Then, algorithm $\mathcal{A}$ computes $g^{ab} \mod p^2$ and $g^{ab} \mod q$. Since $gcd(p^2, q) = 1$, we can compute $g^{ab} \mod n$ by using the Chinese Remainder Theorem.

Conversely, if there exist an algorithm $\mathcal{B}$ that solves the DHP over $\mathbb{Z}_n$, we can solve the DHP over $\mathbb{Z}_{p^2}$ and the DHP over $\mathbb{Z}_q$ by using the algorithm. Without lose of generality, suppose that $(g^a \mod p^2, g^b \mod p^2)$ is given as a challenge where $g \in \mathbb{Z}_{p^2}^*$ is the generator. Then we can make z, x and y as following:

$$\begin{aligned} z &= (g \mod p^2) q (q^{-1} \mod p^2) + p^2 (p^{-2} \mod q) \mod n, \\ x &= (g^a \mod p^2) q (q^{-1} \mod p^2) + p^2 (p^{-2} \mod q) \mod n, \\ y &= (g^b \mod p^2) q (q^{-1} \mod p^2) + p^2 (p^{-2} \mod q) \mod n. \end{aligned}$$

Then we compute $z^a \mod n$ and $z^b \mod n$ as following by Lemma 1 and Lemma 2:

$$\begin{aligned} z^a &= ((g \mod p^2) q (q^{-1} \mod p^2) + p^2 (p^{-2} \mod q))^a \mod n \\ &= ((g \mod p^2) q (q^{-1} \mod p^2))^a + (p^2 (p^{-2} \mod q))^a \mod n \\ &= (g \mod p^2)^a (q (q^{-1} \mod p^2))^a + (p^2 (p^{-2} \mod q))^a \mod n \\ &= (g \mod p^2)^a q (q^{-1} \mod p^2) + p^2 (p^{-2} \mod q) \mod n, \end{aligned}$$

$$\begin{aligned} z^b &= ((g \mod p^2)q(q^{-1} \mod p^2) + p^2(p^{-2} \mod q))^b \mod n \\ &= ((g \mod p^2)q(q^{-1} \mod p^2))^b + (p^2(p^{-2} \mod q))^b \mod n \\ &= (g \mod p^2)^b(q(q^{-1} \mod p^2))^b + (p^2(p^{-2} \mod q))^b \mod n \\ &= (g \mod p^2)^b q(q^{-1} \mod p^2) + p^2(p^{-2} \mod q) \mod n. \end{aligned}$$

Since $x = z^a \mod p^2$ and $x = z^a \mod q$, $x = z^a \mod n$. Similarly, $y = z^b \mod n$. We compute $z^{ab} \mod n$ by using algorithm $\mathcal{B}$ for $(z^a \mod n, z^b \mod n)$ where z is used as a generator. Then, $g^{ab} \mod p^2$ is computed as $(z^{ab} \mod n) \mod p^2 = g^{ab} \mod p^2$. We can solve the DHP over $\mathbb{Z}_q$ in the same way. □

The security of the t_p-DHP is equal to the security of the DHP over $\mathbb{Z}_{p^2}$ and $\mathbb{Z}_q$. However, if the difference between t and k is small, the t_p-DHP is not secure against the authority, since the authority can recover a' for given $g^a \mod n$ where $a = a' + a''p$ and so the remained secret information is $t - k$ bit integer a''. Hence, when $t - k$ is small, the authority can find a'' by brute-forcing. If we choose large t to make it hard to guessing a'' then the t_p-DHP is sufficiently hard to solve against the authority.

Conjecture 3. For every probabilistic polynomial time algorithm $\mathcal{A}$, there exists a negligible function $negl(\cdot)$ such that for sufficiently large t and k,

$$\Pr\left[\mathcal{A}(p,q,A,B) = C \;\middle|\; \begin{array}{cc} p,q \leftarrow \mathcal{P}(k); & n = p^2q; \\ g \leftarrow \mathbb{G}_p; & a,b \leftarrow_R [1, 2^t - 1]; \\ A = g^a \mod n; & B = g^b \mod n; \\ C = g^{ab} \mod n; & \end{array}\right] \leq negl(t,k).$$

Under the hardness of t_p-DHP, we can prove that the privacy enhanced double decryption scheme is secure against a malicious authority. Note that, to achieve sufficient security against the authority, we should use large k than the scheme proposed in Section 3. The proof of Theorem 6 is same to Theorem 4, except the hard problem, the t_p-DHP.

Theorem 6. *The privacy enhanced double decryption scheme is one-way against a malicious authority if and only if the t_p-DHP is hard to solve.*

The one-wayness of privacy enhanced scheme against the ordinary adversary and the authority is based on the t-DHP and the t_p-DHP, respectively. The t-DHP and the t_p-DHP are also YCP, so we can use the general conversion method proposed in [9] to achieve the semantic security against adaptive chosen ciphertext attack. The converted scheme of privacy enhanced scheme is as following: $\mathcal{E}_{pk}(m,r) = (g^{H(m||r)} \mod n, E_\kappa(m||r))$ where E is symmetric encryption function, H and G are two hash functions, and $\kappa = G(h^{H(m||r)} \mod n)$. Semantic security of the converted scheme is proved similar to Theorem 3.

Theorem 7. *In the random oracle model, the converted scheme is IND-CCA2 secure if the computational t-DHP is intractable and the symmetric encryption scheme E is OTE secure. Especially, the scheme is IND-CCA2 secure against the authority if the t_p-DHP is intractable.*

Remark 6. Since the authority has the factoring information, it looks like that the semantic security of the privacy enhanced scheme can be defeated by the author-

ity. However, the previous conversion method guarantee the semantic security of a scheme if the one-wayness of the scheme is based on a kind of YCP. So, the privacy enhanced double decryption scheme is secure against IND-CCA2 adversary.

Since the privacy enhanced scheme achieves the one-wayness and the semantic security against the authority, the scheme is enhanced in the privacy point of view. If an user want to get rid of the apprehension of invasion of privacy, he will renounce the property, double decryption mechanism. However, he can choose whether to use the property or not. The property is not duty anymore in our scheme. So, we say that our scheme is enhanced in the privacy point of view rather perfectly secure against the authority.

Obviously, the security of the privacy enhanced doubled decryption scheme against the authority differs from the other adversaries. However, by choosing sufficiently large k and t, we can make the scheme achieve enough security against both the authority and ordinary adversaries.

5 Conclusion

In this paper, we have proposed an efficient public key cryptosystem with a double decryption mechanism and a modification that offers to an user more higher privacy than the previous double decryption scheme. Compared with [1], our schemes have the following advantages:

1. Efficiency: The length of ciphertext is shorter than that of the previous scheme in the same security level. Moreover, the encryption and decryption of our scheme is faster than those of the previous scheme about 3 times. Especially, the authority's decryption is faster about 4 times than that of the previous scheme.
2. Security (Privacy against the authority): The privacy enhanced double decryption scheme is secure against the authority who knows the factoring of n. In the previous scheme, the authority can recover any ciphertext by using the factoring information. However, in our scheme, the authority's excessive ability is restricted.

However, the efficiency of the basic double decryption scheme that we proposed is better than that of the previous scheme, but the privacy enhanced double decryption scheme is not efficient than that of the previous scheme. So, it is an open problem to design a double decryption scheme that raises the efficiency and enhances the privacy against the authority simultaneously.

References

1. Emmanuel Bresson, Dario Catalano, and David Pointcheval, *A Simple Public-Key Cryptosystem with a Double Trapdoor Decryption Mechanism and Its Applications*, ASIACRYPT 2003, LNCS 2894, pp. 37-54, Springer-Verlag, 2003.
2. Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway, *Relations Among Notions of Security for Public-Key Encryption Schemes*, CRYPTO'98, LNCS 1462, pp. 26-46, Springer-Verlag, 1998.

3. Joonsang Baek, Byoungcheon Lee, and Kwangjo Kim, *Provably Secure Length-Saving Public-Key Encryption Scheme under the Computational Diffie-Hellman Assumption*, ETRI Journal, Volume 22, Number 4, December 2000.
4. Ronald Cramer, and Victor Shoup, *Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption*, EUROCRYPT 2002, LNCS 2332, pp. 45-64, Springer-Verlag, 2002.
5. D. Dolev, C. Dwork, and M. Naor, *Non-malleable cryptography*, Proceedings of the 23rd Annual Symposium on Theory of Computing, ACM, 1991.
6. W. Diffie, and M. E. Hellman, *New Directions in Cryptography*, IEEE Transactions on Informaion Theory, 22(6), 644-654, 1976.
7. Eiichiro Fujisaki, and Tatsuaki Okamoto, *How to Enhance the Security of Public-Key Encryption at Minimum Cost*, PKC'99, LNCS 1560, pp. 53-68, 1999.
8. S. Goldwasser, and S. Micali, *Probabilistic encryption*, Journal of Computer and System Science, Vol.28, No.2, pp.270-299, 1984.
9. Eike Kiltz and John Malone-Lee, *A General Construction of IND-CCA2 Secure Public Key Encryption*, Cryptography and Coding 2003, LNCS 2898, pp. 152-166, 2003.
10. Wenbo Mao, and Chae Hoon Lim, *Cryptanalysis in Prime Order Subgroups of* Z_n^*, ASIACRYPT'98, LNCS 1514, pp. 214-226, 1998.
11. A.J. Menezes, P.C. Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Inc, (1999).
12. Tatsuaki Okamoto, Shigenori Uchiyama, *A New Public-Key Cryptosystem as Secure as Factoring*, EUROCRYPT 98, LNCS 1403, pp. 308-318, Springer-Verlag, 1998.
13. Pascal Paillier, *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*, EUROCRYPT'99, LNCS 1592, pp. 223-238, Springer-Verlag, 1999.
14. Rene Peralta, *Report on Integer Factorization*, available at http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1025_report.pdf, 2001.
15. David Pointcheval, *Chosen-Ciphertext Security for any One-Way Cryptosystem*, Proceedings of PKC'2000, LNCS 1751, pp. 129-146, 2000.

On the (Im)Possibility of Practical and Secure Nonlinear Filters and Combiners*

An Braeken and Joseph Lano

Katholieke Universiteit Leuven, Dept. Elect. Eng.-ESAT/SCD-COSIC,
Kasteelpark Arenberg 10, 3001 Heverlee, Belgium
{an.braeken, joseph.lano}@esat.kuleuven.be

Abstract. A vast amount of literature on stream ciphers is directed to the cryptanalysis of LFSR-based filters and combiners, resulting in various cryptanalytic attacks. In this paper, we present a unified framework for the security of a design against these attacks based on the properties of the LFSR(s) and the Boolean function used. It is explained why building nonlinear filters seems more practical than building nonlinear combiners. We also investigate concrete building blocks that offer a good trade-off in their resistance against these various attacks, and can at the same time be used to build a low-cost synchronous stream cipher for hardware applications.

Keywords: Combination and filter generator, distinguishing attack, correlation attack, algebraic attack, hardware complexity.

1 Introduction

For efficient encryption of data, cryptography mainly uses two types of symmetric algorithms, block ciphers and stream ciphers. In the past decades, block ciphers have become the most widely used technology. However, as block ciphers are often used in a stream cipher mode such as CTR and OFB, stream ciphers may offer equivalent security at a lower cost.

Designing a secure stream cipher appears to be a hard task. In the NESSIE competition, flaws have been found in all candidates. In fact, unexpected biases are often detected in designs, especially if the design is based on relatively new concepts or uses large vectorial Boolean functions of which it is impossible to calculate all biases and correlations beforehand.

By far the most studied designs are the nonlinear filter and combiner generators, which are based on LFSRs in conjunction with a Boolean function. So far, new developments were mostly restricted to the development of separate attacks which were then applied to a design that is particularly vulnerable to the

* This work was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and by the European Commission through the IST Programme under Contract IST2002507932 ECRYPT. An Braeken is a F.W.O. Research Assistant, sponsored by the Fund for Scientific Research - Flanders (Belgium), Joseph Lano is financed by a Ph.D. grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 159–174, 2006.

new attack. Little attention has been given to the cryptographic design requirements that result from these attacks combined. The recent ECRYPT eSTREAM project motivates us to investigate this further.

Although at first sight the vast amount of research on nonlinear filters and combiners seems to put these designs at a disadvantage compared to newer design principles, we believe that they can also benefit from the accumulated knowledge of their underlying mathematics. In fact, if one can develop a stream cipher that is resistant to all these attacks combined and is still easily implementable in hardware, confidence in this classical and transparent design can be higher (of course thorough public evaluation will still be necessary) and hence actual application can be faster.

In this paper, we study the cryptographic design requirements for filters and combiners. In Sect. 3, we study the impact of the most important attacks (distinguishing attacks, correlation attacks and algebraic attacks) on the building blocks. By analyzing building blocks that offer optimal resistance against certain attacks, we establish minimal requirements for the internal state size, the LFSR polynomials and the properties of the Boolean function (number of inputs, Walsh transform, degree, nonlinearity, algebraic immunity, ...). This analysis allows to establish design criteria for filters and combiners respectively. We then study, in Sect. 4, some Boolean functions such as power functions and symmetric functions, which can be interesting design choices as they have some desirable properties and are easy to implement in hardware.

Not all our results could be presented in this extended abstract. For an extended version of this paper containing more explanations, tables, and proofs, please refer to [5].

2 Preliminaries

The two basic models of key stream generators are the combination and the filter generator. A *combination generator* uses several LFSRs in parallel and combines their outputs in a nonlinear way (by means of the combining function). If the output is computed by a nonlinear function (filter function) of some taps of one LFSR, a *filter generator* is obtained. In this section, we list some properties of the two building blocks that are used in these generators, namely LFSRs and Boolean functions.

2.1 Linear Feedback Shift Registers

Definition 1. *A* Linear Feedback Shift Register (LFSR) *of length L is a collection of L 1-bit memory elements $s_t^0, s_t^1, \ldots, s_t^{L-1}$. At each time t the memory is updated as follows:*

$$\begin{cases} s_t^i = s_{t-1}^{i+1} \text{ for } i = 0, \ldots, L-2 \\ s_t^{L-1} = \bigoplus_{i=1}^{L} c_i \cdot s_{t-1}^{L-i}. \end{cases} \tag{1}$$

where the c_i are fixed binary coefficients that define the feedback equation of the LFSR. The LFSR stream $(s_t)_{t \geq 0}$ consists of the successive values in the memory element s_0.

Associated with an L-bit LFSR is its *feedback polynomial P(X)* of degree d, $P(X) = 1 + \sum_{i=1}^{L} c_i \cdot X^i$. The *weight* of the feedback polynomial is equal to its number of nonzero terms. In practical designs, a feedback polynomial is chosen to be *primitive*. This implies that every nonzero initial state produces an output sequence with maximal period $2^L - 1$, which is also called a *pn*-sequence.

For many cryptanalytic attacks, it is useful to search *low-weight multiples* of the feedback polynomial $P(x)$, see [6, 19]. The number $m(D, w)$ of multiples $Q(X) = 1 + \sum_{i=1}^{D} c_i \cdot X^i$ of the polynomial $P(X)$, with degree less than or equal to D and with weight w, can be approximated by [6]:

$$m(D, w) \approx \frac{D^{w-1}}{(w-1)! \cdot 2^L} \,. \tag{2}$$

It is interesting to know from which D_{min} we can expect a first multiple $Q(X)$ of weight w to start appearing. It follows from (2) that:

$$D_{min}(w) \approx (2^L \cdot (w-1)!)^{\frac{1}{w-1}} \,. \tag{3}$$

The most efficient approach, known to date, to search for these low-weight multiples is a birthday-like approach, see [19]. The *precomputation complexity* P needed to find all multiples $Q(X)$ of weight w and degree at most D can be approximated by:

$$P(D, w) \approx \frac{D^{\lceil \frac{w-1}{2} \rceil}}{\lceil \frac{w-1}{2} \rceil !} \,. \tag{4}$$

2.2 Boolean Functions

A *Boolean function* f is a mapping from $\mathbb{F}_2^\varphi$ into $\mathbb{F}_2$. The *support* of f is defined as $\sup(f) = \{\overline{x} \in \mathbb{F}_2^\varphi : f(\overline{x}) = 1\}$. The cardinality of $\sup(f)$ represents the *weight* $\mathrm{wt}(f)$ of the function.

A Boolean function can be uniquely represented by means of its *algebraic normal form (ANF)*:

$$f(\overline{x}) = f(x_0, \ldots x_{\varphi-1}) = \bigoplus_{(a_0,\ldots,a_{\varphi-1}) \in \mathbb{F}_2^\varphi} h(a_0, \ldots, a_{\varphi-1})\, x_0^{a_0} \ldots x_{\varphi-1}^{a_{\varphi-1}} \,, \tag{5}$$

where f and h are Boolean functions on $\mathbb{F}_2^\varphi$. The *algebraic degree* of f, denoted by $\deg(f)$, is defined as the highest number of variables in the terms $x_0^{a_0} \ldots x_{\varphi-1}^{a_{\varphi-1}}$ in the ANF of f.

Alternatively, a Boolean function can be represented by its *Walsh spectrum*:

$$W_f(\overline{\omega}) = \sum_{\overline{x} \in \mathbb{F}_2^\varphi} (-1)^{f(\overline{x}) \oplus \overline{x} \cdot \overline{\omega}} = 2^{\varphi-1} - 2wt(f \oplus \overline{x} \cdot \overline{\omega}) \,, \tag{6}$$

where $\overline{x} \cdot \overline{\omega} = x_0\omega_0 \oplus x_1\omega_1 \oplus \cdots \oplus x_{\varphi-1}\omega_{\varphi-1}$ is the *dot product* of $\overline{x}$ and $\overline{\omega}$. We will use the following two well-known formulae for the Walsh values:

$$\begin{cases} \sum_{\overline{\omega} \in \mathbb{F}_2^\varphi} W_f(\overline{\omega}) = \pm 2^\varphi \\ \sum_{\overline{\omega} \in \mathbb{F}_2^\varphi} W_f^2(\overline{\omega}) = 2^{2 \cdot \varphi} \,, \end{cases} \tag{7}$$

where the second equality is known as Parseval's theorem.

Several properties are of importance for Boolean functions from a cryptographic viewpoint. A function is said to be *balanced* if $\mathrm{wt}(f) = 2^{\varphi-1}$ and thus $W_f(\overline{0}) = 0$. The *nonlinearity* N_f of the function f is defined as the minimum distance between f and any affine function; it can be calculated as $N_f = 2^{\varphi-1} - \frac{1}{2}\max_{\overline{\omega}\in\mathbb{F}_2^n} |W_f(\overline{\omega})|$. The *best affine approximation* $l(\overline{x})$ is associated with this notion. We will say that f has *bias* ϵ if it has the same output as its best affine approximation with probability $0.5 + \epsilon$. It is easy to see that $\epsilon = N_f/2^{\varphi} - 0.5 = \frac{\max_{\overline{\omega}\in\mathbb{F}_2^n} |W_f(\overline{\omega})|}{2^{\varphi+1}}$. A function f is said to be *correlation-immune* [34] of order ρ, $CI(\rho)$, if and only if its Walsh transform W_f satisfies $W_f(\overline{\omega}) = 0$, for $1 \leq wt(\overline{\omega}) \leq \rho$. If the function is also balanced, then the function is called ρ-*resilient*. Two important bounds hold for the bias ϵ:

$$\begin{cases} \epsilon \geq 2^{-\varphi/2-1} \\ \epsilon \geq 2^{\rho+1-\varphi}, \end{cases} \tag{8}$$

where the first bound is due to Parseval's theorem and equality holds only for *bent* functions; the second bound reflects the trade-off between resiliency and nonlinearity.

The lowest degree of the function g from $\mathbb{F}_2^{\varphi}$ into $\mathbb{F}_2$ for which $f \cdot g = \overline{0}$ or $(f \oplus \overline{1}) \cdot g = \overline{0}$ is called the *algebraic immunity (AI)* of the function f [24]. The function g is said to be an *annihilator* of f if $f \cdot g = \overline{0}$. It has been shown [11] that any function f with φ inputs has algebraic immunity at most $\lceil \frac{\varphi}{2} \rceil$.

A *vectorial Boolean function* F from $\mathbb{F}_2^n$ into $\mathbb{F}_2^m$, also called (n, m) S-box, can be represented by an m-tuple $(f_1, \ldots, f_m)$ of Boolean functions f_i on $\mathbb{F}_2^n$ (corresponding to the output bits).

3 Security Analysis

During the last two decades, several classes of attacks have been proposed on the filter and combination generator. In this section, we will thoroughly investigate these different attacks and will derive minimal requirements that the LFSRs and Boolean functions should satisfy. Our goal is to investigate whether it is possible to construct practical and secure filter or combination generators with 80-bit key security and low hardware cost, which implies that we should keep it close to the edge of the minimal requirements while at the same time keeping a reasonable security margin.

For most attacks, our analysis reflects the currently known attacks described in the literature, but we now relate these attacks directly to the mathematical properties of the concrete building blocks used. Our treatment of distinguishing attacks combines and extends the ideas of the recent work done in [25, 17] to concrete distinguishing attacks on all filter and combination generators. It follows that distinguishing attacks are very similar to correlation attacks but are often stronger, as they can use many linear approximations simultaneously and do not need a decoding algorithm. Note also that resynchronization mechanisms are not discussed in this paper. A secure resynchronization mechanism of the scheme is also necessary. We refer to [12, 2] for more details concerning resynchronization attacks.

3.1 Tradeoff Attacks

Time-Memory-Data Tradeoff attacks [3] are generic attacks against stream ciphers. To prevent these attacks, the internal state should be at least twice the key size. Consequently, with an 80-bit key, the LFSR has at least a length of 160 bits. In the following, we will investigate the security of filter and combination generator with an internal state of 256 bits, and thus taking a sufficient security margin. This allows us to quantify our analysis, but of course it is easy to adapt the framework to other security parameters.

3.2 Berlekamp-Massey Attacks

The linear complexity of a bit stream $(s_t)_{t\geq 0}$ is equal to the length of the shortest LFSR generating that stream. For a Boolean function of degree d, the linear complexity LC of the resulting key stream generated by a filter generator is upper bounded by $\sum_{i=0}^{d} \binom{L}{i}$. Moreover, it is very likely that the LC of the key stream is lower bounded by $\binom{L}{d}$ and that its period remains equal to 2^L-1. If the constituent LFSRs of the combination generator have distinct degrees greater than 2 and initial state different from 0, then the LC of the key stream generated by a combination is equal to $f(L_1,\ldots,L_n)$, where the ANF of f is evaluated over the integers. We refer to [33, 22] for more details.

The Berlekamp-Massey attack requires $2\cdot LC$ data and has complexity of LC^2. For a key stream generator with internal size equal to 256 and a Boolean function of sufficiently high degree, this attack is clearly of no concern. A degree-7 function will be sufficient for a nonlinear filter. For combiners the calculation is more complex as it depends on the size of the LFSRs and on the ANF of the Boolean functions, but also here we start having resistance against the Berlekamp-Massey attack from degree 7.

3.3 Distinguishing Attacks

The distinguishing attack we describe here is based on the framework developed in [17] combined with the mathematical results from [25]. We extend the framework for the filter generator and also develop the attack for the combiner generator.

Filter Generator. The idea of the attack is the following. The LFSR stream has very good statistical properties, but of course there are linear relations, which are determined by the feedback polynomial $P(X)$ and its multiples $Q(X)$, where the cryptanalyst first needs to find the latter in a precomputation step.

Given a linear relation $Q(X)$ of weight w in the LFSR stream, the same relation for the corresponding bits of the key stream will hold with some probability different from one half, because the Boolean function f does not destroy all linear properties. This probability is determined by the combined action of all affine approximations of the Boolean function. This interaction is governed by the piling-up lemma and can be expressed as [25]:

$$\varepsilon' = \frac{\sum_{\overline{\omega}=0}^{2^{\varphi}-1} (W_f(\overline{\omega}))^w}{2^{\varphi\cdot w+1}}\,. \tag{9}$$

To distinguish the key stream from random, the number of samples needed is in the order of $\frac{1}{\varepsilon'^2}$, which is also the time complexity of the attack. The data complexity is the sum of the degree of the relation $Q(X)$ and the number of samples needed. It is possible to decrease the data complexity to some extent by using multiple relations simultaneously.

We now study the impact of this attack on the bent functions, as these offer the best resistance against this distinguishing attack. We assume our LFSR has an internal state of 256 bits and investigate the data complexity of the attack as a function of the number of inputs. By combining (7), (8) and (9), we can calculate that the bias for bent functions can be written as:

$$\varepsilon' = 2^{-(\lceil w/2 \rceil - 1)\cdot\varphi - 1}. \tag{10}$$

A cryptanalyst is interested in finding the weight w for which the attack complexity is minimal. For very low weight w, the degree of $Q(X)$ will be prohibitively high as shown by (3). For very high weight w, the bias (10) will be too small. We hence expect an optimal tradeoff somewhere in between.

We have calculated these numbers, and it turns out that no 256-bit LFSR with a Boolean function with less than 20 inputs can be made secure against this distinguishing attack! However, we have to make two important observations:

- The precomputation time to find the necessary multiples $Q(X)$ is very high (in the order of 2^{150}, governed by (3) and (4). A discussion on the amount of precomputation we can allow is an important topic of discussion. Note that this also applies to other attacks such as trade-off attacks, correlation attacks and algebraic attacks.
- There has been some debate on the relevance of distinguishing attacks requiring long key streams during NESSIE [32]. Whereas time complexity is only a matter of resources at the attacker's side, available data depends on what has been encrypted by the sending party. Hence, we propose to limit the maximum amount of key stream generated from a single key/iv pair to 2^{40} bits (practical assumption), after which the scheme should resynchronize. This measure prevents the appearance of low-weight multiples: from (3) it follows that normally no multiples of weight less than 8 exist with degree less than 2^{40}. Now, we can recalculate the best attack complexities, by adding the extra constraint $\log_2(D_{min}) < 40$. It follows that, under this practical restriction, nonlinear filters can be built which are secure against distinguishing attacks starting from 14 inputs. Note that the restriction of a single key stream to 2^{40} bits also provides some protection against other cryptanalytic attacks, as explained below.

Combination Generator. For combination generators, the attack can be improved by using a divide and conquer approach. Let us assume we have a combiner with φ LFSRs and that the Boolean function has resiliency ρ. The average length of one LFSR is thus $\frac{256}{\varphi}$. The attacker now mounts the same distinguishing attack, restricting himself to r LFSRs, where r must be of course strictly

greater than the order of resiliency ρ. Again, we first search for a low-weight multiple of this equivalent $\frac{256 \cdot r}{\varphi}$-length LFSR, and then try to detect the bias that remains of this relation after the Boolean function. From the piling-up lemma, it follows that this bias is as follows:

$$\varepsilon' = \frac{\sum_{\overline{\omega} \in S} (W_f(\overline{\omega}))^w}{2^{\varphi \cdot w + 1}}, \tag{11}$$

where the set S is defined as:

$$S = \{\overline{\omega} | wt(\overline{\omega}) > \rho \text{ and } \overline{\omega} < 2^r\}, \tag{12}$$

assuming, without loss of generality, that the attacked LFSRs are numbered $0, 1, \ldots r-1$. This divide and conquer approach gives the attacker some advantages compared to the case of the nonlinear filter:

- If the resiliency of the Boolean function is low, the length of the attacked equivalent LFSR can be low. First, this will allow the attacker to perform the precomputation step governed by (3) and (4) much faster. Second, the length of the low-weight multiples will be much lower, making the data complexities feasible for much lower weights, where the detectable biases will be much larger as shown by (11). Note that when $\frac{256 \cdot r}{\varphi}$ is very small, we will even be able to mount a powerful weight-2 attack without precomputation step: this will just correspond to the period of the small equivalent LFSR. As shown in [17], such an attack can be easily turned into a key recovery attack.
- If the resiliency of the Boolean function is high, we will still have the advantages explained in the above point, but to a lesser extent. But here the tradeoff between resiliency and nonlinearity (8) will come into play, resulting in higher Walsh values in (11) and hence a higher bias.

It follows that it is much harder to make the combiner model resistant to this distinguishing attack. We have again calculated the best resistance that can be achieved, and it turns out that 36 inputs and an optimally chosen Boolean function would be needed to resist the attack.

Again, we propose to limit the maximum amount of key stream obtained from a single key/iv pair to 2^{40} bits. After recalculating the complexities with this restriction, we now see that, under ideal assumptions for the cryptographer, he can make a combiner which is secure starting from 18 inputs.

It is important to note that these lower bounds will be very hard to approach in reality due to the difficulty of finding practical functions that have properties close to the optimal case described here, and due to the fact that our lower bounds consider equal LFSR lengths. In reality all LFSR lengths need to be distinct, which will allow the attack to improve his attack significantly by attacking the shortest LFSRs. The larger the number LFSRs, the more serious this problem becomes. As a result of this, we believe it is not possible to design a practical nonlinear combiner, with 256 bits internal state, that can resist to this distinguishing attack. This in comparison to the case of the filter generator, where we can succeed to get close to the bounds with actually implementable functions, as evidenced by the power functions described in Sect. 4. We will now develop a similar reasoning for the case of fast correlation attacks.

3.4 Fast Correlation Attacks

Correlation and fast correlation attacks exploit the correlation between the LFSR stream s_t and the key stream z_t. These attacks can be seen as a decoding problem, since the key stream z_t can be considered as the transmitted LFSR stream s_t through a binary symmetric channel with error probability $p = P_{t\geq 0}(z_t \neq s_t)$. For the nonlinear filter generator, p is determined by $0.5+\epsilon$. For the combination generator, a divide-and-conquer attack as in the case of a distinguishing attack is possible. Consequently, it is sufficient to restrict the attack to a subset of LFSRs $\{i_0, \ldots, i_\rho\}$, such that the following holds:

$$P(f(\overline{x}) \neq 0 | x_{i_0} = a_0, \ldots, x_{i_\rho} = a_\rho, \forall (a_0, \ldots, a_\rho) \in \mathbb{F}_2^{\rho+1}) \neq 1/2\,. \tag{13}$$

Here, as defined above, the parameter ρ corresponds with the order of resiliency of the Boolean function.

Then, the attack consists of a fast decoding method for any LFSR code C of length N (the amount of available key stream) and dimension L (the length of the LFSR), where the length N of the code is lower bounded by Shannon's channel coding theorem:

$$N \geq \frac{L}{C(p)} = \frac{L}{1 + p\log_2 p + (1-p)\log_2(1-p)} \approx \frac{\ln(2)L}{2\epsilon^2}\,. \tag{14}$$

If we want to achieve perfect security against this attack for a 256-bit LFSR and allowing at most 2^{40} bits in a single key stream, the bias ϵ should be less than or equal to 2^{-17}. To achieve this, we would need a highly nonlinear Boolean functions with more than 34 inputs. This can never be implemented efficiently in hardware. However, the above criterion is far too stringent as actual decoding algorithms are not able to do this decoding with a good time complexity. We now look at the current complexities of these decoding algorithms. Besides the maximum-likelihood (ML) decoding, which has very high complexity of $L \cdot 2^L$, mainly two different approaches have been proposed in the literature. In the first approach, the existence of sparse parity check equations for the LFSR code are exploited. These parity check equations correspond with the low weight multiples of the connection polynomial. In this way, the LFSR code can be seen as a low-density parity-check (LDPC) code and has several efficient iterative decoding algorithms. In the second approach, a smaller linear $[n, l]$ code with $l < L$ and $n > N$ is associated to the LFSR on which ML decoding is performed. The complexity of both approaches highly depends on the existence of low degree multiples. As shown above, the precomputation time for finding these low degree multiples is very high. Moreover, the approach requires long key streams and suffers from a high decoding complexity.

Note the very strong resemblance between this classical framework for the correlation attacks and the framework we developed in the previous subsection for distinguishing attacks. The main difference between the two is that in the correlation attacks we need a decoding method, whereas in the distinguishing attacks we just need to apply a simple distinguisher. Analysis of the above formulae learns that the complexity of the decoding procedure is much higher than

for the distinguishing procedure. Even with huge improvements of the existing decoding procedures, we do not expect this situation to change. Hence we can conclude that a choice of parameters that makes the design resistant against distinguishing attacks (explained above), will also make it resistant against correlation attacks.

3.5 Algebraic Attacks

In *algebraic attacks* [11], a system of nonlinear equations between input and output is constructed and subsequently solved. The complexity of solving this system of equations highly depends on the degree of these equations. In the usual algebraic attack, equations between one bit of the output of the filter or combination generator and the initial state of the LFSR are searched. These equations are then solved by linearization. The lowest possible degree d of these equations, also called the *Algebraic Immunity (AI)*, is obtained by the annihilators of the filter or combination function and its complement. The total complexity $C(L, d)$ of the algebraic attack on a stream cipher with a linear state of L bits and equations of degree d is then determined by

$$C(L,d) = \left(\sum_{i=0}^{d} \binom{L}{i}\right)^{\omega} = D^{\omega}\,, \tag{15}$$

where ω corresponds to the coefficient of the most efficient solution method for the linear system. We use here Strassen's exponent [35] which is $\omega = \log_2(7) \approx 2.807$. Clearly, the number of required key stream bits is equal to D. Note that in the complexity analysis, the linearization method is used for solving the equations. It is an open question if other algorithms like the Buchberger algorithm, F4 or F5 [18] can significantly improve this complexity. Also, the total number of terms of degree less than or equal to d is considered in the complexity, while in general nothing is known about the proportion of monomials of degree d that appear in the system of equations. Therefore, a sufficient security margin should be taken into account.

It can be calculated that an AI of 5 is currently sufficient to withstand standard algebraic attacks in our framework, for an LFSR of length 256 and aiming at 80-bit security.

The implications for Boolean functions are the following. It has been shown [11] that the AI of a Boolean function with φ inputs can be at most $\lceil \frac{\varphi}{2} \rceil$. Our Boolean function hence needs to have at least 9 inputs. We will of course need to check that the AI of the Boolean function is large enough. The complexity of the algorithm to check if there are equations of degree less than or equal to d (corresponding to AI equal to d) is slightly better than $\binom{\varphi}{d}^{\omega}$ [24], which is feasible for most practical functions of interest here.

Fast algebraic attacks can be much more efficient than the usual algebraic attacks. In the fast algebraic attacks [9], the attacker tries to decrease the degree d of the system of equations even further by searching for relations between the initial state of the LFSR and several bits of the output function simultaneously.

The equations where the highest degree terms do not involve the key stream bits are considered. This is done in an additional precomputation step (complexity $D \cdot \log^2 D$ [20]): linear combinations of the equations are searched which cancel out the highest degree terms in order to obtain equations of degree e. The complexity for solving these equations is now $C(L, e)$. As shown in [20], we have to take into account another complexity as well, namely that of substituting the key stream into the system of equations. This complexity can be approximated by $2 \cdot E \cdot D \cdot \log D$. The required key stream length is equal to $D + E - 1 \approx D$ because $D >> E$. If we have found C such relations of degree e, we can further reduce the data complexity by a factor C, but at the cost of increasing the substitution complexity by this same factor C because we have to repeat the substitution step C times.

The problem with fast algebraic attacks is that, for the moment, very little is known about whether these reductions are possible or not for some Boolean function. The only way we can be sure is by searching for the existence of these relations, but the complexity of this step may become prohibitively high.

Another approach to achieve resistance against fast algebraic attack, without needing this precomputation, is by choosing a very high AI. In this case either the complexity of the substitution step or the complexity of the solving step will become prohibitively high. It can be calculated that we achieve such security starting from an AI of 12. When restricting the amount of key stream bits to 2^{40}, the AI of the function should be greater than or equal to 9.

It should be stressed that not much is known about the strength and the implementation of fast algebraic attacks, and it is hence probable that the algorithms can be improved. The further investigation of fast algebraic attacks is an interesting research topic, and designers are advised to take a reasonable security margin, as evidenced by [10].

4 Boolean Functions for the Filter Generator

From the analysis in the previous section, it follows that the two most important properties a good Boolean function should have are high algebraic immunity and high nonlinearity. Besides, to be used in practice, they should be implementable in hardware at a very low cost. We will now present two classes of functions that have a low-cost hardware implementation.

4.1 Symmetric Functions

Symmetric functions [7] are functions with the very interesting property that their hardware complexity is linear in the number of variables. A symmetric function is a function of which the output is completely determined by the weight of the input vector. Therefore, the truth table $v_f = (v_0, \ldots, v_\varphi)$, also called *value vector*, of the symmetric function f on $\mathbb{F}_2^\varphi$ reduces to a vector of length $\varphi + 1$, corresponding with the function values v_i of the vectors of weight i with $0 \leq i \leq \varphi$. We have identified the following class of symmetric functions with maximum AI:

Theorem 2. *The symmetric function in $\mathbb{F}_2^\varphi$ with value vector*

$$v_f(i) = \begin{cases} 0 \text{ for } i < \lceil \frac{\varphi}{2} \rceil \\ 1 \text{ else} \end{cases} \tag{16}$$

has maximum AI. Let us denote this function by F_k where k is equal to the threshold $\lceil \frac{\varphi}{2} \rceil$.

By Proposition 2 and Proposition 4 of [7], the degree of these functions are determined as follows.

Theorem 3. *The degree of the symmetric function $F_{\lceil \frac{\varphi}{2} \rceil}$ on $\mathbb{F}_2^\varphi$ is equal to $2^{\lfloor \log_2 \varphi \rfloor}$.*

If φ is odd, these functions are trivially balanced because $v_f(i) = v_f(\varphi - i)$ for $0 \leq i \leq \lfloor \frac{\varphi}{2} \rfloor$. As shown in Proposition 5 of [7], trivially balanced functions satisfy the following properties: The derivative $D_{\overline{1}}f$ with respect to $\overline{1}$ is the constant function, and $W_f(\overline{x}) = 0$ for all $\overline{x}$ with $\mathrm{wt}(\overline{x})$ even. For φ even, the functions are not balanced. But by XORing with an extra input variable from the LFSR, this property is immediately obtained.

Another problem is that the nonlinearity of these functions is not high. In particular, $\max_{\overline{w} \in \mathbb{F}_2^\varphi} |W_f(\overline{w})| = 2\binom{\varphi-1}{\frac{\varphi-1}{2}}$ for odd φ and equal to $\binom{\varphi}{\frac{\varphi}{2}}$ for even φ. Therefore, $\epsilon \approx 2^{-3.15}, 2^{-3.26}, 2^{-3.348}$ for $\varphi = 13, 15, 17$ respectively. Note that the nonlinearity increases very slowly with the number of inputs φ: even for 255 input bits, we only get a bias of $2^{-5.33}$.

Remark 1. The nonlinearity of this class of symmetric functions corresponds to the nonlinearity of the functions with maximum AI that are obtained by means of the construction described in [13]. This construction has the best nonlinearity with respect to other constructions that have a provable lower bound on the AI presented in literature so far. An extensive study on the AI and nonlinearity of symmetric Boolean functions is performed in [4]. It has been shown that there exists for some even dimensions n other classes of symmetric functions with maximum AI which have slightly better nonlinearity, but still far too small to be resistant against the distinguishing attack. Also, no symmetric function with better bias in nonlinearity compared with the AI has been found in [4].

4.2 Power Functions

The idea of using a filter function f derived from a power function P on $\mathbb{F}_2^\varphi$ is as follows: we consider the φ input bits to the function P as a word $\overline{x}$ in $\mathbb{F}_2^\varphi$. We then compute the p-th power, $\overline{y} = \overline{x}^p$, of this word. The output of our Boolean function f is then one bit y_i for $i \in \{0, \ldots, \varphi - 1\}$ of this output word $\overline{y} = (y_0, \ldots, y_{\varphi-1})$. Note that all these functions for $i \in \{0, \ldots, \varphi-1\}$ are linearly equivalent to the trace of the power function P. We now discuss the nonlinearity, algebraic immunity and implementation complexity of some interesting power functions.

Nonlinearity. We will investigate Boolean functions derived from highly nonlinear bijective power functions. These functions have bias $\epsilon = 2^{-\frac{\varphi}{2}}$ for φ even and $2^{-\frac{\varphi}{2}-\frac{1}{2}}$ for φ odd, which is very close to the ideal case, the bent functions. The known classes of such highly nonlinear power functions are the inverse [28], Kasami [21], Dobbertin [14], Niho 1 [16] and Niho 2 [15] classes.

Algebraic Immunity. In [8], the AI of the Boolean functions derived from the highly nonlinear power functions is computed up to dimension less than or equal to 14. These results together with our simulations for higher dimensions indicate that most of the highly nonlinear bijective power functions we study do not achieve the optimal AI, but they do reasonably well on this criterion. For instance, the AI of the Boolean function derived from the inverse power function on $\mathbb{F}_{2^{16}}$ is equal to 6 (where 8 would be the maximum attainable). However, as shown by Courtois [10], fast algebraic attacks can be efficiently applied on this function. In particular, there exist 4 annihilators of degree 6 which reduce to degree 4 and there exist 32 annihilators of degree 7 which reduce to degree 3.

Implementation. We now study implementation complexity of some concrete functions and give the nonlinearity and AI of these practical functions. Efficient implementations of the *inverse function* in the field $\mathbb{F}_{2^i}$ for $i \geq 3$ has been studied by several authors, due to the fact that this function is used in the Advanced Encryption Standard. An efficient approach can be obtained by working in composite fields as described in [30]. Based on recursion, the inverse function is decomposed into operations in the field $\mathbb{F}_{2^2}$. A minimal hardware implementation for $\varphi = 16$ requires 398 XOR gates and 75 AND gates, and thus consists of about 1107.5 NAND gates. It is also possible to increase the clock frequency if necessary by pipelining the design.

Hardware implementation of *exponentiation with general exponents* in $\mathbb{F}_{2^\varphi}$ has been well studied [1]. However, it turns out that all classes of bijective highly nonlinear power functions with degree greater than or equal to 6 have a very regular pattern in their exponent, which can be exploited for a more efficient implementation: All exponents e satisfy the property that the vector $\overline{e} = (e_0, \ldots, e_{\varphi-1})$ defining its binary representation, *i.e.*, $e = \sum_{i=0}^{\varphi-1} e_i 2^i$, contains a regular sequence consisting of ones together with at most one bit which is separated of this sequence. The distance between two consecutive elements in the sequence is equal to one except in the case of the Dobbertin functions for $k > 1$. If the weight of this sequence is equal to a power of 2, than this property can be exploited leading to a more efficient implementation.

We will demonstrate this improved implementation on the power function X^{511} in $\mathbb{F}_2^{16}$. The exponent 511 has binary representation 111111111_2. Consequently, it contains a sequence of weight 9, or also a sequence of weight 8 with one extra digit. First, consider the normal basis $\{\alpha, \alpha^2, \alpha^4, \cdots, \alpha^{2^{\varphi-1}}\}$ of $\mathbb{F}_2^\varphi$ for $\varphi = 16$ (a normal basis exists for every $\varphi \geq 1$, see [29]). Computing the power function in this basis will not change the properties of nonlinearity, degree, AI and Walsh spectrum of the output functions, since power functions in different bases are linearly equivalent. Squaring in this basis represents simply a cyclic

shift of the vector representation of that element. Consequently, computing the power 511 of an element $\overline{x} \in \mathbb{F}_2^{16}$, can be computed as follows:

$$\begin{aligned} \overline{x}^{511} &= (\overline{x} \cdot \overline{x}^2) \cdot (\overline{x}^4 \cdot \overline{x}^8) \cdot (\overline{x}^{16} \cdot \overline{x}^{32}) \cdot (\overline{x}^{64} \cdot \overline{x}^{128}) \cdot \overline{x}^{256} \\ &= (\overline{y} \cdot \overline{y}^4) \cdot (\overline{y}^{16} \cdot \overline{y}^{64}) \cdot \overline{x}^{256} \text{ with } \overline{y} = \overline{x} \cdot \overline{x}^2 \\ &= \overline{z} \cdot \overline{z}^{16} \cdot \overline{x}^{256} \text{ with } \overline{z} = \overline{y} \cdot \overline{y}^4 . \end{aligned} \tag{17}$$

Therefore, we only need to perform some shifts together with 4 multiplications in the normal basis of $\mathbb{F}_2^{16}$. These multiplications correspond with $(\overline{x} \cdot \overline{x}^2)$, $(\overline{y} \cdot \overline{y}^4)$, and $(\overline{z} \cdot \overline{z}^{16} \cdot \overline{x}^{256})$. The hardware complexity of such multiplication depends on the basis used to represent the field elements, or more precisely, on the number of ones C_φ in the multiplication matrix. It is known that $C_\varphi \geq 2\varphi - 1$ with equality if and only if the normal basis is optimal [26]. Note that optimal bases do not exist for any dimension φ. The overall gate count of the multiplication is lower bounded by $\varphi C_\varphi \geq 2\varphi^2 - \varphi$ AND gates and $(\varphi - 1)C_\varphi \geq 2\varphi^2 - 3\varphi + 1$ XOR gates [23]. Other implementations may provide even better complexity. Also several algorithms exist for performing normal basis multiplication in software efficiently [27, 31]. Therefore, the number of NAND gates for a multiplication in normal basis is lower bounded by 1906.5 for $\varphi = 16$, 2161.5 for $\varphi = 17$, and 2719.5 for $\varphi = 19$ respectively. If the vector containing the binary representation of the exponent consists of a regular sequence with weight 2^i, then the number of multiplications is equal to i, or $i + 1$ if there is an additional digit defining the complete exponent.

We can conclude that the power functions described here offer a very good tradeoff between ease of implementation and cryptanalytic strength. Moreover, we believe that the implementation of a complete S-box in the design has several advantages. In the first place, we can increase the throughput of the generator by outputting more bits m instead of outputting 1 bit. Therefore, a careful study on the best bias in the affine approximation and the AI of all linear and nonlinear combinations of m output bits need to be performed. Another possibility, which makes the analysis harder but may increase the security, is to destroy the linearity of the state. We could consider a filter generator with memory by simply feeding some remaining bits from the S-box into a nonlinear memory. Another possibility is to feedback bits of the S-box into the LFSR during key stream generation. In both cases, it seems that the added nonlinearity may allow us to increase the throughput. Finally, resynchronization can be performed faster by using all bits of the S-box to destroy as rapidly as possible the linear relations between the bits.

5 Conclusion

In this paper, we have presented a framework for the security of the classical LFSR-based nonlinear filter and combiner generators. We have related the resistance to the most important cryptanalytic attacks (distinguishing attacks, (fast) correlation attacks and (fast) algebraic attacks) to the mathematical properties of the LFSRs and the Boolean function. From our analysis, we are inclined to

prefer the nonlinear filter generator, with a Boolean function having as most important properties high nonlinearity and high algebraic immunity.

These classical and very transparent designs are the only stream cipher building blocks for which a complete analysis of the linear biases, correlations and nonlinear relations is possible. A design that has been thoroughly analyzed with respect to the presented framework could hence be more trustworthy than a design that is based on a new, little studied design strategy.

We have also presented two classes of Boolean functions, the symmetric functions and the power functions, that should be further analyzed as they possess some desired properties and are at the same time easy to implement in hardware.

Further investigation of such LFSR-based schemes remains a necessity. Notably, the understanding of the existence of lower degree equations in fast algebraic attacks is missing. The aim of this paper is to be a step in the direction of the unification of this interesting research field, in which until now too much attention has been given to ad hoc attacks on some designs and not to the relations between the mathematical properties and the attacks.

References

1. Gordon Agnew, Thomas Beth, Ronald Mullin, and Scott Vanstone. Arithmetic operations in GF(2^m). *Journal of Cryptology*, 6(1):3–13, 1993.
2. Frederik Armknecht, Joseph Lano, and Bart Preneel. Extending the resynchronization attack. In Helena Handschuh and Anwar Hasan, editors, *Selected Areas in Cryptography, SAC 2004*, number 3357 in Lecture Notes in Computer Science, pages 19–38. Springer-Verlag, 2004.
3. Steve Babbage. Space/time trade-off in exhaustive search attacks on stream ciphers. Eurocrypt Rump session, 1996.
4. An Braeken. On the algebraic immunity of symmetric boolean functions. Technical report, K.U. Leuven, 2005.
5. An Braeken and Joseph Lano. On the (im)possibility of practical and secure nonlinear filters and combiners (extended version). COSIC technical report, 2005. `https://www.cosic.esat.kuleuven.be/publications/`.
6. Anne Canteaut and Michael Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In B. Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000*, number 1807 in Lecture Notes in Computer Science, pages 573–588. Springer-Verlag, 2000.
7. Anne Canteaut and Marion Videau. Symmetric Boolean functions. *IEEE Trans. Inform. Theory*, 2004. Regular paper. To appear.
8. Claude Carlet and Philippe Gaborit. On the construction of balanced Boolean functions with a good algebraic immunity. Proceedings of First Workshop on Boolean Functions : Cryptography and Applications, Mars 2005, Rouen, 2005.
9. Nicolas Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, number 2729 in Lecture Notes in Computer Science, pages 176–194. Springer-Verlag, 2003.
10. Nicolas Courtois. Cryptanalysis of sfinks. ECRYPT Stream Cipher Project, 2005.
11. Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In E. Biham, editor, *Advances in Cryptology - EUROCRYPT 2003*, number 2656 in Lecture Notes in Computer Science, pages 345–359. Springer-Verlag, 2003. extended version on eprint.

12. Joan Daemen, Rene Govaerts, and Joos Vandewalle. Resynchronization weaknesses in synchronous stream ciphers. In T. Helleseth, editor, *Advances in Cryptology - EUROCRYPT 1993*, number 765 in Lecture Notes in Computer Science, pages 159–167. Springer-Verlag, 1993.
13. Deepak Dalai, Kishan Gupta, and Subhamoy Maitra. Cryptographically significant Boolean functions: Construction and analysis in terms of algebraic immunity. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption, FSE 2005*, Lecture Notes in Computer Science. Springer-Verlag, 2005.
14. Hans Dobbertin. One-to-one highly nonlinear power functions on GF(2^n). *Applicable Algebra in Engineering, Communication, and Computation*, 9:139–152, 1998.
15. Hans Dobbertin. Almost perfect nonlinear power functions on $gf(2^n)$: The Niho case. *Information and Computation*, 151(1-2):57–72, 1999.
16. Hans Dobbertin, Thor Helleseth, Vijay Kumar, and Halvard Martinsen. Ternary m-sequences with three-valued crosscorrelation: New decimations of Welch and Niho type. *IEEE Transactions on Information Theory*, IT-47:1473–1481, November 2001.
17. Hakan Englund and Thomas Johansson. A new simple technique to attack filter generators and related ciphers. In Helena Handschuh and Anwar Hasan, editors, *Selected Areas in Cryptography, SAC 2004*, number 3357 in LNCS, pages 39–53. Springer, 2004.
18. Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *International Symposium on Symbolic and Algebraic Computation — ISSAC 2002*, pages 75–83. ACM Press, 2002.
19. Jovan Golic. Computation of low-weight parity-check polynomials. *Electronics Letters*, 32(21):1981–1982, 1996.
20. Philip Hawkes and Gregory Rose. Rewriting variables: The complexity of fast algebraic attacks on stream ciphers. In Matthew Franklin, editor, *Advances in Cryptology - CRYPTO 2004*, number 3152 in Lecture Notes in Computer Science, pages 390–406. Springer-Verlag, 2004.
21. Tadao Kasami. The weight enumerators for several classes of subcodes of the second order binary Reed-Muller codes. *Information and Control*, 18:369–394, 1971.
22. Edwin Key. An analysis of the structure and complexity of nonlinear binary sequence generators. *IEEE Transactions on Information Theory*, 22:732–736, 1976.
23. James Massey and Jimmy Omura. Computational method and apparatus for finite field arithmetic. *US Patent No. 4, 587,627*, 1986.
24. Willi Meier, Enes Pasalic, and Claude Carlet. Algebraic attacks and decomposition of boolean functions. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, number 3027 in Lecture Notes in Computer Science, pages 474–491. Springer-Verlag, 2004.
25. Havard Molland and Thor Helleseth. An improved correlation attack against irregular clocked and filtered keystream generators. In Matthew Franklin, editor, *Advances in Cryptology - CRYPTO 2004*, number 3152 in Lecture Notes in Computer Science, pages 373–389. Springer-Verlag, 2004.
26. Ronald Mullin, I. Onyszchuk, and Scott Vanstone. Optimal normal bases in GF(p^n). *Discrete Applied Mathematics*, 22:149–161, 1989.
27. Peng Ning and Yiqun Lisa Yin. Efficient software implementation for finite field multiplication in normal basis. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *Third International Conference on Information and Communications Security ICICS 2001*, number 2229 in Lecture Notes in Computer Science, pages 177–188. Springer-Verlag, 2001.

28. Kaisa Nyberg. Differentially uniform mappings for cryptography. In T. Helleseth, editor, *Eurocrypt 1993*, volume 950 of *Lecture Notes in Computer Science*, pages 55–64. Springer-Verlag, 1993.
29. Oystein Ore. On a special class of polynomials. *Trans. Amer. Math.Soc.*, 35:559–584, 1933.
30. Christopher Paar. *Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields*. Doctoral dissertation, Institute for Experimental Mathematics, University of Essen, Germany, 1994.
31. Arash Reyhani-Masoleh and Anwar Hasan. Fast normal basis multiplication using general purpose processors. *IEEE Transaction on Computers*, 52(3):1379–1390, 2003.
32. Greg Rose and Philip Hawkes. On the applicability of distinguishing attacks against stream ciphers. In *Proceedings of the 3rd NESSIE Workshop*, page 6, 2002.
33. Rainer Rueppel. Stream ciphers. In G. Simmons, editor, *Contemporary Cryptology. The Science of Information Integrity*, pages 65–134. IEEE Press, 1991.
34. Thomas Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, IT-30(5):776–780, 1984.
35. Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

Rekeying Issues in the MUGI Stream Cipher

Matt Henricksen and Ed Dawson

Information Security Institute, Queensland University of Technology,
GPO Box 2434, Brisbane, Queensland 4001, Australia
{m.henricksen, e.dawson}@qut.edu.au

Abstract. MUGI [15] is a word-based stream cipher designed for 64-bit architectures. It uses a 128-bit master key and a 128-bit initialization vector to populate a large non-linear feedback shift register (NLFSR) and additional non-linear state (NLS). In standard benchmarks on 32-bit processors, MUGI suffers from poor key agility because it is implemented on an architecture for which it is not designed, and because its NLFSR is too large relative to the size of its master key. This paper proposes a variant of MUGI, entitled MUGI-M, to enhance key agility, and concludes with an analysis of its security and performance characteristics.

Keywords: stream cipher, MUGI, MUGI-M, key initialization, key agility.

1 Introduction

MUGI [15] is a Pseudo Random Number Generator (PRNG) designed for use as a stream cipher. It uses a 128-bit master key and a 128-bit initialization vector. Its design strength of 128 bits is commensurate with the length of the key.

MUGI's structure is based on the PANAMA PRNG [5], which can be used either as a stream cipher or hash function. A schematic generalization of PANAMA and MUGI is shown in Figure 1. The update function Υ is composed of a linear sub-function λ and a non-linear sub-function ρ. The function λ updates the buffer, using input from both the buffer and the state. The function ρ updates the state, using additional input from the buffer. An output filter f operating on the state produces the keystream.

MUGI is targeted to 64-bit architectures, which means that in terms of speed, it is currently non-competitive with many recent word-based stream ciphers. On the Intel Pentium 4, it has a throughput of 25.2 cycles per byte, compared to 3.7, 6.7 and 9.2 cycles per byte respectively for Rabbit [3], Dragon [4], and Turing [13]. This is a situation that will almost certainly change when 64-bit architectures finally become commonplace. MUGI's mediocre performance in software is not due entirely to the mismatch between the algorithmic requirements and implementation characteristics. It has a large state space, which can lead to poor key agility, through a complex and lengthy key initialization process.

In this paper, we show how to improve MUGI's key agility for both 32- and 64-bit architectures. In Section 2, we describe the MUGI keystream generation and key initialization algorithms. In Section 3, we review previous cryptanalysis

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 175–188, 2006.

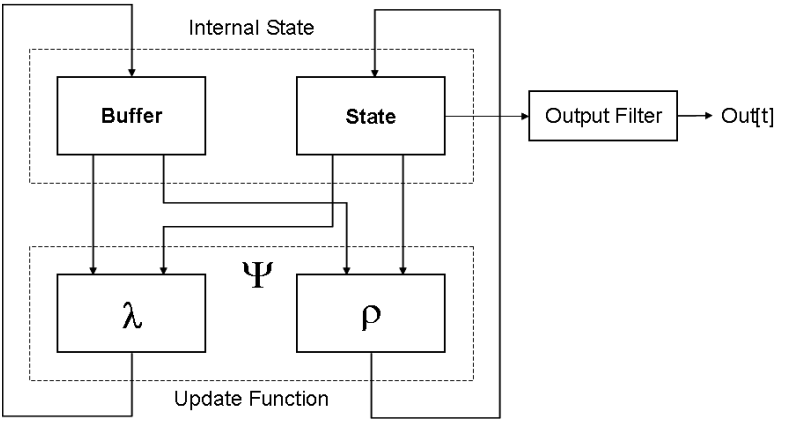


Fig. 1. Generalization of the PANAMA and MUGI Structures

of MUGI, which leads to an interesting insight on the role of the buffer in the cipher. In Section 4, we discuss a peculiarity with the key initialization algorithm. In Section 5, we analyze further the performance of MUGI relative to other word-based stream ciphers, and suggest strategies that could be used to improve it, culminating in an algorithm for a "modified MUGI" in Section 6. In Section 7 we perform a security and implementation analysis for the new algorithm. In Section 8, we summarize the contribution of paper.

2 The MUGI Algorithm

The MUGI algorithm uses 64-bit words. MUGI's internal state contains a 3-stage Non-linear Feedback Shift Register (NLFSR) denoted a, and a 16-stage Linear Feedback Shift Register (LFSR), denoted b. The output filter produces 64 bits of the output from state a at each iteration.

The non-linear function ρ is a target-heavy Feistel network structure:

$$\begin{aligned} a_0[t+1] &= a_1[t] \\ a_1[t+1] &= a_2[t] \oplus F(a_1[t], b_4[t]) \oplus C_1 \\ a_2[t+1] &= a_0[t] \oplus F(a_1[t], b_{10}[t] \lll 17) \oplus C_2 \end{aligned}$$

where C_1 and C_2 are known constants, $(M \lll k)$ indicates leftwise k-bit rotation of M, and F is a function that uses the components of the round function of the Advanced Encryption Standard [6]. Note that the 192-bit state receives at most 128 bits of new material each time ρ is called. Each of the state words is used in a different way: a_0 is used to provide new material to the buffer; a_1 is used for mixing in the F function; and a_2 is used for output and feedback.

The details of the F function are shown in Figure 2. The function has four layers. In the first layer, which resembles key addition in an Substitution Permutation Network (SPN), eight bytes from a buffer word are added to each

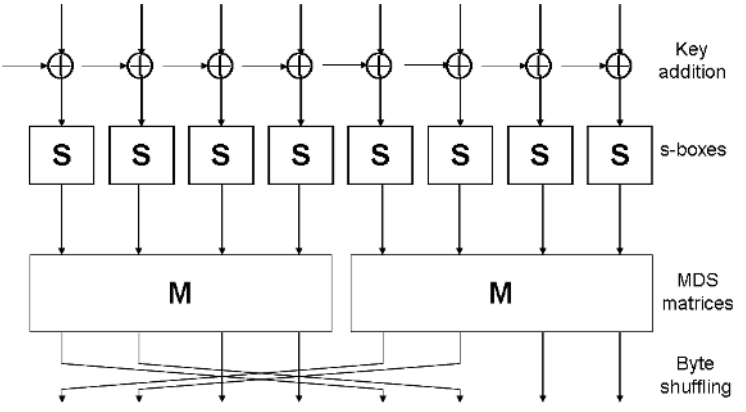


Fig. 2. MUGI F Function

of eight state bytes. In the second layer, the state is modified by eight parallel applications of the AES s-box. The third layer contains a repeated Maximum Distance Separable (MDS) matrix. The final layer consists of a word-based permutation. The polynomials used in the MDS are identical to those used in AES.

Denoting stage i $(0 \leq i \leq 15)$ of the buffer as b_i and stage j $(0 \leq j \leq 2)$ of the state as a_j, the details of function λ are as follows:

$$
\begin{aligned}
&b_i[t+1] = b_{i-1}[t] (i \neq 0, 4, 10) \\
&b_0[t+1] = b_{15}[t] \oplus a_0[t] \\
&b_4[t+1] = b_3[t] \oplus b_7[t] \\
&b_{10}[t+1] = b_9[t] \oplus (b_{13}[t] \lll 32)
\end{aligned}
$$

where $b_i[t+1]$ and $a_i[t+1]$ are the content of stage i of buffer b and respectively state a after the completion of t iterations.

Output Filter

Each application of the Υ function produces new values within the state a. The output filter selects the 64-bit block a_2 to output as the keystream.

Initialization and Rekeying

The initialization process of MUGI consists of five phases. All must be executed in full during rekeying of a master key. Only phases three, four and five are executed during rekeying of an initialization vector.

Phase 1: Master Key Injection. The 192-bit MUGI state a is initialized using the 128-bit master key K. The key is divided into two segments $K_0 \parallel K_1$ and state a set, using known constant C_0, as follows:

$$a_0 = K_0$$
$$a_1 = K_1$$
$$a_2 = (K_0 \lll 7) \oplus (K_1 \ggg 7) \oplus C_0$$

Phase 2: State Mixing and Buffer Initialization. The non-linear state function ρ is used to mix the state a a total of sixteen times, using a null buffer. After each iteration, a stage in the NLFSR buffer is filled with key-dependent material from the state word a_0. The last stage in the buffer is filled first; therefore, the first stage is filled using key material which has undergone the most mixing:

$$b(K)_{15-i} = (\rho^{i+1}(a[-48], 0))_0 \quad 0 \leq i \leq 15$$

Phase 3: Initialization Vector Injection. The 128-bit initialization vector $I = I_0 \parallel I_1$ is added to the mixed state a in a similar way to the key injection.

$$a[-32]_0 = a[-33]_0 \oplus I_0$$
$$a[-32]_1 = a[-33]_1 \oplus I_1$$
$$a[-32]_2 = a[-33]_2 \oplus (I_0 \lll 7) \oplus (I_1 \ggg 7) \oplus C_1$$

Phase 4: Further State Mixing. The state is again mixed, using a null buffer, sixteen times. At the end of this phase, the state is represented by:

$$a[-16] = \rho^{16}(a[-32], 0)$$

Phase 5: State and Buffer Mixing. The rekeying procedure is finished by iterating the state and buffer sixteen times using the Υ function, and discarding the resulting keystream.

$$a[0] = \Upsilon^{16}(a[-16]), b(K)).$$

3 Related Work

In [14] the designers of MUGI analyze their cipher. They claim that MUGI is immune to linear cryptanalysis because the minimum number of active s-boxes within an approximation is 22, and the maximum linear probability of an s-box is 2^{-6}. Consequently the maximum probability of an approximation is 2^{-132}; this is insufficient to attack the cipher given its design strength of 128 bits. They leverage the studied properties of the AES round function to claim immunity against a resynchronization attack that uses differential, linear or integral cryptanalysis.

In [7], it is shown that MUGI is not vulnerable to a linear masking attack due to the difficulty in finding a biased linear combination of the inputs and outputs of the non-linear function ρ. Also the large size of the state (1,216 bits) precludes a time-memory-data attack. The dependence of the state and buffer upon each other makes discovery of divide and conquer and correlation attacks non-trivial, and to date, none have been discovered. They note that MUGI passes all common statistical tests.

In [12], Mihaeljevic studies a variant of MUGI in which MDS matrices are excluded from the F component of the ρ update function. Because MUGI uses the AES s-box, which is well known to produce over-defined and sparse equations, the simplified MUGI can be subjected to an XL attack. However, the report [12] does not produce any definite conclusions about the complexity of the attack, except that increasing the length of the key could increase the design strength above the attack complexity (which would make the attack successful). Also Mihaeljevic [12] need not exclude linear operations like the MDS from the attack; these enable the production of additional equations which should reduce the complexity of the attack, although increase the difficulty in rendering the over-defined equations.

In [10], Golic analyses the linear function λ using a system of recurrences in b_4 and b_{10} , and solved using generating functions. From this, he discovers the period of the subsequences related to the recurrences is equal to or less than 48, and the linear complexity is 32. These properties are considered too small for use in a cryptographic application, although no attack has been forthcoming on this basis. Golic studies a simplified MUGI in which the buffer is made autonomous by decoupling the feedback from the state. Linear cryptanalysis is applied to both the simplified and full versions of MUGI — in both cases, the attack succeeds when compared to the large state size, but requires greater complexity than brute forcing the key. The attack is much easier on the simplified version, proving the success of the non-linear feedback between the buffer and the state. Golic finds that the algorithm is immune to the XL attack due to the large state and complex rekeying algorithm.

In [2], Biryukov and Shamir analyze the non-linear state (NLS) of MUGI. They find that the security of MUGI is very sensitive to small changes in the design of the ρ function, and the output filter, both of which operate on the NLS. For example, they describe practical attacks in which the output filter selects from state words a_0 or a_1, or when a_2 is chosen by the filter after the evaluation of ρ. The work of [10] in determining buffer recurrences in b_4 and b_{10} greatly simplifies the complexity of this last attack. The main part of the paper concerns an attack that allows the contents of the non-linear state to be recovered knowing only words b_4 and b_{10} of the buffer, given only three output words and a time complexity of 2^{32}. However, guessing these buffer words is equivalent in effort to guessing the secret key. Also, knowledge of the state at any point in time does not automatically allow determination of the state at a future point, since it is quickly mixed with unknown buffer words.

4 An Observation on Key Initialization

As seen in Section 2, MUGI rekeying involves five phases. In phase two, the fifteenth word of the buffer (b_{15}) is assigned the output $(\rho^1(a, 0))_0$, which is the value of the state variable a_0 after a single invocation of the ρ function. In the ρ function, the a_0 word is modified simply by replacing its value with that of a_1 (that is, one third of the state is not changed by the ρ function). Since each

buffer word is only updated once in the second phase, at the end of phase two, b_{15} contains the unmodified key word K_1, which entered the state as a_1.

Stages three and four of the initialization do not touch the buffer at all, meaning that at the start of the final stage, after thirty-two rounds of the ρ function, half of the key material is present in the buffer in its unmixed state. An attacker has to work backwards through only sixteen rounds of ρ to obtain K_1. While there is no known way of doing this faster than brute force, this is still significantly less effort than is suggested by the lengthy and complex initialization process.

5 Improving Key Agility of MUGI

Compared to many other contemporary ciphers, MUGI has a large ratio of key size to state size. This can be seen in Table 1, which is ordered by increasing ratio of key to state size.

One implication of a large state size is reduced key agility, since the key initialization algorithm needs to touch each element of the state. A rule of thumb observed in SNOW, Dragon, HC-256 and MUGI, all of which mix the internal state using the update function, is that the function should be called twice for each element in the state. Scream chains each element in its masking table by iterating the update function four times on the previous element. Consequently, MUGI, Scream and HC-256, all of which have large states, also have lengthy key initialization functions and are poor performers in terms of key agility. While Dragon and MUGI have comparable state sizes, Dragon's key is twice the length, providing better security per byte of state. Its update function is much faster, so the key initialization algorithm, at a throughput of 11 cycles/byte, is completed in approximately twenty percent of the time required by MUGI.

There are two obvious strategies that can be considered to improve the performance of MUGI. The first is to migrate the cipher from a 64- to 32-bit design, by halving the size of each of the components, including the stages in the NLFSR and the words within the non-linear state. This has the added advantage that

Table 1. Key to State Size of Modern Word Based Stream Ciphers

Cipher	Key Size (bits)	State Size (bits)	Ratio
Helix [9]	256	160	**1:0.6**
Turing [13]	256	544	**1:2.1**
SNOW [8]	256	576	**1:2.2**
Rabbit [3]	128	513	**1:4.0**
Dragon [4]	256	1,088	**1:4.2**
MUGI [15]	128	1,216	**1:9.5**
RC4 [1]	128	2,048	**1:16.0**
Scream [11]	128	2,432	**1:19.0**
HC-256 [16]	256	65,536	**1:256.0**

the design of MUGI now matches the architecture on which it is most likely to be implemented. It has the fatal weakness that the non-linear state naturally houses a 96-bit rather than 128-bit key. This key size is too small. Also the reduction in size of components necessitates rethinking the design of the core function F, which contains eight 8×8 s-boxes and two 32×32-bit MDS matrices. Using eight 4×4 s-boxes increases the maximum characteristic probability across four rounds from 2^{-132} to 2^{-50}, and using four 8×8 s-boxes increases the maximum probability across four rounds to 2^{-100}. In both cases, this is a significant loss of security. In this case the trade-off of security to benefit efficiency is inappropriate.

An alternative strategy is to leave the non-linear state and its ρ update function as they are, and act upon the deficiencies of the buffer. By reducing the buffer to 8×64-bit stages, for a total state size of $512 + 192 = 704$ bits, the speed of the rekeying strategy is increased significantly, the speed of the update function is slightly increased, and the security is marginally decreased. The state size is still more than five times the size of a 128-bit master key. This is the strategy that will be adopted in the modification of MUGI.

Shrinking the buffer involves altering the taps used for feedback, and also the indices to stages used by the non-linear filter function. As the size of the buffer is halved, it is a natural progression to also halve the indices of the taps and stages, leaving their order unaltered. One effect of this strategy is that some stages receive feedback from adjacent stages.

Another improvement is to remove phase four of the keying scheme. This mixes the non-linear state sixteen times. Consequently, by the end of the initialization, each element of the non-linear state and the buffer has been modified forty-eight and thirty-two times respectively. By removing this stage, each element of the non-linear state and buffer has been altered sixteen times. This brings the cipher into line with the design principles of other ciphers, and the rule of thumb that each element of the state should be touched by a non-linear function (at least) twice.

To remove the property discussed in Section 4, we change the state word that is fed into the buffer in phase two. If a_1 is used as feedback to the buffer, then the state word a_0 reflects the contents of the buffer word last modified. This is a benign property, since it is destroyed immediately upon commencement of phase three. But using a_2 as feedback in phase two avoids this relationship, with the obvious proviso that as it is used post-initialization to generate output, its role in providing feedback to the buffer is localized to the key initialization algorithm.

6 An Improvement: The MUGI-M Algorithm

In the modified algorithm, denoted MUGI-M, the only changes that effect the update sub-function ρ are the changes in the buffer words used as inputs:

$$
\begin{aligned}
a_0[t+1] &= a_1[t] \\
a_1[t+1] &= a_2[t] \oplus F(a_1[t], b_2[t]) \oplus C_1 \\
a_2[t+1] &= a_0[t] \oplus F(a_1[t], b_5[t] \lll 17) \oplus C_2
\end{aligned}
$$

The update sub-function λ operates on the buffer as follows:

$$b_i[t+1] = b_{i-1}[t] (i \neq 0, 2, 5)$$
$$b_0[t+1] = b_7[t] \oplus a_0[t]$$
$$b_2[t+1] = b_1[t] \oplus b_3[t]$$
$$b_5[t+1] = b_4[t] \oplus (b_6[t] \lll 32)$$

The initialization process of MUGI-M consists of four phases. All must be executed in full during rekeying of a master key. Only phases three and four are executed during rekeying of an initialization vector.

Phase 1: Master Key Injection. The 128-bit MUGI-M state a is initialized as per Phase 1 of the MUGI algorithm.

Phase 2: State Mixing and Buffer Initialization. The non-linear state function ρ is used to mix the state a a total of eight times, using a null buffer. After each iteration, a stage in the buffer is filled with key-dependent material from the state word a_2. The last stage in the buffer is filled first; therefore, the first stage is filled using key material which has undergone the most mixing:

$$b(K)_{7-i} = (\rho^{i+1}(a[-16], 0))_2 \quad 0 \leq i \leq 7$$

Phase 3: Initialization Vector Injection. The 128-bit initialization is added to the mixed state a as per Phase 3 of the MUGI algorithm.

Phase 4: State and Buffer Mixing. The rekeying procedure finishes by iterating the state and buffer eight times using the combined Υ function, and discarding the resulting keystream.

$$a[0] = \Upsilon^8(a[-8]), b(K))$$

Test vectors for this algorithm are presented in Appendix A. Code is available from the authors upon request.

7 Analysis of MUGI-M

Table 2 shows the contrast in efficiency between MUGI and MUGI-M on the Intel Pentium 4 (Northwood) processor. In particular, there is an improvement in MUGI-M of 200% in the speed of rekeying an initialization vector, and 170% in full rekeying. There is a modest 30% increase in the speed of the keystream generation, due likely due to reduced register pressure and smaller buffer loops.

The attacks discussed in Section 3 are ineffective against MUGI for the following reasons: the effectiveness of the highly non-linear state function ρ, which leverages the properties of the AES block cipher; the large size of the buffer; the feedback between the internal state and the buffer; and the complex rekeying strategy. None of the attacks rely on properties of the buffer other than

Table 2. Efficiency of MUGI and MUGI-M on the Intel Pentium 4

Cipher	Keystream Generation	Key Initialization (IV)	Key Initialization (Full)	Ratio
Cycles per iteration				
MUGI	181	4987	7540	**1:27.6:41.7**
MUGI-M	140	1652	2784	**1:11.8:20.0**
Cycles per byte				
MUGI	25.2	36.8	55.7	**1:1.5:2.2**
MUGI-M	19.4	12.2	20.6	**1:0.6:1.1**
Ratio	**1.3:1**	**3.0:1**	**2.7:1**	

its size. Golic [10] argues that the properties of the buffer, when considered autonomously, are cryptographically poor. This argument is deflected by the fact that the buffer is coupled to the non-linear state, and that it is unrealistic to map the buffer properties directly to those of the whole cipher. However, from this it can be claimed that by changing the location of the taps in the buffer, we are not altering any special properties of the buffer, which was constructed in an ad-hoc manner. We are aiming to repair the performance of MUGI rather than engender it with additional security properties. In the remainder of this section, the resistance of MUGI-M against classes of individual attacks is considered.

Block-Cipher Style Attacks. Rely on the properties of the non-linear function: for example, the maximum differential and linear probabilities across the function. Given that only the size of the buffer, and the location of its taps have been changed, the analysis of MUGI in [7] remains unchanged. The analysis relies extensively on the properties of the 64-bit F function, which is a modified AES round function. It is well-known that this function is resistant against differential and linear attacks. This is because the s-boxes in the F function have a maximum probability of 2^{-6}, although almost half of the s-box characteristics have a probability of 2^{-7}. To launch a successful attack against the F function requires a differential that incorporates fewer than ten active s-boxes, as $2^{-7\times 10} < 2^{-64}$. The analysis in [7] of the intertwined MDS matrices indicates that they guarantee at least eight active s-boxes over four rounds. If a differential style attack can be launched against MUGI, it will need to use fewer than six words of keystream. The F function exhibits a vulnerability to integral cryptanalysis across no fewer than four, and no more than nine rounds. The synchronous nature of the cipher means that the attacker does not have sufficient control over the inputs to launch it on either MUGI or MUGI-M. The resilience of MUGI-M against block-cipher style attacks appears to be the same as that of MUGI. If an attack of this style affects one, it will presumably affect the other.

Linear Cryptanalysis. The self-evaluation report of MUGI [14] includes an analysis of linear cryptanalysis incorporating both the non-linear state and the buffer. This form of linear cryptanalysis consists of two phases: the first deter-

mines a linear approximation of ρ. In the second, a path is searched to acquire an approximation that consists only of output bits (as the internal state is not available to the attacker). For MUGI-M, the first phase remains unaltered from that of MUGI: if an approximation can be found that includes fewer than twenty-two active s-boxes, linear cryptanalysis may be possible. The second phase does not depend upon the length of the buffer; since the nature of the buffer has not been fundamentally altered, the analysis of MUGI applies equally to MUGI-M.

Time-Memory-Data Trade-Off Attacks. MUGI-M is immune to time-memory-data trade-off attacks because it has a small key size relative to the size of the buffer. For a brute-force equivalent attack with $T = 2^{128}$, $M^2 \times D^2 = 2^{896}$. Assuming that a limit is placed on generating 2^{128} bits of keystream under one key, then to launch an attack requires 2^{287} gigabytes of memory. This is clearly infeasible.

Divide and Conquer Attacks. A successful divide and conquer attack on MUGI in which the components are autonomous, and that determines the contents of the components sequentially, has a complexity of $2^{192} + 2^{1024}$ (rather than the brute-force complexity of $2^{192} \times 2^{1024}$). The shorter buffer length of MUGI-M reduces this complexity to $2^{192} + 2^{512}$. This analysis ignores the fact that the components are not autonomous, and that the complexity may be much higher. The complexity of the attack needs to be less than 2^{127} to be considered successful, given the 128-bit design strength of MUGI. Therefore, divide and conquer attacks are very unlikely to succeed against MUGI-M.

Correlation Attacks. A correlation attack on MUGI or MUGI-M requires a measure of correlation between the NLS and the NLFSR. No measure has been found in either cipher, due to the absence of a perceivable bias in the non-linear filter, and to the feedback between the NLS and the NLFSR. A correlation attack against MUGI-M seems unlikely.

Guess and determine attacks have been successful against a number of word-based ciphers. In a guess and determine attack against a PANAMA-style cipher, a cryptanalyst can adopt one of three approaches: fix elements within the non-linear state and use them guess the contents of the NLFSR; fix elements within the NLFSR and use them to guess the contents of the NLS; or a hybrid approach in which elements from both components are guessed.

MUGI has shown resistance to guess and determine attacks because of the high non-linearity in the ρ function, and the large sizes of both the state and the buffer. Adopting either of the first two approaches outlined is fruitless, because the material guessed exceeds the number of bits in the master key, so a hybrid approach needs to be adopted. While this may be possible, no guess and determine attack has been possible, because no simple relationship between the non-linear state and the buffer has been discovered. As the buffers in MUGI and MUGI-M are similar in structure and size (relative to the master key size), and the ρ function is essentially unchanged, a guess and determine attack on one of the ciphers is likely to apply (with modifications) to the other.

Linear masking attacks depend on two factors: finding a linear approximation to the non-linear filter, and finding a linear combination of the buffer that causes the bias in the non-linear filter to vanish. To date, no effective bias has been discovered in the non-linear filter ρ of MUGI, which is unaltered in MUGI-M. We do not expect that MUGI-M is vulnerable to linear masking attacks.

Algebraic attacks depend upon developing systems of equations on the non-linear components of ciphers. In MUGI-M, the sole non-linear component is the AES s-box, which is well-known to be over-defined. The linear components of the non-linear filter and buffer allow extra equations to be added to the system. In principle, MUGI-M is vulnerable to an XL attack, with a complexity similar to that on MUGI, which shares the same non-linear filter. However, in both cases, the complexity of the XL attack exceeds the design strength of the 128-bit master key [12], and is therefore not practical.

Rekeying Attacks. MUGI-M appears to be secure from rekeying attacks, despite the fact that the key initialization algorithm mixes the non-linear state sixteen instead of forty-eight times, and the buffer sixteen instead of thirty-two times. The level of mixing per buffer stage remains the same.

Also the attacker has no control over any stage in the buffer, except indirectly through the non-linear state. No raw key material enters the buffer at any time.

Consider a resynchronization attack using multiple master keys, in which there are differences between the keys. For extra freedom, the attacker is allowed to control the difference in the initial a_2 state word. Because the F function is optimized against differential cryptanalysis, and because each of the stages in the buffer is chained to previous stages, the attacker very quickly loses the ability to track differences within the keystream. No differentials through the F function are possible after it has been iterated four times. After the population of b_6 and b_7 in phase two of the rekeying, subsequent words are affected by at least four iterations of the F function and therefore activate too many s-boxes for an effective related-key attack to be launched. In phase four, b_6 and b_7 are filled with material dependent upon all buffer words, so the low non-linearity present in these words in phase two is not a weakness.

8 Summary

In this paper we have reviewed past cryptanalysis of the MUGI stream cipher, and pointed out a peculiarity in the key initialization algorithm, whereby one key word was visible in the buffer after thirty-two out of forty-eight iterations of the update function. We determined that MUGI had poor key agility, compared to other word-based stream ciphers because its design targets 64-bit architectures, which are not yet commonly available, and because its large state size requires a lengthy key initialization process. The state size is large relative to the key size, so does not serve well the security-efficiency trade-offs in MUGI's design.

We suggested a variant of the MUGI algorithm, MUGI-M, in which the size of the buffer was halved, and the key initialization algorithm reduced from forty-eight to sixteen steps. This resulted in an improvement of 200% in the speed

of rekeying an initialization vector, and 170% in full rekeying. We analysed the new variant with respect to security and determined that it remains secure against attacks, principally because we made no significant alterations to the non-linear filter, because each stage in the buffer is sufficiently modified by the key initialization algorithm, and because the buffer is still large relative to the key size. This alteration will serve the security-performance trade-off of MUGI well, both now and in the future, when 64-bit architectures, for which MUGI was designed, become commonplace.

Acknowledgements

Many thanks to Minna Yao and the anonymous referees for their feedback on this paper.

References

1. Anonymous. RC4 algorithm revealed. Posting to sci.crypt usenet group on 14 September, 1994. Available at ftp://idea.sec.dsi.unimi.it/pub/security/crypt/code/rc4.revealed.gz.
2. Alex Biryukov and Adi Shamir. Analysis of the non-linear part of MUGI. In Serge Vaudenay, editor, *Proceedings of the 12th International Workshop on Fast Software Encryption*, Lecture Notes in Computer Science. Springer-Verlag, 2005. To appear.
3. Martin Boesgaard, Mette Vesterager, Thomas Pedersen, Jesper Christiansen, and Ove Scavenius. Rabbit: a new high-performance stream cipher. In Joan Daemen and Vincent Rijmen, editors, *Proceedings of the 9th International Workshop on Fast Software Encryption*, volume 2365 of *Lecture Notes in Computer Science*, pages 325–344. Springer-Verlag, 2003.
4. Kevin Chen, Matt Henricksen, Leonie Simpson, William Millian, and Ed Dawson. Dragon: A fast word based cipher. In *Information Security and Cryptology - ICISC '04 - Seventh International Conference*, 2004. To appear in Lecture Notes in Computer Science.
5. Joan Daemen and Craig Clapp. Fast hashing and stream encryption with PANAMA. In Serge Vaudenay, editor, *Proceedings of the 5th International Workshop on Fast Software Encryption*, volume 1372 of *Lecture Notes in Computer Science*, pages 60–74. Springer-Verlag, 1998.
6. Joan Daemen and Vincent Rijmen. Rijndael. In *Proceedings from the First Advanced Encryption Standard Candidate Conference, National Institute of Standards and Technology (NIST)*, August 1998. Available at http://csrc.nist.gov/encryption/aes/.
7. Ed Dawson, Gary Carter, Helen Gustafson, Matt Henricksen, William Millan, and Leonie Simpson. Evaluation of the MUGI psuedo-random number generator. Technical report, CRYPTREC, Information Technology Promotion Agency (IPA), Tokyo, Japan, 2002. Available at www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1035_IPA-MUGI_report_final.pdf.
8. Patrik Ekdahl and Thomas Johansson. Snow - a new stream cipher, 2000. Available at http://www.it.lth.se/cryptology/snow/.

9. Niels Ferguson, Doug Whiting, Bruce Schneier, John Kelsey, Stefan Lucks, and Tadayoshi Kohno. Helix: fast encryption and authentication in a single cryptographic primitive. In Joan Daemen and Vincent Rijmen, editors, *Proceedings of the 9th International Workshop on Fast Software Encryption*, volume 2365 of *Lecture Notes in Computer Science*, pages 345–361. Springer-Verlag, 2003.
10. Jovan Golic. Security evaluation of MUGI. Technical report, CRYPTREC, Information Technology Promotion Agency (IPA), Japan, Tokyo, 2002.
11. Shai Halevi, Don Coppersmith, and Charanjit Jutla. Scream: a software-efficient stream cipher. In Joan Daemen and Vincent Rijmen, editors, *Proceedings of the 9th International Workshop on Fast Software Encryption*, volume 2365 of *Lecture Notes in Computer Science*, pages 195–209. Springer-Verlag, 2003.
12. Mihodrag Mihaeljevic. Report on security evaluation of MUGI stream cipher. Technical report, CRYPTREC, Information Technology Promotion Agency (IPA), Tokyo, Japan, 2002.
13. Gregory Rose and Philip Hawkes. Turing: a fast stream cipher. In Joan Daemen and Vincent Rijmen, editors, *Proceedings of the 9th International Workshop on Fast Software Encryption*, volume 2365 of *Lecture Notes in Computer Science*, pages 307–324. Springer-Verlag, 2003.
14. Dai Watanabe, Soichi Furuya, Hirotaka Yoshida, and Kazuo Takaragi. MUGI psuedorandom number generator, self evaluation, 2001. Available at http://www.sdl.hitachi.co.jp/crypto/mugi/index-e.html.
15. Dai Watanabe, Soichi Furuya, Hirotaka Yoshida, and Kazuo Takaragi. A new keystream generator MUGI. In Joan Daemen and Vincent Rijmen, editors, *Proceedings of the 9th International Workshop on Fast Software Encryption*, volume 2365 of *Lecture Notes in Computer Science*, pages 179–194. Springer-Verlag, 2003.
16. Hongjun Wu. A New Stream Cipher HC-256, 2004. Available at http://eprint.iacr.org/2004/092.pdf.

A Test Vectors for MUGI-M

The following little-endian test vectors for MUGI-M are presented:

```
Key = 00000000000000000000000000000000
IV  = 00000000000000000000000000000000

Keystream =
0E850A7AD4E94A1C 5C97E7FBA492CC60 34738F8D04904D47
79CE86DC89D2E684 34050A91BC2555D0 8C8310A3E543DE40
F2B6B9F612381372 11036D8E55485B69 5323E5B6F05CBF32
389675E756BF490E D61618C9FAAFE00F 51BC3DA8A4C70E50
44147EFBDA308F4A D0AD8E5C38E85FD5 8F397AEA286A7761
C64694622A3599E5

Key = 0E850A7AD4E94A1C5C97E7FBA492CC60
IV  = 34738F8D04904D4779CE86DC89D2E684
```

```
Keystream =
8BB9E439BD1B632F C614E04066FAEA66 1820B17F2D7216B6
8986D48391441B8F E1B8A6D4C6A81815 B91207DC6138669A
2428795E4B67258A 7D6E0786559E0F32 E0B9DC8B34C5A6D8
C59E1BB3FD1ACA53 4395FF4AF7C9A1AC DFFDE7F86661D94D
7A37A985291598A1 AB554E72C2C7EAD2 C9125F4ACAEBE3B4
66DB2836BF75CC34

Key = 8BB9E439BD1B632FC614E04066FAEA66
IV  = 1820B17F2D7216B68986D48391441B8F

Keystream =
F4EB67A12774D27D 6FE1F36A696E8D20 0017C6166A273176
A06F58F0FAEE1B5E C1A8F9081E85FE55 A2FC5569966650F8
C44F926DFEDD99D0 5B6ECCE80E4C2057 67A9F58EED1CABF5
0500EF8D4429B3F4 90F58F5C42F74028 8C4B9D15AA7DFCE1
668491546DC4D799 4D040BCFEB46706E 365E136FC31B8204
BF9CE27566C138B1
```

Tree-Based Key Distribution Patterns

Jooyoung Lee[1] and Douglas R. Stinson[2]

[1] Department of Combinatorics and Optimization
[2] School of Computer Science, University of Waterloo,
Waterloo, Ontario, Canada N2L 3G1
{j3lee, dstinson}@uwaterloo.ca

Abstract. We revisit a key agreement scheme presented by Leighton and Micali [11], generalize the scheme, and present a new framework of tree-based key distribution pattern (TKDP). We presents a method of constructing TKDPs from cover-free families. We show the existence of TKDPs by probabilistic method. We can reduce the upper bounds on the minimum number of rows of $(t, w, \mathcal{T})$-TKDPs, which are obtained from probabilistic methods, asymptotically by a factor of w as compared to Leighton and Micali's schemes by choosing optimal trees $\mathcal{T}$ instead of chains.

Keywords: key predistribution, cover-free family.

1 Introduction

In our model, the network consists of a *trusted authority* (TA) and a set of *users* $\mathcal{U} = \{U_1, \ldots, U_n\}$. Let $2^{\mathcal{U}}$ denote the collection of all subsets of users. $\mathcal{P} \subseteq 2^{\mathcal{U}}$ and $\mathcal{F} \subseteq 2^{\mathcal{U}}$ will denote the collection of *privileged subsets* and the collection of *forbidden subsets*, respectively. $\Gamma = (\mathcal{P}, \mathcal{F})$ is called an *access structure*. Roughly speaking, a *key predistribution scheme* (KPS) with an access structure Γ is a method for the TA to distribute secret shares to each user in the network, so that any user in a privileged subset P can easily compute their group key K_P, while any coalition $F \in \mathcal{F}$ disjoint from P can compute K_P only with negligible probability. In this paper, we will consider an access structure $\Gamma_{(t,w)} = (\mathcal{P}, \mathcal{F})$ such that $\mathcal{P} = \{P \subseteq \mathcal{U} : |P| = t\}$ and $\mathcal{F} = \{F \subseteq \mathcal{U} : |F| = w\}$. Thus any t users can compute a common key while any coalition of at most w users can obtain no information on the key. (Such KPSs are called (t, w)-KPSs.) One method of constructing KPSs uses *key distribution patterns* (KDPs), due to Mitchell and Piper [13]. We will extend the study of KDPs by allowing hashed shares of information in the generation of secret keys.

1.1 Motivation: Leighton and Micali Scheme

This work is motivated by a key agreement scheme proposed by Leighton and Micali [11]. Here we briefly describe their scheme (with slight modification):

1. The TA publishes a (one-way) hash function

$$h : \{0,1\}^l \longrightarrow \{0,1\}^l,$$

where l is a secret key length used for symmetric encryption.

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 189–204, 2006.

2. For each user U_j, a public sequence $A_j \in \{0, \ldots, L-1\}^b$ is chosen in a uniformly random way, where b and L are positive integers. The public sequences act as public identities of users, and they comprise a $b \times n$ matrix $M = (A_1^T | \cdots | A_n^T) = (\alpha_{i,j})$.
3. The TA chooses random seeds $s_i = s(i,0) \in \{0,1\}^l$ for $i = 1, \ldots, b$ and computes hashed shares $s(i,\alpha) = h^\alpha(s_i)$ for $\alpha = 1, \ldots, L-1$. ($h^\alpha(s)$ indicates applying the function h iteratively α times on the input s.)
4. User U_j receives a set of secret shares $s(i, \alpha_{i,j})$ for $i = 1, \ldots, b$. Therefore, b is the number of secret shares assigned to each user.
5. A pairwise key between U_{j_1} and U_{j_2} is defined to be

$$K_{j_1,j_2} = s(1,\delta_1) + \ldots + s(b,\delta_b),$$

where $\delta_i = \max(\alpha_{i,j_1}, \alpha_{i,j_2})$. Addition is defined in $(\mathbb{Z}_2^l, +)$.

We can easily derive a condition that a coalition of w users, $U_{j'_1}, \ldots, U_{j'_w}$ obtain a hashed key between innocent users U_{j_1} and U_{j_2} as follows:

$$\min(\alpha_{i,j'_1}, \ldots, \alpha_{i,j'_w}) \leq \max(\alpha_{i,j_1}, \alpha_{i,j_2}),$$

for all $i = 1, \ldots, b$. The authors of [11] computed the probability that a set of randomly chosen sequences protects pairwise keys against adversarial coalitions of size at most w. They claim that the probability becomes large enough when w is about $(b/e \ln n)^{1/3}$.

Example 1.1. Suppose public sequences $A_1 = (5,2,1)$, $A_2 = (2,3,1)$ and $A_3 = (1,3,3)$ are assigned to users U_1, U_2 and U_3, respectively. Then U_1, U_2 and U_3 receive key sets $\{h^5(s_1), h^2(s_2), h(s_3)\}$, $\{h^2(s_1), h^3(s_2), h(s_3)\}$ and $\{h(s_1), h^3(s_2), h^3(s_3)\}$, respectively. U_1 and U_2 can communicate with each other with key $K_{1,2} = h^5(s_1) + h^3(s_2) + h(s_3)$. An adversary U_3 can compute $h^5(s_1)$ and $h^3(s_2)$, but not $h(s_3)$; therefore the key $K_{1,2}$ is secure against U_3.

1.2 Extension of Leighton and Micali Scheme

We extend Leighton and Micali's scheme in three directions.

Pairwise Key to Group Key. Any t users $U_{j_1}, \ldots, U_{j_t}$ can define their group key to be

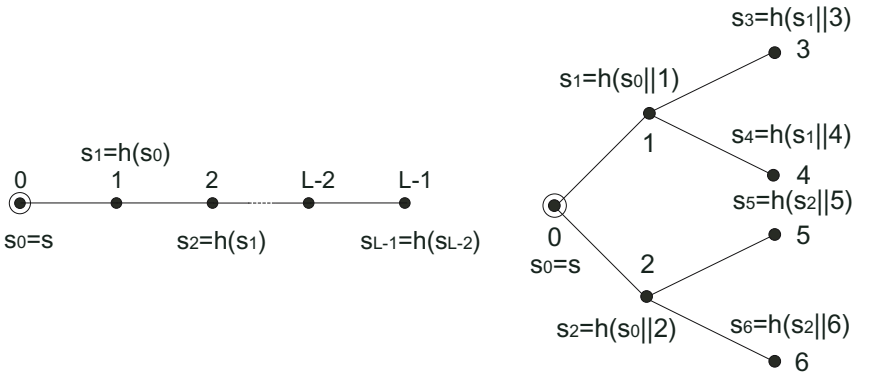
$$K_{j_1,\ldots,j_t} = s(1,\delta_1) + \ldots + s(b,\delta_b),$$

where $\delta_i = \max(\alpha_{i,j_1}, \ldots, \alpha_{i,j_t})$. In order for a coalition of w users, $U_{j'_1}, \ldots, U_{j'_w}$, to obtain the group key, the following should occur:

$$\min(\alpha_{i,j'_1}, \ldots, \alpha_{i,j'_w}) \leq \max(\alpha_{i,j_1}, \ldots, \alpha_{i,j_t}),$$

for all $i = 1, \ldots, b$.

Linear Ordering to Tree Ordering. Suppose that a matrix $M = (\alpha_{i,j})$ of public sequences is determined. For each row of the matrix, a "seed" secret



(a) Linear ordering (b) Tree-based ordering

Fig. 1. Orderings of symbols

share is generated and each entry represents the hash depth applied to the seed. Therefore, we observe a hierarchy or an ordering between entries (symbols) in the sense that one information is enough to compute the other, but not conversely. For two symbols α_1 and α_2, we define $\alpha_1 \leq \alpha_2$ if the information$(= h^{\alpha_2}(s))$ associated with α_2 is easily computed from the information$(= h^{\alpha_1}(s))$ associated with α_1. We can represent the ordering of symbols for each row as a chain structure (Fig. 1(a)), which coincides with the natural linear ordering of integers.

However we can also define nonlinear orderings based on rooted trees. For example, Fig. 1(b) shows an ordering of symbols based on a balanced binary tree of depth 2. Seven hashed shares are derived from a seed for each row, using (public) symbols of the tree in the hash computation. (The symbols are interpreted as binary sequences of a fixed length in the computation.)

Probabilistic Construction to Deterministic Construction. We can construct a matrix M of public sequences in a deterministic way, instead of choosing random columns. For example, cover-free families, orthogonal arrays or ordered designs could be used. We also introduce a symbol ∞ as an entry of M, which means "no information". Based on a chain ordering, we can replace the largest symbol $L-1$ by ∞; since the corresponding hashed share $h^{(L-1)}(s)$ (for some seed s) can be computed by any other single user, it makes no contribution to secure communication. Therefore we would rather assign no information to the symbol in order to save memory storage per user.

1.3 Other Related Works

Blom [3] presented key predistribution schemes that require each user store $w+1$ keys to protect each pairwise key against any coalition attack of at most w users. Blundo *et al.* [4] generalized Blom's scheme to allow for any group of size t to establish its group key. For a group key to be secure against a coalition of w users, each user has to store $\binom{t+w-1}{t-1}$ keys. They also proved the key storage is

information theoretically optimal. Mitchell and Piper [13] presented KPSs using key predistribution patterns, which involve very little arithmetic computation. The KDPs are equivalent to cover-free families, which are widely studied and generalized (for example, [17] and [18]). Such unconditionally secure key predistribution schemes are studied as special families of linear key predistribution schemes [14].

The efficiency of KPSs can be improved by choosing orthogonal arrays or perpendicular arrays as KDPs and then using resilient functions in the generation of keys [16]. Recently, Attrapadung *et al.* [2] studied key predistribution schemes using key chains from pseudo-random sequence generators (PRSGs). In a network of n users, each user stores $O\left(\frac{2^{n-1}}{n}\right)$ keys to establish group keys of any size, which are secure against any coalition attacks, based on the security of the PRSGs.

Since Eschenauer and Gligor's work [8], a series of papers on (pairwise) key predistribution for distributed sensor networks have been published ([5], [9], [10], [12], [15]). The main difference of these KPSs from conventional ones is that they allow certain pairs of nodes to share no common key. The attack model and the metric to evaluate the resiliency are also different, which are based on probabilistic analysis.

1.4 Our Contributions

By extending Leighton and Micali's scheme, we present a new framework of tree-based key distribution patterns (TKDPs). In section 3, we present a simple method of constructing TKDPs from cover-free families. In section 4, we show the existence of TKDPs by probabilistic methods, similar to [7]. Especially, the upper bounds on the minimum number of rows of $(t, w, \mathcal{T})$-TKDPs, which are obtained from probabilistic methods, are asymptotically w times smaller than Leighton and Micali's schemes (with the same parameters), when we choose optimal trees $\mathcal{T}$ instead of chains. One advantage of TKDP-based KPSs over Leighton and Micali's schemes is smaller hash depths are applied to each secret seed, which means less computation is required.

2 Tree-Based Key Distribution Pattern

We are now prepared to present a framework of a *tree-based key distribution pattern* (TKDP). First, we define a rooted tree $\mathcal{T}$ on L vertices, labeled from 0 to $L-1$. Especially, the root vertex is labeled by 0. As we observed in the previous section, a rooted tree defines a partial ordering on L vertices. We say $j_1 < j_2$ *if a vertex* j_1 *is an ancestor of* j_2. For example, we observe that $0 < 3$ and $2 < 5$ in Fig. 1(b). But there is no relation between 1 and 2. We also define $j < \infty$ for every $j \in \{0, \ldots, L-1\}$. From now on, we will identify the set of users with the integers from 1 to n.

Definition 2.1. *Let* $\mathcal{T}$ *be a rooted tree labeled by* $\{0, \ldots, L-1\}$ *and let* $\Gamma = (\mathcal{P}, \mathcal{F})$ *be an access structure. Let* $M = (\alpha_{i,j})$ *be a* $b \times n$ *matrix with*

$\alpha_{i,j} \in \{0, \ldots, L-1\} \cup \{\infty\}$ *for* $1 \le i \le b$ *and* $1 \le j \le n$. *We say that* M *is a* tree-based key distribution pattern $(\Gamma, \mathcal{T})$-*TKDP*(b, n), *provided that for every disjoint pair of* $P \in \mathcal{P}$ *and* $F \in \mathcal{F}$, *there exist* $1 \le i^* = i^*_{(P,F)} \le b$ *and* $j^* = j^*_{(P,F)} \in P$ *such that*

1. $\alpha_{i^*,j} \le \alpha_{i^*,j^*}$ *for all* $j \in P$, *and*
2. $\alpha_{i^*,j} \not\le \alpha_{i^*,j^*}$ *for all* $j \in F$.

The notation $(t, w, \mathcal{T})$-*TKDP*(b, n) *will denote a* $(\Gamma_{(t,w)}, \mathcal{T})$-*TKDP*$(b, n)$.

KPS from TKDP. We construct a KPS with an access structure Γ from a TKDP in the following manner:

1. The TA publishes a $(\Gamma, \mathcal{T})$-TKDP(b, n), denoted $M = (\alpha_{i,j})$, a hash function
$$h : \{0,1\}^{l_1+l_2} \longrightarrow \{0,1\}^{l_1},$$
and a tree $\mathcal{T}$ on a set $\{0, \ldots, L-1\}$ of vertices.
2. For $1 \le i \le b$ and $1 \le j \le L-1$, the TA chooses random values $s_i = s(i, 0)$ of length l_1 and computes hashed shares $s(i, j)$ recursively based on the tree $\mathcal{T}$; if a vertex j_2 is a child of j_1, then $s(i, j_2) = h(s(i, j_1) \parallel j_2)$, where the label of each vertex is represented as a binary sequence of length l_2.
3. User j receives secret shares $s(i, \alpha_{i,j})$ for $1 \le i \le b$, where $s(i, \infty) = \emptyset$ by definition.
4. Let
$$I_P = \{1 \le i \le b : \exists \bar{j} \in P \text{ such that } \alpha_{i,j} \le \alpha_{i,\bar{j}},\ \forall j \in P\}.$$
The key K_P for a privileged set P is defined to be
$$K_P = \sum_{i \in I_P} s(i, \alpha_{i,\bar{j}}).$$
For a given $i \in I_P$, $\bar{j}$ is not necessarily unique, but $\alpha_{i,\bar{j}}$ is.

Any user $j \in P$ can compute every term in K_P by using the function h, while a disjoint coalition $F \in \mathcal{F}$ cannot compute at least one term
$$s(i^*_{(P,F)}, \alpha_{i^*_{(P,F)},\bar{j}}),$$
since $\alpha_{i^*_{(P,F)},j} \not\le \alpha_{i^*_{(P,F)},\bar{j}}$ for all $j \in F$ (by Def. 2.1). In this case, we say *row* $i^*_{(P,F)}$ *protects* P *against* F. If l_1 is the bit length of a secure key for a symmetric encryption, then we can say the KPS is secure with respect to the access structure Γ.

Remark 2.1. We assume a random oracle model for the function h in the sense that we cannot obtain the value of $h(s)$ without query to the random oracle.

Remark 2.2. Each user should store bl_1 bits, or equivalently, b secret keys. Therefore, we would like to have TKDPs that minimize b with the other parameters fixed.

Example 2.1. As we will see in Example 3.1,

$$M = \begin{pmatrix} 0\,0\,3\,0\,5\,6\,7 \\ 1\,0\,0\,4\,0\,6\,7 \\ 0\,2\,0\,4\,5\,0\,7 \\ 1\,2\,3\,0\,0\,0\,7 \\ 0\,2\,3\,4\,0\,6\,0 \\ 1\,0\,3\,4\,5\,0\,0 \\ 1\,2\,0\,0\,5\,6\,0 \end{pmatrix}$$

is a $(2, 2, \mathcal{T}_7)$-TKDP$(7, 7)$, where $\mathcal{T}_7$ is a tree of depth 1 with 7 leaves (Fig. 2).

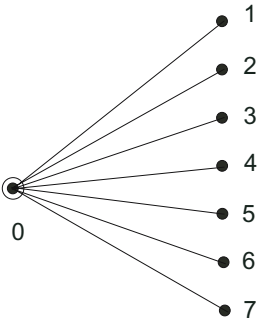


Fig. 2. Star-like tree: $\mathcal{T}_7$

The TA generates random secret seeds s_i for $i = 1, \ldots, 7$. Two users U_1 and U_2 (corresponding to the first two columns) receive sets of keys

$$K_1 = \{s_1, h(s_2 \parallel 1), s_3, h(s_4 \parallel 1), s_5, h(s_6 \parallel 1), h(s_7 \parallel 1)\},$$

and

$$K_2 = \{s_1, s_2, h(s_3 \parallel 2), h(s_4 \parallel 2), h(s_5 \parallel 2), s_6, h(s_7 \parallel 2)\},$$

respectively. For $P = \{1, 2\}$, we have $I_P = \{1, 2, 3, 5, 6\}$. Both U_1 and U_2 can compute the group key

$$K_{1,2} = s_1 + h(s_2 \parallel 1) + h(s_3 \parallel 2) + h(s_5 \parallel 2) + h(s_6 \parallel 1).$$

An adversarial coalition of two users, say, U_3 and U_4, cannot compute $h(s_5 \parallel 2)$ and $h(s_6 \parallel 1)$. In fact, $K_{1,2}$ is secure against any coalition of size 2.

3 Cover-Free Families and TKDPs

In this section, we will show some relations between *cover-free families* (or equivalently, conventional key distribution patterns) and TKDPs. A *set system* is a pair $(X, \mathcal{A})$, where $\mathcal{A}$ is a finite set of subsets of X, called *blocks*.

Definition 3.1. *A set system* $(X, \mathcal{A})$ *is called a* (t, w)-cover-free family *provided that, for any* t *blocks* $B_1, \ldots, B_t \in \mathcal{A}$ *and any* w *other blocks* $A_1, \ldots, A_w \in \mathcal{A}$, *we have*

$$\bigcap_{i=1}^{t} B_i \nsubseteq \bigcup_{j=1}^{w} A_j.$$

A (t, w)-cover-free family will be denoted as a (t, w)-CFF(b, n) if $|X| = b$ and $|\mathcal{A}| = n$. Let $(X, \mathcal{A})$ be a (t, w)-CFF(b, n) such that $X = \{x_1, \ldots, x_b\}$ and $\mathcal{A} = \{A_1, \ldots, A_n\}$. The *incidence matrix* of $(X, \mathcal{A})$ is defined to be a $b \times n$ $(0, 1)$-matrix $M = (\beta_{i,j})$ such that $\beta_{i,j} = 1$ if $x_i \in A_j$, and $\beta_{i,j} = 0$, otherwise.

Now we can regard a cover-free family as a special case of TKDP, as seen in the following lemma.

Lemma 3.1. *Let* $L = 1$, *i.e.,* $\mathcal{T}$ *consists of one vertex. Then* $(t, w, \mathcal{T})$*-TKDPs are equivalent to* (t, w)*-cover-free families.*

Proof. Let $M_T = (\alpha_{i,j})$ be a $(t, w, \mathcal{T})$-TKDP(b, n). Since M_T is a $(0, \infty)$-matrix, we can define a $b \times n$ matrix $M_C = (\beta_{i,j})$ such that

$$\begin{cases} \beta_{i,j} = 1, & \text{if } \alpha_{i,j} = 0; \\ \beta_{i,j} = 0, & \text{if } \alpha_{i,j} = \infty. \end{cases}$$

Then M_C is the incidence matrix of a (t, w)-CFF(b, n). The converse is proved in a similar way.

There is a method of constructing TKDPs based on rooted trees of depth 1 from CFFs. (From now on, rooted trees of depth 1 are called *star-like trees.*) Let $\mathcal{T}_n$ be a star-like tree with root 0 and n leaves (labeled from 1 to n).

Theorem 3.1. *If there exists a* $(t-1, w)$*-CFF*(b, n), *then there exists a* $(t, w, \mathcal{T}_n)$*-TKDP*(b, n).

Proof. Let $M_C = (\beta_{i,j})$ be the incidence matrix of a $(t-1, w)$-CFF(b, n). Then we can define a $b \times n$ matrix $M_T = (\alpha_{i,j})$ as follows;

$$\begin{cases} \alpha_{i,j} = 0, & \text{if } \beta_{i,j} = 1; \\ \alpha_{i,j} = j, & \text{if } \beta_{i,j} = 0. \end{cases}$$

We claim that M_T is a $(t, w, \mathcal{T}_n)$-TKDP(b, n).

The columns of M_C and M_T are indexed from 1 to n; the rows of M_C and M_T are indexed from 1 to b. Let $P = \{j_1, \ldots, j_t\}$ and $F = \{j'_1, \ldots, j'_w\}$ be disjoint subsets of columns. Since M_C is the incidence matrix of a $(t-1, w)$-CFF(b, n), there exists a row i such that $\beta_{i,j_1} = \ldots = \beta_{i,j_{t-1}} = 1$, and $\beta_{i,j'_1} = \ldots = \beta_{i,j'_w} = 0$. Since $\alpha_{i,j_1} = \ldots = \alpha_{i,j_{t-1}} = 0$, we have $\alpha_{i,j} \leq \alpha_{i,j_t}$ for every $j \in P$. On the other hand, we have $\alpha_{i,j'} \nleq \alpha_{i,j_t}$ for every $j' \in F$ since the symbols $\alpha_{i,j'}$ represent distinct leaves of the tree. (In the context of KPS, the group key of P contains a hashed share $h(s_i \parallel j_t)$ as a summand, while the coalition of F cannot compute it.)

Example 3.1. Let $t = w = 2$ and $b = n = 7$. Then we can construct a $(2, 2, \mathcal{T}_7)$-TKDP$(7, 7)$, say M_T, from the incidence matrix M_C of a $(1, 2)$-CFF$(7, 7)$, where

$$M_C = \begin{pmatrix} 1\,1\,0\,1\,0\,0\,0 \\ 0\,1\,1\,0\,1\,0\,0 \\ 1\,0\,1\,0\,0\,1\,0 \\ 0\,0\,0\,1\,1\,1\,0 \\ 1\,0\,0\,0\,1\,0\,1 \\ 0\,1\,0\,0\,0\,1\,1 \\ 0\,0\,1\,1\,0\,0\,1 \end{pmatrix} \quad \text{and} \quad M_T = \begin{pmatrix} 0\,0\,3\,0\,5\,6\,7 \\ 1\,0\,0\,4\,0\,6\,7 \\ 0\,2\,0\,4\,5\,0\,7 \\ 1\,2\,3\,0\,0\,0\,7 \\ 0\,2\,3\,4\,0\,6\,0 \\ 1\,0\,3\,4\,5\,0\,0 \\ 1\,2\,0\,0\,5\,6\,0 \end{pmatrix}.$$

4 Probabilistic Method to Construct TKDPs

We will show the existence of a $(t, w, \mathcal{T})$-TKDP(b, n), denoted $M = (\alpha_{i,j})$, by using the probabilistic method [1]. We would like for the matrix M to have as few rows as possible.

Now we fix a tree $\mathcal{T}$ on a set $V = \{0, \ldots, L-1\}$. Let $S = \{0, \ldots, L-1, \infty\}^n$ be the set of all possible rows of length n with symbols from $\{0, \ldots, L-1, \infty\}$. We define a probability distribution $\mathbf{P}$ on the set S, and a random variable $X(P, F)$ on S for any disjoint pair of $P \in \mathcal{P}$ and $F \in \mathcal{F}$ as follows:

$$X(P, F) = \begin{cases} 0 & \text{if the row protects } P \text{ against } F \\ 1 & \text{otherwise.} \end{cases}$$

Let $\mathcal{M}_{b\times n}$ be the set of all $b \times n$ matrices with symbols from $\{0, \ldots, L-1, \infty\}$. By the probability distribution $\mathbf{P}$, we pick uniformly at random b rows to construct a matrix in $\mathcal{M}_{b\times n}$. The distribution $\mathbf{P}$ and the random variable $X(P, F)$ naturally define the inherited probability distribution $\mathbf{P}_b$ and random variable

$$X_b(P, F) = \begin{cases} 0 & \text{if there exists a row that protects } P \text{ against } F \\ 1 & \text{otherwise.} \end{cases}$$

on $\mathcal{M}_{b\times n}$. Then we have

$$\begin{aligned} \mathbf{Exp}[X_b(P, F)] &= \mathbf{Pr}[X_b(P, F) = 0] \times 0 + \mathbf{Pr}[X_b(P, F) = 1] \times 1 \\ &= (\mathbf{Pr}[X(P, F) = 1])^b . \end{aligned}$$

Finally, we define a random variable

$$X = \sum_{P\in\mathcal{P}, F\in\mathcal{F}, P\cap F=\emptyset} X_b(P, F).$$

If $\mathbf{Exp}[X] < 1$, then there exists a $(t, w, \mathcal{T})$-TKDP(b, n) in $\mathcal{M}_{b\times n}$. In order to evaluate $\mathbf{Exp}[X]$, we define

$$\bar{p}_{t,w} = \min_{P\in\mathcal{P}, F\in\mathcal{F}, P\cap F=\emptyset} \mathbf{Pr}[X(P, F) = 0].$$

Then it is easy to see that

$$\begin{aligned}\mathbf{Exp}[X] &= \sum_{P\in\mathcal{P},F\in\mathcal{F},P\cap F=\emptyset} \mathbf{Exp}[X_b(P,F)] \\ &= \binom{n}{t}\binom{n-t}{w}\mathbf{Exp}[X(P,F)]^b \\ &\le n^{t+w}(1-\bar{p}_{t,w})^b.\end{aligned}$$

A simple computation shows

$$b > \frac{(t+w)\ln n}{-\ln(1-\bar{p}_{t,w})} \implies \mathbf{Exp}[X] < 1, \tag{1}$$

where

$$\frac{(t+w)\ln n}{-\ln(1-\bar{p}_{t,w})} \approx \frac{(t+w)\ln n}{\bar{p}_{t,w}},$$

for sufficiently small $\bar{p}_{t,w}$. Note that $\bar{p}_{t,w}$ is determined by the probability distribution $\mathbf{P}$ on S and the tree $\mathcal{T}$.

Example 4.1. Let $\mathcal{T}$ be a chain on n vertices. Suppose we define

$$\begin{array}{rcll}\mathbf{P} : \{0,\ldots,n-1\}^n & \longrightarrow & [0,1] & \\ \alpha & \mapsto & 1/n! & \text{if } \alpha \text{ is a permutation of the } n \text{ distinct symbols,} \\ & & 0 & \text{otherwise.}\end{array}$$

Since

$$\bar{p}_{t,w} = \frac{t!w!}{(t+w)!},$$

we have a $(t,w,\mathcal{T})$-TKDP(b,n) for

$$b \approx \frac{(t+w)(t+w)!\ln n}{t!w!}.$$

In the following sections, we consider two kinds of probability distributions to generate random rows.

4.1 Independent Random Selection of Symbols

In this section, we fix a tree $\mathcal{T}$ on $V = \{0,\ldots,L-1\}$, and find the optimal probability distribution such that each symbol is chosen independently; each symbol $i \in V \cup \{\infty\}$ is selected with probability p_i, where $0 \le p_i \le 1$ for $i = 0,\ldots,L-1,\infty$, and

$$p_0 + \ldots + p_{L-1} + p_\infty = 1.$$

Then the probability distribution $\mathbf{P}$ on $S = \{0,\ldots,L-1,\infty\}^n$ is derived as follows:

$$\begin{array}{rcl}\mathbf{P} : S & \longrightarrow & [0,1] \\ \alpha & \mapsto & \prod_{i=0}^{L-1} p_i^{wt_i(\alpha)},\end{array}$$

where $wt_i(\alpha)$ is the number of i's appearing in α. Now we have the following lemma.

Lemma 4.1. *Let $\mathbb{P}(v)$ denote the path from the root 0 to $v \in V$ in $\mathcal{T}$ and let $prt(v)$ denote the parent of v. Then we have*

$$\mathbf{Pr}[X(P,F)=0] = \sum_{v \in V} \left(\left(\sum_{i \in \mathbb{P}(v)} p_i \right)^t - \left(\sum_{i \in \mathbb{P}(prt(v))} p_i \right)^t \right) \left(1 - \sum_{i \in \mathbb{P}(v)} p_i \right)^w,$$

for any disjoint $P \in \mathcal{P}$ and $F \in \mathcal{F}$.

Proof. Note that $\mathbf{Pr}[X(P,F)=0]$ is the probability that a row(sequence) protects P against F when it is chosen by the probability distribution $\mathbf{P}$. Suppose that we pick up a sequence $\alpha = (\alpha_1, \ldots, \alpha_n)$. If α protects P against F, then there exists $j^* \in P$ such that $\alpha_j \leq \alpha_{j^*}$, for all $j \in P$ and $\alpha_j \nleq \alpha_{j^*}$, for all $j \in F$. Since the symbol α_{j^*} is unique, we can define $\max(\alpha, P) = \alpha_{j^*}$ for any sequence α that protects P against F.

If $\max(\alpha, P) = v$, then the positions of P should consist of symbols of the path from the root to the vertex v, and contain at least one symbol of v. On the other hand, the positions of F should contain no symbol from the path. Therefore we have

$$\mathbf{Pr}[\max(\alpha,P)=v] = \left(\left(\sum_{i \in \mathbb{P}(v)} p_i \right)^t - \left(\sum_{i \in \mathbb{P}(prt(v))} p_i \right)^t \right) \left(1 - \sum_{i \in \mathbb{P}(v)} p_i \right)^w,$$

for every $v \in V$. Now the lemma is true since

$$\mathbf{Pr}[X(P,F)=0] = \sum_{v \in V} \mathbf{Pr}[\max(\alpha,P)=v].$$

By using the above lemma, we can find optimal probability distributions for some special cases.

***L* = 2.** We have a unique tree $\mathcal{T}$ on 2 vertices. We would like to maximize the probability to construct a $(t, w, \mathcal{T})$-TKDP(b, n), denoted $M = (\alpha_{i,j})$. In this model, symbols 0, 1 and ∞ are selected in a independent random way with probabilities p_0, p_1 and p_∞, respectively. By Lemma 4.1, we have

$$\bar{p}_{t,w} = p_0^t (1-p_0)^w + ((p_0+p_1)^t - p_0^t)(1-p_0-p_1)^w.$$

Removing p_∞, we have to solve the following problem to find an optimal probability distribution:

$$\begin{array}{ll} \text{Maximize} & \bar{p}_{t,w} = p_0^t (1-p_0)^w + ((p_0+p_1)^t - p_0^t)(1-p_0-p_1)^w \\ \text{subject to} & p_0 + p_1 \leq 1,\ p_0,\ p_1 \geq 0. \end{array}$$

Example 4.2. Let $t = 2$, $w = 1$ and $n = 1000$. Then the above problem is reduced to

$$\begin{array}{ll} \text{Maximize} & \bar{p}_{2,1} = -p_0^3 - 2p_0^2 p_1 + p_0^2 - 3p_0 p_1^2 + 2p_0 p_1 - p_1^3 + p_1^2 \\ \text{subject to} & p_0 + p_1 \le 1,\ p_0,\ p_1 \ge 0. \end{array}$$

By elementary calculus, we can show that $\bar{p}_{2,1}$ attains its maximum $\bar{p}^*_{2,1} \approx 0.204$ when

$$p_0 = \frac{12}{23}, \quad p_1 = \frac{6}{23}, \quad \text{and} \quad p_\infty = \frac{5}{23}.$$

By the condition (1), there exists a $(2, 1, \mathcal{T})$-TKDP$(91, 1000)$. On the other hand, if we use a probabilistic method [7] for cover-free families, we can show the existence of a $(2, 1)$-CFF(130,1000) at best. Therefore we have about 40% improvement in the number of rows.

Star-Like Trees with Many Leaves. Suppose $\mathcal{T}$ is a star-like tree with a leaves (labeled from 1 to a). We choose each leaf with the same probability p_1, and set $p_\infty = 0$ and $t = 2$, for simple analysis. Then we have to solve the following problem:

$$\begin{array}{ll} \text{Maximize} & \bar{p}_{2,w} = p_0^2(1-p_0)^w + a((p_0+p_1)^2 - p_0^2)(1-p_0-p_1)^w \\ \text{subject to} & p_0 + ap_1 = 1,\ p_0,\ p_1 \ge 0. \end{array}$$

Put $x = p_0$ and $y = ap_1$. For a sufficiently large a, we have the approximation

$$\begin{aligned} \bar{p}_{2,w} &= (1-y)^2 y^w + a\left(\left(x + \frac{y}{a}\right)^2 - x^2\right)\left(y - \frac{y}{a}\right)^w \\ &= (1-y)^2 y^w + \left(2xy + \frac{y^2}{a}\right)\left(y - \frac{y}{a}\right)^w \\ &\approx \left((1-y)^2 + 2xy\right) y^w \\ &= \left(1 - y^2\right) y^w. \end{aligned}$$

Since

$$\begin{aligned} \frac{d}{dy}\left(\left(1-y^2\right) y^w\right) &= (-2y)\, y^w + w\left(1-y^2\right) y^{w-1} \\ &= \left(w - (w+2)\, y^2\right) y^{w-1}, \end{aligned}$$

we see that $\bar{p}_{2,w}$ attains its (approximate) maximum

$$\bar{p}^*_{2,w} \approx \frac{2}{w+2}\left(\frac{w}{w+2}\right)^{w/2},$$

when $y = \sqrt{\frac{w}{w+2}}$, or equivalently,

$$p_0 = 1 - \sqrt{\frac{w}{w+2}},\ p_1 = \frac{1}{a}\sqrt{\frac{w}{w+2}},\ (\text{and } p_\infty = 0).$$

By the condition (1), we know there exists a $(2, w, \mathcal{T})$-TKDP(b, n) if

$$b > \frac{1}{2}(w+2)^2 \left(1 + \frac{2}{w}\right)^{w/2} \ln n,$$

for sufficiently large w. It is instructive to observe that a similar method shows the existence of $(2, w)$-CFF(b, n) for

$$b > \frac{1}{4}(w+2)^3 \left(1 + \frac{2}{w}\right)^{w} \ln n.$$

4.2 Random Permutations

In this section, we fix a probabilistic distribution, and find the optimal tree ordering. We extend Example 4.1 to a (general) tree $\mathcal{T}$ on a set $V = \{0, \ldots, L-1\}$ of vertices such that $L + 1 \geq n$. We define a probability distribution $\mathbf{P}$ on $S = \{0, \ldots, L-1, \infty\}^n$ as follows:

$$\begin{array}{lll} \mathbf{P} : S \longrightarrow [0,1] & & \\ \quad \alpha \mapsto (L+1-n)!/(L+1)! & \text{if } \alpha \text{ is a permutation of an } n\text{-subset of } V \cup \{\infty\}, \\ \quad\quad 0 & \text{otherwise.} \end{array}$$

Then we have the following lemma.

Lemma 4.2. *Let m_d be the number of vertices whose distance from the root is d. Then we have*

$$\mathbf{Pr}[X(P,F) = 0] = \frac{\sum_{d \geq t-1} m_d \left(\binom{d+1}{t} - \binom{d}{t}\right) \binom{L-d}{w} t! w!}{\binom{L+1}{t+w} (t+w)!},$$

for any disjoint $P \in \mathcal{P}$ and $F \in \mathcal{F}$.

Example 4.3. Let $t = 2$, $w = 1$, and let $\mathcal{T}$ be a binary tree on $L = 7$ vertices as seen in Fig. 1(b). Since $m_0 = 1$, $m_1 = 2$ and $m_2 = 4$, we have

$$\begin{aligned} \mathbf{Pr}[X(P,F) = 0] &= \frac{2 \cdot \binom{2}{2}\binom{6}{1} \cdot 2! + 4 \cdot (\binom{3}{2} - \binom{2}{2})\binom{5}{1} \cdot 2!}{\binom{8}{3} \cdot 3!} \\ &= \frac{31}{84} \approx 0.369. \end{aligned}$$

by Lemma 4.2. Therefore, when we choose a sequence from $S = \{0, \ldots, 6, \infty\}^n$ by the probability distribution $\mathbf{P}$ (for any $n \leq 8$), the sequence protects any two users against another single user with the probability ≈ 0.369.

Proof. Note that $\mathbf{Pr}[X(P,F) = 0]$ is the probability that a row(sequence) protects P against F when it is chosen by the probability distribution $\mathbf{P}$. Suppose that we choose a sequence $\alpha = (\alpha_1, \ldots, \alpha_n)$. If α protects P against F, then there exists $j^* \in P$ such that $\alpha_j \leq \alpha_{j^*}$, for all $j \in P$ and $\alpha_j \nleq \alpha_{j^*}$, for all $j \in F$.

Since the symbol α_{j^*} is unique, we can define $\max(\alpha, P) = \alpha_{j^*}$ for any sequence α that protects P against F.

If $\max(\alpha, P) = v$, then the positions of P should consist of symbols of the path from the root to the vertex v, and contain exactly one symbol of v. On the other hand, the positions of F should contain no symbol from the path. Therefore we have

$$\mathbf{Pr}[\max(\alpha, P) = v] = \frac{\left(\binom{d+1}{t} - \binom{d}{t}\right)\binom{L-d}{w} t!w!}{\binom{L+1}{t+w}(t+w)!},$$

where $d \geq t-1$ is the distance from the root to the vertex v. If $d < t-1$, then $\mathbf{Pr}[\max(\alpha, P) = v] = 0$ since the symbols of P cannot be all distinct. Now the lemma follows since

$$\mathbf{Pr}[X(P, F) = 0] = \sum_{v \in V} \mathbf{Pr}[\max(\alpha, P) = v].$$

In order to find an optimal tree which maximizes $\bar{p}_{t,w}$, we have to solve the following optimization problem:

$$\text{Maximize} \qquad \bar{p}_{t,w} = \frac{\sum_{d \geq t-1} m_d \left(\binom{d+1}{t} - \binom{d}{t}\right)\binom{L-d}{w} t!w!}{\binom{L+1}{t+w}(t+w)!} \tag{2}$$

$$\text{subject to} \qquad \sum_{d=1}^{D} m_d = L - 1, \tag{3}$$

$$D,\ m_1, \ldots m_D \in \mathbb{Z}^+. \tag{4}$$

Let

$$\begin{aligned} C_d &= \frac{\left(\binom{d+1}{t} - \binom{d}{t}\right)\binom{L-d}{w} t!w!}{\binom{L+1}{t+w}(t+w)!} \\ &= \left(\frac{(d+1)!}{(d+1-t)!} - \frac{d!}{(d-t)!}\right)\frac{(L-d)!}{(L-d-w)!} \cdot \frac{(L+1-t-w)!}{(L+1)!} \\ &= \frac{t \cdot d!}{(d+1-t)!} \cdot \frac{(L-d)!}{(L-d-w)!} \cdot \frac{(L+1-t-w)!}{(L+1)!}. \end{aligned}$$

C_d increases at d (as a function of d) if

$$C_{d-1} < C_d \Leftrightarrow d < \frac{(L+1)(t-1)}{t+w-1}.$$

Therefore C_d attains its maximum at

$$d^* = \left\lceil \frac{(L+1)(t-1)}{t+w-1} - 1 \right\rceil,$$

and we can solve the problem (2) with an optimal solution

$$(D^*, m_1^*, \ldots, m_D^*) = (d^*, \overbrace{1, \ldots, 1}^{d^*-1}, L - d^*),$$

and the optimal value

$$\begin{aligned}
\bar{p}^*_{t,w,L} &= \frac{\sum_{d=t-1}^{d^*} m_d^* \left(\binom{d+1}{t} - \binom{d}{t}\right) \binom{L-d}{w} t!w!}{\binom{L+1}{t+w} (t+w)!} \\
&= \frac{\sum_{d=t-1}^{d^*} m_d^* \left((d+1)!/(d+1-t)! - d!/(d-t)!\right) \left((L-d)!/(L-d-w)!\right)}{(L+1)!/(L+1-t-w)!} \\
&= \frac{(L+1-t-w)!}{(L+1)!} \sum_{d=t-1}^{d^*} \frac{m_d^* t \cdot d!}{(d+1-t)!} \cdot \frac{(L-d)!}{(L-d-w)!} \\
&\geq \frac{(L+1-t-w)!}{(L+1)!} \cdot \frac{(L-d^*)t \cdot d^*!}{(d^*+1-t)!} \cdot \frac{(L-d^*)!}{(L-d^*-w)!} \\
&= \frac{(L-d^*)t \cdot \overbrace{d^*(d^*-1)\ldots(d^*-t+2)}^{t-1} \cdot \overbrace{(L-d^*)(L-d^*-1)\ldots(L-d^*-w+1)}^{w}}{\underbrace{(L+1)L\ldots(L-t-w+2)}_{t+w}} \\
&= w\Omega\left(\frac{t!w!}{(t+w)!}\right),
\end{aligned}$$

since

$$\frac{d^*}{L} \approx \frac{t}{t+w}$$

for a sufficiently large L.
The optimal tree looks like a claw, as seen in Fig. 3. Now we conclude that

1. the probability $\bar{p}_{t,w}$ based on the optimal tree is greater than the one based on a chain (asymptotically) by a factor of w, and
2. the maximum hash depth applied to a secret seed is $d^* = \lceil (L+1)(t-1)/(t+w-1) - 1 \rceil$, which is relatively small for a large w.

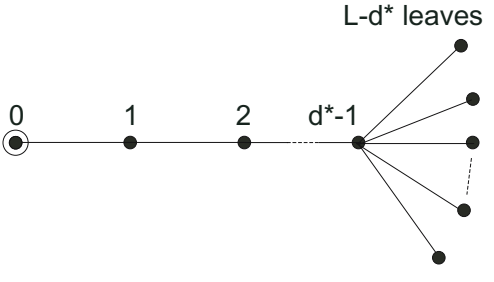


Fig. 3. Optimal tree

Example 4.4. Let $n = L = 1000$, $t = 2$ and $w = 100$. Then we have $d^* = \left\lceil \frac{1001}{101} - 1 \right\rceil = 9$, and $\bar{p}^*_{t,w,L} \geq 0.00764$, while $\frac{t!w!}{(t+w)!} = 0.00019$. By the condition (1), there exists a $(2, 100, \mathcal{T}^*)$-TKDP$(91872, 1000)$, where $\mathcal{T}^*$ is a claw of depth $d^* = 9$.

5 Conclusion

We presented a new framework of tree-based key distribution pattern (TKDP). We constructed TKDPs from cover-free families. We note that other combinatorial structures such as orthogonal arrays, ordered designs and covering arrays also yield TKDPs. (This will be addressed in later work.) We showed the existence of TKDPs by the probabilistic method. Furthermore, we reduced the upper bounds on the minimum number of rows of $(t, w, \mathcal{T})$-TKDPs, which are obtained from probabilistic methods, asymptotically by a factor of w as compared to Leighton and Micali's schemes by choosing optimal trees $\mathcal{T}$ instead of chains. The TKDP-based schemes are expected to have applications to sensor networks since they involve only hash computations in the establishment of pairwise keys, which are known to be more energy-efficient than RSA or elliptic curve operations.

References

1. N. Alon and J. Spencer. *The Probabilistic Method*, Wiley, New York, 1992.
2. N. Attrapadung, K. Kobara and H. Imai. Sequential key derivation patterns for broadcast encryption and key predistribution schemes, *Lecture Notes in Computer Science*, **2894** (2003), 374-391 (Advances in Cryptology - ASIACRYPT '03).
3. R. Blom. An Optimal Class of Symmetric Key Generation Systems, *Lecture Notes in Computer Science*, **209** (1985), 335-338 (Advances in Cryptology - EUROCRYPT '84).
4. C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro and M. Yung. Perfectly-secure key distribution for dynamic conferences, *Lecture Notes in Computer Science*, **740** (1993), 148-168 (Advances in Cryptology - EUROCRYPT '92).
5. H. Chan, A. Perrig, and D. Song. Random Key Predistribution Schemes for Sensor Networks, In *IEEE Symposium on Research in Security and Privacy*, 197–213, May 2003.
6. C.J. Colbourn and J.H. Dinitz, editors. *The CRC Handbook of Combinatorial Designs*, CRC Press, Boca Raton, 1996.
7. M. Dyer, T. Fenner, A. Frieze and A. Thomason, On key storage in secure networks. *Journal of Cryptology*, **8** (1995), 189-200.
8. L. Eschenauer and V.D. Gligor. A Key-Management Scheme for Distributed Sensor Networks, In *Proceedings of the 9th ACM conference on Computer and communications security*, 41–47, November 2002.
9. J. Lee and D.R. Stinson. A combinatorial approach to key predistribution for distributed sensor networks. *the IEEE Wireless Communications and Networking Conference*, CD-ROM, 2005, paper PHY53-06, 6–11, http:// www.cacr.math.uwaterloo.ca/ dstinson/pubs.html.

10. J. Lee and D.R. Stinson. Deterministic key predistribution schemes for distributed sensor networks. *Lecture Notes in Computer Science* **3357** (2004), 294–307 (SAC 2004 Proceedings).
11. T. Leighton and S. Micali, Secret-key agreement without public-key cryptography, *Lecture Notes in Computer Science*, **773** (1994), 456-479 (Advances in Cryptology - CRYPTO '93).
12. D. Liu and P. Ning, Establishing Pairwise Keys in Distributed Sensor Networks, In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, 52–61, October 2003.
13. C.J. Mitchell and F.C. Piper, Key storage in secure networks. *Discrete Applied Mathematics*, **21** (1988), 215-228.
14. C. Padró, I. Grasia, S.M. Molleví and P. Morillo, Linear key predistribution schemes. *Designs, Codes and Cryptography*, **25** (2002), 281-298.
15. M. Ramkumar and N. Memon, HARPS-Hashed Random Preloaded Subset Key Distribution. *Cryptology ePrint Archive*, Report 2003/170 (2003), http://eprint.iacr.org/2003/170.
16. D.R. Stinson and Tran van Trung, Some new results on key distribution patterns and broadcast encryption. *Designs, Codes and Cryptography*, **14** (1998), 261-279.
17. D.R. Stinson and R. Wei, Generalized cover-free families. *Discrete Mathematics*, **279** (2004), 463-477.
18. R. Wei, On cover-free families. *Discrete Mathematics*, to appear.

Provably Secure Tripartite Password Protected Key Exchange Protocol Based on Elliptic Curves*

Sanggon Lee[1], Yvonne Hitchcock[2,**], Youngho Park[3], and Sangjae Moon[4]

[1] Division of Internet Engineering, Dongseo University,
Busan 617-716, Korea
nok60@dongseo.ac.kr
[2] Information Security Institute, Queensland University of Technology,
GPO Box 2434 BRISBANE 4001 Australia
hitchcock@isrc.qut.edu.au
[3] School of Electronics and Electrical Engineering,
Sangju National University, Sangju-si, Gyeongsangbuk-do 742-771, Korea
yhpark@sangju.ac.kr
[4] School of Electrical Engineering and Computer Science,
Kyungpook National University, Daegu 702-701, Korea
sjmoon@ee.knu.ac.kr

Abstract. Joux's tripartite key agreement protocol is one of the most prominent developments in the area of key agreement. Although certificate-based and ID-based authentication schemes have been proposed to provide authentication for Joux's protocol, no provably secure password-based one round tripartite key agreement protocol has been proposed yet. We propose a secure one round password-based tripartite key agreement protocol that builds on Joux's protocol and adapts PAK-EC scheme for password-based authentication, and present a proof of its security.

Keywords: Tripartite key agreement; password-based authentication; provable security; bilinear Diffie-Hellman problem; Joux's protocol.

1 Introduction

A *Key agreement protocol* is the mechanism by which two or more parties can establish a common secret key over a network controlled by an adversary. This secret key is commonly called a session key and can then be used to create a secure communications channel among the parties.

The situation where three or more parties share a key is often called conference keying. The three-party (or tripartite) case is of the most practical importance, not only because it is the most common size for electronic conferences, but also

* This work was supported by University IT Research Center Project of MIC, Korea.
** Research funded by Australian Research Council through Discovery Project DP0345775.

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 205–220, 2006.

because it can be used to provide a range of services for two communicating parties. For example, a third party can be added to chair, or referee a conversation for ad hoc auditing. Also, a three-party key agreement protocol can be used for tree based group key agreement protocols [12].

Joux's tripartite key agreement protocol [10] is one of the most prominent developments in the area of key agreement. This protocol makes use of pairings on elliptic curves and requires each entity to transmit only a single broadcast messages. This should be contrasted with the obvious extension of the Diffie-Hellman protocol to three parties, which requires two broadcasts per entity. However, like the basic Diffie-Hellman protocol, Joux's protocol also suffers from man-in-the-middle attacks because it does not provide key authentication.

To transform Joux's protocol into a secure tripartite protocol that only requires one round, many protocols have been proposed, both certificate-based [1, 17] and ID-based [18]. Another method of authentication is to make use of a password [4, 15, 16]. Certificate-based authentication requires a certificate authority and ID-based authentication requires a trusted dealer with a universal secret key. However, password based authentication does not require any trusted third party. No provably secure password-based one round tripartite key agreement protocol has been proposed so far in the literature.

PAK-EC [15] is a two party password authenticated key agreement protocol built on elliptic curves. Our contribution is to present a provably secure one round password-based tripartite key agreement protocol that builds on Joux's protocol and adapts the PAK-EC scheme for password-based authentication.

In Sect. 2, we describe the model used in the security proof, and in Sect. 3, the proposed protocol is described. In Sect. 4, we prove the protocol secure, and in Sect. 5, we compare the efficiency of our protocol with another one.

2 Security Model

For our proof of security we use the model of Bellare, Pointcheval and Rogaway [3] (which is used by Kate et al. [11] and MacKenzie [16]), and adopt MacKenzie's approach [16]. Our model is for implicitly authenticated key exchange between parties *A, B* and *C* who share a secret. The goal is for them to engage in a protocol such that after the protocol is completed, they each hold a session key that is known to nobody but the three of them. In the following, we will describe our model.

Let I be a nonempty set of participants. We assume each participant $U \in I$ is labeled by a string, and we simply use U to denote this string. We will also use *A,B,C,...* to refer to protocol participants. Each group of three participants, $A, B, C \in I$, who will set up a secret key shared amongst themselves are assumed to share a secret password with each other, π_{ABC}, before the protocol begins.

For a protocol P, each participant is able to execute P multiple times with different partners, and we model this by allowing unlimited number of *instances* of each participant. Instance i (or session number i) of participant $U \in I$ is denoted $\prod_i^U$. To describe the security of the protocol, we assume there is an adversary

$\mathcal{A}$ that has complete control over the environment (mainly the network), and thus provides the input to instances of participants. Formally, the adversary is a probabilistic algorithm with a distinguished query tape. Participants respond to queries written to this tape according to P; the allowed queries are based on and extend the model of Bellare et al. [3]. Oracles exist in one of several possible states: *Accept, Reject, or* $*$. The state $*$ means no decision has yet been reached. In our protocol, an oracle accepts only after receipt of two correctly formatted messages from the two other participants with whom the oracle wishes to establish a shared key, and the transmission of one message. When an oracle accepts, we assume it accepts holding key K that is κ bits in length.

Send (U, i, M)**:** Causes message M to be sent to instance $\prod_i^U$. The instance computes what the protocol says to, the oracle's state is updated, and any outgoing messages are given to $\mathcal{A}$. If this query causes $\prod_i^U$ to accept or terminate, this will also be shown to $\mathcal{A}$. To initiate a session between three participants, the adversary should send a message containing the names of two participants to an unused instance of the other participant.

Execute(*A,i,B,j,C,l***):** Causes P to be executed to completion between $\prod_i^A, \prod_j^B$ and $\prod_l^C$ (where $A, B, C \in I$), and outputs the transcript of the execution. This query captures the intuition of a passive adversary who simply eavesdrops on the execution of P.

Reveal(*U,i***):** Causes the output of the session key held by $\prod_i^U$.

Test(*U,i***):** Causes $\prod_i^U$ to flip a bit b. if $b = 1$ the session key sk_U^i is output; otherwise, a string is drawn uniformly from the space of session keys and output. A **Test** query may be asked at any time during the execution of P, but may only be asked once.

Corrupt(*U***):** This query returns any passwords that U holds.

Partnering: A participant instance that accepts holds a partner-id *pid*, session-id *sid*, and a session key *sk*. Then instances $\prod_i^A, \prod_j^B$, and $\prod_l^C$ (where $A, B, C \in I$) are said to be partnered if all of them accept, they hold (pid_A, sid_A, sk_A), (pid_B, sid_B, sk_B) and (pid_C, sid_C, sk_C), respectively, with $pid_A = \langle B, C\rangle$, $pid_B = \langle A, C\rangle$, $pid_C = \langle A, B\rangle$, $sid_A = sid_B = sid_C$, and $sk_A = sk_B = sk_C$, and no other instance accepts with session-id equal to sid_A, sid_B or sid_C.

Freshness: We define two notions of freshness, as in [3]. Specifically, an instance $\prod_i^U$ is nfs-fresh (fresh with no requirement for forward secrecy) unless either (1) a **Reveal(***U,i***)** query occurs, (2) a **Reveal**(U', j) query occurs where $\prod_j^{U'}$ is a partner of $\prod_i^U$, or (3) a **Corrupt**(U') query occurs for any party U' (for convenience, when we do not make a requirement for forward secrecy, we simply disallow **Corrupt** queries). An instance $\prod_i^U$ is fs-fresh (fresh with forward secrecy) unless either (1) a **Reveal**(U, i) query occurs, (2) a **Reveal**(U', j) query occurs where $\prod_j^{U'}$ is the partner of $\prod_i^U$, or (3) a **Corrupt**(U') query occurs for any party U' before the **Test** query and a **Send**(U, i, M) query occurs for some string M.

We now formally define the authenticated key exchange (*ake*) advantage of the adversary against protocol P. Let $Succ_P^{ake}(\mathcal{A})$ be the event that $\mathcal{A}$ makes a

single **Test** query directed to some fresh instance $\prod_i^U$ that has terminated, and eventually outputs a bit b', where $b' = b$ for the bit b that was selected in the **Test** query. The *ake* advantage of $\mathcal{A}$ attacking P is defined to be

$$Adv_P^{ake}(\mathcal{A}) \stackrel{def}{=} 2Pr[Succ_P^{ake}(\mathcal{A})] - 1.$$

The following fact is easily verified.

Fact 1. $\Pr\big(Succ_P^{ake}(\mathcal{A})\big) = \Pr\big(Succ_{P'}^{ake}(\mathcal{A})\big) + \varepsilon \Leftrightarrow Adv_P^{ake}(\mathcal{A}) = Adv_{P'}^{ake}(\mathcal{A}) + 2\varepsilon.$

3 Password and Pairings-Based Tripartite Key Exchange

We briefly describe some background on pairings on elliptic curves and the BDH assumption, and then present our new tripartite PPK (password protected key exchange) protocol based on Joux's protocol [10].

3.1 Bilinear Pairings and the BDH Assumption

We use the same notation as in [5]. Let G_1 be a cyclic additive group generated by Q, whose order is a prime q, and G_2 be a cyclic multiplicative group of the same order q. We assume that the discrete logarithm problem (DLP) in both G_1 and G_2 is hard. Let $e : G_1 \times G_1 \to G_2$ be a pairing which satisfies the following conditions (where $W, X, Z \in G_1$):

1. Bilinear: $e(W, X+Z) = e(W, X)\cdot(W, Z)$ and $e(W+X, Z) = e(W, Z)\cdot(X, Z)$;
2. Non-degenerate: $e(Q, Q)$ is a generator of G_2 ;
3. Computability: $e(P, Q)$ can be efficiently computed for all $P, Q \in G_1$.

The Weil or Tate pairing on an elliptic curve can be used to derive e [8, 14].

Definition 1 (Bilinear Diffie-Hellman (BDH) Assumption). *Let G_1 and G_2 be as defined above with generators Q and $e(Q, Q)$ respectively. Let* ACCEPTABLE(v) *be a function that returns true if and only if $v \in G_1$. For three values X, Y, and Z, if* ACCEPTABLE(Y)*, and* ACCEPTABLE(Z)*, and $X = aQ$, let* $\mathbf{BDH}(X, Y, Z) = \mathbf{BDH}(X, Z, Y) = \mathbf{BDH}(Y, X, Z) = \mathbf{BDH}(Y, Z, X) = \mathbf{BDH}(Z, X, Y) = \mathbf{BDH}(Z, Y, X) = e(Y, Z)^a$*. (If $X = aQ, Y = bQ$, and $Z = cQ$, then by the definition $\mathbf{BDH}(X, Y, Z) = e(Q, Q)^{abc}$.) Let $\mathcal{D}$ be an algorithm with input (X,Y,Z). Let*

$$Adv_{G_1G_2}^{BDH}(\mathcal{D}) \stackrel{def}{=} Pr\Big[(a, b, c) \stackrel{R}{\leftarrow} Z_q^*; X \leftarrow aQ; Y \leftarrow bQ; Z \leftarrow cQ : \mathbf{BDH}(X, Y, Z) \in \mathcal{D}(X, Y, Z)\Big]$$

Let $Adv_{G_1G_2}^{BDH}(t, n) = max_{\mathcal{D}}\big\{Adv_{G_1G_2}^{BDH}(\mathcal{D})\big\}$, where the maximum is taken over all adversaries of time complexity at most t that output a list containing at most n elements of G_2. The BDH assumption states that for t and n polynomial in the security parameter κ, $Adv_{G_1G_2}^{BDH}(t, n)$ is negligible.

3.2 One Round Tripartite PPK Based on Joux's Protocol

Figure 1 presents a new tripartite PPK protocol based on Joux's protocol. In the protocol, $f_i(A, B, C, \pi)$ is defined to be a function generating a random point on elliptic curve E from A, B, C, and π, as specified by MacKenzie [15], who adapted it from IEEE Standard 1363 [9, Appendix A.11.1].

The protocol can be converted to one round by having A, B and C compute and broadcast m, μ and υ respectively to begin the protocol. Each party then waits for the other two parties' messages and carries out its remaining steps (i.e. B and C now carry out their protocol steps in the same order as is currently specified for A). Such a reordering does not affect the security proof.

We use the terminology "in a PARTICIPANT U ACTION i query to $\prod_j^U$" to mean "in a **Send** query to $\prod_j^U$ that results in the PARTICIPANT U ACTION i procedure being executed." The possible actions with their associated inputs and outputs are shown in Table 1, where A is the initiator, B is the second participant and C is the third participant.

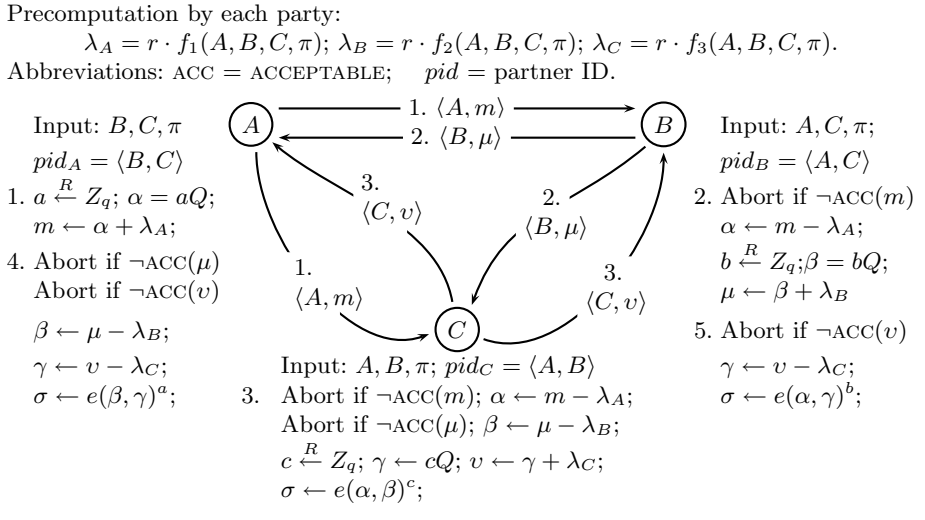


Fig. 1. Tripartite PPK protocol. Session ID is $sid = A||B||C||m||\mu||\upsilon$. Shared session key is $sk = H(\langle A, B, C, m, \mu, \upsilon, \sigma, \lambda_A, \lambda_B, \lambda_C \rangle)$.

Table 1. Inputs and outputs for the participant queries

	A (initiator)		B (second participant)		C (third participant)	
	Input	Output	Input	Output	Input	Output
Action 0	$\langle B, C \rangle$	$\langle A, m \rangle$	$\langle A, m \rangle$	$\langle B, \mu \rangle$	$\langle A, m, B, \mu \rangle$	$\langle C, \upsilon \rangle$
Action 1	$\langle B, \mu, C, \upsilon \rangle$		$\langle C, \upsilon \rangle$			

Let κ be the cryptographic security parameter with $|q| = \kappa$. We use an elliptic curve E over the integers modulo p with coefficients a, b in standard Weiestrass form and $\#E = rq$, with $\gcd(r, q) = 1$. (Currently, $|p| = 162$ and $|q| = 160$ would be considered reasonably secure [9]). The complete specification is below:

$sid \leftarrow pid \leftarrow sk \leftarrow \varepsilon$; $acc \leftarrow term \leftarrow FALSE$
if $state =$ READY **and** ($U \in I$ AND U is the initiator) **then**
{PARTICIPANT A ACTION 0} $\langle A \rangle \leftarrow U$; $\langle B, C \rangle \leftarrow msg - in$ where $B, C \in I$; $a \stackrel{R}{\leftarrow} Z_q$; $\alpha = aQ$; $\lambda_A = r \cdot f_1(A, B, C, \pi)$; $m \leftarrow \alpha + \lambda_A$; $state \leftarrow \langle A, a, m, \lambda_A \rangle$; $msg - out \leftarrow \langle A, m \rangle$; **return** $(msg - out, acc, term, sid, pid, sk, state)$
elseif $state =$ READY **and** ($U \in I$ AND U is the second participant) **then**
{PARTICIPANT B ACTION 0} $\langle B \rangle \leftarrow U$; $\langle A, m \rangle \leftarrow msg - in$, where $A \in I$ and ACCEPTABLE(m); $\lambda_A = r \cdot f_1(A, B, C, \pi)$; $\alpha \leftarrow m \leftarrow \lambda_A$; $b \stackrel{R}{\leftarrow} Z_q$; $\beta = bQ$; $\lambda_B = r \cdot f_2(A, B, C, \pi)$; $\mu \leftarrow \beta + \lambda_B$; $state \leftarrow \langle A, a, m, B, b, \mu, \lambda_A, \lambda_B \rangle$, $msg - out \leftarrow \langle B, \mu \rangle$; **return** $(msg - out, acc, term, sid, pid, sk, state)$
elseif $state =$ READY **and** ($U \in I$ AND U is the third participant) **then**
{PARTICIPANT C ACTION 0} $\langle C \rangle \leftarrow U$; $\langle A, m, B, \mu \rangle \leftarrow msg - in$, where $A, B \in I$ and ACCEPTABLE(m) and ACCEPTABLE(μ); $\lambda_A = r \cdot f_1(A, B, C, \pi)$; $\alpha \leftarrow m - \lambda_A$; $\lambda_B = r \cdot f_2(A, B, C, \pi)$; $\beta \leftarrow \mu - \lambda_B$; $c \stackrel{R}{\leftarrow} Z_q$; $\gamma = cQ$; $\lambda_C = r \cdot f_3(A, B, C, \pi)$; $\upsilon \leftarrow \gamma + \lambda_C$; $\sigma \leftarrow e(\alpha, \beta)^c$; $state \leftarrow DONE$; $msg - out \leftarrow \langle C, \upsilon \rangle$; $sid \leftarrow A||B||C||m||\mu||\upsilon$; $pid \leftarrow \langle A, B \rangle$; $sk \leftarrow H(\langle A, B, C, m, \mu, \upsilon, \sigma, \lambda_A, \lambda_B, \lambda_C \rangle)$; $acc \leftarrow term \leftarrow TRUE$; **return** $(msg - out, acc, term, sid, pid, sk, state)$
elseif $state = \langle A, a, m, \lambda_A \rangle$ **and** ($U \in I$ AND U is the initiator) **then**
{PARTICIPANT A ACTION 1} $\langle B, \mu, C, \upsilon \rangle \leftarrow msg - in$, where $B, C \in I$ and ACCEPTABLE(μ) and ACCEPTABLE(υ); $\lambda_B = r \cdot f_2(A, B, C, \pi)$; $\lambda_C = r \cdot f_3(A, B, C, \pi)$; $\beta \leftarrow \mu - \lambda_B$; $\gamma \leftarrow \upsilon - \lambda_C$; $\sigma \leftarrow e(\beta, \gamma)^a$; $state \leftarrow DONE$; $msg - out \leftarrow \varepsilon$; $sid \leftarrow A||B||C||m||\mu||\upsilon$; $pid \leftarrow (\langle B, C \rangle)$; $sk \leftarrow H(\langle A, B, C, m, \mu, \upsilon, \sigma, \lambda_A, \lambda_B, \lambda_C \rangle)$; $acc \leftarrow term \leftarrow TRUE$; **return** $(msg - out, acc, term, sid, pid, sk, state)$
elseif $state = \langle A, a, m, B, b, \mu, \lambda_A, \lambda_B \rangle$ **and** ($U \in I$ AND U is the second participant) **then**
{PARTICIPANT B ACTION 1} $\langle C, \upsilon \rangle \leftarrow msg - in$, where $C \in I$ and ACCEPTABLE(υ); $\lambda_C = r \cdot f_3(A, B, C, \pi)$; $\gamma \leftarrow \upsilon - \lambda_c$; $\sigma \leftarrow e(\alpha, \gamma)^b$; $state \leftarrow DONE$; $msg - out \leftarrow \varepsilon$; $sid \leftarrow A||B||C||m||\mu||\upsilon$; $pid \leftarrow \langle A, C \rangle$; $sk \leftarrow H(\langle A, B, C, m, \mu, \upsilon, \sigma, \lambda_A, \lambda_B, \lambda_C \rangle)$; $acc \leftarrow term \leftarrow TRUE$; **return** $(msg - out, acc, term, sid, pid, sk, state)$

4 Security of the Protocol

Here we prove that the tripartite PPK protocol is secure, in the sense that an adversary attacking the system cannot determine session keys of fresh instances with greater advantage than that of an online dictionary attack.

Theorem 2. *Let P be the protocol described in Fig. 1 (and formally described above), using groups G_1 and G_2 of order q, with a password dictionary of size N.*

Fix an adversary $\mathcal{A}$ that runs in time t, and makes n_{se}, n_{ex}, and n_{re} queries of type **Send**, **Execute**, *and* **Reveal**, *respectively, and n_{ro} queries to the random oracles. Let t_{op} be the time required to perform a scalar multiplication and a pairing of elliptic curve point in G_1 and an exponentiation in G_2. Then for $t' = O\big(t + (n_{ro}^3 + n_{se} + n_{ex})t_{op}\big)$:*

$$Adv_P^{ake}(\mathcal{A}) = \frac{2n_{se}}{N} + O\left(Adv_{G_1G_2}^{BDH}\left(t', n_{ro}^3\right) + \frac{(n_{se}+n_{ex})(n_{ro}+n_{se}+n_{ex})}{q}\right)$$

Proof. The proof proceeds by introducing a series of protocols $P_0, P_1, \cdots, P_8$ related to P, with $P_0 = P$. In P_8, $\mathcal{A}$ is reduced to a simple online guessing attack that admits straightforward analysis. For each i from 1 to 8, we will prove that the advantage of $\mathcal{A}$ attacking protocol P_{i-1} is at most negligibly more than the advantage of $\mathcal{A}$ attacking protocol P_i. An informal description of the protocols as well as a brief description in brackets of the basis for each proof follows:

P_0 The original protocol P.

P_1 If honest parties choose m, μ, or v values used previously in the protocol, it halts and $\mathcal{A}$ fails. (The probability of collision of nonces is negligible.)

P_2 The protocol answers **Send** and **Execute** queries without making any random oracle queries. Subsequent queries by $\mathcal{A}$ are backpatched, as much as possible, to be consistent with responses to **Send** and **Execute** queries. (This is consistent with P_1 unless $\mathcal{A}$ guesses the output of $f_i(A, B, C, \pi_{ABC})$ queries correctly and uses the guesses in $H(\cdot)$ queries (for $i \in \{1, 2, 3\}$). However, the probability of correctly guessing the outputs is negligible.)

P_3 If an $H(\cdot)$ query is made, it is not checked for consistency against **Execute** queries. That is, instead of backpatching to maintain consistency with an **Execute** query, the protocol responds with a random output. (An instance of the BDH problem can be embedded in the m μ and v values so that if backpatching would have been necessary, $H(\cdot)$ query inputs can be used to solve the BDH problem.)

P_4 If a correct password guess is made against any participant instance (determined by an $H(\cdot)$ query using the correct inputs to compute a session key), the protocol halts and $\mathcal{A}$ automatically succeeds. (This is obvious.)

P_5 If the adversary makes three password guesses against a Participant A instance, the protocol halts and $\mathcal{A}$ fails. (This is shown by embedding an instance of the BDH problem in the f_i queries and A's output. Inputs to the three $H(\cdot)$ queries used to find the session keys can be used to solve the BDH problem.)

P_6 **and** P_7 are similar to P_5, but for B and C respectively, instead of A.

P_8 The protocol uses an internal password oracle that holds all passwords and only accepts simple queries that test whether a given password is correct password for a given three parties. The test for correct password guesses (from P_4) is changed so that whenever the adversary makes a password guess, a query is submitted to the oracle to determine if it is correct. (By inspection P_7 and P_8 are indistinguishable.)

We assume without loss of generality that n_{ro} and $n_{se} + n_{ex}$ are both at least 1. We make the standard assumption that random oracles are built "on the fly," that is, each new query to a random oracle is answered with a fresh random output, and each query that is not new is answered consistently with the previous queries. We also assume that the $f_j(\cdot)$ query is answered in the following way:

In an $f_j(A, B, C, \pi)$ query for $j \in \{1, 2, 3\}$, output $\phi_j[A, B, C, \pi]Q$, where $\phi_j[A, B, C, \pi] \xleftarrow{R} Z_q$. Also put $\psi_j[A, B, C, \pi] = r\phi_j[A, B, C, \phi]$, $\lambda_A \leftarrow r\phi_1[A, B, C, \pi]Q$, $\lambda_B \leftarrow r\phi_2[A, B, C, \pi]Q$, and $\lambda_C \leftarrow r\phi_3[A, B, C, \pi]Q$. Denote $\psi_j[A, B, C, \pi]$ and $\phi_j[A, B, C, \pi]$ as $\psi_j[\pi]$ and $\phi_j[\pi]$ respectively. Thus $\psi_j[\pi] = r\phi_j[\pi]$.

We now define some events, corresponding to the adversary making a password guess against a participant instance, and against three participant instances that are partnered in an **Execute** query. In each case, we also define an associated value for the event, and we note that the associated value is actually fixed by the protocol before the event occurs.

testpw (U, i, V, W, π)**:** This is the event that the adversary makes a password guess against $\prod_i^U$ with $pid_U = \langle V, W\rangle$. Let $\{U, V, W\} = \{A, B, C\}$ where A is initiator, B is the second participant and C is the third participant. For some $m, \mu, \upsilon, \lambda_A, \lambda_B$, and λ_C, $\mathcal{A}$ makes an $H(\langle A, B, C, m, \mu, \upsilon, \sigma, \lambda_A, \lambda_B, \lambda_C\rangle)$ query, and if $U = A$, $\mathcal{A}$ makes a PARTICIPANT U ACTION 0 query with input $\langle B, C\rangle$ and output $\langle A, m\rangle$, and a PARTICIPANT U ACTION 1 query with input $\langle B, \mu, C, \upsilon\rangle$ to $\prod_i^U$. Otherwise, if $U = B$, $\mathcal{A}$ makes PARTICIPANT U ACTION 0 query with input $\langle A, m\rangle$ and output $\langle B, \mu\rangle$, and a PARTICIPANT U ACTION 1 query with input $\langle C, \upsilon\rangle$ to $\prod_i^U$. Otherwise, since $U = C$, $\mathcal{A}$ makes PARTICIPANT U ACTION 0 query with input $\langle A, m, B, \mu\rangle$ and output $\langle C, \upsilon\rangle$ to $\prod_i^U$. $\mathcal{A}$ also makes an $f_1(A, B, C, \pi)$ query returning $\phi_1[\pi]Q$, an $f_2(A, B, C, \pi)$ query returning $\phi_2[\pi]Q$, an $f_3(A, B, C, \pi)$ query returning $\phi_3[\pi]Q$, where $\sigma = \mathbf{BDH}(\alpha, \beta, \gamma)$, $m = \alpha + \lambda_A$, $\mu = \beta + \lambda_B$, $\upsilon = \gamma + \lambda_C$, $\lambda_A = \psi_1[\pi]Q$, $\lambda_B = \psi_2[\pi]Q$, $\lambda_C = \psi_3[\pi]Q$, ACCEPTABLE (m), ACCEPTABLE (μ) and ACCEPTABLE (υ). The event's associated value is $sk_U^i = H(\langle A, B, C, m, \mu, \upsilon, \sigma, \lambda_A, \lambda_B, \lambda_C\rangle)$.

testexecpw (A, i, B, j, C, l, π)**:** This is the event that the adversary makes a password guess against three instances that are partnered in an **Execute** query. For some $m, \mu, \upsilon, \lambda_A, \lambda_B$ and λ_C, $\mathcal{A}$ makes an $H(\langle A,B,C,m,\mu,\upsilon,\sigma,\lambda_A, \lambda_B, \lambda_C\rangle)$ query, and previously $\mathcal{A}$ made an **Execute**(A, i, B, j, C, l) query that generated m, μ, υ, and $f_1(A, B, C, \pi)$, $f_2(A, B, C, \pi)$, and $f_3(A, B, C, \pi)$ queries returning $\phi_1[\pi]Q$, $\phi_2[\pi]Q$, and $\phi_2[\pi]Q$, where $\lambda_A = \psi_1[\pi]Q$, $\lambda_B = \psi_2[\pi]Q$, $\lambda_C = \psi_3[\pi]Q$, $\sigma = \mathbf{BDH}(\alpha, \beta, \gamma)$, $m = \alpha + \lambda_A$, $\mu = \beta + \lambda_B$, and $\upsilon = \gamma + \lambda_C$. The associated value of this event is $sk_A^i = sk_B^j = sk_C^l = H(\langle A, B, C, m, \mu, \upsilon, \sigma, \lambda_A, \lambda_B, \lambda_C\rangle)$.

correctpw: A **testpw**(U, i, V, W, π_{UVW}) event occurred, for some U, i, V, W, where π_{UVW} is the password shared between U, V, and W.

correctpwexec: A **testexecpw**$(A, i, B, j, C, l, \pi_{ABC})$ event occured for A, i, B, j, C, and l, where π_{ABC} is the password shared between A, B, and C.

triplepw($\mathcal{U}$): A **testpw**(U, i, V, W, π) event, a **testpw**$(U, i, V, W, \hat{\pi})$ event and a **testpw**$(U, i, V, W, \tilde{\pi})$ occurred, for some $U, i, V, W, \pi, \hat{\pi}$, and $\tilde{\pi}$ with $\pi \neq \hat{\pi} \neq \tilde{\pi} \neq \pi$.

Protocol P_1. Let E_1 be the event that an m value generated in a PARTICIPANT A ACTION 0 or **Execute** query is equal an m value generated in a previous PARTICIPANT A ACTION 0 or **Execute** query, an m value sent as input in a previous PARTICIPANT B ACTION 0, or PARTICIPANT C ACTION 0 query, or m in a previous $f_i(\cdot)$ query (made by the adversary). Let E_2 be the event that a μ value generated in a PARTICIPANT B ACTION 0 or **Execute** query is equal to a μ value generated in a previous PARTICIPANT B ACTION 0 or **Execute** query, a μ sent as input in a previous PARTICIPANT A ACTION 1 or PARTICIPANT C ACTION 0, or μ value in a previous $f_j(\cdot)$ query (made by the adversary). Let E_3 be the event that a υ value generated in a PARTICIPANT C ACTION 0 or **Execute** query is equal to a υ value generated in a previous PARTICIPANT C ACTION 0 or **Execute** query, a υ sent as input in a previous PARTICIPANT A ACTION 1 or PARTICIPANT B ACTION 1 query, or υ value in a previous $f_j(\cdot)$ query (made by the adversary). Let $E = E_1 \vee E_2 \vee E_3$. Let P_1 be a protocol that is identical to P_0 except that if E occurs, the protocol aborts (and thus the adversary fails).

Theorem 3. *For an adversary $\mathcal{A}$,*

$$Adv_{P_0}^{ake}(\mathcal{A}) \leq Adv_{P_1}^{ake}(\mathcal{A}) + \frac{O\big((n_{se}+n_{ex})(n_{ro}+n_{se}+n_{ex})\big)}{q}.$$

Proof. Consider the last m, μ, or υ value generated. There is a probability of no more than $\frac{n_{ro}+n_{se}+n_{ex}}{q}$ that this value has previously been generated in a **Send**, **Execute** or Random oracle query. There are $n_{se} + n_{ex}$ values that are required to be unique if event E is not to occur. Hence the probability of any of the m, μ, or υ values not being unique is $\frac{O\big((n_{se}+n_{ex})(n_{se}+n_{ex}+n_{ro})\big)}{q}$, and the theorem follows. □

Protocol P_2. Let P_2 be a protocol that is identical to P_1 except that **Send** and **Execute** queries are answered without making any random oracle queries, and subsequent random oracle queries by the adversary are backpatched, as much as possible, to be consistent with the responses to the **Send** and **Execute** queries. Specifically, the queries in P_2 are changed as follows:

Execute (A, i, B, j, C, l)**:** $m \leftarrow \tau[i, A]Q$, where $\tau[i, A] \xleftarrow{R} Z_q$, $\mu \leftarrow \tau[j, B]Q$, where $\tau[j, B] \xleftarrow{R} Z_q$, $\upsilon \leftarrow \tau[l, C]Q$, where $\tau[l, C] \xleftarrow{R} Z_q$ and $sk_A^i \leftarrow sk_B^j \leftarrow sk_C^l \xleftarrow{R} \{0,1\}^\kappa$.

PARTICIPANT A ACTION 0 to $\prod_i^A$: $m \leftarrow \tau[i, A]Q$, where $\tau[i, A] \xleftarrow{R} Z_q$.

PARTICIPANT B ACTION 0 to Π_j^B: $\mu \leftarrow \tau[j, B]Q$, where $\tau[j, B] \xleftarrow{R} Z_q$.

PARTICIPANT C ACTION 0 to Π_l^C: $\upsilon \leftarrow \tau[l, C]Q$, where $\tau[l, C] \xleftarrow{R} Z_q$, and $sk_C^l \xleftarrow{R} \{0,1\}^\kappa$.

PARTICIPANT A ACTION 1 to $\prod_i^A$: if $\prod_l^C$ is paired with instance $\prod_i^A$ and $\prod_j^B$, $sk_A^i \leftarrow sk_B^j \leftarrow sk_C^l$, else if $\prod_l^C$ is paired with instance $\prod_i^A$, $sk_A^i \leftarrow sk_C^l$, else if $\prod_j^B$ is paired with instance $\prod_i^A$ and have a session key sk_B^j, $sk_A^i \leftarrow sk_B^j$, else if this query causes a $\mathbf{testpw}(A, i, B, C, \pi_{ABC})$ event to occur, set sk_A^i to the value associated with that event, else set $sk_A^i \xleftarrow{R} \{0,1\}^\kappa$.

PARTICIPANT B ACTION 1 to $\prod_j^B$: if $\prod_l^C$ is paired with instance $\prod_i^A$ and $\prod_j^B$, $sk_A^i \leftarrow sk_B^j \leftarrow sk_C^l$, else if $\prod_l^C$ is paired with instance $\prod_j^B$, $sk_B^j \leftarrow sk_C^l$, else if $\prod_i^A$ is paired with instance $\prod_j^B$ and has a session key sk_A^i, $sk_B^j \leftarrow sk_A^i$, else if this query causes a $\mathbf{testpw}(B, j, A, C, \pi_{ABC})$ event to occur, set sk_B^j to the value associated with that event, else set $sk_B^j \xleftarrow{R} \{0,1\}^\kappa$.

$H(\langle A, B, C, m, \upsilon, \sigma, \lambda_A, \lambda_B, \lambda_C\rangle)$**:** if this $H(\cdot)$ causes a **testpw** (A, i, B, C, π_{ABC}), **testpw** (B, j, A, C, π_{ABC}), **testpw** (C, l, A, B, π_{ABC}), or **testexecpw** $(A, i, B, j, C, l, \pi_{ABC})$ event to occur, output the associated value of that event, else output a random value from $\{0,1\}^\kappa$.

Note that we can determine whether the appropriate event occurred using the $\phi_1[\pi]$, $\phi_2[\pi]$, $\phi_3[\pi]$, and τ values. Also note that by P_1 and the fact that a participant instance that is paired with any participant C instance copies the session key of the participant C instance (or, if there is no paired participant C instance, then it copies the key of its partner, if such a partner exists), there will never be more than one associated value that needs to be considered in the $H(\cdot)$ query.

Theorem 4. *For any adversary* $\mathcal{A}$, $Adv_{P_1}^{ake}(\mathcal{A}) = Adv_{P_2}^{ake}(\mathcal{A}) + \frac{O(n_{ro})}{q}$.

Proof. In P_1, participant instance $\prod_l^C$ creates a session key sk_C^l that is uniformly chosen from $\{0,1\}^\kappa$, independent of anything that previously occurred, since the $H(\cdot)$ query that determines sk_C^l is new. Also in P_1, for any participant A and B instances $\prod_i^A$ and $\prod_j^B$ that have had an Action 1 query, either:

1. exactly one instance $\prod_l^C$ is paired with $\prod_i^A$ and $\prod_j^B$, in which case $sk_C^l = sk_A^i = sk_B^j$, or
2. only one instance $\prod_l^C$ is paired with $\prod_i^A$ or $\prod_j^B$, in which case $sk_A^i = sk_C^l$ or $sk_B^j = sk_C^l$, or
3. no instance $\prod_l^C$ is paired with $\prod_i^A$ and/or $\prod_j^B$, and $\prod_i^A$ and $\prod_j^B$ may or may not be paired with each other. In both of these cases, either a **testpw** (A, i, B, C, π_{ABC}) or **testpw** (B, j, A, C, π_{ABC}) event occurs, and sk_A^i or sk_B^j is the value associated with that event (i.e. the output of the previous $H(\cdot)$ query associated with that event) or sk_A^i and sk_B^j are uniformly chosen from $\{0,1\}^\kappa$, independent of anything that previously occurred, since the $H(\cdot)$ query that determines sk_A^i and sk_B^j is new.

Finally, for any $H(\langle A, B, C, \cdot, \cdot, \cdot, \cdot, \lambda_A, \lambda_B, \lambda_C\rangle)$ query, either (1) it causes a **testpw** (A, i, B, C, π_{ABC}), **testpw** (B, j, A, C, π_{ABC}), **testpw** (C, l, A, B, π_{ABC}),

or **testexecpw** $(A, i, B, j, C, l, \pi_{ABC})$ event to occur, in which case the output is the associated value of that event, (2) $\lambda_A = r \cdot f_1(A, B, C, \pi_{ABC})$, $\lambda_B = r \cdot f_2(A, B, C, \pi_{ABC})$, and $\lambda_C = r \cdot f_3(A, B, C, \pi_{ABC})$, but the adversary has not made $f_1(A, B, C, \pi_{ABC})$, $f_2(A, B, C, \pi_{ABC})$, and $f_3(A, B, C, \pi_{ABC})$ queries, or (3) the output of $H(\cdot)$ query is uniformly chosen from $\{0,1\}^\kappa$, independent of anything that previously occurred, since this is a new $H(\cdot)$ query.

If the second case for the $H(\cdot)$ query described above occurs, P_1 may be inconsistent with P_2, since the key associated with the relevant session may need to have been returned by P_2, instead of a random value. However, the probability of the adversary correctly guessing the value of λ_A, λ_B, and λ_C in an $H(\cdot)$ query is less than $\frac{1}{q}$. Thus the total probability of an $H(\cdot)$ query causing the second case above is bounded by $\frac{n_{ro}}{q}$. If this case never occurs, then P_2 is consistent with P_1. □

Protocol P_3**.** Let P_3 be identical to P_2 except that in an $H(\langle A, B, C, m, \mu, \upsilon, \sigma, \lambda_A, \lambda_B, \lambda_C\rangle)$ query, there is no **testexecpw**$(A, i, B, j, C, l, \pi_{ABC})$ event check.

Theorem 5. *For any $\mathcal{A}$ running in time t, there is a $t' = O\big(t + (n_{ro} + n_{ex})t_{op}\big)$ such that $Adv^{ake}_{P_2}(\mathcal{A}) \le Adv^{ake}_{P_3}(\mathcal{A}) + 2Adv^{BDH}_{G_1G_2}(t', n_{ro})$.*

Proof. Let E be the event that a **correctpwexec** event occurs. if E does not occur, then P_2 and P_3 are indistinguishable. Let ε be the probability that E occurs when $\mathcal{A}$ is running against protocol P_2. Then $\Pr(Succ^{ake}_{P_2}(\mathcal{A})) \le \Pr(Succ^{ake}_{P_3}(\mathcal{A})) + \varepsilon$, and thus by Fact 1, $Adv^{ake}_{P_2}(\mathcal{A}) \le Adv^{ake}_{P_3}(\mathcal{A}) + 2\varepsilon$.

Now we construct algorithm $\mathcal{D}$ to solve **BDH** by running $\mathcal{A}$ on a simulation of the protocol. Given (X, Y, Z), $\mathcal{D}$ simulates P_3 for $\mathcal{A}$ with these changes.

1. In an **Execute** (A, i, B, j, C, l) query, set $m \leftarrow X + \rho_{i,A}Q$, $\mu \leftarrow Y + \rho_{j,B}Q$, $\upsilon \leftarrow Z + \rho_{l,C}Q$, where $\rho_{i,A}$, $\rho_{j,B}$, $\rho_{l,C} \overset{R}{\leftarrow} Z_1$.
2. When $\mathcal{A}$ finishes, for every $H(\langle A, B, C, m, \mu, \upsilon, \sigma, \lambda_A, \lambda_B, \lambda_C\rangle)$ query, where m, μ, and υ were generated in an **Execute**(A, i, B, j, C, l) query and a $f_1(A, B, C, \pi)$query returned $\phi_1[\pi]Q$ and an $f_2(A, B, C, \pi)$ query returned $\phi_2[\pi]Q$, an $f_3(A, B, C, \pi)$ query returned $\phi_3[\pi]Q$, and $\lambda_A \leftarrow r\phi_1[\pi]Q$, $\lambda_B \leftarrow r\phi_2[\pi]Q$, $\lambda_C \leftarrow r\phi_3[\pi]Q$, add
$$\sigma e(X, Z)^{\psi_3[\pi]-\rho_{l,C}} e\big(\upsilon,\ (\rho_{j,B} - \psi_2[\pi])X + (\rho_{i,A} - \psi_1[\pi])Y\big)^{-1} \cdot$$
$$e\big(\upsilon,\ (\rho_{j,B} - \psi_2[\pi])Q\big)^{\psi_1[\pi]-\rho_{i,A}} e\big((\rho_{j,B} - \psi_2[\pi])X + (\rho_{i,A} - \psi_1[\pi])Y,\ \psi_3[\pi]Q\big) \cdot$$
$$e\big((\rho_{j,B} - \psi_2[\pi])Q,\ (\rho_{j,A} - \psi_1[\pi])Q\big)^{\psi_3[\pi]}$$
to the list of possible values for **BDH**(X, Y, Z), where $\psi_i[\pi] = r\phi_i[\pi]$, for $i = 1, 2, 3$.

This simulation is perfectly indistinguishable from P_3 until E occurs, and in this case, $\mathcal{D}$ adds the correct **BDH**(X, Y, Z) to the list. After E occurs the simulation may be distinguishable from P_3, but E still occurs with probability ε. We assume $\mathcal{A}$ still follows the appropriate time and query bounds (or that the simulator stops $\mathcal{A}$ from exceeding them). $\mathcal{D}$ creates a list of size n_{ro}, and its advantage is

ε. Let t' be the running time of $\mathcal{D}$, and note that $t' = O\big(t + (n_{ro} + n_{ex})t_{op}\big)$. The theorem follows from the fact that $Adv^{BDH}_{G_1G_2}(\mathcal{D}) \leq Adv^{BDH}_{G_1G_2}\big(t', n_{ro}\big)$. □

Protocol P_4. Let P_4 be a protocol that is identical to P_3 except that if **correctpw** occurs then the protocol halts and the adversary automatically succeeds. (P_3 already checks for a **correctpw** event, in the PARTICIPANT A or B ACTION 1 query to determine if the session key has already been determined, and in the $H(\cdot)$ query, to see if the output has already been determined.)

Theorem 6. *For any adversary $\mathcal{A}$, $Adv^{ake}_{P_3}(\mathcal{A}) \leq Adv^{ake}_{P_4}(\mathcal{A})$.*

Proof. Obvious. □

Note that in P_4, until **correctpw** occurs, an $H(\langle A, B, C, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot\rangle)$ query will output a value uniformly chosen from $\{0,1\}^\kappa$, and the session key for an unpaired client instance will be uniformly chosen from $\{0,1\}^\kappa$.

Protocol P_5. Let P_5 be a protocol that is identical to P_4 except that if **triplepw**(A) occurs, the protocol halts and the adversary fails. We assume that when a query is made, the test for **triplepw**(A) occurs before the test for **correctpw**.

Theorem 7. *For any adversary $\mathcal{A}$ running in time t, there is a $t' = O\big(t + (n^3_{ro} + n_{se} + n_{ex})t_{op}\big)$ such that $Adv^{ake}_{P_4}(\mathcal{A}) \leq Adv^{ake}_{P_5}(\mathcal{A}) + 9Adv^{BDH}_{G_1G_2}(t', n^3_{ro})$.*

Proof. Let ε be the probability that the **triplepw**(A) event occurs when $\mathcal{A}$ is running against protocol P_4. Then $\Pr\big(Succ^{ake}_{P_4}(\mathcal{A})\big) \leq \Pr\big(Succ^{ake}_{P_5}(\mathcal{A})\big) + \varepsilon$, and thus by Fact 1, $Adv^{ake}_{P_4}(\mathcal{A}) \leq Adv^{ake}_{P_5}(\mathcal{A}) + 2\varepsilon$.

Now we construct algorithm $\mathcal{D}$ to solve BDH by running $\mathcal{A}$ on a simulation of the protocol. Given (X, Y, Z), $\mathcal{D}$ simulates P_4 for $\mathcal{A}$ with these changes:

1. In an $f_2(A, B, C, \pi)$ query and $f_3(A, B, C, \pi)$ query, set
   $f_2(A, B, C, \pi) = \psi_2[\pi]Y + \psi'_2[\pi]Q$, where $\psi'_2[\pi] \xleftarrow{R} Z_q$,
   $f_3(A, B, C, \pi) = \psi_3[\pi]Z + \psi'_3[\pi]Q$, where $\psi'_3[\pi] \xleftarrow{R} Z_q$,
   and $(\psi_2[\pi], \psi_3[\pi]) \in_R \{(0,1), (2,0), (0,2)\}$.
2. In a PARTICIPANT A ACTION 0 query to a participant instance $\prod^A_i$ with input $\langle B, C\rangle$, set $m \leftarrow X + \rho_{i,A}Q$.
3. Tests for **correctpw** (from P_4) are not made.
4. For every triple of queries $H(\langle A,B,C,m,\mu,\upsilon,\sigma,\lambda_1,\lambda_2,\lambda_3\rangle)$, $H(\langle A,B,C,m,\mu,\upsilon,\hat{\sigma},\hat{\lambda}_1,\hat{\lambda}_2,\hat{\lambda}_3\rangle)$, $H(\langle A,B,C,m,\mu,\upsilon,\tilde{\sigma},\tilde{\lambda}_1,\tilde{\lambda}_2,\tilde{\lambda}_3\rangle)$ where ACCEPTABLE(σ), ACCEPTABLE$(\hat{\sigma})$ and ACCEPTABLE$(\tilde{\sigma})$ are true, and there was a PARTICIPANT A ACTION 0 query to a participant instance $\prod^A_i$ with input $\langle B, C\rangle$ and output $\langle A, m\rangle$, a PARTICIPANT A ACTION 1 query to $\prod^A_i$ with input $\langle B, \mu, C, \upsilon\rangle$, an $f_k(A, B, C, \pi)$ query that returned λ_k, an $f_k(A, B, C, \hat{\pi})$ query that returned $\hat{\lambda}_k$, and an $f_k(A, B, C, \tilde{\pi})$ query that returned $\tilde{\lambda}_k$, for $k \in \{1, 2, 3\}$, add

$$\Big(\sigma^2\hat{\sigma}^{-1}\tilde{\sigma}^{-1}e(\mu,\upsilon)^{r(2\phi_1[\pi]-\phi_1[\hat{\pi}]-\phi_1[\tilde{\pi}])}e(X,\mu)^{r(2\phi_3[\pi]-\phi_3[\hat{\pi}]-\phi_3[\tilde{\pi}])}$$
$$e(X,\upsilon)^{r(2\phi_2[\pi]-\phi_2[\hat{\pi}]-\phi_2[\tilde{\pi}])}e(Y,\upsilon)^{-2r^2(\phi_1[\pi]-\phi_1[\hat{\pi}])}e(Z,\mu)^{-2r^2(\phi_1[\pi]-\phi_1[\tilde{\pi}])}$$
$$e(X,Y)^{-2r^2(\phi_3[\pi]-\phi_3[\hat{\pi}])}e(X,Z)^{2r^2(\phi_2[\pi]-\phi_2[\tilde{\pi}])}e(Y,Z)^{2r^2(r\phi_1[\pi]-\rho_{i,A})}$$
$$e(Q,\mu)^{r\rho_{i,A}(2\phi_3[\pi]-\phi_3[\hat{\pi}]-\phi_3[\tilde{\pi}])-r^2(2\phi_1[\pi]\phi_3[\pi]-\phi_1[\hat{\pi}]\phi_3[\hat{\pi}]-\phi_1[\tilde{\pi}]\phi_3[\tilde{\pi}])}$$
$$e(Q,\upsilon)^{r\rho_{i,A}(2\phi_2[\pi]-\phi_2[\hat{\pi}]-\phi_2[\tilde{\pi}])-r^2(2\phi_1[\pi]\phi_2[\pi]-\phi_1[\hat{\pi}]\phi_2[\hat{\pi}]-\phi_1[\tilde{\pi}]\phi_2[\tilde{\pi}])}$$
$$e(Q,X)^{-r^2(2\phi_2[\pi]\phi_3[\pi]-\phi_2[\hat{\pi}]\phi_3[\hat{\pi}]-\phi_2[\tilde{\pi}]\phi_3[\tilde{\pi}])}$$
$$e(Q,Y)^{-2r^2\rho_{i,A}(\phi_3[\pi]-\phi_3[\hat{\pi}])+2r^3(\phi_1[\pi]\phi_3[\pi]-\phi_1[\hat{\pi}]\phi_3[\hat{\pi}])}$$
$$e(Q,Z)^{-2r^2\rho_{i,A}(\phi_2[\pi]-\phi_2[\tilde{\pi}])+2r^3(\phi_1[\pi]\phi_2[\pi]-\phi_1[\tilde{\pi}]\phi_2[\tilde{\pi}])}$$
$$e(Q,Q)^{-r^2\rho_{i,A}(2\phi_2[\pi]\phi_3[\pi]-\phi_2[\hat{\pi}]\phi_3[\hat{\pi}]-\phi_2[\tilde{\pi}]\phi_3[\tilde{\pi}])}$$
$$e(Q,Q)^{r^3(2\phi_1[\pi]\phi_2[\pi]\phi_3[\pi]-\phi_1[\hat{\pi}]\phi_2[\hat{\pi}]\phi_3[\hat{\pi}]-\phi_1[\tilde{\pi}]\phi_2[\tilde{\pi}]\phi_3[\tilde{\pi}])}\Big)^{\frac{1}{2r^2}}$$

to the list of possible values of $\mathbf{BDH}(X,Y,Z)$.

This simulation is perfectly indistinguishable from P_4 until a $\mathbf{triplepw}(A)$ event or a **correctpw** event occurs. If a $\mathbf{triplepw}(A)$ event occurs, then with probability $\frac{2}{9}$ it occurs for three passwords π, $\hat{\pi}$, and $\tilde{\pi}$ with:

$$\{(\psi_2[\pi],\psi_3[\pi]),(\hat{\psi}_2[\pi],\hat{\psi}_3[\pi]),(\tilde{\psi}_2[\pi],\tilde{\psi}_3[\pi])\}=\{(1,1),(0,2),(2,0)\},$$

in this case $\mathcal{D}$ adds the correct $\mathbf{BDH}(X,Y,Z)$ to the list. If a **correctpw** event occurs before a $\mathbf{triplepw}(A)$ event occurs, then the $\mathbf{triplepw}(A)$ event would never have occurred in P_4, since P_4 would halt. Note that in this case, the simulation may be distinguishable from P_4, but this does not change the fact that a $\mathbf{triplepw}(A)$ event will occur with probability at least ε in the simulation. However, we do make the assumption that $\mathcal{A}$ still follows the appropriate time and query bounds (or at least that the simulation can stop $\mathcal{A}$ from exceeding these bounds), even if $\mathcal{A}$ distinguished the simulation from P_4.

$\mathcal{D}$ creates a list of size less than n_{ro}^3, and its advantage is $\frac{2}{9}\varepsilon$. Let t' be the running time of $\mathcal{D}$, and note that $t'=O\big(t+\big(n_{ro}^3+n_{se}+n_{ex}\big)t_{op}\big)$. Then the theorem follows from the fact that $Adv_{G_1G_2}^{BHD}(\mathcal{D})\leq Adv_{G_1G_2}^{BHD}(t',n_{ro}^3)$. □

Protocol P_6**.** Let P_6 be a protocol that is identical to P_5 except that if $\mathbf{triplepw}(B)$ occurs, the protocol halts and the adversary fails. We assume that when a query is made, the test for $\mathbf{triplepw}(B)$ occurs before the test for **correctpw**.

Theorem 8. *For any adversary $\mathcal{A}$ running in time t, there is a $t'=O\big(t+(n_{ro}^3+n_{se}+n_{ex})t_{op}\big)$ such that $Adv_{P_5}^{ake}(A)\leq Adv_{P_6}^{ake}(A)+9Adv_{G_1G_2}^{BHD}\big(t',n_{ro}^3\big)$*

Proof. Omitted due to lack of space, but similar to that of Theorem 7. □

Protocol P_7**.** Let P_7 be a protocol that is identical to P_6 except that if $\mathbf{triplepw}(C)$ occurs, the protocol halts and the adversary fails. We assume that when a query is made, the test for $\mathbf{triplepw}(C)$ occurs before the test for **correctpw**.

Theorem 9. *For any adversary $\mathcal{A}$ running in time t, there is a $t'=O\big(t+(n_{ro}^3+n_{se}+n_{ex})t_{op}\big)$ such that $Adv_{P_6}^{ake}(\mathcal{A})\leq Adv_{P_7}^{ake}(\mathcal{A})+9Adv_{G_1G_2}^{BHD}(t',n_{ro}^3)$*

Proof. Omitted due to lack of space, but similar to that of Theorem 7. □

Protocol P_8. Let P_8 be a protocol that is identical to P_7 except that there is a new internal oracle (i.e., not available to the adversary) that handles passwords, called a *password oracle*. This oracle generates all passwords during initialization. Then it accepts queries of the form **testpw**(π) and returns TRUE if π is correct, and FALSE otherwise. The protocol is changed only in the method for determining **correctpw**. Specifically, to test if **correctpw** occurs, whenever a **testpw** (A, i, B, C, π), a **testpw** (B, j, A, C, π) or a **testpw** (C, l, A, B, π) event occurs, a **testpw**(π) query is made to password oracle to see if π is correct.

Theorem 10. *For any adversary $\mathcal{A}$, $Adv^{ake}_{P_7}(\mathcal{A}) = Adv^{ake}_{P_8}(\mathcal{A})$.*

Proof. By inspection, P_7 and P_8 are perfectly indistinguishable. □

The probability of the adversary $\mathcal{A}$ succeeding in P_8 is bounded by:
$\Pr(Succ^{ake}_{P_8}(\mathcal{A})) \leq \Pr(correctpw) + \Pr(Succ^{ake}_{P_8}(\mathcal{A})|\neg correctpw)\Pr(\neg correctpw)$.
First, since there are at most $2n_{se}$ queries to the password oracle, and passwords are chosen uniformly from dictionary of size N, $\Pr(correctpw) \leq \frac{2n_{se}}{N}$.

Now we compute $\Pr(Succ^{ake}_{P_8}(\mathcal{A})|\neg correctpw)$. If **correctpw** does not occur, then $\mathcal{A}$ succeeds by making a Test query to a fresh instance $\prod_i^U$ and guessing the bit used in that **Test** query. We will show that the view of the adversary is independent of sk^i_U, and thus the probability of success is exactly $\frac{1}{2}$.

First we examine **Reveal** queries. Recall that since $\prod_i^U$ is fresh, there could be no **Reveal**(U, i) query, and if $\prod_i^{U'}$ is partnered with $\prod_i^U$, no **Reveal**(U', j) query. Second note that since *sid* includes m and μ and v values, if more than three participant instances accept with the same *sid*, $\mathcal{A}$ fails (see P_1). Thus the output of **Reveal** queries is independent of sk^i_U.

Second we examine $H(\cdot)$ queries. As noted in the discussion following the description of P_4, an $H(\cdot)$ query returns random values independent of anything that previously occurred. Thus any $H(\cdot)$ queries that occurs after sk^i_U is set are independent of sk^i_U. But consider the following cases. (1) if U is the third participant, sk^i_U is chosen independently of anything that previously occurred (see P_2). (2) if U is the first or second participant and is unpaired, sk^i_U is chosen independently of anything that previously occurred (see the discussion after P_4). (3) if U is the first or second participant and is paired, then $sk^i_U \leftarrow sk^j_{U'} \leftarrow sk^l_{U''}$, where $\prod_j^{U'}$ and $\prod_l^{U''}$ are the partners of $\prod_i^U$ and U'' is the third participant and $sk^l_{U''}$ is chosen independently of anything that previously occurred (see P_2). This implies that the adversary's view is independent of sk^i_U, and thus the probability of success is exactly $\frac{1}{2}$. Since $\Pr(\neg correctpw) = 1 - \Pr(correctpw)$, we have:

$$\begin{aligned}
\Pr(Succ^{ake}_{P_8}(\mathcal{A})) &\leq \Pr(correctpw) + \\
&\qquad \Pr\left(Succ^{ake}_{P_8}(\mathcal{A})|\neg correctpw\right)\Pr\left(\neg correctpw\right) \\
&\leq \Pr(correctpw) + \frac{1}{2}\big(1 - \Pr(correctpw)\big) \\
&\leq \frac{1}{2} + \frac{\Pr(correctpw)}{2} \leq \frac{1}{2} + \frac{n_{se}}{N}.
\end{aligned}$$

Therefore $Adv^{ake}_{P_7} \leq \frac{2n_{se}}{N}$. Theorem 2 follows from this and Theorems 3–10. □

5 Complexity Comparison

In this section we compare our protocol with Bresson et al.'s [7], which we call BCP. Because BCP is a group protocol, we fixed the group size to 3 for the comparison. Table 2 shows the total cost of computation and communication required. However, the three (symmetric) encryptions and three decryptions using the password as the key in BCP are not shown. Parings are the most expensive type of computation, and normally scalar multiplication on an elliptic curve is faster than exponentiation and finding a square root. We assume that each pairing computation is approximately equal to three exponentiations [2, 8]. Considering this assumption, the sum of pairings and exponentiations of our proposed protocol is about equal to the number of exponentiations of Bresson et al.'s protocol. Our protocol requires extra computations, i.e. 3 scalar multiplications and 18 square root computations. However, these can be pre-computed. In terms of message length, our protocol is better than BCP.

Table 2. The complexity of protocols

Protocols	Computation				Communication		Precomputation
	Pairings	Scalar Multiplications	Exponentiations	SQRT*	Passes	Message Length**	
Proposed	3	3	3	18	3	$3\|p\|$	Y
Bresson et. al.'s	-	-	12		3	$9\|g\|$	N

* Assume one application of $f_i(A, B, C, \pi)$ loops twice in MacKenzie's algorithm [15].
** $|p|$ is the bit size of finite field over which the elliptic curve is defined and $|g|$ is the bit size of the finite field over which the DLP is defined. When the base field is GF(2), normally $|p| \approx 250$ and $|g| \approx 1024$.

6 Conclusion

In this paper we proposed a provably secure one round password-based tripartite key agreement protocol, which builds on Joux's protocol and adapts the PAK-EC scheme for password-based authentication. We proved the security of the proposed protocol using the random oracle model. It is better than an existing protocol by Bresson et al. in terms of message length. Although it requires extra computation compared to Bresson et al.'s, the extra part can be pre-computed.

Acknowledgements. We thank Colin Boyd and Juan Manuel González Nieto for many helpful suggestons and discussions. We also thank the anonymous referees of SAC 2005 for their comments.

References

1. S. Al-Riyami and K. Paterson, "Tripartite authenticated key agreement protocols from pairings," IMA Conference on Cryptography and Coding, LNCS vol. 2898, Springer-Verlag, pp.332-359. 2003.

2. P. Barreto, H. Kim, and M. Scott, "Efficient algorithms for pairing-based cryptosystems," CRYPTO 2002, LNCS 2442, Springer-Verlag, pp.354-368, 2002.
3. M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks." In EUROCRYPT 2000, LNCS vol. 1807, pp.139-155, Springer-Verlag, 2000.
4. S. M. Bellovin and M. Merritt, "Encrypted key exchange: Password-based protocols secure against dictionary attacks," In IEEE Symposium on Research in Security and Privacy, pp.72-84, 1992.
5. S. Blake-Wilson, D. Johnson, and A. Menezese, "Key agreement protocols and their security analysis." In proceedings of the sixth IMA International Conferences on Cryptography and Coding, LNCS vol.1355, pp. 30-45, Springer-Verlag, 1997.
6. S. Blake-Wilson and A. Menezese, "Authenticated Diffie-Hellman key agreement protocols." In S. Tacares and H. Meijer, editors, Selected Areas in Cryptography (SAC'98), LNCS 1556, pp. 339-361, Springer-Verlag, 1998.
7. E. Bresson, O. Chevassut and D. Pointcheval, "Group Diffie-Hellman key exchange secure against dictionary attacks," Proceedings of Asiacrypt '02, LNCS vol. 2501, Springer-Verlag, pp.497-514, 2002.
8. S. Galbraith, K. Harrison and D. Soldera, "Implementing the Tate pairing," Algorithm Number Theory Symposium – ANTS V, LNCS vol. 2369, Springer-Verlag, pp. 324-337, 2002.
9. IEEE, IEEE1363 Standard Specifications for public key cryptography, 2000.
10. A. Joux, "A one round protocol for tripartite Deffie-Hellman." In W. Bosma, editor, Proceedings of Algorithmic Number Theory Symposium – ANTS IV, LNCS vol. 1838, pp.385-394, Springer-Verlag, 2000.
11. J. Kate, R. Ostrovsky, ans M. Young, "Practical password-authenticated key exchange provably secure under standard assumptions." In EUROCRYPT 2001, LNCS vol. 2045, pp.475-494, 2001
12. Y. Kim, A. Perrig and G. Tsudik, "Communication-efficient group key agreement," IFIP SEC 2001, Jun 2001.
13. L. Law, A. Menezes, M. Qu, J. Solinas, and S.A. Vanstone, "An efficient protocol for authenticated key agreement." Technical Report CORR 98-05, Department of C & O, University of Waterloo, 1998.
14. L. Law, A. Menezes, M. Qu, J. Solinas, and S.A. Vanstone, "An efficient protocol for authenticated key agreement." Designs, Codes and Cryptography, vol. 28, no. 2, pp.119-134, 2003.
15. P. MacKenzie, "More efficient password-authenticated key exchange." CT–RSA 2001, LNCS vol. 2020, pp. 361-377, Springer-Verlag 2001.
16. P. MacKenzie, The PAK suit: Protocols for password-authenticated key exchange. DIMACS Technical report 2002-46, October 2002.
17. K. Shim, "Efficient one-round tripartite authenticated key agreement protocol from Weil pairing," Electronic Letters 39, pp.208-209, 2003.
18. F. Zhang, S. Liu and K. Kim, "ID-based one-round authenticated tripartite key agreement protocol with pairings," Cryptology ePrint archive, Report 2002/122.

An Access Control Scheme for Partially Ordered Set Hierarchy with Provable Security

Jiang Wu[1,⋆] and Ruizhong Wei[2,⋆⋆]

[1] School of Computer Science, University of Waterloo,
200 University Ave. West, Waterloo, ON, N2L 3G1, Canada
[2] Department of Computer Science, Lakehead University,
955 Oliver Road, Thunder Bay, ON, P7B 5E1, Canada

Abstract. In a hierarchical structure, an entity has access to another if and only if the former is a superior of the later. The access control scheme for a hierarchy represented by a partially ordered set (poset) has been researched intensively in the past years. In this paper, we propose a new scheme that achieves the best performance of previous schemes and is provably secure under a comprehensive security model.

1 Introduction

In many situations, the hierarchical systems can be represented by a partially ordered set (poset). In such a hierarchy, all users are allocated into a number of disjoint sets of security classes $p_1, p_2, \cdots, p_n$. A binary relation $\leq$ partially orders the set $\mathcal{P} = \{p_1, p_2, \cdots, p_n, \}$. The users in p_j have access to the information held by users in p_i if and only if the relation $p_i \leq p_j$ held in the poset $(\mathcal{P}, \leq)$. If $p_i \leq p_j$, p_i is called a successor of p_j, and p_j is called a predecessor of p_i. If there is no p_k such that $p_i \leq p_k \leq p_j$, the p_i is called an immediate successor of p_j, and p_j is called an immediate predecessor of C_i.

A straightforward access control scheme for poset hierarchy is to assign each class with a key, and let a class have the keys of all its successors. The information belonging to a class is encrypted with the key assigned to that class, therefore the predecessors have access to the information of their successors. This is awkward because the classes in higher hierarchy have to store a large number of keys. In the past two decades, many schemes based on cryptography have been proposed to ease the key management in the hierarchy. Generally, these schemes are aimed to fully or partly achieve the following goals:

- *Support any arbitrary poset.* It is desirable that any arbitrary poset is supported. Some schemes only support special cases of poset such as a tree. Such schemes are considered restrictive in application.
- *Be secure under attacks.* The schemes are supposed to withstand attacks. For example, a user may try to derive the key of a class that is not his/her successor. The schemes should be secure under all possible attacks.

⋆ Research supported by NSERC PGS.
⋆⋆ Research supported by NSERC grant 239135-01.

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 221–232, 2006.

- *Require small storage space.* Any scheme needs a user in a class to store a certain amount of secret or public parameters for key derivation. All the schemes tried to reduce the amount of parameters stored.
- *Support dynamic poset structures.* The structure of a hierarchy may change. Classes may be added to or deleted from the hierarchy. In these cases the users in the classes (not only the ones being added and deleted) need to update the parameters they store. It is desirable that when a change takes place, the number of classes involved in updating their parameters is as small as possible.

Several hierarchical access control schemes have been proposed in the last two decades. [1, 5, 4] are direct access schemes based on the RSA problem. In a direct access scheme, a predecessor can derive the key of a successor directly from the public parameters of that successor. The disadvantages of this group of schemes include large storage spaces and lack of dynamics. [6, 10, 11] are indirect access schemes. In these schemes, to derive the key of a successor, a predecessor has to derive the key of each class between them. The indirect schemes achieve smaller storage spaces and better dynamics than the direct schemes. However, none of the above schemes provided formal security proof under a secure model that covers all possible cryptographic attacks, except in [10] such a model was defined and a proof sketch was given. Yet [9] indicated that a rigorous proof can not be obtained directly from this proof sketch; some possible attack scenarios are not covered by the proof sketch.

In this paper, we propose a new scheme that is superior to the previous schemes in that it provides both good performance and provable security, and is easy to implement. When we talk about security of the hierarchical access control scheme, we refer to the following security model:

Definition 1. *A hierarchical access control scheme for poset hierarchy is secure if for any group of classes in the poset, it is computationally infeasible to derive the key of any class that is not a member of that group, nor a successor of any member of that group.*

Our scheme is an indirect access scheme, which has similar performance in storage and dynamics to other indirect access schemes. The significant part of our scheme is its formal security proof under this comprehensive security model, which the previous indirect access schemes did not provide.

The rest of this paper is organized as follows: Section 2 presents the scheme, Section 3 analyzes its security, Section 4 compares the performance of the schemes, and Section 5 concludes this paper.

2 Proposed Scheme

2.1 Preliminary

Poset Representation. For a given hierarchy structure, its corresponding poset $(\mathcal{P}, \leq)$ can be represented by a Hasse diagram, which is a graph whose

nodes are classes of $\mathcal{P}$ and the edges correspond to the $\leq$ relation (in the rest of the paper we use "node" and "class" interchangeably)[7]. For distinct $p_j \in \mathcal{P}$ and $p_i \in \mathcal{P}$, an edge from p_j to p_i is present if $p_i \leq p_j$ and there is no $p_k \in \mathcal{P}$ such that $p_i \leq p_k$ and $p_k \leq p_j$. When $p_i \leq p_j$, p_j is drawn higher than p_i. Because of that, the direction of the edges is not indicated in a Hasse diagram. Fig. 1 shows an example of poset represented as a Hasse diagram.

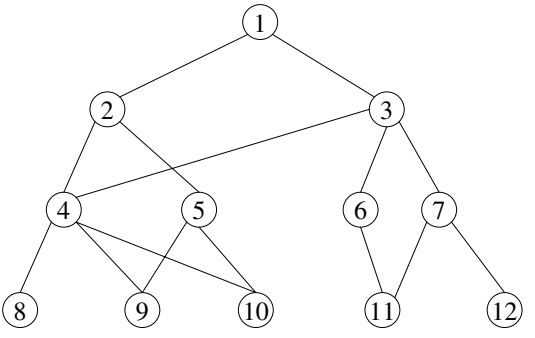


Fig. 1. Example of a Hasse diagram

Auxiliary Function. We construct a function that will be used in our scheme below. Let $p = 2q + 1$ where p, q are all odd primes. Let $\mathbb{G}$ be the subgroup of $\mathbb{Z}_p^*$ of order q. We define a function $f : \mathbb{G} \to [1, q]$ as follows:

$$f(x) = \begin{cases} x; & x \leq q \\ p - x; & x > q \end{cases} \tag{1}$$

For any $x \in \mathbb{Z}_p^*$, if $x \in \mathbb{G}$, then $-x \notin \mathbb{G}$. So the above function is a bijection. If x is a random variable uniformly distributed on $\mathbb{G}$, $f(x)$ is uniformly distributed on $[1, q]$.

2.2 Key Management

The key management of the scheme consists of two procedures: the key generation and the key derivation.

Key Generation

1. The central authority (CA) chooses a group $\mathbb{Z}_p^*$, where $p = 2q + 1$, p and q are both large primes. $\mathbb{G}$ is the subgroup of $\mathbb{Z}_p^*$ of order q.
2. From the top-level classes, the CA traverses the Hasse diagram of the hierarchy with width-first algorithm. For each node p_i, run the following key assignment algorithm to assign its public parameters g_i, $h_{i,j}$ and a secret key k_i:

Algorithm 1. Key Assignment

set g_i to be a unique generator of $\mathbb{G}$
if p_i does not have any immediate predecessor **then**
  set k_i to be a number chosen from $[1, q]$ at random
else if p_i has only one immediate predecessor p_j **then**
  $k_i = f(g_i^{k_j})$
else
  {comment: p_i has more than one immediate predecessors}
  let $\mathcal{X}$ be the set of keys of p_i's immediate predecessors
  $x = \prod_{x_i \in \mathcal{X}} x_i$
  $k_i = f(g_i^x)$
  for all $x_j \in \mathcal{X}$ **do**
    $h_{i,j} = g_i^{x/x_j}$
  end for
end if

For example, the nodes in Fig. 1 will be assigned with the following secret key and public parameters:

Node ID	secret key	public parameters
1	k_1	-
2	$k_2 = f(g_2^{k_1})$	g_2
3	$k_3 = f(g_3^{k_1})$	g_3
4	$k_4 = f(g_4^{k_2 k_3})$	$h_{4,2} = g_4^{k_3}, h_{4,3} = g_4^{k_2}$
5	$k_5 = f(g_5^{k_2})$	g_5
6	$k_6 = f(g_6^{k_3})$	g_6
7	$k_7 = f(g_7^{k_3})$	g_7
8	$k_8 = f(g_8^{k_4})$	g_8
9	$k_9 = f(g_9^{k_4 k_5})$	$h_{9,4} = g_9^{k_5}, h_{9,5} = g_9^{k_4}$
10	$k_{10} = f(g_{10}^{k_4 k_5})$	$h_{10,4} = g_{10}^{k_5}, h_{10,5} = g_{10}^{k_4}$
11	$k_{11} = f(g_{11}^{k_6 k_7})$	$h_{11,6} = g_{11}^{k_7}, h_{11,7} = g_{11}^{k_6}$
12	$k_{12} = f(g_{12}^{k_7})$	g_{12}

Key Derivation. When a node needs to compute the key of one successor, it finds a path from itself to the successor in the Hasse diagram of the hierarchy. Starting from its immediate successor in the path, the node goes through the path, and computes k_i of every successor p_i along the path with the following algorithm:

Algorithm 2. Key Derivation

if p_i has only one predecessor p_j **then**
  $k_i = f(g_i^{k_j})$
else
  {comment: p_j is the predecessor of p_i that is on the path}
  $k_i = f(h_{i,j}^{k_j})$
end if

For example, in Fig. 1, node 1 is to derive the key of node 4. It finds the path 1 $\rightarrow$ 2 $\rightarrow$ 4, and does the following computations:

$k_2 = f(g_2^{k_1})$
$k_4 = f(h_{4,2}^{k_2})$

The correctness of the scheme is easy to be verified by reviewing the procedures in key generation and key derivation.

3 Security Analysis

3.1 Preliminary

On the group $\mathbb{G}$ used in our scheme, two standard assumptions, the discrete logarithm (DL) assumption and decisional Diffie-Hellman (DDH) assumption are believed to hold [2]. Another assumption, named group decisional Diffie-Hellman (GDDH) assumption is proven to hold based on DDH assumption on $\mathbb{G}$ too [8, 3]. To be concrete, let g be a generater of $\mathbb{G}$, a, b, c be random variables uniform on $[1, q]$, $\mathcal{X}$ be a set of random variables uniform on $[1, q]$, l be the binary length of q. Suppose $|\mathcal{X}|$ is polynomially bounded by l. Let $\prod(S)$ indicate the product of all elements in the set S. For any probabilistic polynomial time (in l) algorithms A, any polynomial Q, for l large enough, the three assumptions can be formally expressed as follows:

DL assumption:

$$P_r[A(g, g^a) = a] < \frac{1}{Q(l)} \tag{2}$$

DDH assumption:

$$|P_r[A(g, g^a, g^b, g^{ab}) = 1] - P_r[A(g, g^a, g^b, g^c) = 1]| < \frac{1}{Q(l)} \tag{3}$$

GDDH assumption:

$$|P_r[A(g, g^{\prod(\mathcal{X})}, g^{\prod(S)} | S \subset \mathcal{X}) = 1] - P_r[A(g, g^c, g^{\prod(S)} | S \subset \mathcal{X}) = 1]| < \frac{1}{Q(l)} \tag{4}$$

We give a simple example to explain *GDDH* intuitively. Suppose Alice, Bob and Cathy are to negotiate a shared secret key among them, while all their conversation are open to Eve. Alice chooses a secret number a for herself, in the same way Bob chooses b and Cathy chooses c. They also choose a number g that is known to all including Eve. First Alice computes and announces g^a, Bob g^b, Cathy g^c, then Alice computes and announces $(g^b)^a$, Bob $(g^c)^b$, Cathy $(g^a)^c$. Now each of Alice, Bob and Cathy can computes g^{abc} separately and use it as their common secret key. The *GDDH* assumption says that, while Eve knows $g, g^a, g^b, g^c, g^{ab}, g^{ac}, g^{bc}$, she cannot compute g^{abc}; moreover, given with g^{abc} and a random number, Eve cannot even tell which one is the key and which one is random. In this example, $\mathcal{X} = \{a, b, c\}$, $\{\prod(S) | S \subset \mathcal{X}\} = \{a, b, c, ab, ac, bc\}$.

For convenience, we use the notation from [8] to simplify the expression of (3) and (4), as well as other expressions that are of much greater length in the following parts. When DDH assumption holds, we say that the probabilistic distributions (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) in (3) are polynomially indistinguishable, and rewrite (3) as

$$(g, g^a, g^b, g^{ab}) \approx_{poly} (g, g^a, g^b, g^c).$$

Similarly, if $GDDH$ assumption holds, we say $(g, g^{\prod(\mathcal{X})}, g^{\prod(S)} | S \subset \mathcal{X})$ and $(g, g^c, g^{\prod(S)} | S \subset \mathcal{X})$ in (4) are indistinguishable, and rewrite (4) as

$$(g, g^{\prod(\mathcal{X})}, g^{\prod(S)} | S \subset \mathcal{X}) \approx_{poly} (g, g^c, g^{\prod(S)} | S \subset \mathcal{X}).$$

3.2 Security Proof

The security of our scheme is based on the above three assumptions. In the following parts, we prove the scheme is secure under Definition 1. We suppose the number of nodes in $\mathcal{P}$ is polynomially bounded by l (the binary length of $|\mathbb{G}|$), and all the algorithms considered below are polynomial time (in l) algorithms.

We choose an arbitrary node $p_t \in \mathcal{P}$ and suppose its secret key is k_t. Let $\mathcal{A}$ be the set of predecessors of p_t. We need to prove that, even when all the nodes in $\mathcal{P} - \mathcal{A} - \{p_t\}$ conspire, it is computationally intractable for them to derive k_t.

We group the set $\mathcal{P} - \mathcal{A} - \{p_t\}$ into three subsets: $\mathcal{B}$ the set of nodes in $\mathcal{P} - \mathcal{A}$ which do not have predecessors in $\mathcal{P} - \mathcal{A}$, and which is not p_t; $\mathcal{D}$ the set of nodes that are immediate successors of p_t; $\mathcal{R} = \mathcal{P} - \mathcal{A} - \{p_t\} - \mathcal{B} - \mathcal{D}$. The followings relations between $\mathcal{B}$, $\mathcal{D}$ and $\mathcal{R}$ are direct from their definitions:

- $\mathcal{B} \cup \mathcal{D} \cup \mathcal{R} = \mathcal{P} - \mathcal{A} - \{p_t\}$
- $\mathcal{B} \cap \mathcal{D} = \emptyset$, $\mathcal{R} \cap \mathcal{B} = \emptyset$ and $\mathcal{R} \cap \mathcal{D} = \emptyset$
- the nodes in $\mathcal{R}$ are successors of the nodes in $\mathcal{B}$, or $\mathcal{D}$, or both

An example of the above partition is as follows: in Fig. 1, suppose node 4 is the one we choose as the node p_t, then $\mathcal{A} = \{1, 2, 3\}, \mathcal{B} = \{5, 6, 7\}, \mathcal{D} = \{8, 9, 10\}, \mathcal{R} = \{11, 12\}$.

First we consider when all nodes in $\mathcal{B}$ conspire, what information about k_t they can learn. Suppose the generator assigned to node p_t is g_t, $\mathcal{X}$ is the set of secret keys of the immediate predecessors of node p_t. Let $\prod(S)$ be the product of all elements in the set S. Let $x = \prod(\mathcal{X})$, then $k_t = g_t^x$. The public parameters of p_t are

$$\{g_t, g_t^{\prod(S)} | S \subset \mathcal{X} \text{ and } |S| = |\mathcal{X}| - 1\}$$

The nodes $b_i \in \mathcal{B}$ with generators g_{b_i}, $i \in [1, n]$ may share the same predecessors with node p_t, thus may hold a subset of $\{g_{b_i}^{\prod(S)} | S \subseteq \mathcal{X}\}$ as their public parameters or secret keys. We assume that

$$\{g_{b_i}, g_{b_i}^{\prod(S)} | S \subseteq \mathcal{X}, i \in [1, n]\}$$

is all the information possibly held by nodes in $\mathcal{B}$ that is related to k_t. So the public parameters of p_t, plus the information pertaining to k_t held by $\mathcal{B}$ is a subset of

$$\{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subset \mathcal{X}\} \cup \{g_{b_i}, g_{b_i}^{\prod(S)} | \mathcal{S} \subseteq \mathcal{X}, i \in [1,n]\}$$

The following result formally shows that even all nodes in $\mathcal{B}$ conspire, with the above information, they can not distinguish k_t from a random number on $[1,q]$.

Theorem 1. *Suppose DDH and GDDH assumptions hold on the group $\mathbb{G}$. Let c be a random variable uniform on $[1,q]$. The two distributions*

$$V_{b_n} = \left(g_t^x, \{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subset \mathcal{X}\}, \{g_{b_i}, g_{b_i}^{\prod(S)} | \mathcal{S} \subseteq \mathcal{X}, i \in [1,n]\}\right)$$

and

$$V'_{b_n} = \left(g_t^c, \{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subset \mathcal{X}\}, \{g_{b_i}, g_{b_i}^{\prod(S)} | \mathcal{S} \subseteq \mathcal{X}, i \in [1,n]\}\right)$$

are indistinguishable.

Proof. From GDDH assumption we have

$$\left(g_t^x, \{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subset \mathcal{X}\}\right) \approx_{poly} \left(g_t^c, \{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subset \mathcal{X}\}\right)$$

A polynomial time algorithm can choose z uniformly from $[1,q]$ at random, and reduce the above GDDH distribution pair to

$$V_b = \left(g_t^x, \{g_t, g_t^{\prod(S)} | S \subset \mathcal{X}\}, g_t^z, (g_t^z)^x, \{(g_t^z)^{\prod(S)} | S \subset \mathcal{X}\}\right)$$

$$V'_{im} = \left(g_t^c, \{g_t, g_t^{\prod(S)} | S \subset \mathcal{X}\}, g_t^z, (g_t^z)^c, \{(g_t^z)^{\prod(S)} | S \subset \mathcal{X}\}\right)$$

respectively. It follows that

$$V_b \approx_{poly} V'_{im}. \tag{5}$$

Let c_1 be a random variable uniform on $[1,q]$. Since zc_1 is independent of z and c, from DDH, we have

$$(g_t, g_t^z, g_t^c, g_t^{zc}) \approx_{poly} (g_t, g_t^z, g_t^c, g_t^{zc_1})$$

A polynomial time (in l) algorithm can choose $\mathcal{X}$ that is a set of random variables uniform on $[1,q]$, and whose order is polynomially bounded by l, and reduce the above DDH distribution pair to

$$V'_{im} = \left(g_t^c, \{g_t, g_t^{\prod(S)} | S \subset \mathcal{X}\}, g_t^z, (g_t^z)^c, \{(g_t^z)^{\prod(S)} | S \subset \mathcal{X}\}\right)$$

$$V''_{im} = \left(g_t^c, \{g_t, g_t^{\prod(S)} | S \subset \mathcal{X}\}, g_t^z, (g_t^z)^{c_1}, \{(g_t^z)^{\prod(S)} | S \subset \mathcal{X}\}\right)$$

respectively. It follows that

$$V'_{im} \approx_{poly} V''_{im} \tag{6}$$

Similarly, by choosing z and c uniformly from $[1, q]$ at random, a polynomial time (in l) algorithm can reduce the GDDH distribution pair

$$\left(g_t^{c_1}, \{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subset \mathcal{X}\}\right) \approx_{poly} \left(g_t^x, \{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subset \mathcal{X}\}\right).$$

to

$$V''_{im} = \left(g_t^c, \{g_t, g_t^{\prod(S)} | S \subset \mathcal{X}\}, g_t^z, (g_t^z)^{c_1}, \{(g_t^z)^{\prod(S)} | S \subset \mathcal{X}\}\right)$$
$$V'_b = \left(g_t^c, \{g_t, g_t^{\prod(S)} | S \subset \mathcal{X}\}, g_t^z, (g_t^z)^x, \{(g_t^z)^{\prod(S)} | S \subset \mathcal{X}\}\right).$$

respectively. It follows that

$$V''_{im} \approx_{poly} V'_b \tag{7}$$

From (5), (6) and (7), We conclude

$$V_b \approx_{poly} V'_b$$

i.e.,

$$\left(g_t^x, \{g_t, g_t^{\prod(S)} | S \subset \mathcal{X}\}, g_t^z, \{(g_t^z)^{\prod(S)} | S \subseteq \mathcal{X}\}\right)$$
$$\approx_{poly} \left(g_t^c, \{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subseteq \mathcal{X}\}, g_t^z, \{(g_t^z)^{\prod(\mathcal{S})} | \mathcal{S} \subseteq \mathcal{X}\}\right).$$

By choosing z_i, $i \in [1, n]$ uniformly from $[1, q]$ at random, a polynomial time algorithm can reduce V_b and V'_b to

$$\left(g_t^x, \{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subset \mathcal{X}\}, \{g_t^{zz_i}, (g_t^{zz_i})^{\prod(\mathcal{S})} | \mathcal{S} \subseteq \mathcal{X}, i \in [1, n]\}\right)$$
$$\left(g_t^c, \{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subset \mathcal{X}, \{g_t^{zz_i}, (g_t^{zz_i})^{\prod(\mathcal{S})} | \mathcal{S} \subseteq \mathcal{X}, i \in [1, n]\}\right)$$

It follows that

$$V_{b_n} \approx_{poly} V'_{b_n}.$$

This completes our proof □

Then we consider when the nodes in $\mathcal{B}$ and $\mathcal{D}$ conspire, what information about k_t they can learn. The nodes $d_i \in \mathcal{D}$ assigned with generator g_{d_i}, $i \in [1, m]$ may hold a subset of the following information pertaining to k_t:

$$\{g_{d_i}, g_{d_i}^{k_t} | i \in [1, m]\}.$$

The following theorem shows that even all nodes in $\mathcal{B}$ and $\mathcal{D}$ conspire, with the information they hold, they can not derive k_t:

Theorem 2. *It is intractable for any polynomial time (in l) algorithm to derive g_t^x from*

$$\mathcal{I} = \{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subset \mathcal{X}\} \cup \{g_{b_i}, g_{b_i}^{\prod(\mathcal{S})} | \mathcal{S} \subseteq \mathcal{X}, i \in [1, n]\} \cup \{g_{d_i}, g_{d_i}^{f(g_t^x)} | i \in [1, m]\},$$

i.e., for any polynomial time (in l) algorithm A, any polynomial Q, if l is sufficiently large, then

$$P_r\left[A\left(\mathcal{I}\right) = f(g_t^x)\right] < \frac{1}{Q(l)}.$$

Proof. For convenience, let

$$\mathcal{V} = \{g_t, g_t^{\prod(\mathcal{S})} | \mathcal{S} \subset \mathcal{X}\} \cup \{g_{b_i}, g_{b_i}^{\prod(\mathcal{S})} | \mathcal{S} \subseteq \mathcal{X}, i \in [1, n]\}.$$

Step 1. Assume that there exist a polynomial time (in l) algorithm B, a polynomial Q_1 and a number L, for $l > L$

$$P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^x)}) = f(g_t^x)] \geq \frac{1}{Q_1(l)} \tag{8}$$

where g_d is a generator of $\mathbb{G}$.

Let c be a random variable uniform on $[1, q]$, $Q_2(l) = 2Q_1(l)$. Suppose l is large enough. We consider the following two cases

- **Case 1:** $P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^c)}) = f(g_t^c)] \geq \frac{1}{Q_2(l)}$
  Notice that c is a random variable independent of $\mathcal{V}$. Let $z \in [1, q]$, we define the following algorithm $C(g_d, g_d^z)$:

Algorithm 3. $C(g_d, g_d^z)$

choose a generator of $\mathbb{G}$ as g_t
choose a set of n distinct generators of $\mathbb{G}$ as $\mathcal{B}$
choose a set of random variables uniform on $[1, q]$ as $\mathcal{X}$
compute $\mathcal{V}$ with g_t, $\mathcal{B}$ and $\mathcal{X}$
return $B(\mathcal{V}, g_d, g_d^z)$

  The algorithm C is a polynomial time (in l) algorithm. Since $z = f(g_t^c)$ for some $c \in [1, q]$ (though we do not know c), we have

$$\begin{aligned} P_r[C(g_b, g_b^z) = z] &= P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^c)}) = f(g_t^c)] \\ &\geq \frac{1}{Q_2(l)}. \end{aligned}$$

  This contradicts the DL assumption.
- **Case 2:** $P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^c)}) = f(g_t^c)] < \frac{1}{Q_2(l)}$
  From this inequality and (8), we have

$$\begin{aligned} & P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^x)}) = f(g_t^x)] - P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^c)}) = f(g_t^c)] \\ \geq & \frac{1}{Q_1(l)} - \frac{1}{Q_2(l)} \\ = & \frac{1}{Q_2(l)} \end{aligned} \tag{9}$$

Let $z \in \mathbb{G}$, we define the algorithm $D(\mathcal{V}, z)$ in Algorithm 4.

Algorithm 4. $D(\mathcal{V}, z)$

choose a generator of $\mathbb{G}$ as g_b
if $B(\mathcal{V}, g_d, g_d^{f(z)}) = f(z)$ **then**
  return 1
else
  return 0
end if

D is a polynomial time (in l) algorithm. From (9), we have

$$\begin{aligned} & P_r[D(\mathcal{V}, g_t^x) = 1] - P_r[D(\mathcal{V}, g_t^c) = 1] \\ = & P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^x)}) = f(g_t^x)] - P_r[B(\mathcal{V}, g_d, g_d^{f(g_t^c)}) = f(g_t^c)] \\ \geq & \frac{1}{Q_2(l)}. \end{aligned}$$

That means D can distinguish the two distributions:

$$(\mathcal{V}, g_t^x) \text{ and } (\mathcal{V}, g_t^c).$$

This contradicts to Theorem 1.

Combining Case 1 and Case 2, we conclude that for any polynomial time (in l) algorithm B, any polynomial Q, for sufficiently large l,

$$P_r\left[B(\mathcal{V}, g_d, g_d^{f(g_t^x)}) = f(g_t^x)\right] < \frac{1}{Q(l)} \tag{10}$$

Step 2. Assume there exist a polynomial time (in l) algorithm A, a polynomial Q and a number L such that for $l > L$,

$$P_r\left[A\left(\mathcal{V}, \{g_{d_i}, g_{d_i}^{f(g_t^x)} | i \in [1, m]\}\right) = f(g_t^x)\right] \geq \frac{1}{Q(l)}.$$

Let $B(\mathcal{V}, g_d, g_d^{f(g_t^x)}) = A(\mathcal{V}, \{g_d^{z_i}, g_d^{z_i f(g_t^x)} | i \in [1, m]\})$ where $z_1, \cdots, z_m$ are random variables uniform on $[1, q]$, and m is polynomially bounded by l. We have

$$\begin{aligned} P_r\left[B(\mathcal{V}, g_d, g_d^{f(g_t^x)}) = f(g_t^x)\right] &= P_r\left[A(\mathcal{V}, \{g_d^{z_i}, (g_d^{z_i})^{f(g_t^x)} | i \in [1, m]\} = f(g_t^x)\right] \\ &\geq \frac{1}{Q(l)} \end{aligned}$$

This contradicts (10). Therefore for any polynomial time (in l) algorithm A, any polynomial Q, for sufficiently large l,

$$P_r\left[A\left(\mathcal{V}, \{g_{d_i}, g_{d_i}^{f(g_t^x)} | i \in [1, m]\}\right) = f(g_t^x)\right] < \frac{1}{Q(l)},$$

i.e.,

$$P_r\left[A\left(\mathcal{I}\right) = f(g_t^x)\right] < \frac{1}{Q(l)}.$$

This completes our proof. □

Finally, we consider when all the nodes in $\mathcal{B}$, $\mathcal{D}$, and $\mathcal{R}$ conspire, whether they are able to derive k_p. Since all the nodes in $\mathcal{R}$ are successors of $\mathcal{B}$ or $\mathcal{D}$ or both, the information held by $\mathcal{R}$ can be derived by a polynomial time (in l) algorithm from the information held by $\mathcal{B}$ and $\mathcal{D}$. Thus if $\mathcal{B} \cup \mathcal{D} \cup \mathcal{R}$ can derive k_p, then $\mathcal{B} \cup \mathcal{D}$ can derive k_p. This contradicts to Theorem (2). Therefore we conclude that the scheme is secure under the security model defined in Definition (1).

4 Performance Analysis

4.1 Storage Requirement

Our scheme is an indirect access scheme, and has similar storage requirement with other indirect schemes. In a hierarchy with N nodes where each node has at most M direct predecessors, an indirect scheme assigns each node with one secret key and at most M public parameters. For the direct schemes, to store the public information of one node, the maximum storage is about N numbers, or the product of the N numbers. In a real situation, N would be much greater than M, and N will increase as the scale of the hierarchy increases, while M usually keeps limited, therefore the indirect schemes tend to require less storage than the direct schemes.

4.2 Dynamics

As an indirect hierarchical access scheme, the operation of adding, deleting a node or link in our scheme is similar to other indirect access schemes. When a node is added or deleted, or a link is added to or deleted from a node, only the nodes that are successors of that node will be affected, i.e., the secret key and public parameters of those nodes need to be updated. The direct schemes are quite different. In Akl-Taylor scheme, when a node is added or deleted, all the nodes except for its successors have to update their secret keys and public parameters. In Harn-Lin scheme, when a node is added or deleted, all its predecessors will be impacted. In addition, for these two schemes, to prevent a deleted node to access its former successors, the keys of these successors have to be changed too. In a practical hierarchy, there are much more low level nodes than high level nodes, and it is more likely that the low level nodes will change. Therefore in an indirect scheme, less nodes are affected than in a direct scheme when the hierarchy structure changes. The indirect schemes are more suitable than direct schemes for a dynamic hierarchy.

4.3 Performance Summary

In summary, in view of performance in storage and dynamics, although our scheme does not improve previous indirect schemes, it inherits their performances, which are better than those of the direct schemes.

5 Conclusion

In this paper we proposed a new access control scheme for poset hierarchy. This scheme is concrete and practical for implementation. It supports any arbitrary poset, achieves the best performance of previous schemes in storage and dynamics, and provides a formal security proof under a comprehensive security model. None of the previous schemes achieved all the properties as fully as ours does. Our scheme provides a solution with both practical and theoretical significance for the hierarchical access control problem.

Acknowledgment

The authors wish to thank David Wagner and Kristian Gjøteen for their helpful discussions on the security proof of the scheme. The authors also would like to thank the anonymous referees for their useful comments.

References

1. Selim G. Akl and Peter D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Trans. Comput. Syst.*, 1(3):239–248, 1983.
2. Dan Boneh. The decision diffie-hellman problem. In *ANTS-III: Proceedings of the Third International Symposium on Algorithmic Number Theory*, pages 48–63. Springer-Verlag, 1998.
3. Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. The group diffie-hellman problems. In *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 325–338. Springer, 2003.
4. L. Harn and H.-Y. Lin. A cryptographic key generation scheme for multilevel data security. *Comput. Secur.*, 9(6):539–546, 1990.
5. Stephen J. MacKinnon, Peter D. Taylor, Henk Meijer, and Selim G. Akl. An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Trans. Comput.*, 34(9):797–802, 1985.
6. Ravi S. Sandhu. Cryptographic implementation of a tree hierarchy for access control. *Inf. Process. Lett.*, 27(2):95–98, 1988.
7. Steven Skiena. *Implementing Discrete Mathematics: Combinatorics and Graph Theory With Mathematica*. Perseus Books, 1990.
8. Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-hellman key distribution extended to group communication. In *CCS '96: Proceedings of the 3rd ACM conference on Computer and communications security*, pages 31–37. ACM Press, 1996.
9. Jiang Wu. An access control scheme for partially ordered set hierarchy with provable security. Master's thesis, Lakehead University, Thunder Bay, ON, Canada, 2005.
10. Y. Zheng, T. Hardjono, and J. Pieprzyk. The sibling intractable function family (siff): notion, construction and applications. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science.*, E76-A(1):4–13, 1993.
11. Sheng Zhong. A practical key management scheme for access control in a user hierarchy. *Computers & Security*, 21(8):750–759, 2002.

Breaking a New Hash Function Design Strategy Called SMASH*

Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Austria
{Norbert.Pramstaller, Christian.Rechberger,
Vincent.Rijmen}@iaik.tugraz.at

Abstract. We present a collision attack on SMASH. SMASH was proposed as a new hash function design strategy that does not rely on the structure of the MD4 family. The presented attack method allows us to produce almost any desired difference in the chaining variables of the iterated hash function. Due to the absence of a secret key, we are able to construct differences with probability 1. Furthermore, we get only few constraints on the colliding messages, which allows us to construct meaningful collisions. The presented collision attack uses negligible resources and we conjecture that it works for all hash functions built following the design strategy of SMASH.

Keywords: SMASH, hash functions, cryptanalysis, collision.

1 Introduction

A lot of progress has been made during the last 10 years in the cryptanalysis of dedicated hash functions such as MD4, MD5, SHA-0, SHA-1 [2, 4, 5, 10, 12]. In 2004 and 2005, Wang *et al.* announced that they have broken the hash functions MD4, MD5, RIPEMD, HAVAL-128, SHA-0, and SHA-1 [13, 14]. Due to these recent developments we will have to work on the design and analysis of new hash functions in the future.

A proposal for a new design strategy for dedicated hash functions, called SMASH, has been presented at FSE 2005 by Lars Knudsen [7]. SMASH is a hash function design-strategy that does not follow the structure of the MD4 family. As an example, two specific instances were presented: SMASH-256 and SMASH-512. SMASH-256 and SMASH-512 can be seen as alternatives to SHA-256 and SHA-512 proposed by NIST [9].

We present here a collision attack on SMASH that works independently of the choice that is made for the compression function in the hash function. The attack is based on an extension of the *forward prediction property* already observed in the design document. Furthermore, we exploit the absence of a secret key to construct differentials with probability 1. We are able to construct almost any

* The work in this paper has been supported by the Austrian Science Fund (FWF), project P18138.

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 233–244, 2006.

difference in the chaining variables and we have only few constraints on the colliding messages. This fact allows us to produce meaningful collisions.

We present the collision attack on SMASH in three constructive steps with the goal to break the two specific instances SMASH-256 and SMASH-512. In order to explain the attack we also define two simple instances of SMASH, referred to as SMASH-ORD3 and SMASH-ORDy.

Firstly, we apply the attack on a simple instance of SMASH, referred to as SMASH-ORD3, by choosing the finite field element different than the one for SMASH-256 and SMASH-512. Secondly, we extend this attack to break all simple instances of SMASH, referred to as SMASH-ORDy, but show that it is not successful for SMASH-256 and SMASH-512 due to the maximum message length. Finally, a further extension of the collision attack leads to a collision for SMASH-256 and SMASH-512.

This article is structured as follows. We recall the most important aspects of the SMASH design method in Section 2. In Section 3, we introduce the underlying principles of the attack and apply it to the simple instances SMASH-ORD3 and SMASH-ORDy. We extend the attack to cover SMASH-256 and SMASH-512 in Section 4. We shortly present the most important aspects of SMASH-256 and give an example of a meaningful collision. Section 5, discusses some ideas about how to modify the design-strategy SMASH such that it is immune to the presented attack. We conclude in Section 6. In Appendix A we present the equations and results to produce a collision for SMASH-512.

2 The SMASH Design Method

We present here an overview of the hash function design strategy presented in [7]. Basically, we follow the notation of [7], except that we denote finite field addition by '+', and we stick to the convention of [3] to denote a difference by $h' = h + h^*$.

2.1 Definition of SMASH

Knudsen [7] proposes a new hash function model with a nonlinear compression function f based on a bijective n-bit mapping. Let $m = m_1, m_2, \ldots, m_t$ be the message input after MD strengthening [8], where each block m_i consists of n bits. The hash output h_{t+1} is computed as follows:

$$h_0 = f(\mathrm{iv}) + \mathrm{iv} \tag{1}$$

$$h_i = f(h_{i-1} + m_i) + h_{i-1} + \theta m_i \quad \text{for } i = 1, \ldots, t \tag{2}$$

$$h_{t+1} = f(h_t) + h_t \ . \tag{3}$$

Different to the design strategy of the MD4 family, SMASH applies the compression function f also to the initial value (1) and to the final hash computation (3). This is done in order to avoid pseudo-collision attacks, since the attacker does not have full control over the chaining variable h_0. Applying f also to the final hash computation should make it impossible to predict the final hash value.

The multiplication by θ in (2) is defined as an operation in the finite field $\mathrm{GF}(2^n)$. Note, that for this section and Section 3, θ is an arbitrary field element in $\mathrm{GF}(2^n)$ with the only restriction that $\theta \neq \{0, 1\}$ as mentioned in [7].

The structure of SMASH in (2) exhibits a *forward prediction* property as already described in [7]. Let h_{i-1}, h^*_{i-1} be two intermediate hash values with difference $h'_{i-1} = h_{i-1} + h^*_{i-1}$. Choose a value for m_i and compute $m^*_i = m_i + h'_{i-1}$. Then

$$h'_i = h_i + h^*_i = (1 + \theta)h'_{i-1} \ . \tag{4}$$

2.2 Comparing SMASH with a Block Cipher Based Hash Function

The SMASH design can be compared to a block cipher based hash function operating in the Matyas-Meyer-Oseas mode [11] as shown in Figure 1. In this mode the intermediate hash value is computed as follows:

$$h_i = E_{h_{i-1}}(m_i) + m_i \quad \text{for } i = 1, \ldots, t \ . \tag{5}$$

The underlaying block cipher $E_{h_{i-1}}(m_i)$ in (5) can be replaced by the term $f(h_{i-1}+m_i)+h_{i-1}$ defined in (2). This is a block cipher following the Even-Mansour construction [6]—more precisely, an Even-Mansour construction with key $K = K_1K_2 = h_{i-1}h_{i-1}$. The only difference between the SMASH design and a block cipher based hash function operating in the Matyas-Meyer-Oseas mode is that the message m_i is multiplied by θ prior to the addition to the chaining variable h_i.

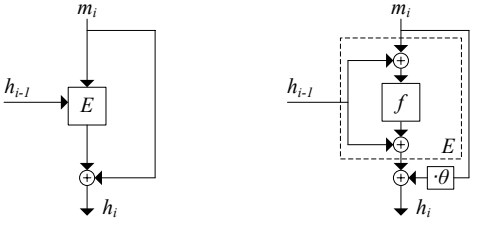


Fig. 1. The Matyas-Meyer-Oseas scheme (left) and the SMASH scheme (right)

3 Observation on the SMASH Design Method

We present here an observation on the design method explained in [7, Section 2]. The observation can be used to break the simple instances SMASH-ORD3 and SMASH-ORDy, but it does not break SMASH-256 nor SMASH-512. In Section 4, we explain how to extend this observation in order to break SMASH-256 and SMASH-512.

3.1 Target

In order to explain our attack, we first consider a simple instance of SMASH. The instance, referred to as SMASH-ORD3, differs from SMASH in the choice of the finite field element θ. We assume that we can choose a θ such that $(1 + \theta)$ has order 3, *i.e.* $(1 + \theta)^3 = 1$. Such a choice is not explicitly forbidden in [7].

3.2 Result

For describing the attack method in Section 3.3 we define the following variables (see also Figure 2):

$$\begin{aligned} x &\text{ an arbitrary 256-bit value} \\ f_1 &= f(m_1 + h_0) \\ f_2 &= f(m_2 + h_1) \\ f_3 &= f(m_3 + h_2) \\ a &= f_1 + f(m_1 + h_0 + x) + \theta x \ . \end{aligned}$$

The variable x defines an arbitrary 256-bit difference. $f_1 \dots f_3$ are the output values of f with message m_i and intermediate chaining variable h_{i-1} as input, *i.e.* without differences. a defines the difference in h_1. Based on these definitions, the following 4-block messages $m = m_1 m_2 m_3 m_4$ and $m^* = m_1^* m_2^* m_3^* m_4^*$ result in the same hash, for any value of z_1, z_2, z_3, and x. Note that a depends on both m_1 and x. In particular, if $x = 0$ then $a = 0$, *i.e.* $m = m^*$.

$$\begin{aligned} m_1 &= z_1 \\ m_2 &= z_2 \\ m_3 &= z_3 \\ m_4 &= z_1 + f_1 + f_2 + f_3 + \theta(m_1 + m_2 + m_3) \\ m_1^* &= z_1 + x \\ m_2^* &= z_2 + a \\ m_3^* &= z_3 + (1 + \theta)a \\ m_4^* &= z_1 + f_1 + f_2 + f_3 + \theta(m_1 + m_2 + m_3) + (1 + \theta)^2 a + x \end{aligned} \tag{6}$$

3.3 Description of the Attack Method

We describe here why we have a collision between the two 4-block messages m and m^* defined in (6). The attack is illustrated in Figure 2.

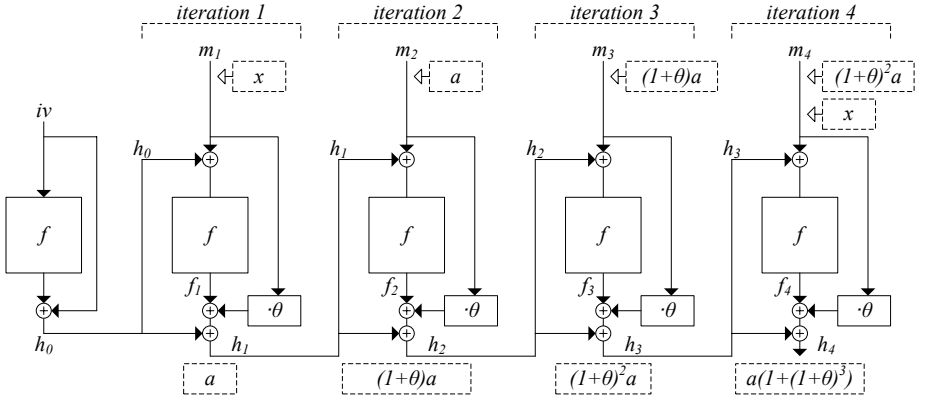


Fig. 2. The attack on SMASH-ORD3. The dashed rectangles denote differences.

The attack is an extension of the *forward prediction property* observed in [7]. It can be verified that the value a is the difference in h_1. We cannot predict the value of a, but we can of course easily compute it once we have chosen an arbitrary z_1 and x.

The basic idea of the attack is to control the propagation of the difference such that the input differences to the function f after the first iteration and prior to the last iteration (*iteration 2* and *iteration 3* in Figure 2) equal zero. In this case the nonlinearity of f does not have any impact on the difference propagation. For the last message block (*iteration 4*) we ensure that the difference $m'_4 = m_4 + m^*_4$ equals the difference $m'_1 = m_1 + m^*_1$ and that $f'_1 = f'_4$. This can be achieved as follows. By choosing the difference in the second message block equal to a, we make sure that the input difference to f equals zero (differences cancel out). Hence, we ensure that the difference in h_2 equals $(1+\theta)a$. Similarly, by choosing the difference in the third message block equal to $(1+\theta)a$, we ensure that the difference in h_3 equals $(1+\theta)^2 a$. This was already observed in [7].

Now we have to determine the last message block in each of the messages. We choose the last message block of the first message, m_4, in such a way that the input of f in the last iteration equals the input of f in the first iteration.

$$\begin{aligned} m_4 &= m_1 + h_0 + h_3 \\ &= m_1 + h_0 + f_3 + h_2 + \theta m_3 \\ &= m_1 + h_0 + f_3 + f_2 + f_1 + h_0 + \theta(m_3 + m_2 + m_1) \\ &= m_1 + f_3 + f_2 + f_1 + \theta(m_3 + m_2 + m_1) \end{aligned}$$

The last block of the second message, m^*_4, is selected in such a way that the difference in the last message block equals $(1+\theta)^2 a + x$. This choice ensures that the two inputs of f in the last iteration (*iteration 4*) equal the inputs in the first iteration (*iteration 1*). Consequently, the outputs of f will be the same as in the first iteration, hence they will have the same difference as in the first iteration: $a + \theta x$. Working out the equations, we see that the difference in h_4 becomes:

$$\begin{aligned} h'_4 = (f(m_1 + h_0) + h_3 + \theta m_4) + (f(m_1 + h_0 + x) + h_3 \\ + (1+\theta)^2 a + \theta(m_4 + (1+\theta)^2 a + x)) \\ = a + \theta x + (1+\theta)^3 a + \theta x \\ = a(1 + (1+\theta)^3)\ . \end{aligned} \tag{7}$$

Since we assumed a θ such that $(1+\theta)$ has order 3, the difference in (7), $h'_4 = a(1 + (1+\theta)^3)$, equals zero and we have produced a collision for our simple SMASH instance SMASH-ORD3. Due to the collision after the last iteration ($h'_4 = 0$), the final hash computation (3) has no impact on the result.

The attack on SMASH-ORD3 can be generalized to break the simple SMASH instances, referred to as SMASH-ORDy. The instances SMASH-ORDy are defined by choosing a θ such that $ord(1+\theta) = y$, where y is an arbitrary value. For a successful attack we then need at least $y+1$ message blocks without counting in the last message block that results from the MD strengthening. For instance,

for SMASH-ORD3 we have $y = 3$ and we have a collision in iteration $y + 1 = 4$. We will see in the next section that for the specific instances SMASH-256 and SMASH-512 this attack strategy does not work anymore. This is due to the number of maximum message blocks that can be hashed with SMASH-256 and SMASH-512.

4 Attacking SMASH-256 and SMASH-512

In this section we explain how the attack can be extended to break the proposed SMASH hash functions. After a description of the general attack strategy, we present some equations and solutions for the specific instance SMASH-256. Furthermore, we present an example for a meaningful collision. Equations and solutions for SMASH-512 are given in Appendix A.

4.1 SMASH-256

The hash function SMASH-256 is a specific instance of the design method SMASH. SMASH-256 is specified by setting $n = 256$, by defining the finite field $\mathrm{GF}(2^{256})$ via the irreducible polynomial $q(\theta)$,

$$q(\theta) = \theta^{256} + \theta^{16} + \theta^3 + \theta + 1, \tag{8}$$

and by defining the compression function f. Due to the chosen padding method, SMASH-256 can process messages with a bit length less than 2^{128}.

Even if the properties of f are not relevant for our attack, we shortly repeat them to give a basic understanding of the SMASH design strategy. The compression function f is composed of several rounds, called H-rounds and L-rounds:

$$f = H_1 \circ H_3 \circ H_2 \circ L \circ H_1 \circ H_2 \circ H_3 \circ L \circ H_2 \circ H_1 \circ H_3 \circ L \circ H_3 \circ H_2 \circ H_1 \ .$$

Both the H-rounds and the L-rounds take as input a 256-bit value and produce a 256-bit output. The underlaying operations are S-Boxes, some linear diffusion layers and variable rotations. The S-Boxes are based on the S-Boxes used for the block cipher Serpent [1]. An exact definition of the H-rounds and L-rounds can be found in [7].

4.2 Brief Description of the Attack

For the attacks on SMASH-ORD3 and SMASH-ORDy we assumed that we can choose a certain finite field element θ. This is not possible for the specific instances of SMASH. For SMASH-256 the finite field element θ is defined as a root of the irreducible polynomial $q(\theta) = \theta^{256} + \theta^{16} + \theta^3 + \theta + 1$, *i.e.* $q(\theta) = 0$. The irreducible polynomial for SMASH-512 is given in Appendix A. In order to show whether the previously described attacks on SMASH-ORD3 and SMASH-ORDy can be applied to SMASH-256 and SMASH-512, we have to compute the order of $(1 + \theta)$ for the specified θ. For SMASH-256, the order of $(1 + \theta)$ is $((2^{256} - 1)/5)$ and for SMASH-512 the order of $(1+\theta)$ is $(2^{512} - 1)$. Therefore, the

attack requires at least $(((2^{256}-1)/5)+1)$ message blocks for SMASH-256 and (2^{512}) message blocks for SMASH-512, respectively. As specified in [7], SMASH-256 can be used to hash messages of bit length less than 2^{128}. This corresponds to $(2^{120}-1)$ message blocks of 256 bits. SMASH-512 is specified for $(2^{247}-1)$ message blocks of 512 bits. Hence, the order of $(1+\theta)$ is for both hash functions larger than the maximum number of message blocks. This means, that we can still produce colliding messages but these messages are no longer valid inputs according to the SMASH-256 and SMASH-512 specification.

However, the attack technique can be generalized further. Previously, we extended the *forward prediction property* by considering message pairs that introduce a non-zero input difference x into f twice: once at the beginning and once at the end of the message. We can extend the property further by considering message pairs that introduce the difference x three or more times. Every time the input difference to f is non-zero, we make sure that the absolute values of the two message blocks equal the values in the first message blocks. Consequently, the output difference of f will be every time the same, namely $(a+\theta x)$. In this way, we can produce almost any desired difference in the chaining variable h_t. In order to find a collision, we want to construct a difference of the form $h'_t = a \cdot q(\theta) = a \cdot 0 = 0 \pmod{q(\theta)}$.

4.3 Equations

In this section, we introduce some notations and list the equations that need to be solved in order to construct pairs of messages that result in a specific difference in h_t. Without loss of generality, we will always work with messages that differ already in the first block, *i.e.* $m'_1 \neq 0$.

We define the following notation. Let d be a function defined for two input values only: $d(0) = 0$, and $d(x) = 1$. Let m'_i denote the difference in message block i. Let $\delta_1 = 1$ and let δ_i with $1 < i \leq t$, be defined as follows:

$$\delta_i = d(m'_i + \sum_{j=1}^{i-1} (1+\theta)^{i-j-1} a \delta_j) \ . \tag{9}$$

Then it is possible to construct two t-block messages with the differences defined by (9), such that the difference in h_t has the following value

$$h'_t = a \sum_{i=1}^{t} (1+\theta)^{t-i} \delta_i \ . \tag{10}$$

The absolute values m_i can be determined as follows. The first block, m_1, can always be selected arbitrarily. If $\delta_i = 0$, then m_i can be selected arbitrarily. If $\delta_i = 1$ and $i > 1$, then m_i has to be equal to $h_{i-1} + m_1 + h_0$.

4.4 Solutions for SMASH-256

The field polynomial $q(\theta)$ can be written as follows:

$$\theta^{256} + \theta^{16} + \theta^3 + \theta + 1 = 1 + (1+\theta)^2 + (1+\theta)^3 + (1+\theta)^{16} + (1+\theta)^{256} \ . \tag{11}$$

Hence, the solution of (10) is given by $\delta_i = 1$ for $i = 1, 241, 254, 255, 257$ and $\delta_i = 0$ for all other $i \leq t = 257$. Given the δ_i, (9) can be solved for the differences m'_i. This gives:

$$\begin{aligned}
m'_1 &= x \\
m'_i &= (1+\theta)^{i-2}a, && 1 < i \leq 240 \\
m'_{241} &= x + (1+\theta)^{239}a \\
m'_i &= (1+\theta)^{i-2}a + (1+\theta)^{i-242}a, && 241 < i < 254 \\
m'_{254} &= x + (1+\theta)^{252}a + (1+\theta)^{12}a \\
m'_{255} &= x + (1+\theta)^{253}a + (1+\theta)^{13}a + a \\
m'_{256} &= (1+\theta)^{254}a + (1+\theta)^{14}a + (1+\theta)a + a \\
m'_{257} &= x + (1+\theta)^{255}a + (1+\theta)^{15}a + (1+\theta)^2 a + (1+\theta)a \ .
\end{aligned}$$

Here, x is an arbitrary 256-bit difference. All other differences are defined by the attack method. As explained above, 253 of the message blocks m_i can be chosen arbitrarily, while the remaining 4 are determined by the attack.

The above-defined differences produce the following difference in the chaining variable h_{257}:

$$h'_{257} = (1+\theta)^{256}a + (1+\theta)^{16}a + (1+\theta)^3 a + (1+\theta)^2 a + a \ .$$

It is clear that $h'_{257} = a \cdot q(\theta) = a \cdot 0 = 0$, and hence we have a collision after iteration 257.

4.5 SMASH-256: Example of Colliding Messages m and m^*

In this section we give an example of two messages[1], m and $m^* = m + m'$, that collide after 257 iterations.

Figure 3 shows the two colliding ASCII coded strings. Each message consists of 258 message blocks. Even if we have already a collision after iteration 257 (see also Table 1), we added an additional message block, $m_{258} = m^*_{258}$, containing the character '>' ($3e_{hex}$). We have chosen the two messages in this way, because the message blocks $m_2 \ldots m_{258}$ and $m^*_2 \ldots m^*_{258}$ are inside the end tag ($< / m_2 \ldots m_{258} >$) and hence are not displayed in a standard *HTML* viewer or web browser (*e.g.* Mozilla Firefox 1.0.3). Therefore, at first sight, only the two message blocks m_1 and m^*_1 are visible.

Using hex notation, Table 1 shows the input message blocks m_i and m^*_i for $i = 1, 241, 254, 255, 257, 258$, the initial chaining variables $h_0 = h^*_0 = f(\mathrm{iv}) + \mathrm{iv}$, the chaining variables h_{257}, h^*_{257}, h_{258}, and h^*_{258}, and the colliding outputs h_{259} and h^*_{259}.

As described in Section 4.4, the messages $m_2 \ldots m_{240}, m_{242} \ldots m_{253}$, and m_{256} can be chosen arbitrarily. In this simple example each of these message blocks contains only space characters (20_{hex}).

[1] For this simple example we omitted padding.

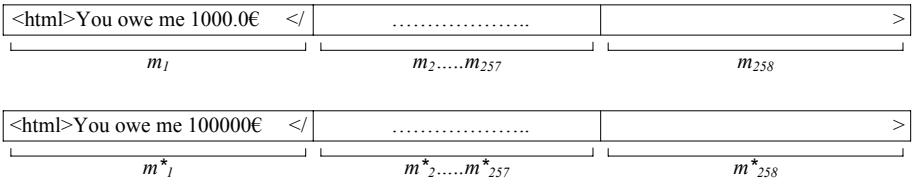


Fig. 3. ASCII coded strings m and m^*

Table 1. Colliding messages m and m^*

h_0=	55214e9e	237290c2	3ff782f7	c2073a8c	2105c5f1	6ccb0855	9c71b7c1	e7ecceac
h^*_0=	55214e9e	237290c2	3ff782f7	c2073a8c	2105c5f1	6ccb0855	9c71b7c1	e7ecceac
m_1=	3c68746d	6c3e596f	75206f77	65206d65	20313030	302e3030	80202020	20203c2f
m^*_1=	3c68746d	6c3e596f	75206f77	65206d65	20313030	30303030	80202020	20203c2f
				.....				
m_{241}=	346a6100	4e3cbc5b	f472d355	b41311b2	4b7df46d	e6b4028f	6aaf9c4d	97a6f169
m^*_{241}=	4afe2771	dd8507d9	a25082bc	dac25578	f34abb1c	5501e05e	d9874798	6aa679d3
				.....				
m_{254}=	60358467	cfde2276	534a4038	d3555d7e	576415d4	5c151dbb	7664ed09	f97bb393
m^*_{254}=	de2cdecd	6e323f7e	de8e653b	7d887168	f94cebb5	7370fc51	0e2e0226	8f1e25ba
m_{255}=	40088790	06da5567	eb2a1d6e	2869d96f	02fb791a	dc8799ca	0df2d9de	9dec9799
m^*_{255}=	f81b73fc	db0f8afe	28947e42	699822e0	6de2b3b5	0b5536fe	c3d1ebb0	7744d316
				.....				
m_{257}=	cc4a7c9c	9b4a99e1	8d275de9	3a44a2e7	4640484b	3cb2abb4	f1af679f	4e6e142f
m^*_{257}=	1a5d75eb	71ea319e	be76a60e	abc9278b	329ff04f	5e932f4d	cc04996a	9e6c4183
h_{257}=	ddc0b465	b42b5072	c34ad69d	b47c8e2c	30a36a7b	218a7bbe	99ffd185	831e8ddf
h^*_{257}=	ddc0b465	b42b5072	c34ad69d	b47c8e2c	30a36a7b	218a7bbe	99ffd185	831e8ddf
m_{258}=	20202020	20202020	20202020	20202020	20202020	20202020	20202020	2020203e
m^*_{258}=	20202020	20202020	20202020	20202020	20202020	20202020	20202020	2020203e
h_{258}=	da7c8fa1	4389e3c5	7299afdd	ad027de9	4c595315	c981c2f8	95390053	37c2fa00
h^*_{258}=	da7c8fa1	4389e3c5	7299afdd	ad027de9	4c595315	c981c2f8	95390053	37c2fa00
h_{259}=	2ffeac86	08bc1142	a3ddf493	6455bcd8	673dea34	c6365ec3	92b1bc79	15c1487e
h^*_{259}=	2ffeac86	08bc1142	a3ddf493	6455bcd8	673dea34	c6365ec3	92b1bc79	15c1487e

5 Discussion

In this section we discuss some observations on SMASH. Firstly, we propose a way how to change the SMASH design strategy such that it is not vulnerable to the presented attack. Secondly, we give some comments on block cipher based hash functions relaying on the SMASH design strategy.

5.1 Using Different Functions f

If a different f is used in each iteration, the attack described in this article seems not to work anymore. This is due to the fact that we expect the difference in the chaining variable h_i in iteration i, where $\delta_i = 1$, to be $(a + \theta x)$. If different functions are used for each iteration this cannot be ensured anymore and hence the presented attack is not successful. A simple method to modify SMASH could for instance be the addition of a counter value in each iteration. However, we did not further investigate these modifications and hence it should not be seen as a solution for this hash function design strategy. Another idea to modify SMASH such that it is immune against the presented attack is given in [7].

5.2 Block Cipher Based Hash Functions

In Section 2.2 we compared the SMASH design strategy with a block cipher based hash function. We have shown that the Matyas-Meyer-Oseas operation mode with a block cipher following the Even-Mansour construction is not secure.

6 Conclusion

We described a collision attack on SMASH-256 and SMASH-512. The attack works independently of the choice of the nonlinear compression function f and requires negligible computation power. We are able to construct meaningful collisions due to the fact that we have only few restrictions on the colliding messages. The attack is based on two observations. Firstly, the property of *forward prediction* which was described in [7]. Secondly, a differential attack on a hash function is easier than on a block cipher, because the attacker has control over the input values. If the attacker ensures that the two inputs to two different instantiations of the compression function are equal, then the two outputs (and hence the output difference) will also be equal in both instantiations.

If the compression function f would be different in every iteration, then it would not be possible to produce the same output difference twice.

Acknowledgements

We would like to thank Lars Knudsen for valuable conversations and for providing a reference implementation of SMASH-256.

References

1. Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A New Block Cipher Proposal. In Serge Vaudenay, editor, *Fast Software Encryption, 5th International Workshop, FSE 1998, Paris, France, March 23-25, 1998, Proceedings*, volume 1372 of *LNCS*, pages 222–238. Springer, 1998.
2. Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *LNCS*, pages 290–305. Springer, 2004.
3. Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. Journal of Cryptology, 4(1):3–72, 1991.
4. Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462, pages 56–71. Springer, 1998.
5. Hans Dobbertin. Cryptanalysis of MD4. In Bart Preneel, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *LNCS*, pages 53–69. Springer, 1996.

6. Shimon Even and Yishay Mansour. A Construction of a Cipher From a Single Pseudorandom Permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology - ASIACRYPT '91, International Conference on the Theory and Applications of Cryptology, Fujiyoshida, Japan, November 11-14, 1991, Proceedings*, volume 739 of *LNCS*, pages 210–224. Springer, 1991.
7. Lars R. Knudsen. SMASH - A Cryptographic Hash Function. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Proceedings*, volume 3557 of *LNCS*, pages 228–242. Springer, 2005.
8. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997. Available online at `http://www.cacr.math.uwaterloo.ca/hac/`.
9. National Institute of Standards and Technology (NIST). FIPS-180-2: Secure Hash Standard, August 2002. Available online at `http://www.itl.nist.gov/fipspubs/`.
10. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Exploiting Coding Theory for Collision Attacks on SHA-1. In *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings to appear*, LNCS. Springer, 2005.
11. Bart Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Katholieke Universiteit Leuven, 1993.
12. Vincent Rijmen and Elisabeth Oswald. Update on SHA-1. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *LNCS*, pages 58–71. Springer, 2005.
13. Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Xiuyuan Yu. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD, August 2004. Preprint, available at `http://eprint.iacr.org/2004/199`.
14. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.

A SMASH-512: Equations and Solutions

To determine the differences m'_i that produce a collision for SMASH-512 we use the same definitions and equations as presented in Section 4.3.

SMASH-512 is specified by setting $n = 512$ and by defining the finite field $\mathrm{GF}(2^{512})$ vie the irreducible polynomial $q(\theta)$

$$q(\theta) = \theta^{512} + \theta^8 + \theta^5 + \theta^2 + 1 \ . \tag{12}$$

The polynomial $q(\theta)$ can be written as follows:

$$q(\theta) = 1 + (1+\theta) + (1+\theta)^2 + (1+\theta)^4 + (1+\theta)^5 + (1+\theta)^8 + (1+\theta)^{512} \ . \tag{13}$$

The solution of (10) is given by $\delta_i = 1$ for $i = 1, 505, 508, 509, 511, 512, 513$ and $\delta_i = 0$ for all other $i \leq t = 513$. Given the δ_i, (9) can be solved for the differences m'_i. This gives:

$$
\begin{aligned}
m'_1 &= x \\
m'_i &= (1+\theta)^{i-2}a, && 1 < i \leq 504 \\
m'_{505} &= x + (1+\theta)^{503}a \\
m'_i &= (1+\theta)^{i-2}a + (1+\theta)^{i-506}a, && 505 < i < 508 \\
m'_{508} &= x + (1+\theta)^{506}a + (1+\theta)^2 a \\
m'_{509} &= x + (1+\theta)^{507}a + (1+\theta)^3 a + a \\
m'_{510} &= (1+\theta)^{508}a + (1+\theta)^4 a + (1+\theta)a + a \\
m'_{511} &= x + (1+\theta)^{509}a + (1+\theta)^5 a + (1+\theta)^2 a + (1+\theta)a \\
m'_{512} &= x + (1+\theta)^{510}a + (1+\theta)^6 a + (1+\theta)^3 a + (1+\theta)^2 a + a \\
m'_{513} &= x + (1+\theta)^{511}a + (1+\theta)^7 a + (1+\theta)^4 a + (1+\theta)^3 a + (1+\theta)a + a \ .
\end{aligned}
$$

Here x is an arbitrary 512-bit difference. All other differences are defined by the attack method. For SMASH-512, 507 of the message blocks m_i can be chosen arbitrarily, while the remaining 6 are determined by the attack.

The above-defined differences produce the following difference in the chaining variable h_{513}:

$$
\begin{aligned}
h'_{513} = {} & (1+\theta)^{512}a + (1+\theta)^8 a + (1+\theta)^5 a \\
& + (1+\theta)^4 a + (1+\theta)^2 a + (1+\theta)a + a \ .
\end{aligned}
$$

It is clear that $h'_{513} = a \cdot q(\theta) = a \cdot 0 = 0$, and hence we have a collision for SMASH-512 after iteration 513.

Analysis of a SHA-256 Variant*

Hirotaka Yoshida[1] and Alex Biryukov[2]

[1] Systems Development Laboratory, Hitachi, Ltd.,
1099 Ohzenji, Asao-ku, Kawasaki-shi, Kanagawa-ken, 215-0013 Japan
hyoshida@sdl.hitachi.co.jp
[2] Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC,
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
abiryuko@esat.kuleuven.ac.be

Abstract. SHA-256 is a cryptographic hash function which was proposed in 2000 as a new generation of SHA functions and was adopted as FIPS standard in 2002. In this paper we will consider a SHA-256 variant and a SHACAL-2 variant in which every arithmetic addition is replaced by XOR operation. We call the SHA-256 variant SHA-2-XOR and the SHACAL-2 variant SHACAL-2-XOR respectively. We will present a differential attack on these constructions by using one-round iterative differential characteristics with probability 2^{-8} we identified. Our result shows that SHACAL-2-XOR with up to 31 rounds out of 64 has a weakness of randomness and that SHA-2-XOR with up to 34 rounds has a weakness of pseudo-collision resistance. Using the 31-round distinguisher, we present an attack on SHACAL-2-XOR with up to 32 rounds. We also show that no 2-round iterative patterns with probability higher than 2^{-16} exist.

Keywords: SHA-256,SHA-2-XOR, SHACAL-2-XOR, Differential cryptanalysis, Pseudo-collision resistance, Iterative patterns.

1 Introduction

A cryptographic hash function is an algorithm that takes input strings of arbitrary (typically very large) length and maps these to short fixed length output strings. The progress in cryptanalysis of cryptographic hash functions has been quite slow until very recently, the cryptographic community has been surprised at the progress of cryptanalysis of hash functions, such as an attack on MD5 [23] for finding collisions and an attack with a new strategy on SHA-0 [2, 3] and an attack for finding multi-collisions. However, these techniques are not applicable to SHA-256 due to its more complex message schedule and round function.

SHA-256 is a cryptographic hash function which was proposed in 2000 as a new generation of SHA functions and was adopted as FIPS standard in 2002 [18]. SHA-256 is constructed from MD(Merkle-Damgård) -construction and Davis-Meyer mode. The compression function of SHA-256 has 64 rounds, two kinds of

* This work was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government.

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 245–260, 2006.

non-linear functions, cyclic rotations, and round-dependent constants. The hash value calculated by SHA-256 is 256 bits long.

The function obtained from the compression function of SHA-256 by removing the feed-forward operation of the Davis-Meier mode is invertible. It was proposed for use as a block cipher by Handschuh and Naccache and named SHACAL-2 [12]. The block cipher was selected as one of the NESSIE finalists. In cryptanalysis of SHACAL-2, there have been several attacks on its reduced versions [14, 22], but with time complexities around 2^{500} for 32 or less rounds.

Although several works have discussed the security of SHA-256[11] and reported interesting differential properties of several consecutive round functions [13], no weakness has been demonstrated for SHA-256 or any SHA-256 variant so far. In this paper we will consider a SHA-256 variant and a SHACAL-2 variant in both of which ADD operations are replaced by XOR operations. We call the SHA-256 variant SHA-2-XOR and the SHACAL-2 variant SHACAL-2-XOR respectively. We will present a differential attack [5] on these ciphers by identifying iterative differential characteristics. We will show how to distinguish the SHACAL-2-XOR from a random permutation. Our result will show that SHACAL-2-XOR with up to 31 rounds has a weakness of randomness and that SHA-2-XOR with up to 34 rounds has a weakness of pseudo-collision resistance. In addition to that, it will also show a property that SHA-2-XOR with up to 31 rounds has a weakness in certain collision resistance we will define.

Hereafter we introduce three kinds of resistance of hash functions for the motivation of our approach in the cryptanalysis of SHA-256: near-collision resistance, pseudo-collision resistance, and randomness.

The importance of the first two requirements is related to collision resistance. *Near-collision* resistance is resistance against attacks finding a pair of hash values which differ in only small number of bit positions. Near-collisions of the SHA-0 hash function have been found, which is an undesirable property [2] for a hash function. In fact, there has been presented a strategy to convert near-collisions into full-collisions [1]. Therefore near-collision resistance is crucial for the collision resistance. *Pseudo-collision* resistance is resistance against finding a collision obtained from more relaxed condition that different initial vectors can be chosen. Pseudo-collision resistance has a particular importance for a hash function constructed by the MD-construction because in this case pseudo-collision resistance for the hash function can be translated into collision resistance for its compression function. The theory of the MD-construction, on which the security of many popular hash functions rely, does not guarantee collision resistance for a hash function without pseudo-collision resistance for its compression function [10]. Recently, a situation where pseudo-collisions could become practical has been considered [16].

Pseudo-randomness of a function is its indistinguishability from a random function. This resistance has a particular importance in some existing applications where one of the requirements for the hash function is randomness. Recently, the strongest version of the HAVAL hash function (in encryption mode) was shown to be non-random [24].

Although in the past these three types of resistance have received less attention than the collision resistance, we expect that situation will change in the near future.

The outline of this paper is as follows. In Section 2, we give a brief description of the SHA-2 algorithm published in [18]. In Section 3 we study the known results on cryptanalysis of the SHA-256 algorithm and the SHACAL-2 algorithm. In Section 4, we present our differential attack on the SHA-2-XOR and SHACAL-2-XOR identifying iterative characteristics. Our conclusions are given in Section 5.

2 Description of the SHA-256 Hash Function and the SHACAL-2 Block Cipher

In this section, we give a brief description of the SHA-256 hash function and the SHACAL-2 block cipher, which is sufficient to understand the concepts introduced in this paper. For a full description of SHA-256 we refer to [18].

SHA-256 is a hash function that is based on the well-known Davies-Meyer construction of hash functions ([17], p. 341). The variable-length message M is divided into 512-bit blocks $M_0, M_1, \ldots, M_{n-1}$. The 256-bit hash value V_n is then computed as follows:

$$V_0 = IV;\ V_{s+1} = \texttt{compress}(V_s, M_s) = E_{M_s}(V_s) + V_s\ for\ 0 \leq s < n,$$

where `compress` is the compression function, IV is a fixed initial value and $E_K(X)$ is the block cipher, SHACAL-2. The function $E_K(X)$ is an iterated design that only uses simple operations on 32-bit words. The 256-bit input V_j is loaded into 8 registers (A, B, C, D, E, F, G, H) and the 512-bit message block is divided into 16 words of 32 bits $(W_0 \ldots W_{15})$ and these words are expanded to a sequence of 64 words through the message schedule:

$$\begin{aligned}
\sigma_0(X) &= ROTR^7(X) \oplus ROTR^{18}(X) \oplus SHR^3(X);\\
\sigma_1(X) &= ROTR^{17}(X) \oplus ROTR^{19}(X) \oplus SHR^{10}(X);\\
W_t &= \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}
\end{aligned}$$

where $ROTR^n$ is right rotation by n bits. SHACAL-2 encrypts the initial value using this sequence as a key.

The 8 registers are updated through a number of rounds. One round of the compression function is depicted in Fig. 1. The SHA-256 compression function consists of 64 rounds. Every round function has arithmetic addition, a round-dependent constant K_i, two linear functions Σ_0, Σ_1, and two non-linear functions CH, MJ.

$$\begin{aligned}
CH(X, Y, Z) &= (X \wedge Y) \oplus (\overline{X} \wedge Z);\\
MJ(X, Y, Z) &= (X \wedge Y) \oplus (Y \wedge Z) \oplus (Z \wedge X);\\
\Sigma_0(X) &= ROTR^2(X) \oplus ROTR^{13}(X) \oplus ROTR^{22}(X);\\
\Sigma_1(X) &= ROTR^6(X) \oplus ROTR^{11}(X) \oplus ROTR^{25}(X),
\end{aligned}$$

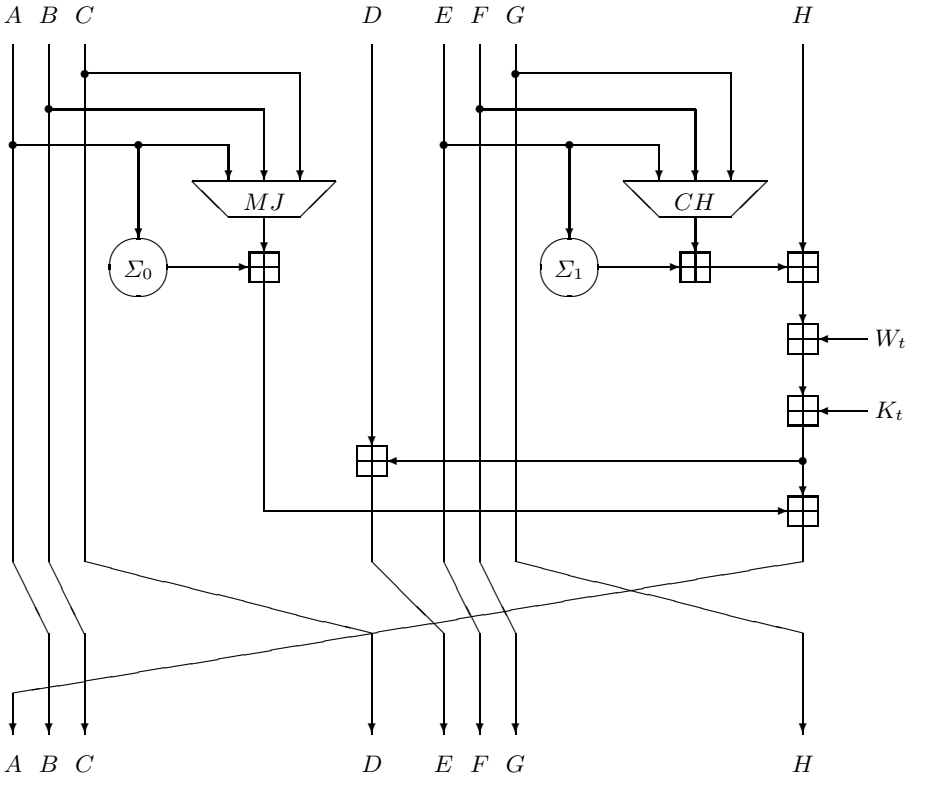


Fig. 1. Round function for SHA-256

where $\overline{X}$ is bitwise complement of X. The t-th round of the compression function updates the 8 registers using the word W_t and the constant K_i as input. The compression function updates the 8 registers according to the following algorithm:

$$\begin{aligned} T1_t(E_t, F_t, G_t, H_t, K_t, W_t) &= H_t + \Sigma_1(E_t) + CH(E_t, F_t, G_t) + K_t + W_t\,; \\ T2_t(A_t, B_t, C_t) &= \Sigma_0(A_t) + MJ(A_t, B_t, C_t)\,; \\ H_{t+1} = G_t; G_{t+1} &= F_t; F_{t+1} = E_t; E_{t+1} = D_t + T1_t\,; \\ D_{t+1} = C_t; C_{t+1} &= B_t; B_{t+1} = A_t; A_{t+1} = T1_t + T2_t. \end{aligned}$$

2.1 Our Variant of SHA-256

In our analysis, we simplify SHA-256 and SHACAL-2 by replacing all the arithmetic addition used in its round function by the XOR operation. This analysis tells us how much the carry propagation caused by the arithmetic addition affect the security of the cipher. It is also interesting for designers to investigate the security of an arithmetic-addition free hash function, because such a hash function has an advantage in its hardware implementation due to a lower gate count.

3 Previous Work

3.1 A Study on the Known Attacks on a Reduced Version of SHACAL-2

In the literature, two kinds of attacks on SHACAL-2 have been demonstrated. In [14], it was shown that the impossible differential attack [4] is applicable to the reduced 30-round SHACAL-2 with a time complexity $2^{495.1}$ and a memory complexity $2^{14.5}$. In [22] it has been shown that the differential-linear attack is applicable to the reduced 32-round SHACAL-2 with a complexity $2^{504.2}$ and a memory complexity $2^{48.4}$ which is the best attack so far. In the table 1, we list the best previous result and our result[1].

Table 1. The best previous result and our result

Attack type	#R	Data	Time	Memory
Impossible Differential attack on SHACAL-2[14]	30	744CP	$2^{495.1}$	$2^{14.5}$
Differential-linear attack on SHACAL-2[22]	32	$2^{43.4}$CP	$2^{504.2}$	$2^{48.4}$
Related-Key Rectangle attack on SHACAL-2[15]	37	$2^{43.2}$RK-CP	$2^{484.95}$	$2^{238.16}$
Distinguisher attack on SHACAL-2-XOR in this paper	31	2^{248}CP	2^{248}	
Differential attack on SHACAL-2-XOR in this paper	32	$2^{243.3}$CP	$2^{246.3}$	2^{22}

#R: # of rounds, CP: Chosen Plaintexts, RK-CP: Related-Key Chosen Plaintexts, Time: Encryption units, Memory: Bytes of memory

3.2 A Study on the Known Results on SHA-256

What has been known as results on cryptanalysis of the SHA-256 algorithm so far are several properties related to resistance of the function against the known attacks [11, 13] where none of the attacks have demonstrated any weakness in SHA-256 or any SHA-256 variant.

Hereafter we study the known results on resistance of SHA-256 against a theoretical attack on SHA-0[8] which have been very important results so far in the following sense: some strong attacks on the SHA algorithms have been developed by improving the attack. Two interesting strategies significantly reducing the complexity in the attack found collisions or near-collisions for the SHA-0 hash functions [2, 3].

We explain the procedure of the attack which is divided into two steps. This attack first finds a sequence of differences which is called local collision with a high-probability. An attacker introduces a 1-bit difference into one message word and then for the following rounds the attacker also introduced differences into the following message words so that the differences in the registers are canceled out, which results in a local collision with several rounds. As a result, the attacker has

[1] This distinguisher uses a differential characteristic for 31 rounds, it can be made more efficient by relaxing conditions of the final rounds. This is done for the differential attack which improves complexity of the attack and allows to recover the secret key bits.

obtained Recent works have given high-probabilities for the local collisions they identified, a probability 2^{-66} in [11], a better probability 2^{-39} in [13]. What we need to take into account in this step is that the attacker can choose the differences he injects whatever he likes, which means he does not care about the message schedule.

Secondly the attack analyzes the message schedule in an attempt to find two messages such that the message schedule will result in the collision obtained in the first step. However, it has been difficult to carry out this step for all the SHA-algorithms except for SHA-0 because of the large influence of the message schedule with respect to difference propagation.

4 Differential Cryptanalysis of SHA-2-XOR and SHACAL-2-XOR

4.1 Search for One-Round Iterative Differential Characteristics

We will search for one round iterative differential characteristics for SHA-2-XOR. We will first determine the constraints which an iterative characteristic should satisfy. Then we will develop an efficient algorithm to find all the differential characteristics satisfying the constraints and find one with the highest probability.

Let us denote the value in the register A at time t by A_t and the difference in this register at time t by dA_t. The t-th round changes the value A_t to A_{t+1} in the register A. The one-round iterative translates into conditions that in each register the differences at time t and at time $t+1$ are the same: $dA_{t+1} = dA_t, dB_{t+1} = dB_t, dC_{t+1} = dC_t, dD_{t+1} = dD_t, dE_{t+1} = dE_t, dF_{t+1} = dF_t, dG_{t+1} = dG_t, dH_{t+1} = dH_t$.

Our purpose here is to translate the constraints into the conditions with differences only at time t. There are 6 registers, in each of which value at time $t+1$ is determined by only one register value at time t. This builds the following simple relations between the differences at time t and the differences at time $t+1$: $dB_{t+1} = dA_t, dC_{t+1} = dB_t, dD_{t+1} = dC_t, dF_{t+1} = dE_t, dG_{t+1} = dF_t, dH_{t+1} = dG_t$. From these relations, 6 constraints $dB_{t+1} = dB_t$, $dC_{t+1} = dC_t$, $dD_{t+1} = dD_t$, $dF_{t+1} = dF_t$, $dG_{t+1} = dG_t$, $dH_{t+1} = dH_t$ are equivalent to the following conditions: $dA_t = dB_t = dC_t = dD_t, dE_t = dF_t = dG_t = dH_t$.

Now we have two remaining constraints $dA_{t+1} = dA_t$, $dE_{t+1} = dE_t$ to transform. We introduce several functions dCH, dMJ,$dT1_t$, $dT2_t$ each of which is the output difference of a sub-function used in the round function. These functions are defined as follows:

$$
\begin{aligned}
dCH &= CH(X,Y,Z) \oplus CH(X',Y',Z'),\\
dMJ &= MJ(X,Y,Z) \oplus MJ(X',Y',Z'),\\
dT1_t &= T1_t(E_t,F_t,G_t,H_t,K_t,W_t) \oplus T1_t(E'_t,F'_t,G'_t,H'_t,K_t,W_t),\\
dT2_t &= T2_t(A_t,B_t,C_t) \oplus T2_t(A'_t,B'_t,C'_t).
\end{aligned}
$$

We rewrite the non-linear functions CH, MJ in terms of their input values and input differences. Let's denote the input differences to the non-linear functions by

$dX_t = X_t \oplus X'_t$ for two input values X_t, X'_t. dCH is calculated as the following:

$$dCH = ((Y \oplus Z) \wedge dX) \oplus (X \wedge dY) \oplus (\overline{X} \wedge dZ) \oplus (dX \wedge dY) \oplus (dX \wedge dZ) \quad (1)$$

In particular, if all the differences to CH are equal, $dX = dY = dZ$, then

$$dCH = (Y \oplus \overline{Z}) \wedge dX. \quad (2)$$

dMJ is calculated as the following:

$$dMJ = MJ(dX, dY, dZ) \oplus ((Y \oplus Z) \wedge dX) \oplus ((Z \oplus X) \wedge dY) \oplus ((X \oplus Y) \wedge dZ). \quad (3)$$

In particular, if all the differences to MJ are equal, $dX = dY = dZ$, then

$$dMJ = dX \quad (4)$$

This tells an important property on MJ that this function behaves linearly if all the input differences are equal[2].

By using the formulas (2),(4) and the constraints obtained so far, $dT1_t$, $dT2_t$, dA_{t+1}, dE_{t+1} are calculated as follows:

$$dT1_t = dE_t \oplus \Sigma_1(dE_t) \oplus ((F_t \oplus \overline{G_t}) \wedge dE_t) \quad (5)$$

$$dT2_t = \Sigma_0(dA_t) \oplus dMJ(A_t, B_t, C_t) = \Sigma_0(dA_t) \oplus dA_t \quad (6)$$

$$dA_{t+1} = dT1_t \oplus dT2_t$$

$$dE_{t+1} = dD_t \oplus dT1_t = dA_t \oplus dT1_t$$

Therefore, the remaining constraints $dA_{t+1} = dA_t$, $dE_{t+1} = dE_t$ are equivalent to the following two conditions:

$$dA_t = dT1_t \oplus dT2_t$$

$$dE_t = dA_t \oplus dT1_t.$$

We can from now on omit the time indexes of differences, e.g. $dA_t = dA$. Then these two conditions are equivalent to following conditions:

$$dA = dT1_t \oplus dE \quad (7)$$

$$dE = dT2_t. \quad (8)$$

By the formula (6), the condition (8) is calculated as follows:

$$dE = \Sigma_0(dA) \oplus dA.$$

[2] This property has been noticed previously, for example see [11].

By the formula (5), the condition (7) is calculated as follows:

$$dA = dE \oplus \Sigma_1(dE) \oplus ((F \oplus \overline{G}) \wedge dE) \oplus dE.$$

Value $F \oplus \overline{G}$ can be considered to be some random value X. This condition is equivalent to the following condition:

$$dA = \Sigma_1(dE) \oplus (X \wedge dE).$$

We have determined the conditions for the existence of iterative. We now are interested in those iterative characteristic that have high probabilities. For an iterative with differences dA, dE, if some register inputs make this condition hold, they also make the other conditions hold. Therefore, we pay a probability only for this condition to hold. We see that we have to pay probability for this equation at bit position j to hold if and only if $dE^{(j)}$ is equal to 1. In particular, an iterative where Hamming weight of dE is the smallest has the best probability. This discussion leads us to the following theorem.

Theorem 1. *For SHA-2-XOR, a differential characteristic with input differences (dA,dB,dC,dD,dE,dF,dG,dH) is a one round iterative if and only if for some 32-bit value X, the input differences dA, dE satisfy the following:*

$$dA = \Sigma_1(dE) \oplus (X \wedge dE). \tag{9}$$
$$dE = \Sigma_0(dA) \oplus dA. \tag{10}$$

If this condition holds, the other differences in the characteristic are determined by dA and dE as follows:

$$dB = dA, dC = dA, dD = dA, dF = dE, dG = dE, dH = dE.$$

Furthermore, iterative where the weight of dE is the smallest has the best probability.

4.2 The Search Algorithm

We have to design an algorithm for practical use of the theorem. By substituting the second condition into the first one, we obtain the following:

$$dA = \Sigma_1(\Sigma_0(dA) \oplus dA) \oplus (X \wedge (\Sigma_0(dA) \oplus dA)).$$

It is sufficient for us to search for dA's which make this equation solvable in terms of X. Looking at this equation per bit leads us to consider a 1-bit equation $I = X \wedge R$. We consider what is the condition on I that the equation has a solution $X = X_0$, in each of two cases, $R = 0, R = 1$. In the case of R equal to 1, there always exists a solution. In the case of R equal to 0, there exists a solution if and only if I is equal to 0. Based on this consideration, now we can develop the following algorithm shown in Table 2 where for a bit string V, its value at bit position j is denoted by $V^{(j)}$.

Table 2. The search algorithm

Step1: Choose a 32-bit value, dA
Step2: Compute $R = \Sigma_0(dA) \oplus dA$.
Step3: Set u to be 0.
Step4: For j=0 to 31 do:
If $R^{(j)}$ is equal to 0, do
Compute $I^{(j)} = (\Sigma_1(\Sigma_0(dA) \oplus dA) \oplus dA)^{(j)}$
If $I^{(j)}$ is equal to 1, increase u by 1.
Otherwise, do nothing.
Step5: If u is equal to 0, then output dA.
Step6: If all possible value for dA have been chosen, then end.
Otherwise go to step1.

4.3 The Best One-Round Iterative Differential Characteristics

The algorithm we designed has identified *all* one round iterative characteristics for SHA-2. The running time was 30 min. Table 3 shows all the one-round iterative differential characteristic with the best probability 2^{-8}.

Table 3. One round iterative differential characteristic with the best probability 2^{-8}

$dA = dB = dC = dD$	$dE = dF = dG = dH$
3b3b3b3b	c0c0c0c0
67676767	18181818
76767676	81818181
9d9d9d9d	60606060
b3b3b3b3	0c0c0c0c
cecececе	30303030
d9d9d9d9	06060606
ecececec	03030303

It was confirmed that one of the best iterative with $dA = b3b3b3b3$, $dE = 0c0c0c0c$ has an experimental probability $259/(2^{16})$ which is around 2^{-8}. We can theoretically tell exactly what happens in one round. The only place where probabilities are paid is the place where the CH function is applied. The difference at the input of CH, 0c0c0c0c becomes 08080808 at the output with a probability 2^{-8}, which is calculated using the following differential property of CH per bit:

$$CH(0,0,0) = 0$$
$$CH(1,1,1) = 0/1 \text{ with probability } 1/2.$$

Note that the eight iterative patterns given in Table 3 are cyclic rotations of the same pattern. In the following section, we show that no 2-round iterative patterns better than a concatenation of two best one-round iteratives exist.

4.4 Search for 2-Round Iterative Differential Characteristics

We search for two-round iterative differential characteristics for SHA-2-XOR. However, we will show that no 2-round iterative patterns with probability higher than 2^{-16}. We first determine the constraints which an iterative pattern should satisfy. Let's denote the value(the difference) in the register A at time t by $A_t(dA_t)$. The t-th first rounds change the value A_t to A_{t+1} in the register A. The constraints are translated into conditions that in each register its difference at time t and its difference at time $t+2$ are the same: $dA_{t+2} = dA_t, dB_{t+2} = dB_t, dC_{t+2} = dC_t, dD_{t+2} = dD_t, dE_{t+2} = dE_t, dF_{t+2} = dF_t, dG_{t+2} = dG_t, dH_{t+2} = dH_t$.

Our purpose here is to translate the constraints into the conditions with differences only at time t. There are 4 registers, in each of which value at time $t+1$ is determined by only one register value at time t. This builds the following simple relations between the differences at time t and the differences at time $t+1$: $dC_{t+2} = dA_t, dD_{t+2} = dB_t, dG_{t+2} = dE_t, dH_{t+2} = dF_t$. From these relations, 4 constraints $dC_{t+2} = dC_t$, $dD_{t+2} = dD_t$, , $dG_{t+2} = dG_t$, $dH_{t+2} = dH_t$ equivalent to the following conditions: $dC_t = dA_t, dD_t = dB_t, dG_t = dE_t, dH_t = dF_t$.

Now we have 4 remaining constraints $dF_{t+2} = dF_t$, $dE_{t+2} = dE_t$, $dA_{t+2} = dA_t$, $dB_{t+2} = dB_t$ from which we can derive the following four conditions:

$$dB_t \oplus \Sigma_1 dE_t = dCH(E_t, F_t, G_t), \tag{11}$$

$$dA_t \oplus \Sigma_1 dF_t = dCH(E_{t+1}, E_t, F_t), \tag{12}$$

$$dF_t \oplus \Sigma_0 dA_t = dMJ(A_t, B_t, C_t), \tag{13}$$

$$dE_t \oplus \Sigma_0 dB_t = dMJ(A_{t+1}, A_t, B_t) \tag{14}$$

In our case, the conditions $dA_{t+1} = dB_t$, $dE_{t+1} = dF_t$ hold. Therefore we need to know what is the differential property of non-linear functions with some conditions on their input differences which is given in Table 4.

Table 4. A differential property on non-linear functions

$dX = dZ$	dY	dCJ	dMJ
0	0	0	0
0	1	0/1	0/1
1	0	0/1	0/1
1	1	0/1	1

We assume that there is an iterative with differences (dA,dB,dC,dD,dE,dF, dG,dH) have a probability at least 2^{-16}. Let us define α, β as follows:

$$\alpha = dCH(E_t, F_t, G_t) \oplus dCH(E_{t+1}, E_t, F_t).$$

$$\beta = dMJ(A_t, B_t, C_t) \oplus dMJ(A_{t+1}, A_t, B_t).$$

We know $\mathrm{Ham}(\alpha) \leq 8$ by studying (11) (12) and Table 4. We can assume $\mathrm{Ham}(\alpha)$ is more than 0, otherwise the search is reduced to the search for one-round

iterative pattens. We also know the following condition on $dE \oplus dF$ that holds for any bit position j,

$$\alpha^{(j)} = 0 \Longrightarrow (dE_t \oplus dF_t)^{(j)} = 0.$$

Hence, the number of possible values for $dE_t \oplus dF_t$ is $2^{Ham(\alpha)}$. By adding (11) and (12) we have the following:

$$dA_t \oplus dB_t = \Sigma_1(dE_t \oplus dF_t) \oplus \alpha \tag{15}$$

On the other hand, we have the following condition by adding (12) and (13),

$$dF_t \oplus dE_t \oplus \Sigma_0(dA_t \oplus dB_t) = \beta$$

Finally we obtain the following condition:

$$dF_t \oplus dE_t \oplus \Sigma_0(\Sigma_1(dE_t \oplus dF_t) \oplus \alpha) = \beta \tag{16}$$

Now we can compute $dA \oplus dB$ and β from α. From the discussion above, we also obtain the following property that holds for any bit position j,

$$(dA_t \oplus dB_t)^{(j)} = 0 \Longrightarrow \beta^{(j)} = 0.$$

However, it was confirmed that none of computed $dA_t \oplus dB_t$ and β satisfy this property. This was done with 2^{32} possible values for α. The total complexity is $2^{32+Ham(\alpha)} = 2^{40}$ elemental computations.

4.5 Pseudo-collision Attack on SHA-2-XOR Using Iterative Differential Characteristic

We present attacks on SHA-2-XOR and SHACAL-2-XOR using iterative differential characteristic we identified. We present two kinds of attacks on SHA-2-XOR.

By definition, to find a pseudo-collision, an attacker can inject differences both into the message schedule and registers. The attacker would require a complexity 2^{128} to find a pseudo-collision for a ideal hash function. We obtain a 15-round iterative with a probability 2^{-120} by concatenating one of the best one-round iterative we identified. This leads to an attack finding a pseudo-collision with a complexity 2^{120} for the 15-round SHA-2-XOR.

Our attack suggests a security model where an attacker can inject differences only into registers. Taking into account the feed-forward operation of the Davis-Meyer mode, to find a collision means to find a differential characteristic for the underlying block cipher where an input difference and an output difference of are same. In the ideal case, if both of an input difference and an output difference are fixed, then the probability that a plaintext pair with the input difference results in the output difference is 2^{-256}. However, SHA-2-XOR with 31 rounds has a probability 2^{-248} which means that 31 rounds of this hash function does not behave as a random hash function.

4.6 Differential Attack on 32-Round SHACAL-2-XOR

As for SHACAL-2-XOR, we can build a 31-round characteristic with a probability 2^{-248} concatenating one of the best iterative differential characteristics we identified. This shows SHACAL-2-XOR with 31 rounds is distinguished from a random permutation. We now attack SHACAL-2-XOR with 32 rounds by using the 30-round differential characteristic. Our goal here is to find the 32-bit key W_{31}. Let δ be the input difference in the 30-round characteristic(e.g. dA_0=dB_0=dC_0=dD_0= `3b3b3b3b`, dE_0=dF_0=dG_0=dH_0= `c0c0c0c0`). We denote a plaintext P at time t by P_t and the value of P_t in the register A by A_t. We denote the difference between a pair of plaintexts (P, P^*) at time t by Δ_t and the difference of d_t in the register A by dA_t. Let (P, P^*) be a pair of plaintexts with the difference δ:$\Delta_0 = \delta$ The pair of corresponding ciphertexts is (P_{32}, P^*_{32}). There are two steps to perform our attack, data collection step, data analysis step. In the data collection step, we encrypt $2^{240} \cdot 10$ plaintext pairs. Then we collect only $2^{16} \cdot 10$ pairs needed for the next step by checking if the corresponding ciphertexts pairs satisfy certain conditions. Let us see what this condition looks like. For the right pairs, the condition $\Delta_{30} = \delta$ holds. For the last 2 rounds, we observe how this difference behaves in case of not paying any probability. Even in this case, in the 4 registers, the differences at time 32 are determined uniquely by the differences at time 30: $dC_{32} = dA_{30}$, $dD_{32} = dB_{30}$, $dG_{32} = dE_{30}$, $dH_{32} = dF_{30}$. By studying how non-linear functions increase the uncertainty of differences, we can see there are 2^{16} candidates for the differences in the other 4 registers at time 32: (dA_{32}, dB_{32},dE_{32},dF_{32}).

In the data analysis step, we find 8bits of 32-bit key W_{31} using 2^8 counters. Each pair suggests one key therefore one counter is 2^8 in average, while the counter for the correct key bits is $2^8 \cdot 10$. This enables us to detect the correct key bits. Using another three iterative characteristics we identified, we can find another 24 bits of W_{32}.

The time complexity of this attack is $2^{246.3}$(= 2^{240}·10·2·4) 32-round SHACAL-2-XOR encryptions and the data complexity of this attack is $2^{243.3}$ chosen plaintexts which are immediately discarded leaving only 2^{17} for the analysis step.

4.7 Improvement of the Pseudo-collision Attack

In the previous section, we identified one-round iterative differential characteristics. Using the best ones with the probability 2^{-8}, we attacked 15 rounds of SHA-2-XOR regarding pseudo-collision resistance. Here we will improve this result and add more rounds.

In the pseudo-collision attack model, the attacker choose any element from the set $I_{all} = \{0,1\}^{256} \times \{0,1\}^{512}$, which is taken as input to the compression function. The main idea in our improvement is to use a subset of I_{all} denoted by I_{sub} for which better probabilities for many rounds are obtained. This idea was already indicated in [19] where it is pointed out that the attacker can choose the message so that the first several rounds follow the characteristic with probability 1. It is quite natural to consider this idea in cryptanalysis of hash functions. Recently, this idea was effectively used in the attacks in [23].

To realize this idea in practice, the attacker first randomly choose an input from I_{all} and then modifies it in a way that certain condition on the register values $E_t, F_t, G_t, t = 0, 1, \ldots, 17$ in the Table 5 is satisfied. Using the resulting the set of modified inputs, we do not have to pay probability for the first 19 rounds.

Now we develop an algorithm of the input modification. Firstly, we fix one of the iterative characteristic to δ as the previous section. Let L be the constant value:`0x08080808` and J be the set of bit positions: {2, 3, 10, 11, 18, 19, 26, 27}. Studying the proof of the above theorem tells us the condition for register values at each time to result in the required difference after 19 rounds.

Table 5. The condition for register values at each time to result in the required difference after 19 rounds

$$
\begin{array}{ll}
(F_0 \oplus G_0)^{(j)} = L^{(j)} & (j \in J) \\
(E_0 \oplus F_0)^{(j)} = L^{(j)} & \\
(E_t \oplus E_{t+1})^{(j)} = L^{(j)} & (j \in J, t = 1, 2, \ldots, 17)
\end{array}
$$

Taking this condition into account, we develop the following algorithm shown in Table 6 where for a bit string V, its value at the bit position j is denoted by $V^{(j)}$.

Table 6. The input modification

Step1: Choose randomly an initial resister values: $A_0, B_0, C_0, D_0, E_0, F_0, G_0, H_0$
Step2: Choose randomly a message block of 16 words: $W_0, W_1, \ldots, W_{15}$
Step3: Replace 8 bits of $G_0^{(j)}$ and $E_0^{(j)}$ by 8 bits of $(F_0 \oplus L)^{(j)} (j \in J)$
Step4: For $t = 0$ to 15 do:
  Compute the value: $\alpha = E_{t+1} \oplus E_t \oplus L$
  Replace 8 bits of $W_t^{(j)}$ by 8bits of $\alpha^{(j)} (j \in J)$
  Apply the t-th round function with the resulting W_t
Step5: Copy the value from W_0 to the variable: W_0^{old}
Step6: Compute the value:
  $\beta = D_{16} \oplus H_{16} \oplus \Sigma_1(E_{16}) \oplus CH(E_{16}, F_{16}, G_{16}) \oplus K_{16} \oplus L \oplus E_{16}$
Step7: Compute $W_{16} = \sigma_1(W_{14}) \oplus W_9 \oplus \sigma_0(W_1) \oplus W_0$.
Step8: Replace 8 bits of $W_{16}^{(j)}$ by 8 bits of $\beta^{(j)} (j \in J)$
Step9: Replace $W_0^{(j)}$ by the value: $(W_{16} \oplus \sigma_1(W_{14}) \oplus W_9 \oplus \sigma_0(W_1))^{(j)} (j \in J)$
Step10: Replace $H_0^{(j)}$ by the value: $(H_0 \oplus W_0 \oplus W_0^{old})^{(j)} (j \in J)$

The algorithm involves a modification of 152 input bits(=19× 8 bits), that is, $E_0^{(j)}, G_0^{(j)}, H_0^{(j)}, W_0^{(j)}, W_1^{(j)}, \ldots, W_{15}^{(j)} (j \in J)$. All the modified inputs with the difference δ results in δ again after 19 rounds, which was experimentally confirmed with 2^{20} randomly chosen inputs. We use 120 input bits out of the remaining 616 bits to add 15 rounds. This leads to an attack finding a pseudo-collision with a complexity 2^{120} for the 34-round SHA-2-XOR.

4.8 An Example of a 23-Round Pseudo-collision for SHA-2-XOR

In the Table 7 Here we list an example of a pseudo-collision producing input to SHA-2-XOR with reduced rounds. Our approach found a 23-round pseudo-collision for SHA-2-XOR with a complexity 2^{32}.

Table 7. A Message and Register values producing a 23-round pseudo-collision for SHA-2-XOR

Message words $W_0, W_1, ..., W_{15}$:			
0xe97ae8e7	0x695655dd	0x57e9383b	0x8c916172
0x68e61dd1	0x2bc71033	0x081dae0f	0x5546e057
0xfd1450ef	0xcb398b6a	0xa16bf40c	0xfc7bb645
0x14b17c9c	0x1b2a8265	0xa17f20c4	0xe8f96137
Register values $(A_0, B_0, C_0, D_0, E_0, F_0, G_0, H_0)$:			
0x4939a45a	0x79ec4172	0xf0ef5249	0x29b5bb6f
0xd92f76e4	0x21962dfe	0xd88e64f6	0x7b624d63

4.9 The Impact on Round-Reduced Versions of the Actual SHA-256

Since our attack on SHA-2-XOR is based on one-round iterative characteristic whose Hamming weight is relatively high, it is unlikely to obtain a high probability for the same characteristic in the case of the actual the SHA-256 hash function. Therefore it is not possible to apply our attack to the actual SHA-256 in a straightforward way.

5 Conclusions

We considered a SHA-256 variant and a SHACAL-2 variant. We presented a differential attack on these ciphers. Our result shows that SHACAL-2-XOR with up to 31 rounds has a weakness of randomness and that SHA-2-XOR with up to 34 rounds has a weakness of pseudo-collision resistance. We also presented an attack on SHACAL-2-XOR with up to 32 rounds by using the 31-round distinguisher.

Acknowledgements

The authors would like to thank Bart Preneel for his suggestions towards this analysis. We also would like to thank Joseph Lano and Souradyuti Paul for helpful comments and useful discussions. We are also grateful to the anonymous referees for their valuable remarks.

References

1. E. Biham, "New Results on SHA-0 and SHA-1," Invited talk presented at SAC 2004.
2. E. Biham, R. Chen "Near-Collision of SHA-0," in *Proceedings of CRYPT 2004*, LNCS 3152, M. Franklin, Ed., pp.290–305, 2004.

3. E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby, "Collisions of SHA-0 and Reduced SHA-1," in *Proceedings of Eurocrypt 2005*, LNCS 3494, R. Cramer, Ed., Springer-Verlag, pp. 36–57, 2005.
4. E. Biham, A. Biryukov, A. Shamir, "Cryptanalysis of SkipJack Reduced to 31 Rounds Using Impossible Differentials," in *Proceedings of Eurocrypt'99*, LNCS 1592, pp.12–23, 1999.
5. E. Biham, A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.
6. A. Biryukov, D. Wagner, "Advanced slide attacks," in *Proceedings of Eurocrypt 2000*, LNCS 1807, B. Preneel, Ed., Springer-Verlag, pp. 589–606, 2000.
7. B. D. Boer, A. Bosselaers, "Collisions for the compression function of MD5," in *Proceedings of Eurocrypt 1993*, LNCS 765, T. Helleseth, Ed., Springer-Verlag, pp. 293–304, 1993.
8. F. Chabaud and A. Joux, "Differential Collisions in SHA-0," in *Proceedings of CRYPTO'98*, LNCS 1462, H. Krawczyk, Ed., pp.56-71, Springer-Verlag, 1998.
9. I. Damgård, "A design principle for hash functions," in *Proceedings of Crypto'89*, LNCS 435, G. Brassard, Ed., Springer-Verlag, pp. 416–427, 1990.
10. H. Dobbertin, "The status of MD5 after a recent attack,", Cryptobytes, Vol. 2, No. 2, pp. 1–6, Summer 1996.
11. H. Gilbert, H. Handschuh, "Security Analysis of SHA-256 and Sisters," in *Proceedings of SAC 2003*, LNCS 3006, M. Matsui and R. Zuccherato, Eds., Springer-Verlag, pp. 175–193, 2004.
12. H. Handschuh, D. Naccache, "SHACAL," Submission to the NESSIE project, 2000. Available from `http://www.gemplus.com/smart/r_d/publications/pdf/HN00shac.pdf`.
13. P. Hawkes, M. Paddon, and G.G. Rose, "On Corrective Patterns for the SHA-2 Family," Cryptology ePrint Archive August 2004. Available from `http://eprint.iacr.org/`.
14. S. Hong, J. Kim, G. Kim, J. Sung, C. Lee, and S. Lee, "Impossible Differential Attack on 30-Round SHACAL-2," in *Proceedings of INDOCRYPT 2003*, LNCS 2904, T. Johansson and S. Maitra, Ed., Springer-Verlag, pp. 97–106, 2003.
15. J. Kim, G. Kim , S. Lee, J. Lim, and J. Song, "Related-Key Attacks on Reduced Rounds of SHACAL-2,"in *Proceedings of INDOCRYPT 2004* , LNCS 3348, A. Canteaut and K. Viswanathan Ed., Springer-Verlag, pp. 175–189 2004.
16. L. R. Knudsen and J. E. Mathiassen, "Preimage and collision attacks on MD2," in *Proceedings of FSE 2005*, LNCS 3557, H. Gilbert and H. Handschuh Ed., Springer-Verlag, pp. 255–267, 2005.
17. A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
18. National Institute of Standards and Technology, FIPS-180-2: "Secure Hash Standard (SHS)," August 2002.
19. V. Rijmen, B. Preneel, "Improved characteristics for differential cryptanalysis of hash functions based on block ciphers," Fast Software Encryption, Lecture Notes in Computer Science 1008, B. Preneel, Ed., Springer-Verlag, 1995, pp. 242-248.
20. R. Rivest, "The MD5 message-digest algorithm," Request for Comments (RFC) 1321, Internet Activities Board, Internet Privacy Task Force, April 1992.
21. M. Saarinen, "Cryptanalysis of Block Ciphers Based on SHA-1 and MD5," in *Proceedings of FSE 2003*, LNCS 2887, T. Johansson, Ed., Springer-Verlag, pp. 36–44, 2003.

22. Y. Shin, J. Kim, G. Kim, S. Hong, and S. Lee, "Differential-Linear Type Attacks on Reduced Rounds of SHACAL-2," in *Proceedings of ACISP 2004*, LNCS 3108, H. Wang, J. Pieprzyk, and V. Varadharajan, Ed., Springer-Verlag, pp. 110–122, 2004.
23. X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu, "Cryptanalysis of the Hash Functions MD4 and RIPEMD," in *Proceedings of Eurocrypt 2005*, LNCS 3494, R. Cramer, Ed., Springer-Verlag, pp. 1–18, 2005.
24. H. Yoshida, A. Biryukov, C. D. Cannière, J. Lano, and B. Preneel, "Non-randomness of the Full 4 and 5-pass HAVAL," in *Proceedings of SCN 2004*, LNCS 3352, C. Blundo and S. Climato, Ed., Springer-Verlag, pp. 324–336, 2005.

Impact of Rotations in SHA-1 and Related Hash Functions*

Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Austria
{Norbert.Pramstaller, Christian.Rechberger,
Vincent.Rijmen}@iaik.tugraz.at

Abstract. SHA-1 uses a single set of rotation constants within the compression function. However, most other members of the MD4 family of hash functions use multiple sets of rotation constants, *i. e.* the rotation amounts change with the step being processed.

To our knowledge, no design rationales on the choice of rotation constants are given on any of these hash functions. This is the first paper that analyzes rotations in iterated hash functions. We focus on SHA-1-like hash functions and use recent developments in the analysis of these hash functions to evaluate the security implications of using multiple sets of rotation constants in the compression function instead of a single set. Additionally, we give some observations on the set of constants used in SHA-0 and SHA-1.

1 Introduction

SHA-0 was introduced in 1993 and SHA-1 was introduced in 1995 without giving any rationales on the design. Both are based on the MD4 design strategy, however the used message expansions are more complex. Additionally, a *single* set of rotation constants instead of *multiple* sets are used during state update, *i. e.* the rotation constants remain the same for all steps. Later on, in 1998, the hash function HAS-160 was specified for use in South Korea's Digital Signature Algorithm. The structure of HAS-160 is very similar to SHA-1. However, one distinct feature is the use of multiple sets of rotation constants (as in MD4 and MD5) instead of a single set. Several questions are open so far:

1. Why were the rotation constants for SHA-1 chosen as they are?
2. Would there be better choices for these constants from a security point of view?
3. Is there a security advantage of using multiple sets of rotation constants instead of a single set?

We attempt to give some answers to these questions. To our knowledge this is the first article which deals with the issue of rotation constants in iterated hash

* The work in this paper has been supported by the Austrian Science Fund (FWF), project P18138.

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 261–275, 2006.

functions. The outline and main contributions of this article are as follows. In Section 2, we give a short description of SHA-1 and HAS-160. Afterwards, in Section 3, we review and comment on currently known analysis strategies for SHA-1. This review serves as an important starting point for some comparisons done later in this article. Looking at HAS-160, we see that due to its non-recursive message expansion the basic building block for most of these strategies (elementary collisions as introduced by [3]) can not be directly applied. However, the Rijmen-Oswald extension [15] can be used to overcome these problems.

Afterwards we turn to the influence of multiple sets of rotation constants. We analyze the effect of multiple sets of rotation constants in simplified models in Section 4. We show that these multiple sets improve the avalanche effect in the first steps of our simplified, linearized model.

Section 5 contains the main contribution. We compare single and multiple sets of rotation constants in hash functions like SHA-1, HAS-160 and variations thereof. We identify two reasons why the complexity of an attack increases when multiple sets are used. Firstly, we show that the weight of collision-producing differences increases and secondly, we show that more conditions are needed due to an optimization trick which is less effective with multiple sets. Here, we also give a first observation on the design of SHA-1. For 80 or more steps, the benefits of multiple sets over a single set of rotation constants is negligible (in terms of Hamming weight of a collision-producing difference). Additionally, we analyze the attack complexity for variants of SHA-1 having different single sets of rotation constants. We show that in the case of full SHA-1 (80 steps), rotating the chaining variable A by *5* to the left and chaining variable B by *2* to the right are the smallest possible values which do not impair security. Finally, we discuss advantages of having these small constants.

1.1 Used Notation and Terminology

Table 1 contains a description of symbols used throughout this article. Note that if negative integers are used as rotation constants, the rotation direction is reversed from left to right. Whenever we talk about Hamming weight of differential

Table 1. Used notation

notation	description
$A \oplus B$	addition of A and B modulo 2 (XOR)
$A + B$	addition of A and B modulo 2^{32}
$A \vee B$	logical OR of two bit-strings A and B
M_t	input message word t (32-bits), index t starts with 0
W_t	expanded input message word t (32-bits), index t starts with 0
$A \lll n$	bit-rotation of A by n positions to the left, $0 \leq n \leq 31$
step	the SHA-1 compression function consists of 80 steps
round	the SHA-1 compression function consists of 4 rounds $= 4 \times 20$ steps
N	number of steps of the compression function

patterns we refer to the smallest Hamming weight we found using an adapted version [13] of Leon's algorithm [7] for finding low-weight words in linear codes.

2 Description of Used Hash Functions

In this section, we shortly describe SHA-1 and the differences of HAS-160 compared to SHA-1.

2.1 SHA-0 and SHA-1

The SHA family of hash functions is described in [11]. Briefly, their compression function consists of two phases: a message expansion and a state update transformation. These phases are explained in more detail in the following. SHA-0 and SHA-1 share the same state update, but SHA-0 has a simpler message expansion. Both SHA-0 and SHA-1 consist of 80 steps. Since we will study variable-step versions in this article, we denote the number of steps by N.

Message Expansion. In SHA-1, the message expansion is defined as follows. The input is a 512-bit message, denoted by a row vector m. The message is also represented by 16 32-bit words, denoted by M_t, with $t = 0, 1, \ldots, 15$.

In the message expansion, this input is expanded linearly into N 32-bit words W_t, also denoted as the $32N$-bit expanded message word w. The words W_t are defined as follows.

$$W_t = M_t, \quad t = 0, \ldots, 15 \tag{1}$$

$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1, \quad t > 15 \tag{2}$$

The message expansion of SHA-0 is very similar, but uses:

$$W_t = W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}, \quad t > 15\ . \tag{3}$$

Consequently, a bit at a certain position i in one of the words of w only depends on the bits at corresponding positions in the words of m.

State Update Transformation. The state update transformation starts from a (fixed) initial state for 5 32-bit registers and updates them in N steps, using one word W_t in every step. Figure 1 illustrates one step of the state update transformation. The function f depends on the step number: steps 0 to 19 (round 1) use the *IF-function* and steps 40 to 59 (round 3) use the *MAJ-function*.

$$f_{\mathrm{if}}(B, C, D) = BC \oplus \overline{B}D \tag{4}$$

$$f_{\mathrm{maj}}(B, C, D) = BC \oplus BD \oplus CD \tag{5}$$

The remaining rounds use a 3-input XOR. A round constant K_t is added in every step. There are four different constants; one for each round. After the last application of the state update transformation, the initial register values are

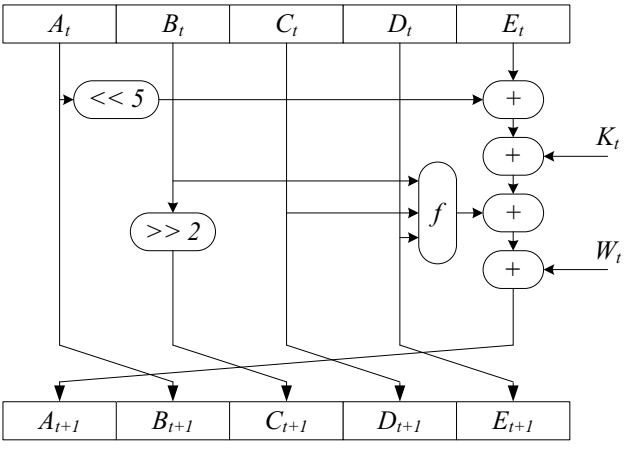


Fig. 1. One step of the state update transformation of SHA-1

added to the final values, and the result is either the input to the next iteration function or the final digest.

2.2 HAS-160

HAS-160 [17] is designed for use with the South Korean KCDSA digital signature algorithm [16]. The output length is 160 bits. A security evaluation of KCDSA by Lim *et al.* can be found in [9, 5]. An independent English description of HAS-160 is available [10, 8]. HAS-160 can be seen as a predecessor of the HAS-V family of hash functions proposed in [12]. The design is based on SHA-1, however some features are distinct. Subsequently, only the differences to SHA-1 are described.

Round Constants. HAS-160 uses a different set of round constants. We do not need their actual values in this article.

Message Expansion. In SHA-0 and SHA-1, 16 input message words M_t are expanded into 80 expanded message words W_t using a recursive definition. In HAS-160, the 16 input words are expanded into 20 words (differently for each round) and permuted for each of the four rounds. For actual permutation tables and expansion tables, refer to [10, 8].

Boolean Functions in the State Update. The only difference to SHA-1 is the 3-input Boolean function used for steps 40-59. We denote this function f_{has3}.

$$f_{\text{has3}}(B, C, D) = C \oplus (B \vee \overline{D}) \tag{6}$$

The impact of this difference with respect to collision-search attacks is analyzed in Section 5.2.

Rotations in the State Update. In SHA-0 and SHA-1, the chaining variable A_t is rotated by 5 bit-positions to the left before it is input to a modular addition.

In HAS-160, this single rotation constant is replaced by multiple constants, *i. e.* each step within a round rotates A_t differently. The actual values are

$$S_1(t \bmod 20) = \{5, 11, 7, 15, 6, 13, 8, 14, 7, 12, 9, 11, 8, 15, 6, 12, 9, 14, 5, 13\} , \quad 0 \leqslant t \leqslant 79 . \tag{7}$$

In SHA-0 and SHA-1, the chaining variable B_t is rotated by 30 bit-positions to the left before it becomes variable C_{t+1}. In HAS-160, this single rotation constant is replaced by multiple rotation constants for each round. The actual values are

$$\begin{aligned} S_2(t) &= 10, \quad 0 \leqslant t \leqslant 19 , \\ S_2(t) &= 17, \quad 20 \leqslant t \leqslant 39 , \\ S_2(t) &= 25, \quad 40 \leqslant t \leqslant 59 , \\ S_2(t) &= 30, \quad 60 \leqslant t \leqslant 79 . \end{aligned} \tag{8}$$

Note that this concept of having multiple sets of rotation constants is different to what is referred to as data dependent rotations (DDR).

3 Outline of Recent Attacks on SHA-0 and SHA-1

In this section we give an overview and comment on all the analysis techniques that were used in recent years to analyze SHA-0 or SHA-1. The content of this section is the basis for our approach to compare variants of SHA-1 later in this article.

3.1 Differential Characteristics

Most recent collision attacks use the following strategy. Firstly, a differential characteristic through the compression function of the hash function is constructed. Secondly, messages are constructed, which follow the characteristic.

3.2 Original Chabaud-Joux Approach

In the original approach of Chabaud and Joux [3], the differential characteristic is determined by constructing a linear approximation for all the nonlinear elements of SHA. Subsequently, Chabaud and Joux look for a differential characteristic through this linear approximation. Since a differential characteristic propagates in a deterministic way through a linear function, the characteristic is determined completely by the choice of input difference. Hence, there are 2^{512} different characteristics. A fraction of 2^{-160} of these, results in a zero output difference (a collision).

Chabaud and Joux use the same linear approximation in every step. Consequently, every local collision contains the same number of corrections, *i. e.* 5. They impose the additional constraint that the pattern of perturbations is a valid expanded message, which accounts for another reduction factor 2^{-160}. Hence, there remain 192 "free" bits.

3.3 Rijmen-Oswald Extension

In [15], it is proposed to drop the condition that the perturbation pattern should be a valid expanded message. Any *collision-producing difference*, *i. e.* input difference that produces output difference zero in the linearized model, can be used. This approach increases the number of free bits to 352. The approach still results in collisions that are linear combinations of local collisions, each consisting of a perturbation and 5 corrections, but there are now less restrictions on the perturbation pattern.

Until now, it hasn't been demonstrated that this extension can result in better differential characteristics for SHA-1. However, for other hash functions, the improvement could be significant.

3.4 Multi-block

In multi-block collisions, we can also use differentials that don't result in a zero output. For instance, in a two-block collision, all we require is that the output difference in both blocks is equal, because then, final feed-forward will result in cancelation of the differences (with a certain probability). For a z-block collision, we get $512z-160$ free bits ($512z-320$ if we require that the perturbation pattern is a valid expanded message).

3.5 Exploiting Non-linearity—Improvements by Wang *et al.*

If we study the differentials used by Wang *et al.* [21, 19], then we see that they create even more freedom by allowing differential characteristics that don't follow the linear approximation in the first steps. The propagation of differences through nonlinear functions is non-deterministic and this can be exploited. The possibility to exploit non-linear behavior is also observed in [2]. The Rijmen-Oswald extension can be adapted to exploit this additional freedom.

3.6 Removing Conditions

For the second step of the attack, constructing a pair of messages that follows this characteristic, a number of conditions on message words and intermediate chaining variables need to be fulfilled. As already observed in [3], conditions on the first steps can be pre-fulfilled. Using the fact that their exist neutral bits in the compression function, this approach was extended to cover the first 20-22 steps of SHA-0 [1]. Wang *et al.* employ a different technique called message modification in their collision-search attacks on SHA-0 [21] and SHA-1 [19] to pre-fulfill the conditions in more than 20 steps. Note that a variant of this technique is also used in the analysis of MD4 and MD5 [18, 20, 6].

3.7 Decrease Final Search Complexity

There are still several ways to speed up the attack. Firstly, it is advantageous to choose the bit position of message differences to be in the MSB, since there a possible change in the carry has no effect. The reason for this is that in this case

no condition on message words and chaining variables are necessary to prevent this carry. A simple optimization is therefore to rotate each word of the difference such that the number of MSBs of value 1 is maximized.

A second trick deals with the implementation of the final search. After pre-fulfilling some conditions, the remaining conditions can only be fulfilled using random trials. There are two natural ways to talk about the time-complexity of this final search. One is to use the numbers of message pairs needed. In this case, the time complexity can be estimated by 2^c, where c corresponds to the number of conditions that cannot be pre-fulfilled. Another way of looking at it is to use the number of steps as a means to express time complexity. It seems natural to define the time-complexity 1 to be the N steps of the compression function.

A "good" pre-computed 13-word or 14-word message-pair can be used as a starting point. Depending on the conditions on the message words m_{14}, m_{15} and m_{16} we get up to 96 degrees of freedom for our final search. If all degrees of freedom are used without finding a collision, a new pre-computed message pair is needed. However, we can stop our step-computations after the first condition with probability $p_1 = 1/2$, after the second condition $p_2 = 1/4$ and so on. Since we assume random trials for conditions we cannot pre-fulfill, we always estimate the probability for fulfilling the conditions to be 0.5.

Starting from step 13 or 14, on average 10 steps are enough. Therefore we estimate the time-complexity of our final search to be 2^{c-3} (for $N = 80$). Joux *et al.* [4] use a similar reasoning and arrive at 2^{c-2}. The difference is that there the same amount of computation is assumed for the second message pair. However, the steps for the second message pair are only needed if all conditions are fulfilled in order to check if the messages really collide.

3.8 Application of Attacks to HAS-160

Due to the non-recursive structure of the message expansion of HAS-160, a direct application of the Chabaud-Joux approach is not possible. However, using the approach described in [15], generating a differential characteristic for HAS-160 is straightforward. Some details on constructing messages that follow this characteristic are given in Section 5.2.

4 Rotations During the Step Update—Analysis of Simplified Models

The original step update function of HAS-160 is defined as follows:

$$A_{t+5} = (A_{t+4} \lll S_0) + f(A_{t+3}, A_{t+2} \lll S_1, A_{t+1} \lll S_1) + A_t \lll S_1 + W_t + K_t, \tag{9}$$

whereas S_0 has different values for each step of a round and S_1 has different values for different rounds as defined in Section 2.2. Note that the SHA-1 state update has the same structure, but S_0 has always a value of 5 and S_1 has always a value of 30.

In order to analyze the impact of the multiple constants in S_0, we use a simplified model of the state update. We replace the modular addition by an XOR and the function f by a 3-input XOR. In a first approximation, we consider only two chaining variables, and therefore only one bit-rotation.

$$A_{t+2} = (A_{t+1} \lll S_0) \oplus A_t \tag{10}$$

If we introduce a single-bit difference in one of the chaining variables of Equation 10, we observe an increase of the Hamming weight of this difference with increased number of steps. Our first observation is that whenever S_0 is constant and a multiple of 2, the number of affected bits decreases. Summing over all 80 steps, we get a maximum of 283 affected bits in the chaining variables when we apply a single-bit difference in the first chaining variable. When we do the same computations for multiple rotation constants we get a total of 1077 affected bits. Note that if half of the bits would be affected in each step, we would arrive at $80 \times 16 = 1280$ affected bits.

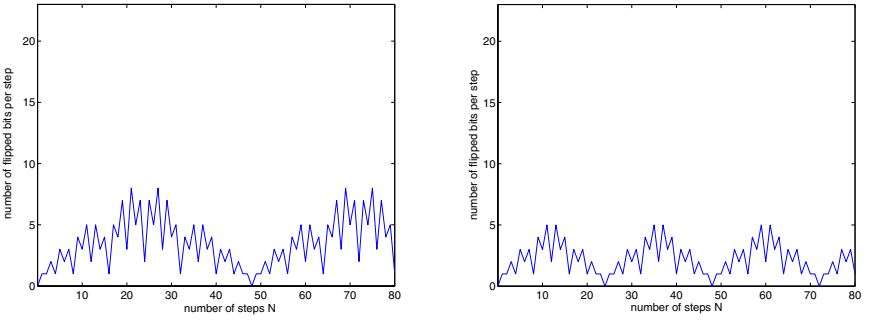


Fig. 2. Number of affected bits per step for constant bit-rotations. The constant is not a multiple of 2 in the left figure. In the right figure, the used constant is a multiple of 2.

Figure 2 gives another point of view: the number of affected bits per step for a single rotation constant. Here we distinguish between cases where the value for the bit-rotations is a multiple of 2, and where this is not the case. The symmetry in Figure 2 can be explained by our simplified and linearized model. If we apply the same method to compute the number of affected bits for the case of multiple rotation constants, we get the result shown in Figure 3.

We observe a much steeper increase in the first rounds. Due to the multiple rotation constants, differences do not cancel out early. Later on, the ideal 16 affected bits per round are reached.

Next, we extend our model to three chaining variables to contain the second bit-rotation of variable B. The resulting equation is as follows:

$$A_{t+3} = (A_{t+2} \lll S_0) \oplus A_{t+1} \oplus A_t \lll S_1 \ . \tag{11}$$

Our simulation results and conclusions are pretty similar to the case for two chaining variables. Therefore we omit them. However, we found an example

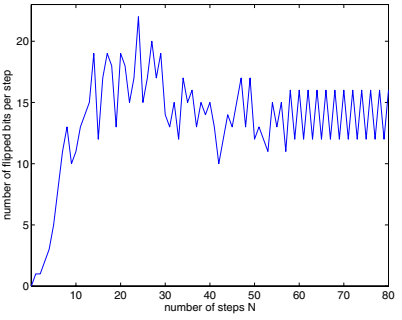


Fig. 3. Number of affected bits per step for multiple rotation constants

were we "outperformed" the ideal case: we used a constant 24-bit rotation for A and "pseudo-random" rotations for B. Using this setting, we arrived at 1352 bit-flips after 80 steps.

5 Impact of Multiple Rotation Constants on the Attack Complexity

In this section, we are comparing several variants of SHA-1. We use the approach described in [15] to find low-weight input differences, which in turn can be used to analyze the complexity of a collision-search attack. Even if we consider the recent results by Wang *et al.* , comparing Hamming weights using this method is sound since the underlying principle is the same.

Quote from [12]: "*The variable shift amount seems to provide better immunity against attacks such as differential collision in SHA-0 [3]. The generalization of inner collisions to a full compression function seemed to be harder with variable shift amounts.*"

The method of [3] assumes a message expansion defined by a recursion, which is a reason for the difficulties of applying this approach to HAS-160. However, these problems are overcome if the Rijmen-Oswald extension is applied.

Multiple rotation constants account for a slightly increased Hamming weight of collision-producing differences, which in turn slightly increases the number of conditions that have to be fulfilled in the final search for a collision. Later on, we will show that this increase is negligible after 80 or more steps. There are two reasons why multiple rotation constants result in higher collision-search complexity:

1. Higher Hamming weight of the collision-producing difference in the linearized model
2. It is less likely to take advantage of some condition-reducing effects.

5.1 Higher Hamming Weight of the Collision-Producing Difference in the Linearized Model

We consider the first point now. In order to study the effect of different rotation constants in an actual design, we searched for low-weight collision-producing

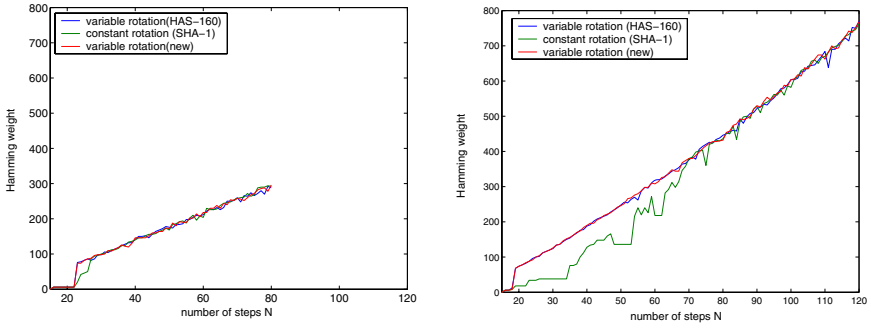


Fig. 4. Weight of collision-producing differences for single and multiple sets of rotation constants. On the left, the HAS-160 message expansion is computed for up to 80 steps. On the right, the SHA-1 message expansion is computed for up to 120 steps.

differences in variants of SHA-1, where we slightly changed the state update transformation.

Firstly, we compare the state update transformations used by SHA-1 and HAS-160. The result is depicted in Figure 4.

We consider three different state update variations. The original HAS-160 state update having multiple sets of rotation constants, the original SHA-1 state update having a single set of rotation constants and a new state update having different multiple sets of rotation constants. These variations of the state update are combined with both the HAS-160 message expansion (depicted on the left) and the SHA-1 message expansion (depicted on the right).

When looking at the values, we observe that using the HAS-160 message expansion instead of the SHA-1 message expansion actually decreases the best found Hamming weight. We also see that the lower Hamming weights for versions using a single set of rotation constants catch up on the Hamming weights of the variants with multiple sets with increased number of steps. In the case of the HAS-160 message expansion, this happens after 30 steps. In the case of SHA-1 the difference between single and multiple sets of rotation constants vanishes after 80 steps. This gives us a first hint on the choice made by the designers of SHA.

Observation 1. *The difference between a single set rotation constants and multiple sets of rotation constants vanishes with increased number of steps. In contrast to the HAS-160 message expansion, the SHA-1 message expansion delays this process until step 80.*

Secondly, we evaluate the effect of different single sets of rotation constants for SHA-1. Instead of rotating variable A by 5 positions to the left, we evaluated the attack complexity for all possibilities from 0-31. The results are depicted in Figure 5. In the step-reduced version, we see considerable differences for the chosen rotation constants of A. The constant 5, which was chosen for SHA-1, is in this setting favorable for the attacker. However, with increased number

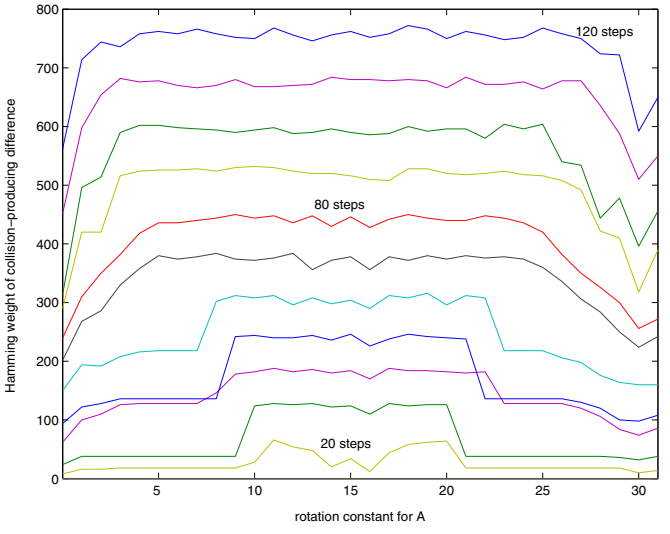


Fig. 5. Hamming weight of collision-producing differences for all possible bit-rotations of A and 20 to 120 steps of SHA-1

of steps, this advantage vanishes. After 80 steps, five bit-rotations are already enough to arrive at the plateau of Hamming weights found.

We apply the same technique for chaining variable B. Instead of rotating variable B by 30 positions to the left, we again evaluated the attack complexity for all possibilities from 0-31. The results are depicted in Figure 6.

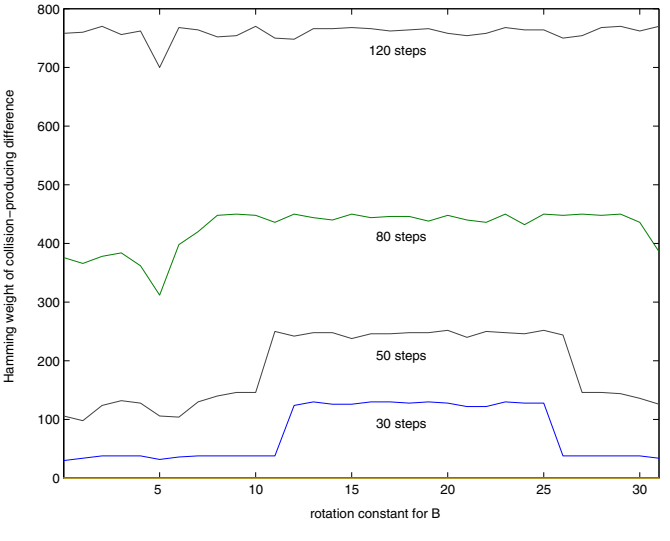


Fig. 6. Hamming weight of collision-producing differences for all possible bit-rotations of B and 30 to 120 steps of SHA-1

In the step-reduced version, we see considerable differences for the chosen rotation constants of B. The constant 30, which was chosen for SHA-1, is in this setting again favorable for the attacker. However, with increased number of steps, this advantage vanishes. After 80 steps, the value 30 (or -2) is already enough to arrive at the plateau of Hamming weights found.

Observation 2. *In the case of full SHA-1 (80 steps),* 5 *is the lowest possible value for rotating A and* 30 *is the highest possible value for rotating B to result in comparatively high Hamming weights for collision-producing differences.*

The advantage of having these constants is as follows: Let's consider platforms where constant-time shifters or rotators (see *e. g.* [14]) are neither implemented in hardware nor as microcode. There, rotating B by *e. g.* 2 positions to the right instead of more is faster. Note that these observations cannot be seen as a design criterium for the SHA family since they do not apply to SHA-0. Refer to Appendix A for details.

5.2 Impact of Multiple Rotation Constants on the Condition Generating Phase

Let us now consider the second point mentioned above: the assumption, that it is less likely to take advantage of some condition-reducing effects due to multiple sets of rotation constants. This refers to the second step of our analysis: deriving conditions on chaining variables and input message words to make the real hash function behave like the linearized model.

Before looking at the effect of multiple sets of rotation constants, the effect of the new non-linear Boolean function introduced in HAS-160 is analyzed: the f_{MAJ} function used in SHA-1 has the nice property (for an attacker) that whatever (non-zero) input difference is applied, it is always possible to find conditions on the inputs which modifies the output difference towards an XOR-like behavior. This ensures that every possible collision-producing message difference in the linearized model can lead to a real collision, assuming a number of conditions is fulfilled.

Table 2. Conditions that need to be fulfilled in order to have a differential behavior identical to that of an XOR

input differences	f_{xor}	f_{if}	f_{maj}	f_{has3}
000	0	*always*	*always*	*always*
001	1	$B = 0$	$B \oplus C = 1$	$B = 0$
010	1	$B = 1$	$B \oplus D = 1$	*always*
011	0	*never*	$C \oplus D = 1$	$B = 0$
100	1	$C \oplus D = 1$	$C \oplus D = 1$	$D = 1$
101	0	$B \oplus C \oplus D = 0$	$B \oplus D = 1$	$B \oplus D = 0$
110	0	$B \oplus C \oplus D = 0$	$B \oplus C = 1$	$D = 1$
111	1	$C \oplus D = 0$	*always*	$B \oplus D = 0$

As illustrated in Table 2, the new Boolean function does not increase the difficulty for an attacker to find conditions. As opposed to f_{if} (input difference 011), we can always find conditions on the inputs of f_{has3} to make it behave like an XOR for all input differences.

The new Boolean function does not put additional hurdles for an attack. Due to multiple sets of rotation constants aligning differences to optimize the carry-drop effect (see Section 3.7) is less effective. At this point, it is difficult to estimate the influence on the attack complexity compared to SHA-1, since the bit-rotation of the SHA-1 message expansion has a similar effect.

6 Conclusion

We have analyzed the effect of multiple sets of rotation constants in HAS-160 and compared them to the single set of rotation constants used in SHA-1. The bottom line is that multiple sets increase the attack complexity, the difference to a single set however vanishes for increased number of steps. In our comparisons, the Hamming weight of collision-producing differences in a linearized model was used as a means to compare attack complexities on a relative scale. We also gave some observations on the design of the compression function of SHA-1. For 80 or more steps of SHA-1, the benefits of having multiple sets of rotation constants instead of a single set are negligible. We finally observe that the chosen values for rotations used in the state update of SHA-1 are *on the edge* as far as the provided security level is concerned. Without impairing security, rotating the chaining variable A by *5* to the left and chaining variable B by *2* to the right are the smallest possible values. Platforms without constant-time shifters benefit from this choice.

References

1. Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *LNCS*, pages 290–305. Springer, 2004.
2. Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and Reduced SHA-1. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 36–57. Springer, 2005.
3. Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462, pages 56–71. Springer, 1998.
4. Antoine Joux, Patrick Carribault, William Jalby, and Christophe Lemuet. Full iterative differential collisions in SHA-0, 2004. Preprint.

5. KCDSA Task Force Team. The Korean Certificate-based Digital Signature Algorithm, 1998. Available at `http://grouper.ieee.org/groups/1363/P1363a/contributions/kcdsa1363.pdf`.
6. Vlastimil Klima. Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications, 2005. Preprint, available at `http://eprint.iacr.org/2005/102`.
7. Jeffrey S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, 1988.
8. Chae Hoon Lim. The revised version of KCDSA, 2000. Unpublished Manuscript, available at `http://dasan.sejong.ac.kr/~chlim/pub/kcdsa1.ps`.
9. Chae Hoon Lim and Pil Joong Lee. A Study on the Proposed Korean Digital Signature Algorithm. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology - ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Beijing, China, October 18-22, 1998, Proceedings*, volume 1514 of *Lecture Notes in Computer Science*, pages 175–186. Springer, 1998.
10. Jack Lloyd. A Description of HAS-160, 2003. Available at `www.randombit.net/papers/has160.html`.
11. National Institute of Standards and Technology (NIST). FIPS-180-2: Secure Hash Standard, August 2002. Available online at `http://www.itl.nist.gov/fipspubs/`.
12. Nan Kyoung Park, Joon Ho Hwang, and Pil Joong Lee: HAS-V: A New Hash Function with Variable Output Length. In Douglas R. Stinson and Stafford E. Tavares, editors, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 2012, pages 202–216. Springer, 2001.
13. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Exploiting Coding Theory for Collision Attacks on SHA-1. In *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings to appear*, LNCS. Springer, 2005.
14. Jan M. Rabaey. *Digital Integrated Circuits*. Prentice Hall, 1996.
15. Vincent Rijmen and Elisabeth Oswald. Update on SHA-1. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *LNCS*, pages 58–71. Springer, 2005.
16. TTA. Digital Signature Mechanism with Appendix - Part 2 : Certificate-based Digital Signature Algorithm, TTAS.KO-12.0011/R1, 2000.
17. TTA. Hash Function Standard - Part 2: Hash Function Algorithm Standard (HAS-160), TTAS.KO-12.0011/R1, 2000.
18. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 1–18. Springer, 2005.
19. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.

20. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.
21. Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *LNCS*, pages 1–16. Springer, 2005.

A Single Sets of Rotation Constants for SHA-0

We evaluate the effect of different single sets of rotation constants for SHA-0. Instead of rotating variable A by 5 positions to the left, we evaluated the attack complexity for all possibilities from 0-31. The results are depicted in Figure 7.

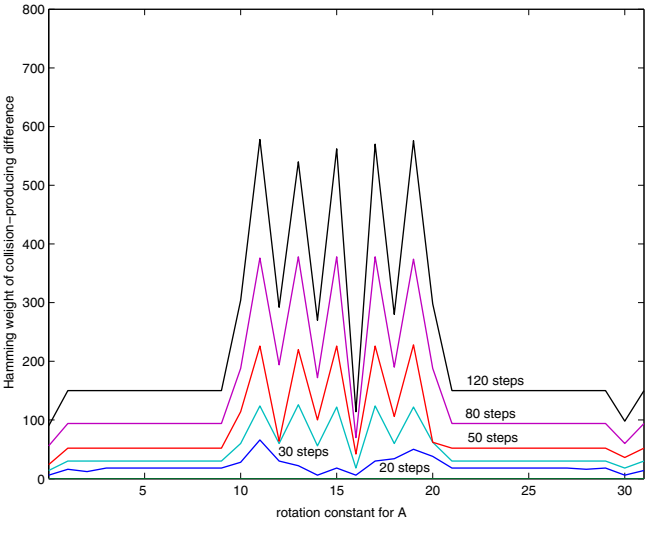


Fig. 7. Hamming weight of collision-producing differences for all possible bit-rotations of A and 20 to 120 steps of SHA-0

Using the Hamming weight for the rotation constant 5 as a starting point, we see that higher as well as lower Hamming weights for collision-producing differences are possible when choosing different rotation constants. This holds for all considered variants from 20 to 120 steps.

A Scalable, Delegatable Pseudonym Protocol Enabling Ownership Transfer of RFID Tags
(Extended Abstract)

David Molnar, Andrea Soppera, and David Wagner

UC Berkeley, British Telecom, and UC Berkeley
dmolnar@eecs.berkeley.edu, andrea.2.soppera@bt.com,
daw@eecs.berkeley.edu

Abstract. The ability to link two different sightings of the same Radio Frequency Identification (RFID) tag enables invasions of privacy. The problem is aggravated when an item, and the tag attached to it, changes hands during the course of its lifetime. After such an *ownership transfer*, the new owner should be able to read the tag but the old owner should not.

We address these issues through an RFID *pseudonym protocol*. Each time it is queried, the RFID tag emits a different pseudonym using a pseudo-random function. Without consent of a special Trusted Center that shares secrets with the tag, it is infeasible to map the pseudonym to the tag's real identity. We present a scheme for RFID pseudonyms that works with legacy, untrusted readers, requires only one message from tag to reader, and is scalable: decoding tag pseudonyms takes work logarithmic in the number of tags. Our scheme further allows for *time-limited delegation*, so that we can give an RFID reader the power to disambiguate a limited number of pseudonyms without further help from the Trusted Center. We show how RFID pseudonyms facilitate the transfer of ownership of RFID tags between mutually distrustful parties.

Our scheme requires only limited cryptographic functionality from the tag: we need a pseudo-random function (PRF) and the ability to update tag state or to generate random numbers. Tag storage and communication requirements are modest: we give example parameters for a deployment of one million tags in which each tag stores only 128 bits, makes 6 PRF evaluations, and sends 158 bits each time it is read.

Keywords: RFID, privacy, pseudonym protocol, cryptography.

1 Introduction

Radio Frequency Identification (RFID) technology holds great promise, but it also raises significant privacy concerns. The term RFID represents a family of emerging technologies that enable object identification without physical or visual contact. The main idea is to give a unique identity to every object by attaching a tag. A tag is a small chip, with an antenna, that stores a unique ID and other information which can be sent to a reading device. The reading device uses a database to link the tag ID with information about the object it is attached to.

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 276–290, 2006.

Today's RFID systems do not authenticate the tag, so it is easy for an attacker to impersonate other tags. This allows an attacker to mislabel goods for illicit gain, e.g. causing an expensive item to be reported as a cheap one at checkout time. Future systems will need to provide a way for readers to authenticate tags and prevent such impersonation attacks.

The other main concern in RFID systems is the privacy of the user. Today, tags can be read remotely and invisibly by any reader. This leads to unwanted consequences, such as the surreptitious tracking of objects and people through time and space. For instance, any party could use the RFID tags to track people's movements without authorization, since the ability to recognize an RFID tag allows for tracking items, and by extension, the people associated with them. A future protocol should prevent unauthorized readers from violating the privacy of users.

Some of the early work in this area has proposed protocols for mutual authentication between the tag and the reader [9, 6]. Mutual authentication protects privacy, because the tag can insist that the reader authenticate itself and prove it is authorized before releasing the tag identity. However, mutual authentication is overkill for many RFID applications, because in most cases we simply want to know the tag's identity, and mutual authentication incurs an unnecessarily high performance overhead. Moreover, these mutual authentication schemes cannot be used with existing standard reader protocols. It would be better to have solutions that are compatible with legacy readers, which can only read and pass along a tag identifier.

We propose a cryptographic scheme that protects privacy while retaining many of the legitimate benefits of current RFID technology. The main idea is to introduce an RFID *pseudonym scheme* and to use a *Trusted Center* (TC) to enforce the desired privacy policy and limit which readers may read each tag. Each time the tag is read, it generates a new pseudonym and sends this pseudonym to the reader. The Trusted Center is able to decode this pseudonym and obtain the tag's identity. Online readers can simply contact the Trusted Center and request that the pseudonym be decoded (if allowed by the privacy policy). In addition, we provide mechanisms so that the decoding can be performed anywhere in the network, enabling us to support legacy readers and offline operations.

Our scheme provides two new features not seen in prior RFID protocols, namely *time-limited delegation* and *ownership transfer*. Delegation enables a reader to decode a particular tag's pseudonyms without any further assistance from the Trusted Center, by transferring the secrets associated with that tag to the reader. Time-limited delegation allows to provide a controlled form of delegation, where the reader receives only the ability to recognize the next q pseudonyms for this tag (where q can be chosen arbitrarily). We can use time-limited delegation to reduce the exposure if an adversary breaks into the reader: instead of losing the secrets for all tags for all time, we lose only what was delegated to that particular reader. Delegation also gives us a way to tolerate poor quality network connections between the reader and Trusted Center, since the reader does not need network access once it has received its delegated secrets.

Finally, we show how to use delegation to help Alice and Bob, who both trust the same Trusted Center but do not trust each other, securely transfer an RFID-tagged item from one to the other. After the transfer, Bob has assurance that Alice can no longer read the RFID tag on the item, even though she could before. Our methods for ownership transfer require minimal or no online interaction by the Trusted Center itself.

We present two versions of our scheme. The first version stores a counter on the tag and provides all of the features discussed so far. For tags that do not support any form of writable non-volatile state, we also design a second version that requires only a random number generator and read-only state. However, this second version does not support time-limited delegation or ownership transfer.

Our scheme seems to be practical. It can leverage the existing infrastructure of readers. The tag need only provide support for symmetric-key cryptography and either a counter or a random number generator. These requirements appear to be reasonable for a large class of RFID applications, including many deployments that have already raised significant privacy concerns.

2 Towards a Secure RFID Tag Protocol

We begin by outlining the features our protocol is designed to provide and some of the key challenges in achieving these goals.

Pseudonyms. Our main goal is to allow authorized readers to identify and authenticate the RFID tag, while preventing unauthorized readers from determining anything about the identity of tags they interact with. One possible approach would be to require readers to authenticate themselves to the tag before they are allowed to read its contents; however, this would require changing the communication protocol between tags and readers, and thus would mean that existing readers would have to be replaced. Therefore, our approach is to build a RFID pseudonym protocol [8]. In our scheme, the RFID tag replies with a unique pseudonym that changes each time it is queried. The pseudonym is generated based on some secret key that is stored on the tag and known to authorized readers, so that authorized readers can identify the tag. However, without that secret, the pseudonym provides no information about the tag's identity. In particular, pseudonyms are unlinkable, so that unauthorized readers will be unable to tell if two pseudonyms came from the same tag. In this way, possession of the secret key controls the ability to link sightings of the same tag.

The tag-reader protocol is very simple: the reader interrogates the tag, and the tag responds with its current pseudonym. Our use of pseudonyms allows the scheme to be compatible with legacy readers, because the reader does not need to know anything about the way that pseudonyms are generated or decoded. Instead, the reader can forward the pseudonym it received to some other entity, and that other entity can recover the tag's identity from the pseudonym.

Privacy. It is important to be able to specify a privacy policy for each tag, restricting which readers are authorized to read that tag. Our architecture in-

cludes a central trusted entity, which we call the Trusted Center (TC), whose role is to manage and enforce these privacy policies. When a tag is enrolled into the system, it is loaded with a secret key generated for it by the TC. The TC keeps a database listing, for each tag, the secret key provided to that tag, the information associated with that tag (such as its identity), and that tag's privacy policy. Given any pseudonym from an enrolled tag, the TC can decode the pseudonym and determine the identity of the tag using the secret keys stored in its database.

Note that we do not require the existence of a single global Trusted Center that is trusted by everyone in the world. Although it would be possible to set up a worldwide key infrastructure with a single globally trusted root (e.g., administered by a consortium of tag manufacturers), this is not necessary. For example, a library deploying RFID could act as its own Trusted Center, enrolling a tag and writing secrets to it when the tag is applied to a library book. If libraries do not need to read each other's tags, then no library need trust any other.

In our system, the Trusted Center acts as a trusted third party that manages the privacy policy associated to tags. We envision that the Trusted Center might provide a way for the owner of each tag to specify a privacy policy for that tag, listing which readers are authorized to decode this tag's pseudonyms. Manufacturers might also specify a default policy when the tag is created, allowing us to support both opt-in and opt-out policies. When the Trusted Center receives a request from some reader to decode a particular pseudonym, the Trusted Center can decode the pseudonym, consult the tag's privacy policy, and decide whether to reveal the tag's identity to this reader.

This provides a simple way for users to delegate access only to specific readers. In the future, a RFID infrastructure might consist of thousands or even millions of RFID readers deployed across the planet, and we need a way for legitimate readers to read the tag. In a naive implementation, the Trusted Center might give a copy of the tag's secret key to each reader that is authorized to read the tag. However, this form of delegation is too coarse-grained, because the reader then permanently receives the ability to identify this tag for all time. We may not wish to place this much trust in every RFID reader that ever encounters the tag, because then compromise of any one reader could endanger the privacy of many users. The challenge is to provide time-limited delegation, where a reader's ability to read a tag can be limited to a particular time period.

Time-limited Delegation. Controlling delegation is easy if all readers are online—the reader can simply act as a dumb relay, passing on the pseudonym from the tag to Trusted Center and letting the TC reply with the identity of the tag (if permitted by this tag's privacy policy). However, this approach requires a costly interaction between the reader and TC every time a tag is read. Because today's readers may repeatedly broadcast queries to all tags within range at a rate of 50 times per second or so, the burden on the TC and the database may be very high: if there are 10 tags within range, we require 500 round-trip interactions per second with the TC, multiplied times the number of readers. We instead focus on the problem of offline delegation.

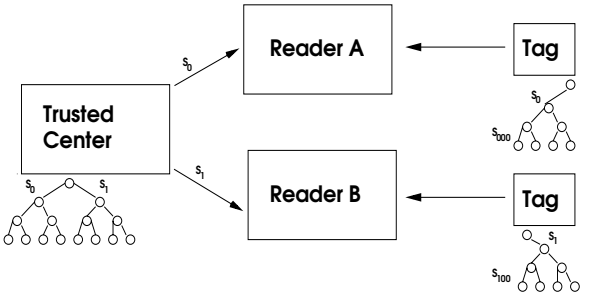


Fig. 1. The Trusted Center delegates access to two different readers

In our scheme, the TC can compute a time-limited secret that provides only the ability to disambiguate pseudonyms for a particular tag for a limited number of times. In particular, the TC computes a secret that allow to recognize the next q pseudonyms from this tag, where q is arbitrary and can be specified by the privacy policy. This secret can be communicated to the reader through any secure channel, and for the next q tag-reads the reader does not need to interact with the TC in any way. After reading the tag q times, the reader loses the ability to link tag readings and must contact the Trusted Center to ask for re-authorization. (See Figure 1.)

Delegation is helpful for cases where readers have intermittent or low-bandwidth connectivity. When a reader first sees a tag it is unable to recognize, the reader can send the pseudonym it received to the TC. If this reader is authorized for this tag, the TC can return not only the tag's identity but also a secret that allows reading the tag for a limited time—say, for 1000 queries—without requiring further interaction with the TC. Delegation provides benefits even for online readers, because the locality in tag sightings can be used to greatly improve performance and reduce communication with the TC.

Our scheme also supports recursive delegation: after we delegate to Alice limited-access to the tag, she can further re-delegate to Bob the power to query this tag, and Bob can further delegate to Carol, and so on. Moreover, the rights delegated can be limited arbitrarily at each step. For instance, if Alice receives a secret that lets her identify the tag for the next 100 queries, she can compute a secret for Bob that will let him read the tag for the next 40 queries, a secret for Bill that lets Bill read the tag for the 30 queries after that, and so on. To the best of our knowledge, no previous protocol for RFID privacy has addressed delegation, let alone provided support for recursive delegation.

Ownership Transfer. A related problem to delegation is that of *ownership transfer*, where Alice gives an RFID-tagged item to Bob. After the transfer of ownership, Bob should be able to read the item but Alice should not. Pseudonyms allow us to cleanly deal with ownership transfer from Alice to Bob. If Alice has not been delegated the ability to disambiguate pseudonyms, no further work is needed: once Bob registers his ownership of this tag, the TC can simply deny any future requests from Alice to read this tag. If Alice has been delegated

Scheme	T_{Reader}	S_{Reader}	T_{TC}	S_{TC}	# Msg	Comm	Delegation?
OSK [8]	$O(N)$	$O(N)$	NA	NA	1	$O(1)$	No
AO [1]	$O(N^{2/3})$	$O(N^{2/3})$	NA	NA	1	$O(1)$	No
MW [6]	$O(\log N)$	$O(1)$	NA	NA	$O(\log N)$	$O(\log N)$	No
Basic	$O(D)$	$O(D)$	$O(\log N)$	$O(2^{d_1})$	1	$O(\log N)$	Yes
Optimized	$O(D)$	$O(D)$	$O(\log N)$	$O(1)$	1	$O(\log N)$	Yes

Fig. 2. Comparison to previous RFID privacy schemes. Here T_{TC} and S_{TC} stand for the time and storage requirements of the Trusted Center, with the Reader requirements marked similarly. N is the total number of tags in the system, d_1 is the depth of the Trusted Center's tree, and D is the number of tags delegated to a particular reader. In practice, we expect $D \ll N$. The Optimized Scheme uses a PRF to generate the TC's tree of secrets and truncates the tag outputs, as described in Section 6.

secrets that let her read this tag, we have two methods for ensuring Alice can no longer link a tag after it is passed to Bob. Both are described in more detail in Section 5.

Scalable Lookup. A major technical challenge in the design of such systems is how to make them scale to a large number of tags. Consider a TC with a database of N tags. Naively, decoding a pseudonym might require a linear scan through all N tag keys, which may not be practical for an RFID system with $N = 10^6$ tags. Instead, we design a scheme with logarithmic complexity: the TC does just $O(\log N)$ work to disambiguate a pseudonym. In Figure 2, we compare our system to previous RFID pseudonym schemes.

Delegation incurs some performance overhead at the readers. In our scheme, a reader that has received D delegations will require $O(D)$ work per tag queried. In practice we expect D will be small compared to the total number of tags; for example, D might be the number of tags in a single shipment of goods. Therefore, we expect this performance level to be adequate in practice.

3 Notation

We use a pseudo-random function (PRF) $F : K \times \{0,1\}^n \to \{0,1\}^n$ and a pseudo-random generator (PRG) $G : K \to K \times K$. Also, we use $G_0(k)$ and $G_1(k)$ to denote the first and second part of $G(k)$, respectively, so that $G(k) = (G_0(k), G_1(k))$.

In practice we might use AES as the PRF. Recent results on low-gate-count implementations of AES suggest that AES may be within reach for all but the lowest-end RFID tags [2]. We might also define the PRG in terms of the PRF, for instance defining G by $G_b(k) = F_k(0^{n-1}\, b)$, so that the tag needs only a single cryptographic primitive. One should be careful to ensure that the inputs to the PRF when used for PRG-emulation are disjoint from the inputs to the PRF elsewhere in the protocol, for instance by having the first bit of the PRF input indicate which mode it is being used in.

If $s \in \{0,1\}^*$ is a bitstring, we use $s_{1..i}$ to denote the first i bits of s, and $\text{len}(s)$ to denote the length of s (in bits). Also, we place the nodes of a complete depth-d binary tree in one-to-one correspondence with $\{0,1\}^{\leq d}$, the set of bitstrings of length at most d. The empty string represents the root of the tree. If s is any internal node, $s\,0$ and $s\,1$ are used to represent its left and right children, respectively. Thus each bitstring of length $\leq d$ identifies a node in the binary tree by specifying the path from the root that reaches it. We sometimes also use s to refer to the integer $s_1 2^{n-1} + \cdots + s_{n-1} 2 + s_n$ obtained by viewing the string s as a number written in big-endian binary notation.

If $f : S' \to T$ is a function and $S \subseteq S'$, let $f|_S : S \to T$ denote the restriction of f to S. When given a function $H : \{0,1\}^{\leq d_1} \to K$ defined on $\{0,1\}^{\leq d_1}$, we will extend it to a function defined on all of $\{0,1\}^*$ using the recurrence $H(s\,b) = G_b(H(s))$.

4 Our Protocol

Tree of Secrets. Our protocol is based around a tree of secrets of depth $d = d_1 + d_2$ as shown in Figure 3. Each node in the tree has its own k-bit secret key. For simplicity, we describe our scheme in terms of a complete binary tree $\{0,1\}^{\leq d}$, though we will later generalize this to larger branching factors.

The first d_1 levels of the tree contain node secrets that are chosen uniformly and independently at random by the Trusted Center during system initialization (see algorithm TC.GenTC in Figure 5). Each node at depth d_1 corresponds to

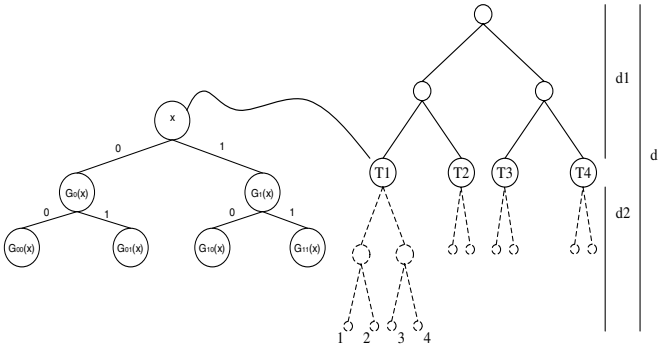


Fig. 3. An example tree of secrets for four tags in our RFID pseudonym scheme. The nodes drawn with solid lines correspond to secrets shared only between the tags T1,...,T4 and the Trusted Center. Each of these secrets is drawn uniformly at random and independently of each other. The dashed line nodes are secrets in *delegation trees*, where keys at child nodes are derived by the GGM construction from the key at their parent. On each read, a tag updates its state to use the next leaf in the delegation tree for its next pseudonym. To delegate limited-time access to a tag, the Trusted Center can give out subtrees of the delegation tree; for example, the immediate parent of 1 and 2 allows learning T1's identity in time periods 1 and 2, but not in time periods 3 and 4.

a unique tag. When the tag is enrolled into the system, it receives all keys on the path from its node to the root. Therefore, each tag only needs capacity to store d_1 secrets.

The next d_2 levels of the tree contain secrets that are derived using a GGM tree construction [3]: each node is labelled with a secret, and the secrets for its children are derived by applying a PRG. Knowing a secret at level $\geq d_1$ allows computation of the secrets for every descendent in the subtree rooted at that node, but nothing else.

Formally, the TC chooses a function $H : \{0,1\}^{\leq d_1} \to K$ uniformly at random, and $H(s)$ denotes the key associated with node s in the tree. We extend the function $H : \{0,1\}^{\leq d_1} \to K$ to a function $H : \{0,1\}^{\leq d} \to K$ by the rule $H(s\,b) = G_b(H(s))$ for all $s \in \{0,1\}^{\geq d_1}$, $b \in \{0,1\}$. For the rest of this paper, we assume this extension is implicitly performed whereever necessary, and we do not distinguish between H and its extended version.

Each tag receives $H|_S$ for some prefix-closed set $S = \{t_{1..1}, \ldots, t_{1..d_1}\}$ corresponding to the path to the root. This means that the tag effectively learns the function $H|_{S'}$, where $S' = \{t_{1..1}, \ldots, t_{1..d}\}$, though it only needs to store the first d_1 secrets in this list.

Tag Responses. Each tag T keeps a counter $T.c$. The counter identifies a leaf at level d of the tree; thus, each counter value corresponds to a new pseudonym for this tag. The tag responds to a query from a reader by generating a random number r and sending a pseudonym

$$(r, p) = (r, (F_{h(c_{1..1})}(r), F_{h(c_{1..2})}(r), ..., F_{h(c_{1..d})}(r)))$$

where the $h(c_{1..i})$ values represent secrets along the path in the tree of secrets from the root to the tag's current leaf $T.c$. The tag then increments the counter c. See Figure 4.

Notice that because the counter c is incremented on each query, the tag will use a different path of secrets, and therefore a different pseudonym, for every query. This is what enables delegation, because we can give the reader a subtree of secrets that will expire after a certain number of tag reads. The tag's workload is quite modest: only $d + d_2$ invocations of the PRF are needed per query. By

Tag State: (initialized by TC.EnrollTag)
c, a counter in $\{0,1\}^d$.
S, a set with $S \subseteq \{0,1\}^{\leq d_1}$.
h, where $h : S \to K$.

Algorithm Tag.Respond():
1. Pick $r \in_R \{0,1\}^k$ uniformly at random.
2. Set $p := (F_{h(c_{1..1})}(r), F_{h(c_{1..2})}(r), \ldots, F_{h(c_{1..d})}(r))$.
3. Set $c := c + 1$.
4. Return (r, p).

Fig. 4. Algorithms and state for the RFID tag

varying the branching factor and depth of the tree, we can trade off between the complexity of Tag.RESPOND and the complexity for the reader. See Section 6.

Decoding Pseudonyms. Given a pseudonym (r, p), it is possible to use the tree structure to efficiently decode this pseudonym and discover the identity of the tag that generated this pseudonym. The main idea is to use a depth-first search to find a path in the tree that matches the response p. We start at the root of the tree of secrets. At each node s, we can check whether the left child $s\,0$ or the right child $s\,1$ matches entry p_i in the response by checking whether $F_{s0}(r) = p_i$ or $F_{s1}(r) = p_i$, respectively. In this way, wrong paths can be quickly pruned. See TC.IDENTIFYTAG in Figure 5.

Given a pseudonym, this procedure lets the TC identify the tag's real identity ID. Based on the identity of the tag, the identity of the reader, and the privacy policy for this tag, the TC can then decide whether to reveal ID to the reader. This provides a mechanism for enforcing a privacy policy regarding which readers are allowed to learn which tag IDs.

Delegation. Our protocol also allows the TC to delegate access to a certain interval of pseudonyms to an offline reader. This can be thought of as allowing the reader to perform the mapping itself from a pseudonym (r, p) to the tag's identity ID, but only if the tag's counter value is in a prescribed interval $[L, R]$ (for some $1 \leq L \leq R \leq 2^d$).

Recall that each leaf of the tree corresponds to a different pseudonym for a tag. To delegate access to leaves in an interval $[L, R]$, the Trusted Center first determines the smallest set $S \subseteq \{0,1\}^{\geq d_1}$ of tree nodes that cover the interval $[L, R]$. We say that S covers $[L, R]$ if for all $x \in [L, R]$, there exists $s \in S$ so that s is a prefix of x. The Trusted Center then sends $H|_S$ to the reader along with the tag's identity. Now, when the reader sees the pseudonym (r, p), the reader no longer needs to communicate with the Trusted Center. Instead, the reader can perform a depth-first search starting at each node in S, since $H|_S$ contains everything the reader needs to know to perform this search. See Figures 5 and 6.

After the tag updates itself past the leaf R, however, the reader can no longer recognize any subsequent pseudonyms from this tag. This is because the counter Tag.c will have updated past the subtree of secrets known to the reader. The reader's access to the tag has effectively expired, and at this point the reader must re-apply to the TC if it wants continued access.

Note that decoding a pseudonym takes the reader $O(D)$ invocations of the PRF (for $D = |S|$), since the reader must check every value in its delegated subset S for a match with the tag's response.

Second Version: Eliminating the Counter. In low- and middle-end RFID technologies, writing permanent state such as a counter on each tag read may be difficult, making our first protocol inapplicable. For example, the EPC Gen II specification requires a random number generator, but EPC tags are read at a distance of several meters and may not have enough power available for writes.

TC State:
$H : \{0,1\}^{\leq d_1} \to K$, a function.

Algorithm TC.GenTC():
1. Let $H : \{0,1\}^{\leq d_1} \to K$ be a random function, i.e., pick $H(s) \in_R K$ uniformly at random for each bitstring s of length at most d_1.

Algorithm TC.EnrollTag(ID):
1. Find the smallest integer $t \in \{0,1\}^{d_1}$ that hasn't been assigned to any other tag. Assign t to this tag.
2. Set $S := \{t_{1..j} : 1 \leq j \leq d_1\}$.
3. Return $(t\, 0^{d_2}, S, H|_S)$ as the state for this tag.

Algorithm TC.Delegate(L, R):
1. Let S denote the minimal subset of $\{0,1\}^{\geq d_1}$ such that for all x with $L \leq x \leq R$, there exists $s \in S$ so that s is a prefix of x.
2. Return $H|_S$.

Algorithm TC.IdentifyTag(r, p):
1. Return DFS($r, p, 1, \epsilon$), where ϵ denotes the empty bitstring.

Algorithm DFS($r, p = (p_1, .., p_d), i, s$):
1. If $i = d + 1$, return $\{s_{1..d_1}\}$.
2. Set $ids := \emptyset$.
3. If $F_{H(s\,0)}(r) = p_i$ then set $ids := ids \cup \text{DFS}(r, p, i + 1, s\,0)$.
4. If $F_{H(s\,1)}(r) = p_i$ then set $ids := ids \cup \text{DFS}(r, p, i + 1, s\,1)$.
5. Return ids.

Fig. 5. Algorithms and state for the Trusted Center

We now design a second version of the protocol that eliminates the need for updateable non-volatile state, assuming the tag can generate random numbers on demand. We replace the counter Tag.c with a d-bit value whose first d_1 bits are fixed at the unique value t (as before) and whose last d_2 bits are chosen uniformly at random for each query. Thus, the tag uses the same subtree to generate pseudonyms, but instead of walking along the leaves from left to right one at a time, it instead picks a random leaf each time it is read. The Trusted Center's algorithms remain unchanged in either case. Unfortunately, the second version of our protocol does not support time-limited delegation or ownership transfer.

Security and Privacy. Our protocol provides replay-only security against impersonation attack and privacy against a radio-only adversary. Informally, this is because each pseudonym emitted by a tag is indistinguishable from other pseudonyms unless the secret keys are known; we give formal definitions and proofs in the full version of the paper [5].

Our protocol provides replay-only security against impersonation attack even if an adversary can compromise tags. This is because each tag has at least

Reader State: (updated by TC.DELEGATE)
$h : S \to K$, for some $S \subseteq \{0,1\}^{\geq d_1}$, with S initialized to $\emptyset$.

Algorithm READER.IDENTIFYTAG(r, p):
1. Set $ids := \emptyset$.
2. For each $s \in S$ such that no prefix of s is in S, do:
3. Set $ids := ids \cup \mathrm{DFS}(r, p, \mathrm{len}(s) + 1, s)$.
4. Return ids.

Fig. 6. Algorithms and state for the reader

one secret not shared with any other tag; to perform a successful non-replayed impersonation, the adversary would need to predict the value of a PRF keyed with such a secret.

Privacy, on the other hand, degrades under tag compromise. This is because tags may share secrets in the tree of secrets. The amount of degradation depends on the branching factor of the tree. At one extreme, a single-level tree with a branching factor of N loses no privacy under tag compromise. At the other extreme, two randomly chosen tags in a binary tree have a chance of $1/2^k$ of sharing k secrets. Each deployment can pick the branching factor that makes the best tradeoff between privacy loss under tag compromise and reader complexity. Even at high branching factors, however, our scheme provides benefits via delegation.

5 Ownership Transfer

Ownership transfer in RFID is the following problem: Alice gives an RFID tag to Bob. How do we prevent Alice from later reading the RFID tag? This problem is crucial for limiting the trust required in readers which may need to read tags at some point in the tag's lifetime.

In the case that Alice has not been delegated access to the RFID tag, ownership transfer in our model is simple. The Trusted Center is notified of the transfer and updates a privacy policy associated with the tag. Afterwards, Alice requests access to the tag's ID. The Trusted Center then checks the privacy policy, sees Alice no longer owns the item, and denies access. In case Alice has been already been delegated access to the tag, we introduce two methods for ownership transfer.

Soft Killing. In the first method, *soft killing*, Bob queries the Trusted Center and learns how many leaves were delegated to Alice. Suppose this number is k. Bob then reads the tag $k+1$ times. The tag will then have updated past Alice's access, so she will no longer be able to disambiguate the tag's pseudonyms. Notice that even if Bob knows how many leaves were delegated to Alice, he still cannot distinguish a tag delegated to Alice from any other tag without Alice's help; this is because the tag will emit a new, pseudorandom, pseudonym on each read. Therefore knowing the number of leaves delegated to Alice does not hurt the privacy of our protocol.

The benefit of soft killing is that it does not require shared secrets between the tag and reader. The downside is that soft killing requires many tag reads. Soft killing also opens up the possibility for a denial of service attack if an adversary reads the tag many times; Alice can recover from this by simply asking the Trusted Center to delegate more access.

Increasing The Tag Counter. In the second method, we allow Bob to increase the counter on a tag from c to c'. Bob does so by sending the tag a random seed r, after which Bob and the tag can perform mutual authentication and establish a secure channel with the shared secret $F_{h(c)}(r)$. Bob then sends c', plus a proof that Bob knows the secret for the leaf c', to the tag over the secure channel. The tag checks that $c' > c$, so Bob can only increase the tag's counter, not decrease it. By doing so, Bob can "leapfrog" the tag over Alice's delegated leaves and be sure that Alice can no longer read the tag. Increasing the counter requires only one read, but also requires the tag to implement a substantially more complex protocol.

6 Optimizations and Weakening Assumptions

Reducing TC Storage. In our protocol as described, the Trusted Center must generate and store 2^{d_1+1} independent random values. We can reduce this storage to a single key by instead having the Trusted Center use a PRF with a master key mk that is never revealed to any other party. The PRF evaluated at a nodeID yields the secret for the node: $H(s) = F_{mk}(s)$ for $s \in \{0,1\}^{\leq d_1}$.

Random Number Generation. In some RFID technologies, it may be difficult to generate random numbers. If the tag can support writable non-volatile state, we can replace the random number generator with a PRF run in counter mode. See Figure 7. We stress that the key rk used for random-number generation is not shared with any reader at any time.

Truncating PRF Values. Instead of sending full PRF values in a tag response, it is more efficient to send truncated versions. This reduces communication overhead at the cost of following false paths during the depth-first search. To avoid misidentification of tags, we recommend truncating only at the internal nodes and sending the full-length PRF output at the leaves. If internal nodes are truncated to a bits, the tag's response becomes (r, p) where $p := (F_{h(c_{1..1})}(r) \bmod 2^a, ..., F_{h(c_{1..d-1})}(r) \bmod 2^a, F_{h(c_{1..d})}(r))$. With full-length values at the leaves, the probability of misidentification is negligible.

PRNG.Initialize()
1. Initialize `ctr` to 0.
2. Pick secret key $rk \in_R K$.

PRNG.GetNextNonce()
1. Return F_{rk}(`ctr++`).

Fig. 7. Generating nonces with a PRF and a counter

Number of Tags	Tag Storage	Communication	Tag Computation	Reader Computation
2^{20}	128 bits	158 bits	6	$6 \cdot 2^{10}$
2^{30}	192 bits	168 bits	7	$7 \cdot 2^{10}$
2^{40}	256 bits	178 bits	8	$8 \cdot 2^{10}$

Fig. 8. Concrete resource use of our scheme for some example parameters. We use a branching factor of 2^{10} in all cases, use a 64-bit r value with truncation, and we assume tags will be read at most 2^{20} times. Tag and reader computation are both measured in expected number of PRF evaluations.

When PRF responses are truncated, identifying a tag requires searching through the tree, and this search might follow false paths that do not correspond to the true tag identity. If the branching factor is exactly 2^a, it is possible to show that the search process is a birth-death process and that the expected complexity of the search is $O(2^a \times \lg N) = O(2^a \times d)$.

Branching Factor and Concrete Examples. Truncation greatly reduces communication overhead while only slightly impacting the complexity of tag identification. For instance, with a binary tree of depth $d = 40$, we might truncate PRF values to 1 bit at internal nodes and use a 64-bit PRF output at the leaves. With these parameters, the response p will be 103 bits long, while the search complexity remains minimal.

In practice, we would use trees with branching factors much larger than 2. A larger branching factor reduces the depth of the tree, thus reducing tag storage and computation, at the cost of more computation for the Trusted Center and reader. For example, consider an RFID system with $N = 2^{20}$ tags, each of which will be read at most 2^{20} times. We construct a four-layer tree of secrets with branching factor $1024 = 2^{10}$ at all levels. Each tag stores two 64-bit secrets s_1, s_2, with the second secret being the root of a GGM tree that covers the final two tree levels. Each pseudonym requires two PRF invocations to compute s_3, s_4 and four PRF invocations to compute the response. Total tag storage is $2 \cdot 64 = 128$ bits and total tag computation is 6 applications of the PRF. If we truncate the tag's responses to 10 bits at internal nodes and 64 bits at the leaf, and use a 64-bit r, the tag's total communication is $64 + 30 + 64 = 158$ bits. The work for the reader, on the other hand, is only $6 \cdot 2^{10}$ applications of the PRF. We show concrete parameters for this and some other examples in Figure 8.

7 Related Work

Weis et al. provide "hash lock" protocols for private mutual authentication [9]. As we have discussed, mutual authentication is not needed in scenarios when only tag identification is required, and it incurs significant performance costs. Their schemes also require readers to perform work linear in the number of total tags and do not support time-limited delegation to offline readers. Because they choose independent secrets for each tag, however, they do not suffer from privacy loss under tag compromise.

Molnar et al. show how to use a tree of secrets to achieve mutual authentication protocol with complexity logarithmic in the number of tags [6]. Their scheme requires at least 3 rounds of communication between tag and reader, while we use only one message from tag to reader. More importantly, their work does not support delegation, nor does it work with legacy readers. Our work uses a similar tree construction to achieve logarithmic work, but applies the idea to RFID pseudonyms. Our recursive tree-walking scheme has some similarities with the traitor tracing scheme of Naor et al. [7].

Ohkubo et al. introduce a scheme for RFID pseudonyms [8]. In their protocol, recovering the tag identity requires work linear in the number of possible tags, while we achieve logarithmic work. They propose storing the expected next output of each RFID tag as an optimization, but this cannot be kept up to date unless without online reader-TC interaction on every tag read. Avoine and Oechslin propose a time-space tradeoff technique that improves the complexity of the Ohkubo et al. protocol to $O(N^{2/3})$ time with a table of size $O(N^{2/3})$, but their protocol does not support delegation as ours does [1]. Both protocols could be extended to support a form of delegation by giving out the individual secrets for each time period, but this requires much more state on the reader. On the other hand, both schemes avoid the problem of privacy loss under tag compromise, because all tags have independently chosen secrets.

The time-memory tradeoff of Avoine and Oeschslin also requires picking a hash chain length in advance. Once a tag exceeds the number of readings set by this length, the entire table must be recomputed before that tag can be read. This is a problem because readers may read a tag 50 times per second. Further, there are active attacks that violate our privacy goals. An adversary that has the ability to trigger a legitimate reader to query a tag and learn whether the reader accepts or rejects can determine if the tag has been read more times than provided for by the hash chain length; this would enable the adversary to distinguish tags. While these concerns are alleviated by longer chain lengths, these lengths also increase the amount of space and time required for the time-memory tradeoff.

Juels gives a scheme for one-use RFID pseudonyms [4]. Unlike our protocol, Juels's scheme does not require a PRF; a simple XOR is enough. Juels also discusses methods for rotating and changing pseudonyms to support ownership transfer. His protocol, however, only allows a tag to emit a limited number of pseudonyms before it must be refreshed through interaction with a trusted reader. Juels outlines an extension to the protocol which removes this restriction using a PRG, but this method requires tight synchronization between reader and tag. Further, his protocol does not work with legacy readers, and it does not support delegation as ours does. Again, in Juels's system, compromising one tag does not aid the adversary in identifying another.

8 Conclusions

We have described a new cryptographic protocol for RFID privacy. In our scheme, tags generate pseudonyms that can only be decoded with knowledge of the appropriate secrets, and privacy is protected by controlling which parties

are given access to these secrets. The key ingredient of our protocol is a set of secrets organized in a tree format. This tree structure enables many powerful features, including support for legacy readers, disconnected operation, flexible privacy policies, delegation to authorized readers, time-limited delegation, recursive delegation, and ownership transfer between users. At the same time, our scheme is practical and scalable: it requires only a few PRF invocations and a modest amount of communication between the tag and reader, even for very large deployments. We believe our protocol could enhance the privacy protection for a wide range of current and future deployments of RFID technology.

Acknowledgements

We thank Gildas Avoine, Michael Backes, Trevor Burbridge, Etienne Dysli, Arnaud Jacquet, Pascal Junod, Vivekanand Korgaonkar, Philippe Oechslin, and the anonymous reviewers for helpful comments. The authors also gratefully acknowledge support from NSF grant CCR-0093337, the Sloan Research Fellowship, and British Telecom. David Molnar was supported by an Intel OCR Fellowship and an NSF Graduate Fellowship.

References

1. Gildas Avoine and Philippe Oechslin. A scalable and provably secure hash-based RFID protocol. In *IEEE PerSec*, 2005.
2. Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. In *CHES*, 2004.
3. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.
4. Ari Juels. Minimalist cryptography for RFID tags, 2003. `http://www.rsasecurity.com/rsalabs/staff/bios/ajuels/publications/minimalist/index.html`.
5. David Molnar, Andrea Soppera, and David Wagner. A scalable, delegatable pseudonym protocol enabling ownership transfer of RFID tags (full version). To appear in Cryptology ePrint Archive, 2005. `eprint.iacr.org/2005/`.
6. David Molnar and David Wagner. Security and privacy in library RFID: Issues, practices, and architectures. In *ACM CCS*, 2004.
7. D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In *CRYPTO*, 2001.
8. Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. Cryptographic approach to a privacy friendly tag. In *RFID Privacy Workshop, MIT*, 2003.
9. Stephen A. Weis, Sanjay E. Sarma, Ronald L. Rivest, and Daniel W. Engels. Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems. In *Security in Pervasive Computing*, 2004.

Reducing Time Complexity in RFID Systems

Gildas Avoine[1], Etienne Dysli[1], and Philippe Oechslin[1,2]

[1] EPFL, Lausanne, Switzerland
[2] Objectif Sécurité, Gland, Switzerland

Abstract. Radio frequency identification systems based on low-cost computing devices is the new plaything that every company would like to adopt. Its goal can be either to improve the productivity or to strengthen the security. Specific identification protocols based on symmetric challenge-response have been developed in order to assure the privacy of the device bearers. Although these protocols fit the devices' constraints, they always suffer from a large time complexity. Existing protocols require $O(n)$ cryptographic operations to identify one device among n.

Molnar and Wagner suggested a method to reduce this complexity to $O(\log n)$. We show that their technique could degrade the privacy if the attacker has the possibility to tamper with at least one device. Because low-cost devices are not tamper-resistant, such an attack could be feasible. We give a detailed analysis of their protocol and evaluate the threat. Next, we extend an approach based on time-memory trade-offs whose goal is to improve Ohkubo, Suzuki, and Kinoshita's protocol. We show that in practice this approach reaches the same performances as Molnar and Wagner's method, without degrading privacy.

Keywords: RFID, time complexity, time-memory trade-off.

1 Introduction

Sometimes presented by the media as the next technological revolution after the Internet, Radio Frequency Identification (RFID) aims to identify objects remotely, with neither physical nor visual contact. They consist of transponders inserted into objects, readers which communicate with the transponders using a radio channel and a database which contains information on the objects.

This technology is not fundamentally new and concerns a whole range of applications. The first RFID application may have been the Royal British Air Force's "Identify Friend or Foe" system, which was used during the Second World War to identify friendly aircrafts. RFID systems have also been used for a few years in commercial applications, for example in contactless smart cards used on public transport. However, the boom that RFID technology enjoys today is chiefly due to the standardization [12, 6] and development of low-cost devices, so-called *tags*. This new generation of RFID tags has opened the door to hitherto unexplored applications. For example in supply chains as suggested by the EPC Global Inc. [6], to locate people in amusement parks [20], to combat the counterfeiting of expensive items [14], to trace livestock [5], to label books in libraries [16], etc.

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 291–306, 2006.

However, these tags also bring with them security and privacy issues. Security issues rely on classic attacks, e.g., denial of service, impersonation of tags or channel eavesdropping. These attacks are rendered more practicable because of the tags' lack of computational and storage capacity. More details on the technical aspects of the tags can be found for example in [14, 7, 8]. Current research deals with these problems but most of them are inherent to the technology itself, and applications have to make do with them. For these reasons, the RFID technology is more suited for bringing functionality (e.g., [6, 20, 5, 16]) rather than security (e.g., [14, 4]).

Nevertheless, whether it has a security or a functionality goal, radio frequency identification raises issues linked to privacy, in particular the problem of traceability of objects and thus indirectly of people [2]. Other technologies also permit the tracking of people, e.g., video surveillance, GSM, Bluetooth, and are extensively used by law enforcement agencies among others. However, RFID tags would permit everybody to track people using only low-cost equipment. This is strengthened by the fact that tags cannot be switched off, they can be easily hidden, their lifespan is not limited, and analyzing the collected data can be efficiently automated. Whether the defenders of RFID minimize the problem of privacy and its detractors amplify it, the fact that some companies have had to renounce this technology after being boycotted by associations [21] which defend individuals' liberty shows that we need to address this problem. Several palliative ways have been explored in order to solve this problem. For example, Juels, Rivest, and Szydlo proposed the "blocker tag" [15] whose goal is to prevent the readers from identifying the tags. With a very different approach, Garfinkel stated the "RFID Bill of Rights" which relates the fundamental rights of the tags' bearers. Today's challenge is to find protocols which allow authorized parties to identify the tags without an adversary being able to track them, thus getting to the root of the privacy problem.

The reason that we cannot use well-known authentication protocols comes from the fact that such protocols do not preserve the privacy of the prover. In other words, the verifier can check whether or not the identity claimed by the prover is true, but he cannot guess it himself: the prover must send his identity in clear which in turn allows an adversary to track him.

Asymmetric cryptography could easily solve this problem: the prover encrypts his identity with the public key of the verifier. Thus, no eavesdropper is able to identify the prover. Unfortunately, asymmetric cryptography is too heavy to be implemented within a tag. Certain classes of tags are simply not able to use cryptography, e.g., Class 0 and Class 1 tags (according to the EPC [6] classification). Several protocols suited to these tags have been proposed (see for example [13,22,11,9] or [1] for a more exhaustive list) but even if they can reduce the capabilities of the attacker – which make them an attractive option – none of them can assure strong privacy. Therefore, we will not consider these tags below; instead, in this paper, we will focus on tags which are capable of embedding a symmetric cryptographic function, either a hash function or a secret-key cipher,

as the suited implementation of AES suggested by Feldhofer, Dominikus, and Wolkerstorfer [7].

The problem remains that both prover and verifier need to share a common key if a secret-key cipher is used instead of a public-key one. In RFID systems, provers (tags) are not tamper-resistant. Therefore an attacker who tampers with a tag can track its past events if she had access to its previous interactions with readers, e.g., from readers' log files. One could point out that the ease of tampering with a tag is counter-balanced by the difficulty of getting access to it. That is the case with sub-dermal tags for example, or bracelet tags in amusements parks which could be re-initialized when the customer gives back his bracelet [20]. Nevertheless, using a common key for all the tags would be a pity: an attacker who tampers with one tag, e.g., her own tag, would also be able to track all the other tags in the system.

Consequently, another approach consists of using a unique key for each tag such that only the verifier (system) knows all these keys. However, this approach, which is the one taken by several reference papers [16, 19, 23], suffers from an expensive time complexity on the system's side. Indeed, because only symmetric cryptographic functions can be used, the system needs to explore its entire database in order to retrieve the identity of the tag it queries. If n is the number of tags managed by the system, $O(n)$ cryptographic operations are required in order to identify one tag. The advantage of the system over an attacker is that the system knows in which subset of identifiers it needs to search while the attacker has to explore the full range of identifiers.

We will address this problem in the rest of the paper. First of all, we will introduce a real life example in a library. We will use this example to compare the protocols which will be considered in this work. Section 3 will be devoted to the basic secret-key challenge-response protocol, denoted CR and the technique suggested by Molnar and Wagner [16], denoted CR/MW, whose goal is to reduce the complexity of CR. We will prove that this technique degrades the privacy when an attacker is able to tamper with at least one tag. In Section 5, we will deal with the protocol of Ohkubo, Suzuki, and Kinoshita [18], denoted OSK. Relying on a previous abstract [3], we will show in Section 6 how a time-memory trade-off can significantly reduce the identification time of OSK. This variant, called OSK/AO, is as efficient as CR/MW but does not degrade privacy. We will finally summarize our results in Section 7.

2 A Practical Example in a Library

In order to illustrate our comparison between CR, CR/MW, OSK, and OSK/AO, we consider a real life scenario in a library, where tags are used to identify books. Several libraries already use this technology, for example the libraries of Santa Clara (USA), Heiloo (Netherlands), Richmond Hill (Canada), and K.U. Leuven (Belgium). In a library scenario, it is realistic to assume that the tags can contain a secret-key cipher or a hash function because they are not disposable. Thus, a slightly higher cost is conceivable.

In the next sections, we assume that the system relies on a single computer which takes $\theta = 2^{-23}$ seconds to carry out a cryptographic operation, either hashing or encrypting a 128-bit blocks. The library manages 2^{20} tags. As described by Avoine and Oechslin in [2] and also by Molnar and Wagner in [16], we assume that tag singulation and collision avoidance are private and performed at a lower layer. Identification of several tags is therefore sequential. Current implementations allow a single reader to read several hundreds of tags per second, meaning that the system should spend at the most a few milliseconds to identify one tag. In the following sections, t_{P} will denote the average time to identify one tag using a protocol P. Because certain applications (in libraries, in amusement parks, etc.) may use numerous readers, the system should not become a bottleneck in terms of computation. Thus, the system should be capable of identifying the whole set of tags it manages in a few seconds only (e.g., for real-time inventories).

3 Description of Molnar and Wagner's Protocol

Several challenge-response protocols suited to RFID have been suggested during the last years, e.g., [16, 19, 7, 23]. We describe below those suggested by Molnar and Wagner, based on a pseudo-random function.

3.1 Challenge-Response Building Block

The Molnar and Wagner's challenge-response building block (CR), depicted on Figure 1, provides mutual authentication of the reader and the tag in a private way. It shall prevent an attacker from impersonating, tracing or identifying tags.

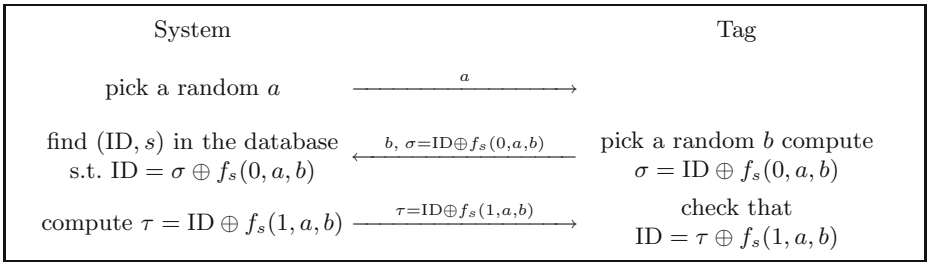


Fig. 1. Challenge-response protocol of Molnar and Wagner

Let ID be the tag's identifier which is stored in both the database of the system and the tag. They also share a secret key s. To initiate the authentication, the reader sends a nonce a to the tag. Next, the tag picks a random b and answers $\sigma := \mathrm{ID} \oplus f_s(0,a,b)$, where f_s is a pseudo-random function. The system retrieves the identity of the tag by finding the pair (ID, s) in its database such that $\mathrm{ID} = \sigma \oplus f_s(0,a,b)$. This completes the authentication of the tag. Now, in order to achieve mutual authentication, the system sends back $\tau := \mathrm{ID} \oplus f_s(1,a,b)$ to the tag. The tag can thus verify the identity of the reader by checking that $\mathrm{ID} = \tau \oplus f_s(1,a,b)$.

3.2 Efficiency

In order to identify a tag, the system must carry out an exhaustive search on the n secrets stored in its database. Therefore the system's workload is linear in the number of tags. More precisely, the average number of cryptographic operations required to identify one tag is $n/2$ and therefore we have $t_{\mathsf{CR}} = \frac{n\theta}{2}$. With the parameters given in Section 2, we have $t_{\mathsf{CR}} \approx 62$ ms which is too high in practice. Since CR does not scale well in a system with many tags, we next examine the three-based technique of Molnar and Wagner [16], whose main strength is the reduction of the system's workload from $O(n)$ to $O(\log n)$.

3.3 Tree-Based Technique

The technique suggested by Molnar and Wagner [16], namely CR/MW, relies on a tree structure in order to reduce the identification complexity. Instead of searching a flat space of secrets, let's arrange them in a balanced tree with branching factor δ. The tags are the leaves of this tree and each edge is associated with a value. Each tag has to store the values along the path from the root of the tree to itself. This sequence makes up its *secret*, and each value is called a *block* of secret. On the other side, the reader knows all secrets. We describe the protocol below.

Setup. Let n be the number of tags managed by the system and $\ell := \lceil \log_\delta n \rceil$ be the depth of the tree with a branching factor δ. Each edge in the tree is valuated with a randomly chosen secret $r_{i,j}$ where i is the level in the tree and j is the number of the branch. Figure 2 represents such a tree with parameters $n = 9$ and $\delta = 3$. The secret of a given tag is the list of the values $r_{i,j}$ from the root to the leaf. For example, the secret of T_5 on Figure 2 is $[r_{1,1}, r_{2,5}]$.

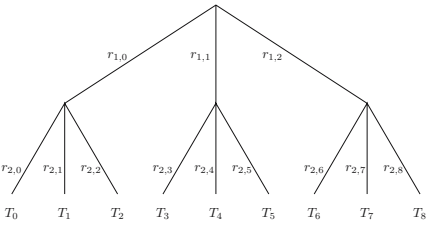


Fig. 2. Tree of tags' secrets

Interrogation. The tag is queried level by level from the root to the leaves. At each level i, CR/MW runs CR for each secret of the explored subtree. That is the reader tries every edge in turn in order to know on which one the tag is. If CR fails for all current level's secrets, the tag rejects the reader and the protocol stops. If the reader has been successfully authenticated at each level the protocol succeeds. Note that CR inevitably does not need to be performed δ times per level in practice. One run is enough if the reader checks the tag's answer with all current level's secrets, as described below.

Identification. At each level i, the system has to search in a set of δ secrets for the one matching the tag's secret. Given that $[s_1, \ldots, s_\ell]$ denotes a secret, the system has thus to compute $\delta/2$ times $f_{s_i}(0, a, b)$ on average at level i, meaning that $\frac{\delta}{2}\ell$ operations are required in order to identify one tag. Thus we have

$$t_{\mathsf{CR/MW}} = \frac{\delta\theta}{2} \log_\delta n.$$

The identification of one tag is far below the threshold of a few milliseconds. Identifying the whole system takes more than 2 minutes when $\delta = 2^{10}$ and decreases to 2 seconds when $\delta = 2$. However, we will see in Section 4 that having a small branching factor enables to trace the tags.

4 Privacy-Weakening Attacks

4.1 Tampering with Only One Tag

We examine in this section how the tree technique suggested by Molnar and Wagner allows tracing a tag when the attacker is able to tamper with some tag. The attack consists of three phases:

1. The attacker has one tag T_0 (e.g., her own) she can tamper with and thus obtain its complete secret. For the sake of calculation simplicity, we assume that T_0 is put back into circulation. When the number of tags in the system is large, this does not significantly affect the results.
2. She then chooses a target tag T. She can query it as much as she wants but she cannot tamper with it.
3. Given two tags T_1 and T_2 such that $T \in \{T_1, T_2\}$, we say that the attacker succeeds if she definitely knows which of T_1 and T_2 is T. We define the probability to trace T as being the probability that the attacker succeeds. To do that, the attacker can query T_1 and T_2 as many times as she wants but, obviously, cannot tamper with them.

We assume that the underlying challenge-response protocol assures privacy when all the blocks of secrets are chosen according to a uniform distribution. We consequently assume that the attacker cannot carry out an exhaustive search over the secret space. Hence, the only way for an attacker to guess a block of secret of a given tag is to query it with the blocks of secret she obtained by tampering with some tag. When she tampers with only one tag, she obtains only one block of secret per level in the tree. Thus, she queries T, and then T_1, and T_2 with this block. If either T_1 or T_2 (but not both) has the same block as T_0, she is able to determine which of them is T. If neither T_1 nor T_2 has the same block as T_0, she cannot answer. Finally, if both T_1 and T_2 have the same block as T_0, she cannot answer, but she can move on the next level of the tree because the authentication of the reader succeeded. We formalize the analysis below. We denote the secrets of T, T_0, T_1, and T_2 by $[s_1, \cdots, s_\ell]$, $[s_1^0, \cdots, s_\ell^0]$, $[s_1^1, \cdots, s_\ell^1]$, and $[s_1^2, \cdots, s_\ell^2]$ respectively. We consider a given level i where s_i^1 and s_i^2 are in the same subtree. Four cases must be considered:

- $C_i^1 = ((s_i^0 = s_i^1) \wedge (s_i^0 \neq s_i^2))$ then the attack succeeds,
- $C_i^2 = ((s_i^0 \neq s_i^1) \wedge (s_i^0 = s_i^2))$ then the attack succeeds,
- $C_i^3 = ((s_i^0 \neq s_i^1) \wedge (s_i^0 \neq s_i^2))$ then the attacks definitively fails,
- $C_i^4 = (s_i^0 = s_i^1 = s_i^2)$ then the attacks fails at level i but can move onto level $i+1$.

When the number of tags in the system is large, we can assume that

$$\Pr\left(C_i^1\right) = \Pr\left((s_i^0 = s_i^1)\right) \times \Pr\left((s_i^0 \neq s_i^2)\right).$$

The same assumption also applies to C_i^2, C_i^3, and C_i^4. Thus we have

$$\Pr\left(C_i^1 \vee C_i^2\right) = \frac{2(\delta-1)}{\delta^2} \quad (1 \leq i \leq \ell) \quad \text{and} \quad \Pr\left(C_i^4\right) = \frac{1}{\delta^2}.$$

The overall probability P that the whole attack succeeds is therefore

$$\begin{aligned} P &= \Pr\left(C_1^1 \vee C_1^2\right) + \sum_{i=2}^{\ell} \left(\Pr\left(C_i^1 \vee C_i^2\right) \times \prod_{j=1}^{i-1} \Pr\left(C_j^4\right) \right) \\ &= \frac{2(\delta-1)}{\delta^2} + \sum_{i=2}^{\ell} \left(\frac{2(\delta-1)}{\delta^2} \left(\frac{1}{\delta^2}\right)^{i-1} \right) = 2(\delta-1)\frac{1-\left(\frac{1}{\delta^2}\right)^{\ell}}{1-\frac{1}{\delta^2}}\frac{1}{\delta^2}. \end{aligned}$$

Remembering that $\delta^\ell = n$ yields $P = \dfrac{2}{\delta+1}\left(1 - \dfrac{1}{n^2}\right)$. The curve of P when $n = 2^{20}$ is the curve plotted on Figure 3 with $k_0 = 1$.

4.2 Tampering with Several Tags

We now consider the case where the attacker can tamper with more tags, e.g., she borrows several books in the library in order to tamper with their tags. We examine the influence of the number of opened tags on the probability of tracing the target tag. As before each opened tag is put back into circulation to simplify calculations. When n is large, this does not affect the results. As in the previous section, we denote the secrets of T, T_1, and T_2 by $[s_1, \cdots, s_\ell]$, $[s_1^1, \cdots, s_\ell^1]$, and $[s_1^2, \cdots, s_\ell^2]$ respectively. We consider a given level i where s_i^1 and s_i^2 are in the same (one-level) subtree. Let $\mathcal{K}_i$ denote the set of blocks of this (one-level) subtree which are known by the attacker and let $\mathcal{U}_i$ denote the set of those which are unknown by the attacker. k_i denotes the number of blocks in $\mathcal{K}_i$. Five cases must be considered:

- $C_i^1 = ((s_i^1 \in \mathcal{K}_i) \wedge (s_i^2 \in \mathcal{U}_i))$ then the attack succeeds,
- $C_i^2 = ((s_i^1 \in \mathcal{U}_i) \wedge (s_i^2 \in \mathcal{K}_i))$ then the attack succeeds,
- $C_i^3 = ((s_i^1 \in \mathcal{K}_i) \wedge (s_i^2 \in \mathcal{K}_i) \wedge (s_i^1 \neq s_i^2))$ then the attack succeeds,
- $C_i^4 = ((s_i^1 \in \mathcal{U}_i) \wedge (s_i^2 \in \mathcal{U}_i))$ then the attacks definitively fails,
- $C_i^5 = ((s_i^1 \in \mathcal{K}_i) \wedge (s_i^2 \in \mathcal{K}_i) \wedge (s_i^1 = s_i^2))$ then the attacks at level i fails but can move onto level $i+1$.

Thus, we have for all i such that $1 \leq i \leq \ell$:

$$\Pr(C_i^1 \vee C_i^2 \vee C_i^3) = \frac{2k_i}{\delta}\left(1 - \frac{k_i}{\delta}\right) + \left(\frac{k_i}{\delta}\right)^2 \left(1 - \frac{1}{k_i}\right)$$
$$= \frac{k_i}{\delta^2}(2\delta - k_i - 1),$$

$$\text{and} \qquad \Pr(C_i^5) = \frac{k_i}{\delta^2}.$$

The overall probability P that the attack succeeds is therefore

$$P = \Pr(C_1^1 \vee C_1^2 \vee C_1^3) + \sum_{i=2}^{\ell} \left(\Pr\left(C_i^1 \vee C_i^2 \vee C_i^3\right) \times \prod_{j=1}^{i-1} \Pr\left(C_j^5\right) \right)$$
$$= \frac{k_1}{\delta^2}\left(2\delta - k_1 - 1\right) + \sum_{i=2}^{\ell} \left(\frac{k_i}{\delta^2}(2\delta - k_i - 1) \prod_{j=1}^{i-1} \frac{k_j}{\delta^2} \right).$$

We now compute k_1, i.e., the number of different blocks known by the attacker at level 1, given that k_0 is the number of tags tampered with by the attacker. We have

$$k_1 = \delta\left(1 - (1 - \frac{1}{\delta})^{k_0}\right)$$

$$\text{and then} \qquad k_i = \delta\left(1 - (1 - \frac{1}{\delta})^{g(k_i)}\right) \quad (2 \leq i \leq \ell),$$

$$\text{where} \qquad g(k_i) = k_0 \prod_{j=1}^{i-1} \frac{1}{k_j}.$$

Results are plotted on Figure 3. We would like to highlight the surprising behavior of P when the branching factor is small. This is due to the fact that neither $\Pr(C_i^1 \vee C_i^2 \vee C_i^3)$ nor $\Pr(C_i^5)$ are monotonous and they reach their optimum at different values of δ. Table 1 supplies a few values in order to illustrate our attack.

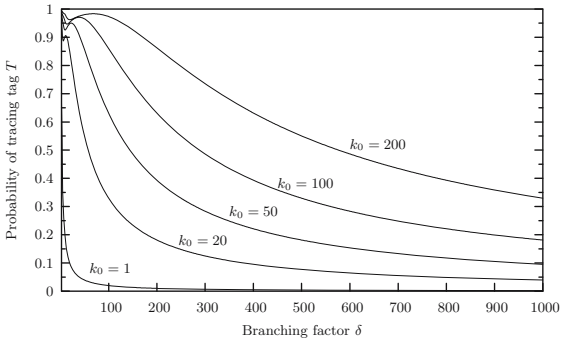


Fig. 3. Probability of tracing a tag when the attacker tampered with k_0 tags

Table 1. Probability that the attack succeeds according to the branching factor δ, given that k_0 tags have been opened and the system contains 2^{20} tags

k_0 \ δ	2	20	100	500	1000
1	66.6%	9.5%	1.9%	0.3%	0.1%
20	95.5%	83.9%	32.9%	7.6%	3.9%
50	98.2%	94.9%	63.0%	18.1%	9.5%
100	99.1%	95.4%	85.0%	32.9%	18.1%
200	99.5%	96.2%	97.3%	55.0%	32.9%

4.3 Notes on the Original Tree-Based Technique

In the original tree-based scheme proposed by Molnar and Wagner in [16], the blocks of secret of the tags were not chosen according to a uniform distribution. Instead, subtrees of a given level have the same set of blocks of secrets. This seems to be due to a typo in the setup algorithm of [16]. The attack is obviously more efficient on this original scheme because, the k_is are larger (for a same value of k_0).

5 Ohkubo, Suzuki, and Kinoshita's Protocol

5.1 Description

The protocol proposed by Ohkubo, Suzuki, and Kinoshita [18] relies on hash chains. When a tag is requested by a reader, it sends a hash of its current identifier and then renews it using a second hash function. Obviously, only the system is able to link all the values sent by the tag while an attacker cannot. More precisely, the scheme works as follows.

Setup. The personalization of a tag T_i consists of storing in its memory a random identifier s_i^1, which is also recorded in the system's database. Thus, the database initially contains the set of random values $\{s_i^1 \mid 1 \leq i \leq n\}$. Two hash functions G and H are chosen. One hash function is enough if a one-bit parameter is added to the function.

Interrogation. When the system queries T_i, it sends an identification request to the tag and receives back $r_i^k := G(s_i^k)$ where s_i^k is the current identifier of T_i. While T_i is powered, it replaces s_i^k by $s_i^{k+1} := H(s_i^k)$. The exchanges between the system and the tag are depicted on Figure 4.

Identification. From r_i^k, the system has to identify the corresponding tag. In order to do this, it constructs the hash chains from each n initial value s_i^1 until it finds the expected r_i^k or until it reaches a given maximum limit m on the chain length. The lifetime of the tag is *a priori* limited to m identifications. However, when a tag is scanned by a reader (in the library), its field in the database can

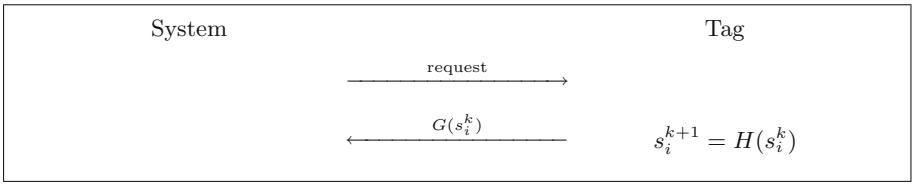


Fig. 4. Protocol of Ohkubo, Suzuki, and Kinoshita

be refreshed. The threshold m is therefore the number of read operations on a single tag between two updates of the database. A suited size for m could be 128, meaning that a tag can be queried 128 times before updating its entry in the database.

5.2 Replay Attack Avoidance and Reader Authentication

Like CR, OSK assures privacy because the information sent by the tag is indistinguishable from a random value, in the random oracle model. The main advantage of OSK compared with CR is that it also assures forward privacy, meaning that if an attacker can tamper with a tag, she is not able to track its past events. However, OSK does not prevent replay attacks. Common techniques to avoid replay attacks are usually incremental sequence number, clock synchronization, or a fresh challenge sent by the verifier. This latter option is the most suited to RFID tags. We propose therefore to modify OSK as depicted on Figure 5. Note that OSK does not provide authentication of the reader. However, this feature can be obtained if the system sends a third message containing $G(s_i^{k+1} \oplus w)$ where w is a fixed and public non-zero binary string.

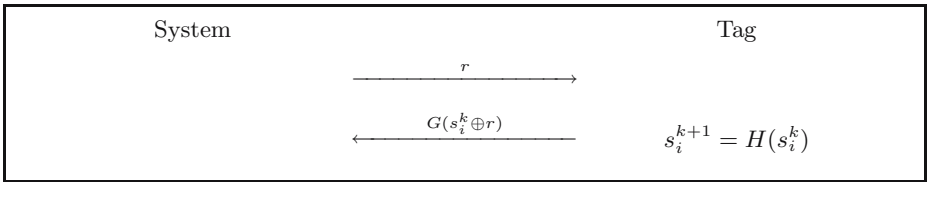


Fig. 5. Modified protocol of Ohkubo, Suzuki, and Kinoshita

5.3 Efficiency of OSK

Outside the library, tags can be queried by foreign readers. This avoids maintaining synchronization between the tag and the system. Therefore the complexity in terms of hash operations in order to identify one tag is $t_{\mathsf{OSK}} = mn\theta$ on average (2 hash operations are carried out $mn/2$ times). With the parameters given in Section 2 and chains of length 128, we have $t_{\mathsf{OSK}} \approx 16$ seconds. Note that if we had considered that readers of the library may read foreign tags (hold by people in the library), then the complexity would tend towards to $2mn$ because the

system would have to explore the whole database to determine whether or not a tag is owned by the system. Note that even if tags and readers were able to stay synchronized, the complexity of OSK cannot be better than CR if no additional memory is used.

6 Using a Time-Memory Trade-Off to Improve OSK

We recall and detail in this section our previous work [3] on OSK. Thus we will be able to compare CR/MW and OSK/AO.

6.1 Time-Memory Trade-Off

To reduce the complexity of OSK, we propose to improve how the data is managed by the system, without modifying the exchanges between tags and readers; so, the privacy of OSK remains. For that, we suggest to use a specific time-memory trade-off based on Hellman's original work [10] and Oechslin's optimizations [17]. This type of trade-off reduces the amount of work T needed to invert any given value in a set of N outputs of a one-way function F with help of M units of memory. The efficiency follows the rule $T = N^2\gamma/M^2$ where γ is a small factor depending on the probability of success and the particular type of trade-off being used. Compared to a brute-force attack, the trade-off can typically reduce the amount of work from N to $N^{2/3}$ using $N^{2/3}$ units of memory.

The basic idea of time-memory trade-off techniques consists in chaining (almost) all the possible outputs of F using a *reduction function* R which generates an arbitrary input of F from one of its outputs. By alternating F and R on a chosen initial value, a chain of inputs and outputs of F can be built. If enough chains of a given length are generated, most outputs of F will appear at least once in any chain. The trade-off comes from the fact that only the first and the last element of each chain is stored. Thus, a substantial memory space is saved, but computations will be required on-the-fly to invert a given element. Given one output r of F that should be inverted, a chain starting at r is generated. If r was part of any stored chain, a last element of a chain in the table will eventually be reached. Looking up the corresponding start of the chain, we can regenerate the complete chain and find the input of F that yields the given output r. To assure a high success rate, several tables have to be generated with different reduction functions. The exact way of doing this is what differentiates existing trade-off schemes.

In what follows, we will use the *perfect rainbow* tables [17] which have been shown to have better performances than other types of tables. The characteristic of the rainbow tables is that each column of a table has a different reduction function. So, when two chains collide, they do not merge (except if they collide at the same position in the chain). When the residual merged chains are removed during the precomputation step, the tables are said to be perfect. With such tables and a probability of success of 99.9%, we have $\gamma = 8$.

6.2 Adapting the Trade-Off to Our Case

The time-memory trade-off technique described above cannot be directly applied to our case. Indeed, the input of F must cover all the identifiers but no more.

Otherwise, the system would have no advantage over the attacker. Consequently, it is important to choose F such that its input space is as small as possible. We define the function F as follows:

$$F : (i, k) \mapsto r_i^k = G(H^{k-1}(s_i^1))$$

where $1 \leq i \leq n$ and $1 \leq k \leq m$. Thus, given the number of the tag and the number of the identification, F outputs the value which will be sent by the tag. We also need to define an arbitrary reduction function R such that

$$R : r_i^k \mapsto (i', k')$$

where $1 \leq i' \leq n, 1 \leq k' \leq m$. For example, we take

$$R(r) = (1 + (r \bmod n), 1 + (\left\lfloor \frac{r}{n} \right\rfloor \bmod m)).$$

There are still two important points that distinguish classical time-memory trade-offs from ours.

Firstly, the brute force method of OSK needs $n|s|$ units of memory to store the n values s_i^1 while usual brute-force methods do not require any memory. Thus, it makes sense to measure the amount of memory needed by the trade-off in multiples of $n|s|$. We call c the ratio between the memory used by the trade-off and the memory used by the brute-force. The memory used to store the tables is a multiple of the size of a chain while it is a multiple of s in the case of the brute-force. A stored chain is represented by its start and end points which can be either the output of F or its input. In our case the input is smaller, we therefore choose to store two pairs of (i, k), thus requiring $2(|n| + |m|)$ bits of memory. The conversion factor from units of brute-force to units of trade-off is $\mu = |s|/(2|n| + 2|m|)$. In the scenarios we are interested in, μ is typically between 2 and 4.

Secondly, when used in the trade-off, F is more complex than when used in the brute-force. Indeed, in the brute-force, the hash chains are calculated sequentially, thus needing just one H and one G calculation at each step. In the trade-off, i and k are arbitrary results from R and have no incremental relation with previous calculations. Thus, on average, each step computes $(m-1)/2+1$ times the function F and G once. We can now rewrite the trade-off relation:

$$T = \frac{N^2}{M^2}\gamma = \frac{n^2m^2}{(c-1)^2\mu^2n^2}(\frac{m-1}{2} + 1)\gamma \approx \frac{m^3\gamma}{2(c-1)^2\mu^2}.$$

We now show how this issue can be mitigated. So far, among the c shares of memory, $(c-1)$ shares are used to store the chains, and 1 share is used to store the n values s_i^1. If we not only store the first element of the chains, but also the element at the middle of the chain, we sacrifice even more memory but we reduce the average complexity of F. We will have only $(c-2)$ shares of the memory available for the tables, but F will have a complexity of $\frac{m-1}{4} + 1$ (we need to generate only a quarter of a chain on average). We have therefore a

trade-off between the memory sacrificed to store the intermediary points and the complexity of F. In general, if we store x values per chain, sacrificing x shares of memory, the complexity of the trade-off becomes:

$$T = \frac{n^2m^2}{(c-x)^2\mu^2n^2}(\frac{m}{2x}+1)\gamma \approx \frac{m^3\gamma}{2x(c-x)^2\mu^2}.$$

The optimal complexity is achieved when $x = \frac{c}{3}$. So we have

$$T \approx \frac{3^3}{2^3}\frac{m^3\gamma}{c^3\mu^2}.$$

Since a pair of (i,k) is 27 bits large (20 bits for i and 7 bits for k) we need at most 54 bits to store one chain. We can thus store more than two chains in the same amount of memory it takes to store one s ($\mu \geq 2$). Assuming that all calculations are carried out on a single back-end equipped with $\frac{c(n|s|)}{8} = 2^{24}c$ bytes of memory and that we choose a success rate of 99.9% ($\gamma = 8$) the time to read a tag with our method is

$$t_{\mathsf{OSK/AO}} \approx \frac{6^9\theta}{c^3} \text{ seconds.}$$

For example, with 1 GB of RAM (i.e., c=64), we have $t_{\mathsf{OSK/AO}} \approx 0.004$ milliseconds. Precomputation takes $nm^2\theta/2$ seconds, that is to say about 17 minutes. The technique used for that can be found in [3].

Note that the time-memory trade-off cannot be applied directly to the modified OSK suggested in Section 5.2. This is due to the randomization of the tag's answer. In order to apply our time-memory technique on the modified version of OSK, the tag must answer with both $G(s_i^k)$ and $G(s_i^k \oplus r)$. The former value enables the reader to identify the tag and the latter one allows detecting replay attacks.

7 Final Comparison and Conclusion

First of all, we consider the storage aspect. On the tag side, the storage of the identifiers becomes a real problem with CR/MW when δ is small. Having a large δ is therefore preferable. Storage is the main drawback of OSK/AO because precomputation and storage of tables is required. In the example given in Section 6, 1 GB of RAM is used. Today, such a memory is available on Joe Blow's computer.

Next, we address the complexity question. Both CR/MW and OSK/AO are parameterizable. CR/MW depends on δ which can be chosen between 2 and n. Obviously, the case $\delta = n$ leads to CR. Having $\delta > \sqrt{n}$ is possible but in this case the tree is no longer complete. Actually, a typical value could be $\delta = \sqrt{n}$. On the other hand, OSK/AO depends on the available memory. Table 2 gives a numerical comparison of CR, CR/MW, OSK, and OSK/AO.

We now consider the privacy issue. While CR is secure, CR/MW degrades the privacy because, when an attacker is able to tamper with at least one tag (e.g., her own tag), she is also able to trace other tags in a probabilistic way. We

Table 2. Time to identify one tag

Scheme (parameter)	Time (millisecond)
CR	62.500
CR/MW ($\delta = 2^{10}$)	0.122
CR/MW ($\delta = 2$)	0.002
OSK	16'000.000
OSK/AO (342 MB)	0.122
OSK/AO (1.25 GB)	0.002

have shown that the probability to trace tags decreases when the computation complexity grows. Thus, CR/MW can be seen as a trade-off between privacy and complexity. We proved that the probability to trace tags is far from being negligible. For example, when the branching factor is $\delta = 2^{10}$, the probability to trace a tag is about 0.1% when only one tag has been opened, but it is about 32.9% when 200 tags have been tampered with (see Table 1). OSK/AO inherits from the security proofs of OSK, in particular the fact that OSK is forward private, because it modifies neither the information exchanged, nor the content of the tag. It only improves the way the system manages and stores the data.

Thus, we can say that the main advantage of CR/MW rests on the fact that it does not require precomputation. Moreover the number of tag readings with OSK/AO is limited by the chain length while it is not with CR/MW (however overpassing this threshold does not threaten the privacy). Hence, when CR/MW is used, we recommend using a large branching factor in order to limit the privacy threat.

Finally, one may think that trade-off techniques could be used to improve CR/MW. Unfortunately, this seems difficult and cannot be done using the same approach because the answers of the tags in CR/MW are randomized. This implies carrying out a time-memory trade-off on a larger space.

Acknowledgment

The authors would like to thank Matthieu Finiasz and Serge Vaudenay for their helpful comments on this work, as well as David Molnar and Andrea Soppera. Gildas Avoine is supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

References

1. Gildas Avoine. Security and privacy in RFID systems. Online bibliography available at `http://lasecwww.epfl.ch/~gavoine/rfid/`.
2. Gildas Avoine and Philippe Oechslin. RFID traceability: A multilayer problem. In Andrew Patrick and Moti Yung, editors, *Financial Cryptography – FC'05*, volume 3570 of *Lecture Notes in Computer Science*, pages 125–140, Roseau, The Commonwealth Of Dominica, February – March 2005. IFCA, Springer-Verlag.

3. Gildas Avoine and Philippe Oechslin. A scalable and provably secure hash based RFID protocol. In *International Workshop on Pervasive Computing and Communication Security – PerSec 2005*, pages 110–114, Kauai Island, Hawaii, USA, March 2005. IEEE, IEEE Computer Society Press.
4. Steve Bono, Matthew Green, Adam Stubblefield, Ari Juels, Avi Rubin, and Michael Szydlo. Security analysis of a cryptographically-enabled RFID device. In *14th USENIX Security Symposium*, pages 1–16, Baltimore, Maryland, USA, July-August 2005. USENIX.
5. Susy d'Hont (Editor). International news letter of the TI RFID group. Electronic Newsletter, Issue 20, November 2000.
6. Electronic Product Code Global Inc. `http://www.epcglobalinc.org`.
7. Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. In Marc Joye and Jean-Jacques Quisquater, editors, *Workshop on Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 357–370, Boston, Massachusetts, USA, August 2004. IACR, Springer-Verlag.
8. Klaus Finkenzeller. *RFID Handbook*. Wiley, England, second edition, 2003.
9. Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. Universal reencryption for mixnets. In Tatsuaki Okamoto, editor, *The Cryptographers' Track at the RSA Conference – CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 163–178, San Francisco, California, USA, February 2004. Springer-Verlag.
10. Martin Hellman. A cryptanalytic time-memory trade off. *IEEE Transactions on Information Theory*, IT-26(4):401–406, July 1980.
11. Dirk Henrici and Paul Müller. Tackling security and privacy issues in radio frequency identification devices. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive Computing*, volume 3001 of *Lecture Notes in Computer Science*, pages 219–224, Vienna, Austria, April 2004. Springer-Verlag.
12. International Organization for Standardization. `http://www.iso.org`.
13. Ari Juels. Minimalist cryptography for low-cost RFID tags. In Carlo Blundo and Stelvio Cimato, editors, *The Fourth International Conference on Security in Communication Networks – SCN 2004*, volume 3352 of *Lecture Notes in Computer Science*, pages 149–164, Amalfi, Italia, September 2004. Springer-Verlag.
14. Ari Juels and Ravikanth Pappu. Squealing euros: Privacy protection in RFID-enabled banknotes. In Rebecca Wright, editor, *Financial Cryptography – FC'03*, volume 2742 of *Lecture Notes in Computer Science*, pages 103–121, Le Gosier, Guadeloupe, French West Indies, January 2003. IFCA, Springer-Verlag.
15. Ari Juels, Ronald Rivest, and Michael Szydlo. The blocker tag: Selective blocking of RFID tags for consumer privacy. In Vijay Atluri, editor, *Conference on Computer and Communications Security – CCS'03*, pages 103–111, Washington, DC, USA, October 2003. ACM, ACM Press.
16. David Molnar and David Wagner. Privacy and security in library RFID: Issues, practices, and architectures. In Birgit Pfitzmann and Peng Liu, editors, *Conference on Computer and Communications Security – CCS'04*, pages 210–219, Washington, DC, USA, October 2004. ACM, ACM Press.
17. Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO'03*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630, Santa Barbara, California, USA, August 2003. IACR, Springer-Verlag.

18. Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. Cryptographic approach to "privacy-friendly" tags. In *RFID Privacy Workshop*, MIT, Massachusetts, USA, November 2003.
19. Keunwoo Rhee, Jin Kwak, Seungjoo Kim, and Dongho Won. Challenge-response based RFID authentication protocol for distributed database environment. In Dieter Hutter and Markus Ullmann, editors, *International Conference on Security in Pervasive Computing – SPC 2005*, volume 3450 of *Lecture Notes in Computer Science*, pages 70–84, Boppard, Germany, April 2005. Springer-Verlag.
20. SafeTzone. `http://www.safetzone.com`.
21. Stop RFID. `http://www.stoprfid.org`.
22. István Vajda and Levente Buttyán. Lightweight authentication protocols for low-cost RFID tags. In *Second Workshop on Security in Ubiquitous Computing – Ubicomp 2003*, Seattle, Washington, USA, October 2003.
23. Stephen Weis, Sanjay Sarma, Ronald Rivest, and Daniel Engels. Security and privacy aspects of low-cost radio frequency identification systems. In Dieter Hutter, Günter Müller, Werner Stephan, and Markus Ullmann, editors, *International Conference on Security in Pervasive Computing – SPC 2003*, volume 2802 of *Lecture Notes in Computer Science*, pages 454–469, Boppard, Germany, March 2003. Springer-Verlag.

Accelerated Verification of ECDSA Signatures

Adrian Antipa[1], Daniel Brown[1], Robert Gallant[1], Rob Lambert[1], René Struik[1], and Scott Vanstone[2]

[1] Certicom Research, Canada
{aantipa, dbrown, rgallant, rlambert, rstruik}@certicom.com
[2] Dept. of Combinatorics and Optimization, University of Waterloo, Canada
savansto@uwaterloo.ca

Abstract. Verification of ECDSA signatures is considerably slower than generation of ECDSA signatures. This paper describes a method that can be used to accelerate verification of ECDSA signatures by more than 40% with virtually no added implementation complexity. The method can also be used to accelerate verification for other ElGamal-like signature algorithms, including DSA.

1 Introduction

The elliptic curve digital signature algorithm (ECDSA) [1, 3, 7] is a widely standardized variant of the original ElGamal signature scheme. As is the case with most ElGamal signature schemes, ECDSA has the property that signature verification is about twice as slow as signature generation (and many times slower if the signer is afforded the luxury of precomputation). The opposite is true in the RSA signature scheme with small encryption exponent e, where signature verification is many times faster than generation. Thus speeding up ECDSA signature verification is a problem of considerable practical importance.

This paper describes a new method that can be used to accelerate signature verification for ECDSA and related signature schemes. The method is quite simple, is independent of the techniques employed for field arithmetic and elliptic curve arithmetic, and requires very little additional resources, such as memory or code space. The advantages of the new method are most apparent for ECDSA*, a slightly modified version of ECDSA. However, the advantages can also be enjoyed if the signer appends a small number of bits ("side information") to standardized ECDSA signatures. We emphasize that, other than this extra information, the new method does not require any changes to the standardized specifications of ECDSA. Thus, the extended signatures are still conformant with the existing ECDSA standards.

In the most favorable setting, if one uses an elliptic curve of prime order over a prime field, only 1 or 2 bits of side information are needed to accelerate signature verification by about 40%.

The remainder of the paper is organized as follows. In §2 we describe ECDSA* and show that its security is equivalent to that of ECDSA. Some relevant mathematical background is collected in §3. The new verification methods for ECDSA* and ECDSA are presented and analyzed in §4. Summary conclusions appear in §5.

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 307–318, 2006.

2 Modified DSA and ECDSA

We next present a modification of DSA and ECDSA in the general setting of a cyclic group.

1. *System-wide parameters.* Let $\mathbb{G}$ be a (cyclic) group of prime order n with generator G and identity element $\mathcal{O}$. Let $f : \mathbb{G} \to \mathbb{Z}_n$ be a suitable conversion function. Let $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_n$ be a collision-resistant hash function.
2. *Initial set-up.* Each communicating party A selects a random integer $d \in [1, n-1]$ and publishes its public key $Q = dG$. The parameter d is kept private to A.
3. *Signature generation.*
   Input: Message $m \in \{0,1\}^*$, private key d.
   Output: Signature (R, s).
   ACTIONS:
   (a) Select a random integer $k \in [1, n-1]$. Compute $R := kG$ and $r := f(R)$.
   (b) Compute $s \in \mathbb{Z}_n$ from the equation $\mathcal{H}(m) \equiv sk - dr$ (**mod** n).
   (c) If $r = 0$ or $s = 0$, go to Step 3a; otherwise, return (R, s).
4. *Signature verification.*
   Input: Signature (R, s), message $m \in \{0,1\}^*$, public key $Q \in \mathbb{G}$ associated with entity A.
   Output: Acceptance or rejection of signature as originating from A.
   ACTIONS:
   (a) Compute $r := f(R)$.
   (b) Verify that r and s are integers in the interval $[1, n-1]$. If any verification fails, return `'reject signature'`.
   (c) Verify that $R = s^{-1}(eG + rQ)$, where $e := \mathcal{H}(m)$. If verification fails, return `'reject signature'`; otherwise, return `'accept signature'`.

Note 1. The function f is usually defined over a superset of $\mathbb{G}$. If so, one can usually evaluate $f(R)$ without explicitly checking that $R \in \mathbb{G}$ first, since if the verification equation holds and if $s \in \mathbb{Z}_n^*$, then $R \in \mathbb{G}$ (since $G, Q \in \mathbb{G}$).

Observe that if the signature verification equation holds, then $s^{-1}(eG+rQ) = R$ and, in particular, $f(s^{-1}(eG + rQ)) = f(R) = r$. The *original scheme* is the one with signature (r, s) and verification equation $f(s^{-1}(eG + rQ)) = r$, rather than the corresponding quantities (R, s) and $s^{-1}(eG + rQ) = R$ in the modified scheme.

It is easy to see that, in either scheme, a signature σ obtained from the signature generation algorithm with input (m, d) is always accepted by the corresponding signature verification algorithm with input (σ, m, Q), where $Q = dG$.

The following result shows that the original and the modified schemes are equally secure.

Theorem 2. *Consider the original and the modified signature schemes. The following statements are equivalent:*

1. (r, s) *is a valid signature with respect to* (m, Q) *in the original signature scheme;*
2. (R, s) *is a valid signature with respect to* (m, Q) *in the modified signature scheme for precisely one point* R *in the set* $\varphi(r) := \{X \in \mathbb{G} \mid f(X) = r\}$.

Moreover, one can efficiently convert a valid signature in either scheme to a valid one in the corresponding scheme. Thus, the original and the modified schemes are equally secure.

Proof. Let $R := s^{-1}(eG + rQ)$ for some $s \in \mathbb{Z}_n^*$. One has $R \in \mathbb{G}$, since $G, Q \in \mathbb{G}$. It follows that $R \in \varphi(r)$ if and only if $f(R) = r$. The result now follows by comparing the conditions under which either scheme accepts signatures. □

We will exploit the relationship between a signature scheme and its modified scheme in the remainder of the paper. In particular, we are interested in specific instantiations that correspond to DSA and ECDSA.

DSA and DSA.* The DSA*scheme is an instantiation of the modified signature scheme with the following system-wide parameters: $\mathbb{G} \subseteq \mathbb{Z}_p^*$ is a subgroup of prime order n of the multiplicative group $\mathbb{Z}_p^*$, where p is a suitably chosen prime number, and $|\mathbb{Z}_p^*| = p - 1 = nh$. The conversion function $f : \mathbb{Z}_p \to \mathbb{Z}_n$ is defined by $f(x) := ((x \textbf{ mod } p) \textbf{ mod } n)$. We define the hash function $\mathcal{H}$ by $\mathcal{H}(x) :=$ SHA-1(x) $(\textbf{mod } n)$. For these parameters, the original scheme corresponds to the *Digital Signature Algorithm* (DSA) as standardized in [3].

ECDSA and ECDSA.* The ECDSA*scheme is an instantiation of the modified signature scheme with the following system-wide parameters: $\mathbb{G} \subseteq E(\mathbb{F}_q)$ is a subgroup of prime order n of some suitably chosen elliptic curve group $E(\mathbb{F}_q)$, $\mathbb{F}_q$ is a finite field, and $|E(\mathbb{F}_q)| = nh$. (We assume h is small, so $n \approx q$.) The conversion function $f : E(\mathbb{F}_q) \to \mathbb{Z}_n$ is defined by $f(x, y) := \overline{x}$ $(\textbf{mod } n)$, where (x, y) is an elliptic curve point in affine representation and where $\overline{x}$ is the standard[1] integer representation of the field element $x \in \mathbb{F}_q$. We define the hash function $\mathcal{H}$ by $\mathcal{H}(x) :=$ SHA-1(x) $(\textbf{mod } n)$. For these parameters, the original scheme corresponds to the *Elliptic Curve Digital Signature Algorithm* (ECDSA) as standardized in [1, 3].

3 Mathematical Preliminaries

In this section, we introduce some well-known results from number theory that facilitate the exposition in the rest of the paper.

Definition 3. *(Simultaneous Diophantine Approximation Problem)*
Let $\xi_1, \ldots, \xi_\ell \in \mathbb{R}$, *let* $\varepsilon > 0$, *and let* B *be a positive integer. Find integers* $p_1, \ldots, p_\ell, q$ *with* $q \in [1, B]$ *such that*

[1] see, e.g., [1].

$$\left|\xi_i - \frac{p_i}{q}\right| \leq \frac{\varepsilon}{q} \quad \textit{for all } i, 1 \leq i \leq \ell. \tag{1}$$

This problem has a solution with $\varepsilon \geq B^{-1/\ell}$, as was shown by Dirichlet [2] based on an argument involving the pigeon-hole principle.

Theorem 4. *[2], [6, Theorem 200] Let $\xi_1, \ldots, \xi_\ell \in \mathbb{R}$ and let B be a positive integer. Then the system of inequalities*

$$\left|\xi_i - \frac{p_i}{q}\right| < \frac{1}{qB} \quad \textit{for all } i, 1 \leq i \leq \ell$$

has an integer solution with $p_1, \ldots, p_\ell, q \in \mathbb{Z}$ and $q \in [1, B^\ell]$.

We are mainly interested in the following corollary, which states that integers in $\mathbb{Z}_n$ can be written as rational numbers involving integers that are significantly smaller (in absolute value) than n.

Corollary 5. *Let $\alpha_1, \ldots, \alpha_\ell \in \mathbb{Z}$ and let B be a positive integer. Then the system of congruence relationships*

$$v(\alpha_1, \ldots, \alpha_\ell) \equiv (u_1, \ldots, u_\ell) \ (\textbf{mod } n)$$

has an integer solution with $|u_1|, \ldots, |u_\ell| < n/B$ and $v \in [1, B^\ell]$.

Proof. We apply Theorem 4 with $\xi_i := \alpha_i/n$. Let $p_1, \ldots, p_\ell, q$ be integers with $q \in [1, B^\ell]$, such that

$$\left|\frac{\alpha_i}{n} - \frac{p_i}{q}\right| < \frac{1}{qB} \ \text{ for all } i, 1 \leq i \leq \ell.$$

Consequently, one has

$$|q\alpha_i - p_i n| < n/B \ \text{ for all } i, 1 \leq i \leq \ell.$$

Now, define $u_i := q\alpha_i - p_i n$ and $v := q$. Since $v\alpha_i \equiv u_i \ (\textbf{mod } n)$, the result follows. □

We give some examples that turn out to be useful later on. All examples assume that n is a prime number and that $B^\ell < n$, so as to ensure that v is invertible modulo n. We will ignore rounding issues involving the parameter B, since these obscure the exposition and are irrelevant in our applications (which use very large integers n).

Example 6. ($\ell = 1$) Let n be a prime number. Any integer $x \in \mathbb{Z}_n$ can be written as $x \equiv u/v \ (\textbf{mod } n)$, where u and v are integers such that $|u| < n^{1-\varepsilon}$ and $1 \leq v \leq n^\varepsilon$ (take $B := n^\varepsilon$ with $0 < \varepsilon < 1$). In particular, one may have $|u| < \sqrt{n}$ and $1 \leq v \leq \sqrt{n}$ (take $\varepsilon := \frac{1}{2}$) or $|u| < n^{2/3}$ and $1 \leq v \leq \sqrt[3]{n}$ (take $\varepsilon := 1/3$).

This example can be generalized as follows.

Example 7. ($\ell \geq 1$) Let n be a prime number. Any integers $x_1, \ldots, x_\ell \in \mathbb{Z}_n$ can be written as $x_i \equiv u_i/v \pmod{n}$ $(1 \leq i \leq \ell)$, where $u_1, \ldots, u_\ell$, and v are integers such that $|u_1|, \ldots, |u_\ell| < n^{1-\varepsilon}$ and $1 \leq v \leq n^{\ell\varepsilon}$ (take $B := n^\varepsilon$ with $0 < \varepsilon < 1/\ell$). In particular, one may have $|u_1|, \ldots, |u_\ell| < n^{\ell/(\ell+1)}$ and $1 \leq v \leq n^{\ell/(\ell+1)}$ (take $\varepsilon := 1/(\ell+1)$) or $|u_1|, \ldots, |u_\ell| < n^{2\ell/(2\ell+1)}$ and $1 \leq v \leq n^{\ell/(2\ell+1)}$ (take $\varepsilon := 1/(2\ell+1)$).

It is well-known that the problem of finding good solutions to the simultaneous Diophantine approximation problem is equivalent to the problem of finding 'short' vectors in a particular lattice. We make this statement more precise and (for completeness) provide a short proof.

Theorem 8. *[8] Let $\xi_1, \ldots, \xi_\ell \in \mathbb{R}$, let $0 < \varepsilon < 1$, and let B be a positive integer. Consider the lattice $\mathcal{L}(A) = \{\mathbf{x}A \mid \mathbf{x} \in \mathbb{Z}^{\ell+1}\}$ that is generated by the rows of the matrix A defined by*

$$A = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \\ \xi_1 & \xi_2 & \cdots & \xi_\ell & \varepsilon/B \end{pmatrix}$$

The lattice $\mathcal{L}(A)$ has a nonzero vector $\mathbf{y}$ with (small) norm $||\mathbf{y}||_\infty \leq \varepsilon$ if and only if the simultaneous Diophantine approximation problem defined by Equation (1) has an integer solution $p_1, \ldots, p_\ell, q \in \mathbb{Z}$ with $q \in [1, B]$.

Proof. Let $p_1, \ldots, p_\ell, q$ be integers, with $q \in [1, B]$, such that the system of inequalities defined by Equation (1) is satisfied. Then $\mathbf{y} = \mathbf{x}A$, with $\mathbf{x} := (-p_1, \ldots, -p_\ell, q)$, has norm $||\mathbf{y}||_\infty \leq \varepsilon$, since $\mathbf{y} = (q\xi_1 - p_1, \ldots, q\xi_\ell - p_\ell, q\varepsilon/B)$. Conversely, if $\mathbf{y} = (q\xi_1 - p_1, \ldots, q\xi_\ell - p_\ell, q\varepsilon/B)$ has norm $||\mathbf{y}||_\infty \leq \varepsilon$, then $|q| \in [1, B]$ and $p_1, \ldots, p_\ell, q$ (or their negatives, if $q < 0$) are a solution to the system of inequalities defined by Equation (1). □

Thus, one can use any algorithm that finds 'short' vectors in lattices to find good solutions to the simultaneous diophantine approximation problem.

Although the L^3 lattice basis reduction algorithm is commonly used to find short vectors, our ultimate goal is to improve the time to verify a signature, and the L^3 algorithm may be too cumbersome to meet this goal. Later, we wish to write an element $x \in \mathbb{Z}_n$ as $x \equiv u/v \pmod{n}$, where u and v are integers such that $|u| < \sqrt{n}$ and $|v| < \sqrt{n}$, as in Example 6. We outline how to use the Euclidean algorithm directly to find u and v, and refer the reader to [4] for more details.

When the Extended Euclidean Algorithm is used to find the greatest common divisor of n and x (which is 1, since n is prime), the algorithm produces a sequence of equations

$$s_i n + t_i x = r_i \text{ for } i = 0, 1, 2, \ldots, \tag{2}$$

where $s_0 = 1$, $t_0 = 0$, $r_0 = n$, $s_1 = 0$, $t_1 = 1$, $r_1 = x$, and $r_i \geq 0$ for all i. At each step of the algorithm, the value of $|r_i|$ decreases, and the extended Euclidean algorithm stops when this value becomes 1. At that iteration i of the algorithm when r_i first becomes less than $\sqrt{n}$, it can be shown that very likely the value of $|t_i|$ is also close to $\sqrt{n}$. Choosing $u = r_i$ and $v = t_i$, we see from (2) that $x \equiv u/v$ (**mod** n), and so we have found appropriate values of u and v.

The procedure above produces integers u and v with $|u|, |v| \sim \sqrt{n}$ such that $x \equiv u/v$ (**mod** n). It is easy to see that the same procedure can be used to write $x \equiv u/v$ (**mod** n) with $|u| \sim n^{2/3}$ and $|v| \sim n^{1/3}$ or, more generally, with $|u| \sim n^{1-\varepsilon}$ and $|v| \sim n^{\varepsilon}$ (where $0 < \varepsilon < 1$). Thus, all representations of an integer $x \in \mathbb{Z}_n$ described in Example 6 can be realized using the extended Euclidean algorithm. This procedure can be generalized to produce simultaneous representations of integers $x_1, \ldots, x_\ell \in \mathbb{Z}_n$ in this format, as described in Example 7, using lattice basis reduction techniques for $(\ell+1)$-dimensional lattices. This can be done quite efficiently for $\ell = 1, 2, 3$ using, e.g., the techniques described in [9].

4 Fast Verification of ElGamal-Like Signatures

In this section, we describe a method for considerably speeding up the verification of signatures in ElGamal-like signature schemes, as described in §2. We then exploit the relationship between these signature schemes and their modified schemes to explore speed-ups of signature verifications in the latter. Although our method applies quite generally, we are mainly interested in the concrete speed-ups of signature verifications in ECDSA*(§4.1) and ECDSA (§4.2).

For the modified signature schemes, checking a signature (R, s) over some message m relative to public key Q involves checking that $s^{-1}(eG + rQ) = R$, where $e := \mathcal{H}(m)$. Here, the point G is a system-wide parameter and can be considered fixed, while the points Q and R cannot be considered fixed, since these vary according to signer and message. The main idea of the paper is that one may replace this equation by any (nonzero) scalar multiple hereof and evaluate the transformed equation instead. Thus, one may replace the original signature verification equation by

$$(ves^{-1})G + (vrs^{-1})Q - vR = \mathcal{O} \tag{3}$$

for any $v \in \mathbb{Z}_n^*$. In particular, if one chooses v such that both v and $u := vrs^{-1}$ (**mod** n) are integers that are significantly smaller than n, such as having only half the bit-length of n, one can often considerably speed-up the computation of the left-hand side of this equation and, thereby, the verification itself. The potential acceleration is suggested by the observation that for most groups $\mathbb{G}$ of practical interest to us, the cost of computing a multiple kP of some unknown point $P \in \mathbb{G}$ is proportional to the bit-size of the scalar multiple k involved. If P is a fixed point and some storage is available, one can accelerate the point multiplication kP by precomputing some data that depends solely on P and storing this for subsequent use. If P is an unknown point, similar acceleration techniques can be used, but are less effective, since one cannot amortize their

cost over time. A similar observation can be made regarding the cost of multiple point multiplications, where one computes a linear combination of points in $\mathbb{G}$ and can accelerate the computation by carrying out common elements hereof only once, rather than for several points separately. Also here, the cost of computing a multiple point multiplication usually depends on the maximum bit-size over all scalar multiples corresponding to unknown points. This suggests that one can indeed considerably accelerate the verification of Equation (3) by choosing $v \in \mathbb{Z}_n^*$ such that the scalars $u := vrs^{-1} (\mathbf{mod}\ n)$ and v corresponding to variable points Q and $-R$, respectively, both have relatively small bit-size.

One can find a suitable value for v by writing $x := r/s\ (\mathbf{mod}\ n)$ as $x \equiv u/v\ (\mathbf{mod}\ n)$, where u and v are integers of approximately half the bit-size of n or, more generally, of $1 - \varepsilon$ and ε times the bit-size of n, respectively, where $0 < \varepsilon < 1$ (see Example 6). For efficient algorithms for computing u and v of this form, see §3.

We now compare in more detail the cost of checking the original signature verification equation with that of checking the transformed equation discussed above, to give some more insight into the potential acceleration. We then give some concrete speed-ups for ECDSA*and ECDSA in §4.1 and §4.2.

We denote by t the bit-length of n, i.e., $t := \lceil \log_2(n+1) \rceil$, where n is the order of the group $\mathbb{G}$. For convenience, we denote $k \sim L$ iff $|k| \lesssim L$. We assume that sufficient storage is available to store a few points of $\mathbb{G}$.

We distinguish two cases, depending on whether a (nontrivial) fixed multiple of the public key Q is available. This multiple of Q may be made available to the verifier by including it in the signer's certificate for Q.

Case 1: Only one point multiple of Q available (Q itself)
One can write the original signature verification equation in the following form:

$$\lambda G + \mu Q = R, \text{ where } \lambda, \mu \sim n, \tag{4}$$

Similarly, one can write the transformed signature verification equation in the following form:

$$\lambda_0 G_0 + \lambda_1 G_1 + uQ - vR = \mathcal{O}, \text{ where } \lambda_0, \lambda_1, u, v \sim \sqrt{n}, \tag{5}$$

where $G_0 := G$ and where $G_1 := 2^{\lceil t/2 \rceil} G$ is a precomputed multiple of G.

Notice that the transformed equation involves 4 half-size point multiplications, whereas the original equation involves 2 full-size point multiplications.

Case 2: Two point multiples of Q available (Q itself and another multiple of this point)
One can write the original signature verification equation in the following form:

$$\lambda_0 G_0 + \lambda_1 G_1 + \mu_0 Q_0 + \mu_1 Q_1 = R, \text{ where } \lambda_0, \lambda_1, \mu_0, \mu_1 \sim \sqrt{n}, \tag{6}$$

where $G_0 := G$, where $G_1 := 2^{\lceil t/2 \rceil} G$ is a precomputed multiple of G, where $Q_0 := Q$, and where $Q_1 := 2^{\lceil t/2 \rceil} Q$ is a precomputed multiple of Q.

Similarly, one can write the transformed signature verification equation in the following form:

$$\lambda_0 G_0 + \lambda_1 G_1 + \lambda_2 G_2 + u_0 Q_0 + u_1 Q_1 - vR = \mathcal{O}, \text{ where } \lambda_0, \lambda_1, \lambda_2, u_0, u_1, v \sim \sqrt[3]{n}, \tag{7}$$

where $G_0 := G$, where $G_1 := 2^{\lceil t/3 \rceil} G$ and $G_2 := 2^{2\lceil t/3 \rceil} G$ are precomputed multiples of G, where $Q_0 := Q$, and where $Q_1 := 2^{\lceil t/3 \rceil} Q$ is a precomputed multiple of Q.

Notice that the transformed equation involves 6 third-size point multiplications, whereas the original equation involves 4 half-size point multiplications.

4.1 Fast Verification of ECDSA*Signatures

In the previous section, we obtained a potential acceleration of signature verifications for a family of ElGamal-like signature schemes. Here, we explore the concrete speed-up for ECDSA*.

From the case analysis in the previous section, it follows that the transformation of the signature verification equation significantly reduces the size of the scalar multiples involved in this equation (if one can store a single precomputed multiple of the generator of $\mathbb{G}$). As a result, we might expect a considerable speed-up of signature verifications, since this eliminates a large number of point doubling operations, a common element in multiple point multiplications. We expect significant improvements, both for prime curves and for random binary curves. For Koblitz curves, we can only expect a marginal improvement, since for those curves the Frobenius map (which assumes the role of point doubling) comes almost for free.

A precise analysis is difficult to give, due to the large number of point multiplication methods available. We compare the cost of checking the original signature verification with that of checking the transformed equation by combining the Non-Adjacent Form (NAF) representation for scalars with the multiple point multiplication method, and using the Joint Sparse Form (JSF); for details, see [10], [5, Chapter 3, §3.3.3]. We distinguish the same two cases as in the previous section.

We adopt the following notation: By A and D, we denote the cost of a single point addition and point doubling operation, respectively. By m, we denote the bit-length of finite field elements in $\mathbb{F}_q$, i.e., $m := \lceil \log_2 q \rceil$. We assume the elliptic curve $E(\mathbb{F}_q)$ to have a small co-factor h, so that $m \approx \lceil \log_2 n \rceil$. As before, we denote $k \sim L$ iff $|k| \lesssim L$.

Case 1: Only one point multiple of Q available (Q itself)
Consider the original signature verification equation in the format of Equation (4) and represent λ and μ in JSF. Compute the left-hand side of this equation via multiple point multiplication. Since $\lambda, \mu \sim n$, this gives a running time of approximately $(m/2+2)A + mD$ operations. Similarly, consider the transformed signature verification equation in the format of Equation (5) and represent λ_0 and λ_1, resp. u and v, in JSF. Compute the left-hand side of this equation via multiple point multiplication. Since $\lambda_0, \lambda_1, u, v \sim \sqrt{n}$, this gives a running time of approximately $(m/2+4)A + (m/2)D = (2 \cdot m/4 + 4)A + (m/2)D$ operations[2].

[2] Here, 4 point additions account for computing $G_0 \pm G_1$ and $Q_0 \pm Q_1$, which is necessary for using the 2 JSFs.

Case 2: Two point multiples of Q available (Q itself and another multiple of this point)
Consider the original signature verification equation in the format of Equation (6) and represent λ_0 and λ_1, resp. μ_0 and μ_1, in JSF. Compute the left-hand side of this equation via multiple point multiplication. Since $\lambda_0, \lambda_1, \mu_0, \mu_1 \sim \sqrt{n}$, this gives a running time of approximately $(m/2+4)A + (m/2)D = (2 \cdot m/4 + 4)A + (m/2)D$ operations. Similarly, consider the transformed signature verification equation in the format of Equation (7) and represent λ_0 and λ_1, λ_2 and u_0, resp. u_1 and v, in JSF. Compute the left-hand side of this equation via multiple point multiplication. Since $\lambda_0, \lambda_1, \lambda_2, u_0, u_1, v \sim \sqrt[3]{n}$, this gives a running time of approximately $(m/2+6)A + (m/3)D = (3 \cdot m/6 + 6)A + (m/3)D$ operations.

Rough analysis for P-384 curve. We provide a rough analysis of the relative efficiency of the ECDSA*verification procedure described in this paper compared to the traditional procedure for ECDSA verification. Our analysis is based on the elliptic curve curve P-384 defined over a 384-bit prime field and specified by NIST [3]. All NIST prime curves have co-factor $h = 1$, so by Hasse's theorem, the bit-size of the order of the cyclic group $\mathbb{G}$ is approximately $m = 384$. We consider each of the scenarios described under Case 1 and Case 2 above. We ignore the relatively minor cost of running the Extended Euclidean Algorithm to compute the half-size integers u and v.

We assume the points to be represented using Jacobian coordinates and that the cost S of a squaring in $\mathbb{Z}_q$ is slightly less than the cost M of a multiplication (we take $S \approx 0.8M$). With Jacobian coordinates, one has $A \approx 8M + 3S$ and $D \approx 4M + 4S$ (see [5, Table 3.3]). Substitution of $S \approx 0.8M$ now yields $A \approx 10.4M$ and $D \approx 7.2M$ and, hence, $D/A \approx 0.69$.

If the verifier has access only to the public key Q of the signer and not to a multiple hereof (Case 1), the cost of verifying the signature using the traditional verification method is roughly $194A + 384D \approx 459A$, while the corresponding figure for the new method is $196A + 192D \approx 328A$. As a result, the new method yields a speed-up of 40% over the old method.

Similarly, if the verifier has access to the public key Q of the signer and a suitable multiple hereof (Case 2), the corresponding figures are roughly $196A + 192D \approx 328A$ for the traditional method and $198A + 128D \approx 286A$ for the new method. Thus, in this case, the new method yields a speed-up of 15% over the old method.

Implementation on ARM7TDMI platform. We implemented the fast verification procedure of ECDSA*signatures on an ARM7TDMI processor running at a 50MHz clock rate and compared this with the traditional ECDSA verification procedure. We assumed the same scenario as considered in the rough analysis above (in particular, we chose the same curve P-384), but *did* consider the cost of computing the half-size integers u and v required for fast verification. We restricted ourselves to Case 1. The following data was obtained:

signature generation time:	∼100 ms;
traditional ECDSA verification time:	209 ms;
fast ECDSA*verification time:	154 ms.
speed-up:	36%.

The experimental data supports the rough analysis we did above for the signature verification step.

4.2 Fast Verification of ECDSA Signatures

In the previous section, we obtained a speed-up of ECDSA*signature verification. To make this efficiency improvement applicable to ECDSA as well, one needs to convert the ECDSA signature (r, s) over some message m to a corresponding ECDSA*signature (R, s) over the same message, i.e., one needs to reconstruct the ephemeral elliptic curve point R from the signature component r. Obviously, one could compute $R := s^{-1}(eG + rQ)$ directly, but this has the same computational cost as the traditional method for ECDSA signature verification and can, therefore, never yield an acceleration of signature verification. In this section, we consider alternative and more efficient mechanisms for reconstructing R from r. First, however, we provide a general framework.

By Theorem 2, one has $R \in \{(x, y) \in \mathbb{G} \mid f(\overline{x}) = r\}$ (the set of *candidate points*). Notice that for each $r \neq 0$, the set of candidate points has even cardinality, since elliptic curve points and their inverses have the same x-coordinate (and $\mathbb{G}$ has no points of order 2). Thus, one cannot uniquely extract the value of R from the set of candidate points alone, since candidate points come in pairs $(R, -R)$. It turns out, however, that in most cases there is only 1 candidate point (up to taking inverses in $\mathbb{G}$). In all cases, one can apply the fast ECDSA*verification procedure for each candidate point that is not discarded based on some side information and accept the ECDSA signature if and only if any verification using such an *admissible point* succeeds. By Theorem 2, this alternative procedure is equivalent to the original signature verification procedure for ECDSA.

The ECDSA verification cost via this procedure depends on the number of admissible points. In particular, if the set of admissible points is a singleton set, then ECDSA signature verification has the same cost as ECDSA*signature verification and the speed-up determined in §4.1 is attainable. We are now ready to discuss alternative mechanisms for reconstructing R from r.

Append explicit side information to ECDSA signatures. A simple method to make sure that the set of admissible points contains only one point is to supplement a traditional ECDSA signature (r, s) with some additional information, so as to ensure that one can uniquely (and efficiently) reconstruct R from r and this side information.

To illustrate this, consider an elliptic curve $E(\mathbb{Z}_q)$ of prime order n defined over the prime field $\mathbb{Z}_q$. We consider two cases. If $n > q$, the x-coordinate of R is equal to r. Thus, a single bit of side information is sufficient to efficiently determine the y-coordinate of R. If $n < q$, one can deduce from Hasse's theorem that $q < 2n$. Hence, the x-coordinate of R is equal to r or $r + n$, with the correct

value being determined by a single extra bit of information. Thus, in this case, two bits of side information are sufficient to efficiently determine the point R.

More generally, if the elliptic curve $E(\mathbb{F}_q)$ defined over the finite field $\mathbb{F}_q$ has co-factor h, the x-coordinate of R can assume at most $h+1$ values (h if $|E(\mathbb{F}_q)| > q$). Thus, $\lceil \log_2(h+1) \rceil + 1$ bits of side information are sufficient to efficiently determine the point R.

Since most elliptic curves used in practice have a small co-factor (e.g., the NIST curves [3] have co-factor $h = 1$ (for prime curves) or $h = 2, 4$ (for binary curves)), adding a few bits of side information suffices to uniquely reconstruct R from r.

Observe that sending r with side information is similar to sending R in compressed form, but allows the use of ECDSA, rather than requiring the use of the modified scheme ECDSA*.

The general procedure for ECDSA signature is described below.

Accelerated ECDSA signature verification.
Input: Signature (r, s), message $m \in \{0,1\}^*$, public key $Q \in \mathbb{G}$.
Output: Acceptance or rejection of signature relative to Q.
ACTIONS:

1. Verify that r and s are integers in the interval $[1, n-1]$. If any verification fails, return `'reject signature'`.
2. Compute the set of candidate points $\varphi(r) := \{(x, y) \in \mathbb{G} \mid f(\overline{x}) = r\}$.
3. Determine the set of admissible points $\mathcal{R} := \overline{\varphi}(r) \subseteq \varphi(r)$ by filtering out those candidate points that do not satisfy side constraints. If this set is empty, return `'reject signature'`.
4. Compute $e := \mathcal{H}(m)$.
5. Write $x := r/s$ as $x \equiv u/v$ (**mod** n), where u and v are integers that are significantly smaller than n.
6. Select an arbitrary point $R \in \mathcal{R}$. Compute $S := (v \cdot es^{-1})G + uQ - vR$. Set $\mathcal{R} := \mathcal{R} \setminus \{R\}$.
7. While ($S \neq \mathcal{O}$ and $\mathcal{R} \neq \emptyset$) do the following:
   (a) Select an arbitrary point $R' \in \mathcal{R}$.
   (b) Compute $S' := S + v(R - R')$.
   (c) Set $(R, S) := (R', S')$ and $R := \mathcal{R} \setminus \{R\}$.
8. If $S = \mathcal{O}$, return `'accept signature'`; otherwise, return `'reject signature'`.

Analysis of computational workload. The computational workload of the above algorithm is determined by the cost of computing admissible points and the cost of ECDSA*signature verifications. If an ECDSA signature gives rise to t admissible candidate points, then the expected workload of the above algorithm is $(t+1)/2$ ECDSA*verifies. For example, if $h = 1$ and no side information is available, then $t = 2$ and the average workload is $1\frac{1}{2}$ ECDSA*verifies, which is still less than a traditional verification. If side information is available and $t = 1$ then only a single ECDSA*verification is required. In general, it can be shown that there are at most $2(h+1)$ possible choices for the R-value. Clearly, the most favourable case is where $t = 1$. Note that the incremental cost of computing another ECDSA*verification is the cost of computing $v(R - R')$.

5 Conclusions

We have presented a method for accelerating ECDSA verification by roughly 40%. The only price one pays for the speedup is that a small number of bits needs to be appended to traditional ECDSA signatures. We emphasize that this change does not affect conformance to the existing ECDSA standards.

The speedups are also applicable to verification of DSA signatures $\sigma = (r, s)$. However, the side information one needs to efficiently recover R from r will be greater than the size of σ itself. Thus the advantage that DSA enjoys over RSA in terms of signature size is lost.

It is also evident that the speedups apply to the elliptic curve versions of many other variants of the ElGamal signature scheme.

References

1. ANSI X9.62-1998, *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, American National Standard for Financial Services, American Bankers Association, January 7, 1999.
2. G.L. Dirichlet, 'Verallgemeinerung eines Satzes aus der Lehrere von Kettenbrüchen nebst einigen Anwendungen auf die Theorie der Zahlen,' Bericht über die zur Bekanntmachung geeigneter Verhandlungen der Königlich Preussischen Akademie der Wissenschaften zu Berlin, pp. 93-95, 1842.
3. FIPS Pub 186-2, *Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication 186-2, US Department of Commerce/National Institute of Standards and Technology, Gaithersburg, Maryland, USA, January 27, 2000. (Includes change notice, October 5, 2001.)
4. R. Gallant, R. Lambert, S.A. Vanstone, 'Fast Point Multiplication on Elliptic Curves with Efficient Endomorphisms,' in *Proceedings of Advances in Cryptology – CRYPTO 2001*, Lecture Notes in Computer Science, Vol. 2139, pp. 190-200, 2001.
5. D.R. Hankerson, A.J. Menezes, S.A. Vanstone, *Guide to Elliptic Curve Cryptography*, New York: Springer, 2003.
6. G.H. Hardy, E.M. Wright, *An Introduction to the Theory of Numbers*, Fifth Edition, Oxford: Oxford University Press, 2000.
7. D.J. Johnson, A.J. Menezes, S.A. Vanstone, 'The Elliptic Curve Digital Signature Algorithm (ECDSA),' *International Journal of Information Security*, Vol. 1, pp. 36-63, 2001.
8. L. Lovász, 'An Algorithmic Theory of Numbers, Graphs and Convexity,' CBMS-NSF Regional Conference Series in Applied Mathematics, Band 50, SIAM Publications, 1986.
9. P. Nguyen, D. Stehlé, 'Low-Dimensional Lattice-Basis Reduction Revisited,' in *Proceedings of Algorithmic Number Theory – ANTS VI*, Lecture Notes in Computer Science, Vol. 3076, pp. 338-357, 2004.
10. J. Solinas, 'Low-Weight Binary Representations for Pairs of Integers,' Centre for Applied Cryptographic Research, Corr 2001-41, University of Waterloo, Ontario, Canada, 2001.

Pairing-Friendly Elliptic Curves of Prime Order

Paulo S.L.M. Barreto[1] and Michael Naehrig[2]

[1] Escola Politécnica, Universidade de São Paulo,
Av. Prof. Luciano Gualberto, tr. 3, n. 158,
BR 05508-900, São Paulo (SP), Brazil
pbarreto@larc.usp.br
[2] Lehrstuhl für Theoretische Informationstechnik,
Rheinisch-Westfälische Technische Hochschule Aachen,
Sommerfeldstr. 24, D-52074 Aachen, Germany
mnaehrig@ti.rwth-aachen.de

Abstract. Previously known techniques to construct pairing-friendly curves of prime or near-prime order are restricted to embedding degree $k \leqslant 6$. More general methods produce curves over $\mathbb{F}_p$ where the bit length of p is often twice as large as that of the order r of the subgroup with embedding degree k; the best published results achieve $\rho \equiv \log(p)/\log(r) \sim 5/4$. In this paper we make the first step towards surpassing these limitations by describing a method to construct elliptic curves of prime order and embedding degree $k = 12$. The new curves lead to very efficient implementation: non-pairing operations need no more than $\mathbb{F}_{p^4}$ arithmetic, and pairing values can be compressed to one third of their length in a way compatible with point reduction techniques. We also discuss the role of large CM discriminants D to minimize ρ; in particular, for embedding degree $k = 2q$ where q is prime we show that the ability to handle $\log(D)/\log(r) \sim (q-3)/(q-1)$ enables building curves with $\rho \sim q/(q-1)$.

Keywords: elliptic curves, pairing-based cryptosystems.

1 Introduction

A non-supersingular elliptic curve over $\mathbb{F}_p$ is called pairing-friendly if it contains a subgroup of order r whose embedding degree k is not too large, which means that computations in the field $\mathbb{F}_{p^k}$ are feasible. The optimal case occurs when the entire curve has prime order and the desired embedding degree.

Pairing-friendly curves of prime or near-prime order are absolutely essential in certain pairing-based schemes like short signatures with longer useful life. For instance, the length of BLS signatures [6] is the size of the base field p; at the 128-bit security level the scheme should be defined on a group of 256-bit order r and be mapped to a finite field of roughly 3072-bit size p^k. In the optimal case, the embedding degree should be $k = 12$. Of course, other systems would benefit as well, since the space requirements for all quantities involved in cryptographic protocols except pairing values would be kept to a minimum

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 319–331, 2006.

(pairing compression techniques [18, 11] would help reducing the bandwidth for pairing values as well).

The pioneering work of Miyaji, Nakabayashi, and Takano [15] describes how to construct elliptic curves of prime order and embedding degree $k \in \{3, 4, 6\}$. Such curves are now dubbed MNT curves, and satisfy $p \sim r$ by the Hasse bound. Extensions of the original MNT construction to curves of near-prime order were investigated by Scott and Barreto [19], and more recently by Galbraith, McKee, and Valença [10][1]. Unfortunately, those methods are restricted to $k \leqslant 6$ and hence only allow for a tradeoff: one has to choose between increasing the base field to a 512-bit prime p (thus doubling the signature size, which ceases to be "short") or contenting oneself with the lower security level of a 1536-bit finite field $\mathbb{F}_{p^6}$.

Let $\rho \equiv \log(p)/\log(r)$ be the ratio between the bit lengths of the finite field and the order of the subgroup with embedding degree k. Several methods have been proposed to construct curves with arbitrary k, including an algorithm credited to Cocks and Pinch [4, chapter 9] and related methods due to Barreto, Lynn, and Scott [1] and to Dupont, Enge, and Morain [9]. In general these algorithms only achieve $\rho \sim 2$.

Algebraic methods may produce curves with ρ closer to unity for certain values of k. Such techniques include the families of curves described by Barreto, Lynn, and Scott [3], and by Brezing and Weng [8]. The latter presents the best known results, achieving $\rho \sim 5/4$ for families of curves with $k = 8$ or $k = 24$, and $\rho \sim (k + 2)/(k - 1)$ for prime k (hence $\rho \leqslant 5/4$ for prime $k \geqslant 13$). Such ratios are already useful under many circumstances. Still, for most embedding degrees the value of ρ is larger than this; for instance, the best published value for $k = 12$ is $\rho \sim 3/2$. Besides, the use of prime k precludes many optimizations that are only possible for even k [2], making the computation of pairings much less efficient.

In spite of all these efforts, constructing pairing-friendly curves with prime order has remained an elusive open problem since it was posed by Boneh *et al.* [6, section 3.5] (see also [7, section 4.5]).

This paper is organised as follows.

Our main contribution, described in section 2, is a surprisingly simple algorithm to construct curves of prime order and embedding degree $k = 12$. The resulting security enhancement is even better than the lower bound of $k = 10$ required by Boneh *et al.*. Using the proposed method, even a 160-bit signature maps to 1920-bit field, where the best algorithms to compute discrete logarithms are worse than Pollard-rho in the elliptic group itself.

We next discuss how to compress the representations of points and pairing values for the proposed curves in section 3. It turns out that non-pairing operations need arithmetic over fields no larger than $\mathbb{F}_{p^4}$; in some cases, this can be improved to only $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ arithmetic. Besides, it is possible to compute pair-

[1] Interestingly, the latter work also considers the construction of hyperelliptic curves of genus $g = 2$ analogous to MNT elliptic curves, for which the range of embedding degrees is $k \in \{5, 8, 10, 12\}$, but the security ratio k/g is still bound by 6.

ings compressed to one third of their length without resorting to full arithmetic on fields larger than $\mathbb{F}_{p^4}$; again, under certain circumstances one can restrict field operations to $\mathbb{F}_{p^2}$, and attain *sixfold* compression.

Finally, we show in section 4 that the ability to handle large complex multiplication (CM) discriminants may have a positive influence on the minimization of ρ. In particular, for embedding degree $k = 2q$ where q is prime we describe how to build curves with $\rho \sim q/(q-1)$ and $\log(D)/\log(r) \sim (q-3)/(q-1)$. Such discriminants are thus much smaller than expected from random curves with the same embedding degree. We also briefly discuss the possibility of building curves of nearly-prime order over extension fields.

We conclude and summarise our results in section 5.

2 The Proposed Method for $k = 12$

Theorem 1. *There exists an efficient algorithm to construct elliptic curves of prime order (of nearly arbitrary bitlength) and embedding degree $k = 12$ over a prime field.*

Proof. We follow the strategy of parametrising $p(x)$, $n(x)$ and $t(x)$, and using the property that $n \mid \Phi_k(t-1)$ as in [1]. Since Φ_{12} is quartic and we know from the Hasse bound that $n \sim p \sim t^2$, we must take $t(x)$ to be a quadratic polynomial such that $n(x)$ is a quartic factor of $\Phi_{12}(t-1)$.

Galbraith *et al.* showed [10] that the only quadratic polynomials $u(x)$ such that $\Phi_{12}(u(x))$ splits into two irreducible quartic factors are $u(x) = 2x^2$ and $u(x) = 6x^2$. Setting the trace of the Frobenius to be $t(x) = 6x^2 + 1$, we obtain

$$\Phi_{12}(t(x) - 1) = n(x)n(-x),$$

where $n(x) = 36x^4 + 36x^3 + 18x^2 + 6x + 1$. From the relation $n = p + 1 - t$ we get the irreducible polynomial $p(x) = n(x) + t(x) - 1 = 36x^4 + 36x^3 + 24x^2 + 6x + 1$. The CM norm equation becomes

$$DV^2 = 4p - t^2 = 3(6x^2 + 4x + 1)^2.$$

Assuming that, for some x_0, both $n = n(x_0)$ and $p = p(x_0)$ evaluate to prime numbers, the CM method for discriminant $D = 3$ [13, 16] produces curves of form $E(\mathbb{F}_p) : y^2 = x^3 + b$, with $b \neq 0$.

Finding b is actually very simple: take the smallest $b \neq 0$ such that $b + 1$ is a quadratic residue mod$p$ and the point $G = (1, \sqrt{b+1} \bmod p)$, which is clearly on the curve[2], satisfies $nG = O$. This method is a simplification of the technique described in [12, section A.14.4] and quickly converges to a suitable b.

We see that the bitlength m of the curve order can be easily tailored by a suitable choice of x_0, namely, start with the smallest $x \sim 2^{m/4}$ such that $n(x)$ has bitlength m and increase it until finding some x_0 such that $n(x_0)$ and $p(x_0)$ are prime. □

[2] Since the curve order $n(x)$ is a large prime, there is no point of form $(0, y)$, which would necessarily have order 3.

In summary, we have the following parametrisation, where x may take either positive or negative values:

$$\begin{aligned} t &= 6x^2 + 1, \\ n &= 36x^4 + 36x^3 + 18x^2 + 6x + 1, \\ p &= 36x^4 + 36x^3 + 24x^2 + 6x + 1, \\ DV^2 &= 108x^4 + 144x^3 + 84x^2 + 24x + 3 = 3(6x^2 + 4x + 1)^2. \end{aligned}$$

The choice $u(x) = 2x^2$ as indicated in [10] is not considered, since in this case DV^2 factors as a square free product of irreducible polynomials which in general leads to an enormous CM discriminant D and therefore is not practical. This would also be the case if one took $u(x)$ to be a linear polynomial. So far the only known choice leading to a favourable factorization of the CM norm equation is $t(x) = 6x^2 + 1$.

Algorithm 1 shows how the CM method simplifies in our setting. The algorithm takes as input value the desired bitlength of the primes p and n, and returns instances of these primes computed as indicated in the proof of theorem 1, plus a parameter $b \in \mathbb{F}_p$ such that the curve $E(\mathbb{F}_p) : y^2 = x^3 + b$ has order n over the field $\mathbb{F}_p$, and the coordinate y of a sample generator $G = (1, y)$. Notice that the CM construction only ensures that the order of a curve satisfying the norm equation $3V^2 = 4p - t^2$ has one of the six forms $\{p+1\pm t, p+1\pm(t\pm 3V)/2\}$ [12, section A.14.2.3, item 6], meaning that not all choices of b will produce a curve with the prescribed order n and making it necessary to filter a suitable b by checking the curve order, as done on line 22. Since the probability that $b+1$ is a quadratic residue on line 19 is about $1/2$ and a fraction about $1/6$ of all values of b produce a curve of the right order, it is expected that the algorithm will test about 12 values of b (and compute nG about six times on line 22) until it stops.

Appendix A gives a few examples of cryptographic interest. Algorithm 1 tends to produce the smallest p and n of the desired bitlength, but it is straightforward to modify it so that the output parameters meet other simple criteria (for instance, the examples in appendix A were selected to maximize p and n while keeping all other parameters as simple as possible). It is still an open problem to determine the distribution of prime pairs (n, p) and hence the expected number of times the loop between lines 3 and 14 will be executed. Empirical evidence suggests that finding such pairs is fairly easy, taking only a fraction of a second on a common PC.

The parametrisation given above may also be expressed in the language of [8]. Choosing a polynomial $u(x)$ such that $\Phi_k(u(x))$ splits may then be interpreted as choosing a suitable number field in the following way. Let $l(x)$ be an irreducible factor of $\Phi_k(u(x))$ and consider the number field

$$K = \mathbb{Q}[x]/(l(x)).$$

As $u(x)$ is a root of Φ_k over K it is a primitive k-th root of unity modulo l. If D is chosen such that $-D$ is a square in K we set $t(x) \leftarrow u(x)^i + 1 \bmod l(x)$

Algorithm 1. Constructing a curve of prime order with $k = 12$

INPUT: the approximate desired size m of the curve order (in bits).
OUTPUT: parameters p, n, b, y such that the curve $y^2 = x^3 + b$ has order n over $\mathbb{F}_p$ and the point $G = (1, y)$ is a generator of the curve.

Let $P(x) \equiv 36x^4 + 36x^3 + 24x^2 + 6x + 1$
Compute the smallest $x \approx 2^{m/4}$ such that $\lceil \log_2 P(-x) \rceil = m$.
loop
    $t \leftarrow 6x^2 + 1$
    $p \leftarrow P(-x), \quad n \leftarrow p + 1 - t$
    if p and n are prime **then**
        exit loop
    end if
    $p \leftarrow P(x), \quad n \leftarrow p + 1 - t$
    if p and n are prime **then**
        exit loop
    end if
    $x \leftarrow x + 1$
end loop
$b \leftarrow 0$
repeat
    repeat
        $b \leftarrow b + 1$
    until $b + 1$ is a quadratic residue mod p
    Compute y such that $y^2 = b + 1 \bmod p$
    $G \leftarrow (1, y)$ on the curve $E : y^2 = x^3 + b$
until $nG = O$
return p, n, b, y.

and $V(x) \leftarrow (u(x)^i - 1)/\sqrt{-D} \bmod l(x)$ where $i \in \{1, \ldots, k-1\}$. If $p(x) = (t(x)^2 - DV(x)^2)/4$ is irreducible, one sets $n = p + 1 - t$. We are able to check for the ratio $\deg(p)/\deg(l)$ to be less than a certain given bound. Choosing $u(x) = 6x^2$ and $D = 3$ yields the above parametrisation as well.

Contrary to the case $k = 12$, finding parametrisations when $\varphi(k) > 4$ (but keeping k reasonably small) seems a rather difficult problem. The method suggested in [10] to find quadratic polynomials $u(x)$ such that $\Phi_k(u(x))$ splits, implies finding integer or rational points on an elliptic curve. Increasing $\varphi(k)$ leads to a higher number of indeterminates and also increases the number of equations to deal with. To combine them into a single equation of an elliptic curve may in this case be impossible. The computation of a resultant as suggested in [10] only reduces the number of indeterminates by one and thus in general will not help. One may try to find polynomials $u(x)$ of degree greater than two such that $\Phi_k(u(x))$ splits, but this results in higher degree equations to be solved. We leave it as an open problem the task of buidling curves of prime order and $\varphi(k) > 4$.

Furthermore, for efficiency reasons in the pairing computation it is desirable to generate curves of prime order n such that n has a low Hamming weight. Constructing such curves for $k = 12$ or $\varphi(k) > 4$ is still a research problem.

3 Point and Pairing Compression

We now consider two efficiency improvements enabled by the proposed curves, namely, point compression and pairing compression, both in general to about one third, and in some cases to one sixth, of the requirements one would expect in naïve implementations.

The basic idea for point compression is not only to restrict the first pairing argument to $E(\mathbb{F}_p)$, but also to take the second argument $Q \in E(\mathbb{F}_{p^{12}})$ as the image $\psi(Q')$ of a point on a sextic twist $E'(\mathbb{F}_{p^2})$ such that $n \mid \#E'(\mathbb{F}_{p^2})$, where $\psi : E'(\mathbb{F}_{p^2}) \to E(\mathbb{F}_{p^{12}})$ is an injective group homomorphism. This way one would work only with $E(\mathbb{F}_p)$ and $E'(\mathbb{F}_{p^2})$ for non-pairing operations like key generation, and map from $E'(\mathbb{F}_{p^2})$ to $E(\mathbb{F}_{p^{12}})$ only when actually computing 'twisted' pairings $e'(P, Q') \equiv e(P, \psi(Q))$. As it turns out, it is possible to do better than this, namely, to work with smaller fields, as we will show.

Lemma 1. *There exists $\xi \in \mathbb{F}_{p^2}^*$ such that $X^6 - \xi$ is irreducible over $\mathbb{F}_{p^2}[X]$ whenever $p \equiv 1 \pmod 6$.*

Proof. Since $p^2 \equiv 1 \pmod 6$, the order of $\mathbb{F}_{p^2}^* = p^2 - 1$ is a multiple of 6. Thus for any primitive root $a \in \mathbb{F}_{p^2}$, the cube roots of unity are $\{1, \zeta, \zeta^2\}$ where $\zeta \equiv a^{(p^2-1)/3}$. Hence every cube $u^3 \in \mathbb{F}_{p^2}^*$ has three distinct cube roots, namely, $\{u, \zeta u, \zeta^2 u\}$, which means that only one third of the elements of $\mathbb{F}_{p^2}^*$ are cubes. Analogously, only one half of the elements of $\mathbb{F}_{p^2}^*$ are squares. Therefore, there must be some element $\xi \in \mathbb{F}_{p^2}^*$ that is neither a square nor a cube, and hence $X^6 - \xi$ is irreducible over $\mathbb{F}_{p^2}[X]$. □

A sensible strategy to obtain ξ without resorting to full $\mathbb{F}_{p^{12}}$ arithmetic is to set $1/\xi = \lambda^2\mu^3$ where $\lambda \in \mathbb{F}_p$ is a noncube and $\mu \in \mathbb{F}_{p^2}$ is a nonsquare. Although any $\xi \in \mathbb{F}_{p^2}^*$ provided by lemma 1 may be used to represent $\mathbb{F}_{p^{12}}$ as $\mathbb{F}_{p^2}[X]/(X^6 - \xi)$ and to define a sextic twist $E'(\mathbb{F}_{p^2}) : y'^2 = x'^3 + b/\xi$, one must choose it so as to ensure that $n \mid E'(\mathbb{F}_{p^2})$. If a particular ξ produces a twist of wrong order, an easily computable alternative is ξ^5 (note that ξ^2, ξ^3, and ξ^4 are not suitable since they produce quadratic and cubic twists). We leave it as an exercise for the reader to show that the correct twist order is $\#E'(\mathbb{F}_{p^2}) = (p+1-t)(p-1+t) = n(2p-n)$.

Let $z \in \mathbb{F}_{p^{12}}$ be a root of $X^6 - \xi$. The corresponding map $\psi : E'(\mathbb{F}_{p^2}) \to E(\mathbb{F}_{p^{12}})$ is $(x', y') \mapsto (z^2x', z^3y')$. Notice that $x = z^2x' \in \mathbb{F}_{p^6}$ and $y = z^3y' \in \mathbb{F}_{p^4}$. Also, since any element of $\mathbb{F}_{p^{12}}$ has the form $a_5z^5 + a_4z^4 + a_3z^3 + a_2z^2 + a_1z + a_0$ the computation of $\psi(Q')$ does not incur any multiplication overhead, and its rather sparse structure favours efficient implementation of the pairing algorithm.

These considerations open the way to compressing pairing values to one third or even one sixth of their length in a way that is compatible with point compression or point reduction, i.e. the techniques of keeping only one point coordinate and respectively keeping a small fraction of the other coordinate or discarding it entirely. Notice that the map $(x', y') \mapsto (z^2x', z^3y')$ produces a point on $E(\mathbb{F}_{p^{12}})$ whose x-coordinate is in $\mathbb{F}_{p^6}$ and whose y-coordinate is in $\mathbb{F}_{p^4}$.

Now suppose that we keep only the y-coordinate of the point $Q' = (x', y')$ on the twist $E'(\mathbb{F}_{p^2}) : y'^2 = x'^3 + b/\xi$ (this is contrary to the conventional

choice of keeping only the x-coordinate). There are three points associated to this y-coordinate corresponding to the three cube roots of $y'^2 - b/\xi$. One of the points is Q'. Upon mapping onto $E(\mathbb{F}_{p^{12}})$, it turns out that those points map to *conjugates* over $\mathbb{F}_{p^4}$ (i.e. their images are of form Q, $[p^4]Q$ and $[p^8]Q$) provided Q' is an n-torsion point, which already exists because of the choice of the twist $E'(\mathbb{F}_{p^2})$. Let ϕ be the p-th power Frobenius endomorphism and $\mathrm{tr}_{\mathbb{F}_{p^6}} : E(\mathbb{F}_{p^{12}}) \to E(\mathbb{F}_{p^6}),\ R \mapsto R + \phi^6(R)$ the trace map over $\mathbb{F}_{p^6}$. An explicit computation leads to the following lemma.

Lemma 2. *Let $Q' = (x', y') \in E'(\mathbb{F}_{p^2})$ and let $Q = \psi(Q')$. Then $\mathrm{tr}_{\mathbb{F}_{p^6}}(Q) = Q + \phi^6(Q) = O$.*

The Frobenius endomorphism has two eigenspaces in $E(\mathbb{F}_{p^{12}})[n]$ for the eigenvalues 1, p. The 1-eigenspace consists of all points in $E(\mathbb{F}_p)$ while the p-Eigenspace is the set of points of trace zero. Therefore we obtain the following lemma which shows that for an n-torsion point whose $\mathbb{F}_{p^6}$-trace is O computing the p-multiple is the same as computing the Frobenius endomorphism.

Lemma 3. *Let $Q \in E(\mathbb{F}_{p^{12}})[n]$. Then $\mathrm{tr}_{\mathbb{F}_{p^6}}(Q) = O$ iff $\phi(Q) = [p]Q$.*

Let $Q = \psi(Q')$ for an n-torsion point Q' on the sextic twist. From lemma 2 we see that we may apply lemma 3 and compute $[p^4]Q = \phi^4(Q) = ((z^2)^{p^4}x', z^3y')$ as well as $[p^8]Q = \phi^8(Q) = ((z^2)^{p^2}x', z^3y')$. The points Q, $[p^4]Q$, and $[p^8]Q$ share the same y-coordinate and therefore have to be the images under ψ of the above mentioned three points corresponding to the given y-coordinate on the twist.

The above shows that the pairing values computed from the three points are also conjugates over $\mathbb{F}_{p^4}$ (i.e. they are of the form e, e^{p^4} and e^{p^8}). Thus, the $\mathbb{F}_{p^4}$-trace of these pairing values is the same for any of the three points. In other words, the choice of the cube root is irrelevant to compute the compressed pairing $\mathrm{tr}_{\mathbb{F}_{p^4}}(e'(P, Q')) = e'(P, Q') + e'(P, Q')^{p^4} + e'(P, Q')^{p^8}$, whose length is one third of the length of $e'(P, Q')$. This reduction also allows for implicit trace exponentiation analogous to the laddering techniques described in [18]; similar effects may be achieved in a torus-based setting as suggested in [11]. Threefold pairing compression may thus be used in the identity-based encryption scheme by Boneh and Franklin [5] for example.

One even may go one step further and discard not only the x-coordinate of Q' but also one bit of its y-coordinate so as not to distinguish between y' and $-y'$ (or, equivalently, between Q' and $-Q'$). Doing so means working with *equivalence classes* of points rather than with the points themselves. The equivalence classes are defined by the isomorphism $(x', y') \mapsto (\zeta_3 x', -y')$ where $\zeta_3^3 = 1$, hence each class contains six finite points $\overline{Q'} \equiv \{(x', \pm y'), (\zeta_3 x', \pm y'), (\zeta_3^2 x', \pm y')\}$, except for the zero class $\overline{O'}$ which contains only the point at infinity O'. With help of the following lemma we can show that pairing compression up to one sixth of the actual pairing length is possible.

Lemma 4. *Let ζ be an n-th root of unity in $\mathbb{F}_{p^{12}}$ and $\mathrm{tr}_{\mathbb{F}_{p^2}} : \mathbb{F}_{p^{12}} \to \mathbb{F}_{p^2}$ the finite field trace over $\mathbb{F}_{p^2}$. Then $\mathrm{tr}_{\mathbb{F}_{p^2}}(\zeta^{-1}) = \mathrm{tr}_{\mathbb{F}_{p^2}}(\zeta)$.*

Proof. Since n divides $\Phi_{12}(t-1)$ it divides $\Phi_{12}(p) = p^4 - p^2 + 1$. So n also divides $p^6 + 1 = (p^2+1)(p^4 - p^2 + 1)$. Therefore since ζ is an n-th root of unity we have $\zeta^{-1} = \zeta^{p^6}$. We now see that $\mathrm{tr}_{\mathbb{F}_{p^2}}(\zeta^{-1}) = \zeta^{-1} + \zeta^{-p^2} + \zeta^{-p^4} + \zeta^{-p^6} + \zeta^{-p^8} + \zeta^{-p^{10}} = \zeta^{p^6} + \zeta^{p^8} + \zeta^{p^{10}} + \zeta + \zeta^{p^2} + \zeta^{p^4} = \mathrm{tr}_{\mathbb{F}_{p^2}}(\zeta)$. □

Since all pairing values are n-th roots of unity in $\mathbb{F}_{p^{12}}$ it follows that $\mathrm{tr}_{\mathbb{F}_{p^2}}(e'(P,Q')) = \mathrm{tr}_{\mathbb{F}_{p^2}}(e'(P,Q')^{-1}) = \mathrm{tr}_{\mathbb{F}_{p^2}}(e'(P,-Q'))$. Together with the transitivity of traces using the above condition on $\mathbb{F}_{p^4}$- traces, this yields that the $\mathbb{F}_{p^2}$-traces of the pairing values are equal for all points in $\overline{Q'}$. Therefore it makes sense to define the compressed pairing $\varepsilon(P,\overline{Q'}) \equiv \mathrm{tr}_{\mathbb{F}_{p^2}}(e(P,\psi(Q')))$, whose second argument is an equivalence class of points on the sextic twist of the base curve containing the first argument.

The advantage of this approach is that one can not only work exclusively on $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ for non-pairing operations, but also represent points on $E'(\mathbb{F}_{p^2})$ using less storage than one $\mathbb{F}_{p^2}$ element, yet obtain a unique compressed pairing value over $\mathbb{F}_{p^2}$. We point out that the compression ratio of 1/6 is better than what is attainable on any practical supersingular Abelian variety, namely, 8/30, as shown by Rubin and Silverberg [17]. This observation also suggests that the proposed prime pairs (p,n) might define an extension of XTR [14] with a sixfold (rather than threefold) compression ratio, but obtaining the required method for implicit exponentiation remains an open problem. Therefore, sixfold pairing compression is currently indicated for cryptographic schemes where the pairing value is not meant to be further processed like in the BLS signature scheme [6].

3.1 Computing Cube Roots

If the x-coordinate corresponding to a point on $E(\mathbb{F}_p)$ or $E'(\mathbb{F}_{p^2})$ with a given y-coordinate is needed, one may obtain it by simply computing a cube root of $u = y^2 - b/\xi$. We now briefly discuss how to efficiently compute cube roots in half of the fields of interest for the proposed curves with $k = 12$.

Each prime number of form $p(x) = 36x^4 + 36x^3 + 24x^2 + 6x + 1$ is congruent to $6x^2 + 6x + 1 \pmod 9$ and hence $p(x) \equiv 1 \pmod 9$ or $p(x) \equiv 4 \pmod 9$ for all $x \in \mathbb{Z}$.

In the latter case, if $u \in \mathbb{F}_p^*$ is a cube then $u^{(p-1)/3} = 1$. Since $(2p+1)/9$ is the inverse of $3 \pmod{(p-1)/3}$, one cube root $r \in \mathbb{F}_p^*$ of u is clearly given by

$$r \leftarrow u^{(2p+1)/9}.$$

Therefore, computing cube roots modulo $p \equiv 4 \pmod 9$ only takes one exponentiation. Notice that all examples given in appendix A fall into this category.

For recovering the x-coordinate of points on $E'(\mathbb{F}_{p^2})$ given only their y-coordinate one needs to compute a cube root in $\mathbb{F}_{p^2}^*$, and for $p \equiv 4 \pmod 9$ we have $p^2 \equiv 7 \pmod 9$. Now if $u \in \mathbb{F}_{p^2}^*$ is a cube then $u^{(p^2-1)/3} = 1$. Since $(p^2+2)/9$ is the inverse of $3 \pmod{(p^2-1)/3}$, one cube root $r \in \mathbb{F}_{p^2}^*$ of u is given by

$$r \leftarrow u^{(p^2+2)/9}.$$

Thus again the computation of a cube root only takes one exponentiation.

In both cases, if it is not known whether u is a cube one must check that the result is correct, i.e. that $r^3 = u$.

4 Considerations on Composite Order

Under some circumstances, a reasonably small cofactor may be quite acceptable. For instance, if 256-bit prime fields do not have a substantial impact on bandwidth occupation, the Brezing-Weng family of curves for $k = 8$ and $\rho \sim 5/4$ could provide roughly 200-bit group orders and map the discrete logarithm on the curve to a 2048-bit finite field. Besides, as we already pointed out even values of k are advantageous from the point of view of efficient implementation of the pairing algorithm. It is thus interesting to investigate ways to produce more curves that meet the conditions that k be even and $\rho > 1$ be as small as possible (say, $\rho \leqslant 5/4$).

A naïve approach to solving the norm equation $DV^2 = 4h\Phi_k(t-1) - (t-2)^2$, namely, by choosing t and hoping to be able to handle the resulting D, is in general bound to failure since $D \sim t^{\varphi(k)}$, where $\varphi(k)$ is Euler's totient function. For instance, for $k = 2q$ where q is an odd prime we expect to find $D \sim t^{q-1}$.

However, it would be quite simple to obtain curves with $k = 2q$ if we could handle a CM discriminant D as large as t^{q-3}, attaining $\rho \equiv \log(p)/\log(r) \sim q/(q-1)$ as the following reasoning reveals. Let the trace of Frobenius have the form $t = -4u^2 + 2$ for some u (notice that t is negative), and let $x = t - 1$. Assume that $\Phi_k(x)$ takes a prime value. Then set:

$$\begin{aligned}
h &= -(x-1)/4, \\
r &= \Phi_k(x) \\
&= x^{q-1} - x^{q-2} + x^{q-3} - x^{q-4} + x^{q-5} - \cdots - x + 1 \\
&= x^{q-1} - x^{q-3}(x-1) - x^{q-5}(x-1) - \cdots - x^2(x-1) - (x-1), \\
p &= hr + t - 1, \\
DV^2 &= 4hr - (t-2)^2 \\
&= -(x-1)x^{q-1} + x^{q-3}(x-1)^2 + x^{q-5}(x-1)^2 + \cdots + (x-1)^2 - (x-1)^2 \\
&= -(x-1)x^2[x^{q-3} - (x-1)(x^{q-5} + x^{q-7} + \cdots + 1)].
\end{aligned}$$

By construction, the $-(x-1)x^2$ factor is a square, so D is the square-free part of $z = x^{q-3} - (x-1)(x^{q-5} + x^{q-7} + \cdots + 1)$. Since $p = hr + t - 1$, it is also clear that $\rho \sim q/(q-1)$. For instance, if $k = 10$ (i.e. $\rho \sim 5/4$) we get $z = x^2 - x + 1$, and a simple search produces parameters like these:

$t = -931556989582$: 40 bits

$r = 753074106157227719531468778253698105623799226081$: 160 bits

$p = 175382861816372173247473133505975362972517516867279787545493$: 197 bits

$\rho \sim 1.237425$

$D = 867798424841873127503473$: 80 bits

Another example, now for $k = 14$ (i.e. $\rho \sim 7/6$) where $z = x^4 - x^3 + x^2 - x + 1$:

$t = -82011134$: 27 bits

$r = 30425445052504605008506791451346026107875713536 1$: 158 bits

$p = 62380632801537057549473290765999408254813645346833338 89$: 183 bits

$\rho \sim 1.153987$

$D = 45236739484946456935793243535361$: 106 bits

Unfortunately, with currently available CM technology the only case where this construction is tractable occurs for $k = 6$, where we get $D = 1$ but also $\rho \sim 3/2$, much worse than plain MNT curves that attain $\rho \sim 1$.

4.1 Curves over Extension Fields

Another interesting observation is that, while none of the currently known methods to construct pairing-friendly curves for arbitrary k is able to produce curves over an extension field $\mathbb{F}_{p^m}$, it may be possible to fill this gap if sufficiently large D can be handled. As Galbraith *et al.* point out [10], parametrising $t = 5x^2 + 1$ causes $\Phi_5(t-1)$ to split as $\Phi_5(t-1) = r(x)r(-x)$, where $r(x) = 25x^4 + 25x^3 + 15x^2 + 5x + 1$. We observe that with cofactor $h = 4$, this gives $hr + t - 1 = (10x^2 + 5x + 2)^2$, a perfect square. This means that finding an odd value $x \in \mathbb{Z}$ such that r and $p = 10x^2 + 5x + 2$ are both prime enables constructing an elliptic curve over a finite field $\mathbb{F}_{p^2}$ with near-prime order $n = 4r$.

The CM equation here has the form $DV^2 = 5(5x^2 \pm 2x + 1)(15x^2 \pm 10x + 3)$. Solving a Pell-like equation can make one but not both of the factors $5x^2 \pm 2x + 1$ or $15x^2 \pm 10x + 3$ to assume the form dy^2 for small d and some y. One might hope that techniques like Hensel lifting could reduce the square-free part of the other factor to $O(x)$, but it is not clear how to harmonise such techniques to solutions of the Pell-like equation. As a consequence, we expect that $D \sim p \sim r^{1/2}$; practical values of p would need $D \sim 2^{100}$ at least.

Nevertheless, such a parametrisation hints that algebraic methods to build ordinary pairing-friendly curves over extension fields might exist for other embedding degrees, and deserved further research.

5 Conclusion

We have presented a very simple algorithm to construct pairing-friendly curves of prime order and embedding degree $k = 12$. This closes (and actually exceeds) an open problem proposed by Boneh *et al.* [6, section 3.5], increasing the security level of most pairing-based cryptosystems, while also reducing up to sixfold the bandwidth requirements of either points or pairing values. Such levels of security and compression are better than what is attainable with any supersingular Abelian variety up to at least genus 6. We leave it as an open problem the task of extending the method for higher values of k.

We have also discussed ways to produce curves of composite order and reasonably small cofactor as long as large discriminants fall within reach of CM methods, and pointed out the possibility of closing yet another problem, namely, building pairing-friendly curves of nearly-prime order over extension fields. Further exploration of such possibilities is left for future research.

Acknowledgments

We are grateful to Peter Birkner, Dan Boneh, Steven Galbraith, Florian Heß, Tanja Lange, Ben Lynn, Alfred Menezes, Mike Scott, Fré Vercauteren, and Felipe Voloch for their valuable comments on an earlier version of this work.

References

1. P. S. L. M. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. In *Security in Communication Networks – SCN'2002*, volume 2576 of *Lecture Notes in Computer Science*, pages 263–273. Springer-Verlag, 2002.
2. P. S. L. M. Barreto, B. Lynn, and M. Scott. On the selection of pairing-friendly groups. In *Selected Areas in Cryptography – SAC'2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 17–25. Springer-Verlag, 2003.
3. P. S. L. M. Barreto, B. Lynn, and M. Scott. Efficient implementation of pairing-based cryptosystems. *Journal of Cryptology*, 17(4):321–334, 2004.
4. I. Blake, G. Seroussi, and N. Smart. *Advances in Elliptic Curve Cryptography*. Number 317 in London Mathematical Society Lecture Note Series. Cambridge University Press, Cambridge, UK, 2005.
5. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.
6. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology – Asiacrypt'2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer-Verlag, 2002.
7. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.
8. F. Brezing and A. Weng. Elliptic curves suitable for pairing based cryptography. Cryptology ePrint Archive, Report 2003/143, 2003. Available from `http://eprint.iacr.org/2003/143`.
9. R. Dupont, A. Enge, and F. Morain. Building curves with arbitrary small MOV degree over finite prime fields. *Journal of Cryptology*, 18(2):79–89, 2005.
10. S. Galbraith, J. McKee, and P. Valença. Ordinary abelian varieties having small embedding degree. Cryptology ePrint Archive, Report 2004/365, 2004. Available from `http://eprint.iacr.org/2004/365`.
11. R. Granger, D. Page, and M. Stam. On small characteristic algebraic tori in pairing-based cryptography. Cryptology ePrint Archive, Report 2004/132, 2004. Available from `http://eprint.iacr.org/2004/132`.
12. IEEE Computer Society, New York, USA. *IEEE Standard Specifications for Public-Key Cryptography – IEEE Std 1363-2000*, 2000.

13. G.-J. Lay and H. G. Zimmer. Constructing elliptic curves with given group order over large finite fields. In *Algorithmic Number Theory Symposium – ANTS-I*, volume 877 of *Lecture Notes in Computer Science*, pages 250–263. Springer-Verlag, 1994.
14. A. K. Lenstra and E. R. Verheul. The xtr public key system. In *Advances in Cryptology – Crypto'2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 1–19, Santa Barbara, USA, 2000. Springer-Verlag.
15. A. Miyaji, M. Nakabayashi, and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Transactions on Fundamentals*, E84-A(5):1234–1243, 2001.
16. F. Morain. Building cyclic elliptic curves modulo large primes. In *Advances in Cryptology – Eurocrypt'1991*, volume 547 of *Lecture Notes in Computer Science*, pages 328–336. Springer-Verlag, 1991.
17. K. Rubin and A. Silverberg. Supersingular abelian varieties in cryptology. In *Advances in Cryptology – Crypto'2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 336–353. Springer-Verlag, 2002.
18. M. Scott and P. S. L. M. Barreto. Compressed pairings. In *Advances in Cryptology – Crypto'2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 140–156, Santa Barbara, USA, 2004. Springer-Verlag.
19. M. Scott and P. S. L. M. Barreto. Generating more MNT elliptic curves. *Designs, Codes and Cryptography*, 2005. To appear.

A Some Curves of Prime Order and $k = 12$

All of the following curves satisfy the equation $E(\mathbb{F}_p) : y^2 = x^3 + 3$, with prime order n and trace of the Frobenius t. A sample generator for any of them is $G = (1, 2)$. In all cases $p \equiv 3 \pmod 4$ and $p \equiv 4 \pmod 9$ (to simplify the computation of square and cube roots), and the bitlengths of p and n are equal. The field $\mathbb{F}_{p^2}$ is represented as $F_p[X]/(X^2 + 1)$, and i is a root of $X^2 + 1$. The sextic twist for all examples has the form $E'(\mathbb{F}_{p^2}) : y'^2 = x'^3 + 3/\xi$, where $1/\xi = \lambda^2\mu^3 = -8 + 8i$, $\lambda = 2$, and $\mu = 1 + i$. The field $\mathbb{F}_{p^{12}}$, if desired, can be represented as $F_{p^2}[X]/(X^6 - \xi)$.

160 Bits

$$p = 1461501624496790265145448589920785493717258890819$$
$$n = 1461501624496790265145447380994971188499300027613$$
$$t = 120892581430521795886320 7$$

192 Bits

$$p = 6277101719531269400517043710060892862318604713139674509723$$
$$n = 6277101719531269400517043709981664699044017441600365563 89$$
$$t = 7922816241420296897963795 3335$$

224 Bits

$$p = 26959946667149205758383469736921695435015736735261155141423417423923$$
$$n = 26959946667149205758383469736921690242718878200571531029749235996909$$
$$t = 519229685853468962411674181427015$$

256 Bits

$$p = 115792089237314936872688561244471742058375878355761205198700409522629\backslash 664518163$$
$$n = 115792089237314936872688561244471742058035595988840268584488757999429\backslash 535617037$$
$$t = 340282366920936614211651523200128901127$$

Minimality of the Hamming Weight of the τ-NAF for Koblitz Curves and Improved Combination with Point Halving

Roberto Maria Avanzi[1,⋆], Clemens Heuberger[2,⋆⋆], and Helmut Prodinger[3,⋆⋆⋆]

[1] Faculty of Mathematics and Horst Görtz Institute for IT Security, Ruhr-University Bochum, Germany
roberto.avanzi@ruhr-uni-bochum.de
[2] Institut für Mathematik B, Technische Universität Graz, Austria
clemens.heuberger@tugraz.at
[3] Department of Mathematics, University of Stellenbosch, South Africa
hproding@sun.ac.za

Abstract. In order to efficiently perform scalar multiplications on elliptic Koblitz curves, expansions of the scalar to a complex base associated with the Frobenius endomorphism are commonly used. One such expansion is the τ-adic NAF, introduced by Solinas. Some properties of this expansion, such as the average weight, are well known, but in the literature there is no proof of its *optimality*, i.e. that it always has minimal weight. In this paper we provide the first proof of this fact.

Point halving, being faster than doubling, is also used to perform fast scalar multiplications on generic elliptic curves over binary fields. Since its computation is more expensive than that of the Frobenius, halving was thought to be uninteresting for Koblitz curves. At PKC 2004, Avanzi, Ciet, and Sica combined Frobenius operations with one point halving to compute scalar multiplications on Koblitz curves using on average 14% less group additions than with the usual τ-and-add method without increasing memory usage. The second result of this paper is an improvement over their expansion. The new representation, called the *wide-double-NAF*, is not only simpler to compute, but it is also optimal in a suitable sense. In fact, it has minimal Hamming weight among all τ-adic expansions with digits $\{0, \pm 1\}$ that allow one halving to be inserted in the corresponding scalar multiplication algorithm. The resulting scalar multiplication requires on average 25% less group operations than the

⋆ This paper was in part written while this author was visiting the Institut für Mathematik, Technische Universität Graz, supported by the START-project Y96-MAT of the Austrian Science Fund. The author's research described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

⋆⋆ Supported by the grant S8307-MAT of the Austrian Science Fund.

⋆⋆⋆ Supported by the grant NRF 2053748 of the South African National Research Foundation.

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 332–344, 2006.

Frobenius method, and is thus 12.5% faster than the previously known combination.

Keywords. Koblitz curves, scalar multiplication, point halving, τ-adic expansion, integer decomposition.

1 Introduction

The use of elliptic curves to design cryptosystems [8, 6] has become increasingly relevant in the last years and it is nowadays regulated by standards [15, 16]. The basic operation of such a cryptosystem is the *scalar multiplication*, i.e. given a point $\boldsymbol{P}$ and an integer s, to compute $s\boldsymbol{P}$. Such an operation is usually performed by a method called double-and-add: it consists in writing the scalar s as $\sum_{i=0}^{\ell} s_i 2^i$ and in evaluating $s\boldsymbol{P} = \sum_{i=0}^{\ell} s_i 2^i \boldsymbol{P}$ by a Horner-like scheme.

Some families of elliptic curves have arithmetic properties that permit very fast implementations of the scalar multiplication, making them especially attractive for the applications. Among them, the *Koblitz curves* [7] defined by

$$E_a\colon y^2 + xy = x^3 + ax^2 + 1 \qquad \text{with} \qquad a \in \{0, 1\} \tag{1}$$

over a finite field $\mathbb{F}_{2^n}$ are of particular relevance. The good performance of Koblitz curves is due to the Frobenius endomorphism τ. This is the map induced on the curve by the Frobenius automorphism of the field extension $\mathbb{F}_{2^n}/\mathbb{F}_2$, that maps a field element to its square. The evaluation of τ is much faster than the usual group law on the curve: τ is performed by squaring the coordinates, and if a suitable representation of the field $\mathbb{F}_{2^n}$ is chosen, this operation has almost negligible computational cost. The basic idea is to rewrite the scalar to the "base of τ" instead of to the base of 2, so that a "τ-and-add" scalar multiplication method using τ in place of doublings [13, 14] can be deployed. *In this paper we give a proof that the commonly used expansion to the base of τ, Solinas' τ-NAF, is optimal, i.e. its weight is minimal among all the τ-adic representations of the same integer with digits $\{0, \pm 1\}$ – in fact we prove the stronger result that the sum of the absolute values of its digits is minimal among all τ-adic expansions with rational integer coefficients.*

Point halving [5, 10] is the inverse operation to point doubling and applies to *all* elliptic curves over binary fields, not only to Koblitz curves. Its evaluation is 2 to 3 times faster than that of a doubling and it is possible to rewrite the scalar multiplication algorithm using halving instead of doubling. The resulting method is very fast, but on Koblitz curves it is slower than the τ-and-add method.

In [2] it is proposed to insert a halving in the "τ-and-add" method to further speed up scalar multiplication. This approach brings a non-negligible speedup (on average 14% with suitable representations of the fields) with respect to the use of the τ-NAF, but it is not optimal. *We show how to get an optimal representation of the scalar under the same assumptions, and we analyse the complexity.* The scalar multiplication performed using our representation is now on average 25% faster than the Frobenius method, up from 14%.

In the next section some mathematical background is recalled. Sections 3 and 4 are respectively devoted to the minimality of the τ-NAF and to the wide-double-NAF, our optimal improvement of the results from [2]. In particular § 4.5 contains a simple analysis of the Hamming weight of the wide-double-NAF. In Section 5 we conclude.

2 Background

2.1 Koblitz Curves

We consider a curve E_a defined over $\mathbb{F}_{2^n}$ by equation (1), with a (unique) subgroup G of large prime order p (standards require the cofactor to be at most 4). Set $\mu = (-1)^{1-a}$. Recall that τ is induced by the map $x \mapsto x^2$. The equation of E_a is invariant under τ, hence the map permutes the $\mathbb{F}_{2^n}$-rational points on it. G is invariant, too. It is well-known [14, Section 4.1] that for each $\boldsymbol{P} \in E_a(\mathbb{F}_{2^n})$, we have $(\tau^2 + 2)\boldsymbol{P} = \mu\tau(\boldsymbol{P})$. Thus we can identify τ with a complex number satisfying

$$\tau^2 + 2 = \mu\tau \ . \tag{2}$$

For $z \in \mathbb{Z}[\tau]$, a τ-expansion of z is an expression $\mathbf{s} = (\ldots, s_2, s_1, s_0) \in \{-1, 0, 1\}^{\mathbb{N}_0}$ where only finitely many $s_j \neq 0$ and $\mathsf{val}_\tau(\mathbf{s}) := \sum_{j\geq 0} s_j \tau^j = z$. The Hamming weight of $\mathbf{s}$ is the number of $j \geq 0$ such that $s_j \neq 0$.

For simplicity, when there is no risk of ambiguity, we write $\mathbf{s}\boldsymbol{P}$ *in place of* $\mathsf{val}(\mathbf{s})P = \sum_{j\geq 0} s_i \tau^i(\boldsymbol{P})$*, for any point* $\boldsymbol{P}$ *and* τ*-adic expansion* $\mathbf{s}$*.*

The τ-adic non-adjacent form (τ-NAF for short) of an integer $z \in \mathbb{Z}[\tau]$ is a decomposition $\mathbf{s}$ as above with the *non-adjacency* property $s_j s_{j+1} = 0$ for $j \geq 0$, similarly to the classical NAF [9]. The average *density* (that is the average ratio of non-zero bits related to the total number of bits) of a τ-NAF is $1/3$. Each integer z admits a unique τ-NAF. Its computation is easy (see for example [14]).

If $m \in \mathbb{Z}$ has a τ-expansion $\mathbf{s}$ and $\boldsymbol{P} \in E_a(\mathbb{F}_{2^n})$, $m\boldsymbol{P}$ can be computed by evaluating $\sum_{j\geq 0} s_j \tau^j(\boldsymbol{P})$ by a Horner-like scheme called τ-and-add. Clearly, the Hamming weight corresponds to the number (plus 1) of additions on the curve E_a.

Before using the τ-adic expansion in scalar multiplication we have to reduce the integer m modulo $(\tau^n - 1)/(\tau - 1)$ (note that τ^n is the identity on the curve) to keep the length of the expansion bounded by n plus a small constant. The τ-NAF of m mod $(\tau^n - 1)/(\tau - 1)$ is called the *reduced* τ-NAF of m. Solinas [13, 14] has a different, in practice faster approach.

2.2 Point Halving

Let now $E/\mathbb{F}_{2^n}$ be a generic elliptic curve defined by an equation

$$E \ : \ y^2 + xy = x^3 + ax^2 + b \qquad \text{with} \qquad a, b \in \mathbb{F}_{2^n}$$

(it is not necessarily a Koblitz curve) and a subgroup $G \leq E(\mathbb{F}_{2^n})$ of large prime order. Halving a point $\boldsymbol{P}$ means to find a point $\boldsymbol{R}$ such that $2\boldsymbol{R} = \boldsymbol{P}$. As described in [5, 10, 11], point halving can be performed by two field multiplications

(denoted by M) in the field $\mathbb{F}_{2^n}$, solving an equation of the type $\lambda^2+\lambda=c$ for λ (EQ) and extraction of a square root ($\sqrt{}$). An elliptic curve addition (in affine coordinates, usually the fastest system for elliptic curves in even characteristic) is done by one field inversion (I), two multiplications and one squaring (S). A point doubling requires $I+2M+2S$.

With a polynomial basis, according to [4], $S\approx\frac{1}{7.5}M$ for $n=163$ and $\frac{1}{9}M$ for $n=233$. Following [3] we assume that, on average, $I\approx 8\,M$ when $n=163$ and $I\approx 10\,M$ when $n=233$. In $\mathbb{F}_{2^{233}}$, a field defined by a trinomial, a square root can be computed in $\approx\frac{1}{8}M$ [3, Example 3.12]. For $\mathbb{F}_{2^{163}}$ only a generic method is currently known, so $\sqrt{}\approx\frac{1}{2}M$. EQ takes, experimentally $\approx\frac{2}{3}M$. If a normal basis is used S, $\sqrt{}$ and EQ have negligible costs, $I\approx 3\,M$. It is then clear that a point halving can be performed in a fraction of the time required by an addition or of a doubling.

According to the very thorough analysis in [3], halving is about two times faster than doubling. We refer the interested reader to [5, 10, 11, 3] for details.

2.3 Frobenius-Cum-Halving

Avanzi, Ciet, and Sica [2] combine the τ-NAF with a single point halving, thereby reducing the amount of point additions from $n/3$ to $2n/7$. They can therefore claim an asymptotic speed-up of $\approx 14.29\%$ on average. Their fundamental idea is that it is possible, using a single point halving, to replace some sequences of a τ-NAF having density $1/2$ and containing at least three non-zero coefficients with sequences having weight 2. Their starting point is the observation that (2) implies $\tau^3+2\tau=\mu\tau^2=\mu(\mu\tau-2)=\tau-2\mu$, hence

$$2=-\mu\big(1+\tau^2\big)\tau\ . \tag{3}$$

Therefore $2\boldsymbol{P}$ can be computed as $-\mu\big(1+\tau^2\big)\tau\boldsymbol{P}$ – which is in fact computationally more expensive, hence this relation is per se not useful. But it can be used to build telescopic sums: In fact, if $\boldsymbol{P}=2\boldsymbol{R}$ and $\boldsymbol{Q}=\tau\boldsymbol{R}$, then, for example, $(1-\tau^2+\tau^4-\tau^6+\tau^8)\boldsymbol{P}=-\mu(1+\tau^{10})\boldsymbol{Q}$, and the second expression requires less group operations than the first one even if we consider the cost of computing $\boldsymbol{Q}$.

In [2] there are three different types of sums like the one we have just seen, each of arbitrary length. For example, the first such family is of the following form

$$\left(\sum_{j=0}^{k-1}(-1)^j\tau^{2j}\right)\boldsymbol{P}=-\mu(1+(-1)^{k-1}\tau^{2k})\boldsymbol{Q}\ .$$

Their algorithm takes an input τ-NAF $\mathbf{s}$. The output is a pair of τ-adic expressions $\mathbf{s}^{(1)}$ and $\mathbf{s}^{(2)}$ with the property that

$$\begin{aligned}\mathbf{s}\boldsymbol{P}=\mathbf{s}^{(1)}\boldsymbol{P}+\mathbf{s}^{(2)}\boldsymbol{Q}&=((-\mu)(1+\tau^2)\mathbf{s}^{(1)}+\mathbf{s}^{(2)})\boldsymbol{Q}\\&=((\mu-\tau)\mathbf{s}^{(1)}+\mathbf{s}^{(2)})\boldsymbol{Q}=(\bar\tau\mathbf{s}^{(1)}+\mathbf{s}^{(2)})\boldsymbol{Q}\ ,\end{aligned} \tag{4}$$

where $\bar\tau$ denotes the *complex conjugate* of τ. Because of this, we call the expression $\left(\begin{smallmatrix}\mathbf{s}^{(1)}\\ \mathbf{s}^{(2)}\end{smallmatrix}\right)$ a $(\bar\tau,1)$-*double expansion*. Note that $\bar\tau$, being an element of $\mathbb{Z}[\tau]$,

operates on the points of the curve, and it is natural to ask what it does: It corresponds to the operation, which we may also denote by $\bar{\tau}$, such that $\tau(\bar{\tau}P) = \bar{\tau}(\tau P) = 2P$.

The *Hamming weight* of a double expansion $\left(\begin{smallmatrix}\mathbf{s}^{(1)}\\ \mathbf{s}^{(2)}\end{smallmatrix}\right)$ is defined to be the sum of the Hamming weights of $\mathbf{s}^{(1)}$ and $\mathbf{s}^{(2)}$. The input is scanned from right to left and whenever one of the above blocks is found in $\mathbf{s}$, then it is removed from $\mathbf{s}$ and the corresponding equivalent expression for $\boldsymbol{Q}$ placed in $\mathbf{s}^{(2)}$. At the end, what "remains" of $\mathbf{s}$ constitutes $\mathbf{s}^{(1)}$. We call the resulting expansion $\left(\begin{smallmatrix}\mathbf{s}^{(1)}\\ \mathbf{s}^{(2)}\end{smallmatrix}\right)$ the *ACS expansion*, from the initials of its inventors. The ACS expansion has an average density of 2/7.

The method can be interleaved with the τ-NAF recoding, because the latter also operated from right to left, and can be performed twice to generate $\mathbf{s}^{(1)}$ and $\mathbf{s}^{(2)}$ independently without having to store them. A variant of the τ-and-add method is proposed that uses $\mathbf{s}^{(1)}$ and $\mathbf{s}^{(2)}$ independently to perform the scalar multiplication without precomputing $\boldsymbol{Q}$ or storing intermediate representations. We present it as Algorithm 1 in a simpler "non interleaved" form for ease of reading, but also because we shall use it with a different recoding in Subsection 4.2. Note that it uses the inverse Frobenius operation τ^{-1} in place of τ, which is fine because it is an operation of negligible cost (like squaring) in the cases when a normal basis for $\mathbb{F}_{2^n}$ is used [1], and still very fast if the field extension is defined by a trinomial [3, § 3.2].

Note that the values ℓ_i are NOT needed in advance. In fact, the recoding of $\mathbf{s}$ into $\mathbf{s}^{(2)}$ first and $\mathbf{s}^{(1)}$ later can be done without knowing ℓ_i in advance: the results in [14] guarantee that the length of $\mathbf{s}$ will be $\approx n$, those in [2] that $\ell_i \approx n$ and the value will be known at the end of the recoding (and of the corresponding τ-and-add loop) so that they can be used immediately after that.

For the other cases, a right-to-left method based on τ is proposed. In this case the recoding must be stored first.

3 Optimality of the τ-NAF

Let τ-NAF($\mathbf{s}$) denote the τ-NAF of $\mathsf{val}_\tau(\mathbf{s})$. For any τ-expansion $\mathbf{s}$ with any (rational) integer digits, define its *cost function* as $c(\mathbf{s}) := \sum_{j\geq 0} |s_j|$. As in the case of the binary nonadjacent form introduced by Reitwiesner [9], the τ-NAF minimizes the Hamming weight. In fact, we prove the following stronger result.

Theorem 1. *Let $z \in \mathbb{Z}[\tau]$. The cost of the τ-NAF of z is minimum over all τ-expansions of z. In particular, the τ-NAF has minimal Hamming weight among all expansions with digits $\{0, \pm 1\}$.*

Proof. We prove this claim by induction on $c(\mathbf{s})$.

Without loss of generality, we may assume that $s_0 > 0$. We choose $k \in \mathbb{Z}$ such that $1 \leq s_0 - 2k \leq 2$. We have

$$\mathsf{val}_\tau(\ldots, s_3, s_2, s_1, s_0) = \mathsf{val}_\tau(\ldots, s_3, s_2 - k, s_1 + \mu k, s_0 - 2k) =: \mathsf{val}_\tau(\mathbf{s}').$$

Algorithm 1. Scalar multiplication algorithm from [2]

INPUT: A Koblitz curve E_a with corresponding parameter $\mu = (-1)^{1-a}$, a point $\boldsymbol{P}$ of odd order on E_a and a joint expansion $\binom{\mathbf{s}^{(1)}}{\mathbf{s}^{(2)}}$ of length approximately n
OUTPUT: $\mathbf{s}^{(1)}\boldsymbol{P} + \mathbf{s}^{(2)}\boldsymbol{Q}$

1. ℓ_i will contain the length of $\mathbf{s}^{(i)}$
2. $\boldsymbol{X} \leftarrow s_0^{(2)}\boldsymbol{P}$
3. **for** $j = 1$ **to** $\ell_2 - 1$ **do**
4. $\quad \boldsymbol{X} \leftarrow \tau^{-1}\boldsymbol{X}$, and $\boldsymbol{X} \leftarrow \boldsymbol{X} + s_j^{(2)}\boldsymbol{P}$ $\quad \left[\text{Now } \boldsymbol{X} = \tau^{-\ell_2+1}\mathbf{s}^{(2)}\boldsymbol{P}\right]$
5. $\boldsymbol{X} \leftarrow \tau^{\ell_2-n}\boldsymbol{X}$, $\boldsymbol{X} \leftarrow \frac{1}{2}\boldsymbol{X}$ $\quad \left[\text{Now } \boldsymbol{X} = \mathbf{s}^{(2)}\tau\left(\frac{1}{2}\boldsymbol{P}\right)\right]$
6. $\boldsymbol{X} \leftarrow \boldsymbol{X} + s_0^{(1)}\boldsymbol{P}$
7. **for** $j = 1$ **to** $\ell_1 - 1$ **do**
8. $\quad \boldsymbol{X} \leftarrow \tau^{-1}\boldsymbol{X}$, and $\boldsymbol{X} \leftarrow \boldsymbol{X} + s_j^{(1)}\boldsymbol{P}$
$\qquad \left[\text{Now } X = \tau^{-\ell_1+1}\left(\mathbf{s}^{(1)}\boldsymbol{P} + \mathbf{s}^{(2)}\tau\left(\frac{1}{2}\boldsymbol{P}\right)\right) = \tau^{-\ell_1+1}s\boldsymbol{P}\right]$
9. $\boldsymbol{X} \leftarrow \tau^{\ell_1-1-n}\boldsymbol{X}$
10. **return** $(\boldsymbol{X})$

Of course, $c(\mathbf{s}') = c(\mathbf{s}) + |s_2 - k| - |s_2| + |s_1 + \mu k| - |s_1| + (s_0 - 2k) - s_0 \le c(\mathbf{s})$. Since $c(\ldots, s_3', s_2', s_1') < c(\mathbf{s}') \le c(\mathbf{s})$, we may replace this expansion by its τ-NAF by induction hypothesis without increasing its cost c. We conclude that $\mathsf{val}_\tau(\mathbf{s}) = \mathsf{val}_\tau(\mathbf{s}'')$ for some $\mathbf{s}''$ such that $s_0'' \in \{1, 2\}$, $(\ldots, s_3'', s_2'', s_1'')$ is in τ-NAF and $c(\mathbf{s}'') \le c(\mathbf{s})$.

We note that for arbitrary t_3, t_4, we have

$$\mathsf{val}_\tau(1, 0, 2) = \mathsf{val}_\tau(0, \mu, 0), \tag{5a}$$
$$\mathsf{val}_\tau(0, -\mu, 2) = \mathsf{val}_\tau(-1, 0, 0), \tag{5b}$$
$$\mathsf{val}_\tau(t_3, 0, \mu, 2) = \mathsf{val}_\tau(-\mu + t_3, 0, 0, 0) \tag{5c}$$

(note that the cost c of the left hand side is always larger than that of the right hand side) and

$$\mathsf{val}_\tau(t_3, 0, 0, 2) = \mathsf{val}_\tau(-\mu + t_3, 0, -\mu, 0), \tag{6a}$$
$$\mathsf{val}_\tau(t_4, 0, -1, 0, 2) = \mathsf{val}_\tau(1 + t_4, -\mu, 0, \mu, 0), \tag{6b}$$
$$\mathsf{val}_\tau(t_3, 0, \mu, 1) = \mathsf{val}_\tau(-\mu + t_3, 0, 0, -1), \tag{6c}$$
$$\mathsf{val}_\tau(0, -\mu, 1) = \mathsf{val}_\tau(-1, 0, -1). \tag{6d}$$

In the last four equalities, the cost c of the left hand side is not smaller than that of the right hand side and the last three or two digits of the right hand side are

already in nonadjacent form. We consider the equivalences (5) and (6) as replacement rules: "replace an occurrence of the left hand side by the corresponding right hand side". Applying these rules on $\mathbf{s}''$ and then using the induction hypothesis for the resulting expansion (in the case of the rules in (5)) or on the left part of the resulting expansion (i.e., excluding the last two or three digits) in the case of the rules in (6), our claim is proved.

4 The Wide-Double-NAF

4.1 Definition and Uniqueness

We consider $(\bar{\tau}, 1)$-*double expansions* $\binom{\mathbf{s}^{(1)}}{\mathbf{s}^{(2)}}$, where $\mathbf{s}^{(1)}$ and $\mathbf{s}^{(2)}$ are just any τ-expansions of arbitrary elements of $\mathbb{Z}[\tau]$. We say that two such expansions $\binom{\mathbf{s}^{(1)}}{\mathbf{s}^{(2)}}$ and $\binom{\mathbf{s}'^{(1)}}{\mathbf{s}'^{(2)}}$ are *equivalent* if $\bar{\tau}\,\mathsf{val}_\tau(\mathbf{s}^{(1)}) + \mathsf{val}_\tau(\mathbf{s}^{(2)}) = \bar{\tau}\,\mathsf{val}_\tau(\mathbf{s}'^{(1)}) + \mathsf{val}_\tau(\mathbf{s}'^{(2)})$; in this case we write $\binom{\mathbf{s}^{(1)}}{\mathbf{s}^{(2)}} \equiv \binom{\mathbf{s}'^{(1)}}{\mathbf{s}'^{(2)}}$.

If we have a point $\boldsymbol{P} \in E_a(\mathbb{F}_{2^n})$ and set $\boldsymbol{Q} = \tau(\frac{1}{2}\boldsymbol{P})$, the relation $\binom{\mathbf{s}^{(1)}}{\mathbf{s}^{(2)}} \equiv \binom{\mathbf{s}'^{(1)}}{\mathbf{s}'^{(2)}}$ implies that $\mathsf{val}_\tau(\mathbf{s}^{(1)})\boldsymbol{P} + \mathsf{val}_\tau(\mathbf{s}^{(2)})\boldsymbol{Q} = \mathsf{val}_\tau(\mathbf{s}'^{(1)})\boldsymbol{P} + \mathsf{val}_\tau(\mathbf{s}'^{(2)})\boldsymbol{Q}$.

The Hamming weight of a double expansion $\binom{\mathbf{s}^{(1)}}{\mathbf{s}^{(2)}}$ is defined to be the sum of the Hamming weights of $\mathbf{s}^{(1)}$ and $\mathbf{s}^{(2)}$.

Let now $\mathbf{s}$ be the τ-NAF of an $m \in \mathbb{Z}$. We will construct a double expansion $\binom{\mathbf{s}^{(1)}}{\mathbf{s}^{(2)}}$ such that $\binom{\mathbf{s}}{\mathbf{0}} \equiv \binom{\mathbf{s}^{(1)}}{\mathbf{s}^{(2)}}$ and with minimal Hamming weight.

Definition 1. *A double expansion* $\binom{\mathbf{s}^{(1)}}{\mathbf{s}^{(2)}}$ *is called a* wide-double-NAF*, if* $s_j^{(i)} = \pm 1$ *implies that* $s_{j+2} = s_{j+1} = \binom{0}{0}$ *and* $s_j^{(i')} = 0$*, where* $i' = 3 - i$ *and* $j \geq 0$*.*

This means that in the language of regular expressions, a wide-double-NAF can be written as

$$\left(\varepsilon + \begin{smallmatrix}1\\0\end{smallmatrix} + \begin{smallmatrix}\bar{1}\\0\end{smallmatrix} + \begin{smallmatrix}0\\1\end{smallmatrix} + \begin{smallmatrix}0\\\bar{1}\end{smallmatrix} + \begin{smallmatrix}0\,1\\0\,0\end{smallmatrix} + \begin{smallmatrix}0\,\bar{1}\\0\,0\end{smallmatrix} + \begin{smallmatrix}0\,0\\0\,1\end{smallmatrix} + \begin{smallmatrix}0\,0\\0\,\bar{1}\end{smallmatrix}\right)\left(\begin{smallmatrix}0\\0\end{smallmatrix} + \begin{smallmatrix}0\,0\,1\\0\,0\,0\end{smallmatrix} + \begin{smallmatrix}0\,0\,\bar{1}\\0\,0\,0\end{smallmatrix} + \begin{smallmatrix}0\,0\,0\\0\,0\,1\end{smallmatrix} + \begin{smallmatrix}0\,0\,0\\0\,0\,\bar{1}\end{smallmatrix}\right)^*. \tag{7}$$

where as customary $\bar{1}$ denotes -1 (here, and in what follows the bar over 1 and μ will denote negation and not complex conjugation).

We first prove a uniqueness result.

Theorem 2. *If* $\mathbf{s}$ *and* $\mathbf{s}'$ *are* equivalent *wide-double-NAFs, then they are equal.*

The proof relies on the following extension of Solinas' [14] Lemma 28, that he used to prove the uniqueness of the τ-NAF.

Lemma 1. *Consider* $z = \sum_{j\geq 0} s_j \tau^j \in \mathbb{Z}[\tau]$*. Then*

(i) z *is divisible by* τ *in* $\mathbb{Z}[\tau]$ *if and only if* $s_0 \equiv 0 \pmod 2$*,*
(ii) z *is divisible by* τ^2 *in* $\mathbb{Z}[\tau]$ *if and only if* $s_0 + 2s_1 \equiv 0 \pmod 4$*,*
(iii) z *is divisible by* τ^3 *in* $\mathbb{Z}[\tau]$ *if and only if* $s_0 - 2\mu s_1 - 4s_2 \equiv 0 \pmod 8$*.*

The first two assertions of Lemma 1 are in Solinas' paper, the proof of the third one is straightforward and we omit it.

Proof of Theorem 2. Let $\binom{\mathbf{s}^{(1)}}{\mathbf{s}^{(2)}} \equiv \binom{\mathbf{s}'^{(1)}}{\mathbf{s}'^{(2)}}$ be two wide-double-NAFs. Without loss of generality, we may assume that $\binom{s_0^{(1)}}{s_0^{(2)}} \neq \binom{s'^{(1)}_0}{s'^{(2)}_0}$ and that $s_0^{(i)} = 1$ for some $i \in \{1, 2\}$, which implies $s_0^{(i')} = 0$ for $i' = 3 - i$ by definition of a wide-double-NAF. By (4), we have

$$\sum_{j\geq 0} \left(s_j^{(1)} - s'^{(1)}_j\right)(-\mu)(1+\tau^2)\tau^j + \sum_{j\geq 0} \left(s_j^{(2)} - s'^{(2)}_j\right)\tau^j = 0 \ . \tag{8}$$

From Lemma 1(i) we conclude that $\left(s_0^{(1)} - s'^{(1)}_0\right)(-\mu) + \left(s_0^{(2)} - s'^{(2)}_0\right) \equiv 0 \pmod 2$. Since $s_0^{(i)} = 1$ and $s_0^{(i')} = 0$, we conclude that $\binom{s'^{(1)}_0}{s'^{(2)}_0} \neq \binom{0}{0}$. This implies that $s_j^{(k)} = s'^{(k)}_j = 0$ for $1 \leq j, k \leq 2$. We set $c = -\mu(s_0^{(1)} - s'^{(1)}_0)$ and $d = (s_0^{(2)} - s'^{(2)}_0)$. From (8) we conclude that $c(1+\tau^2) + d$ is divisible by τ^3. Hence by Lemma 1

$$0 \equiv (c + d) - 4c \equiv d - 3c \pmod 8 \tag{9}$$

but, by assumption, $(c, d) \neq (0, 0)$ and $|c| + |d| = 2$. This contradicts (9).

4.2 Existence and Use

One important property of the ACS expansion $\binom{\mathbf{s}^{(1)}}{\mathbf{s}^{(2)}}$ is that for each column at most one digit is not vanishing. The same property is, by definition, satisfied by the wide-double-NAF. Any $(\bar{\tau}, 1)$-double expansion with this property can be easily viewed, by virtue of (4), as a recoding of the number $\frac{2}{\tau}z$ to the base τ with digit set $\{0, \pm 1, \pm\bar{\tau}\}$. This allows us to make a very simple computation of the wide-double-NAF.

We have the following result:

Lemma 2. *Consider $z = s_0 + s_1\tau \in \mathbb{Z}[\tau]$. Then $\frac{2}{\tau}z = (\mu s_0 + 2s_1) - s_0\tau$. Also*

(i) $z \equiv 1 \pmod{\tau^3}$ *if and only if* $s_0 - 2\mu s_1 \equiv 1 \pmod 8$,
(ii) $z \equiv -1 \pmod{\tau^3}$ *if and only if* $s_0 - 2\mu s_1 \equiv -1 \pmod 8$,
(iii) $z \equiv \bar{\tau} \pmod{\tau^3}$ *if and only if* $s_0 - 2\mu s_1 \equiv 3\mu \pmod 8$,
(iv) $z \equiv -\bar{\tau} \pmod{\tau^3}$ *if and only if* $s_0 - 2\mu s_1 \equiv -3\mu \pmod 8$.

Sketch of the proof. The first assertion is easily verified. For (i) we have $s_0 + s_1\tau \equiv 1 \pmod{\tau^3}$ if and only if τ^3 divides $(s_0 - 1) + s_1\tau$, and at this point Lemma 1 can be applied. To prove (iii) and (iv) it is better to work with $\mu - \tau$ in place of $\bar{\tau}$.

Note that there are *eight* congruence classes modulo τ^3 (cfr. [14]) of which four correspond to elements that are divisible by τ (see Lemma 1 above). It is now clear how we can produce the wide-double-NAF of any element of $\mathbb{Z}[\tau]$: Algorithm 2 serves the purpose thereby giving also an existence proof.

The correctness is easy to prove (it is an almost immediate consequence of Lemmas 1 and 2 and of the definition of wide-double-NAF). The termination

Algorithm 2. Wide-double-NAF recoding

INPUT: An integer $s_0 + s_1\tau \in \mathbb{Z}[\tau]$

OUTPUT: Its wide-double-NAF $\left(\begin{smallmatrix}\mathbf{s}^{(1)}\\ \mathbf{s}^{(2)}\end{smallmatrix}\right)$

$(s_0, s_1) \leftarrow (\mu s_0 + 2s_1, -s_0)$ [Multiply by $\frac{2}{\tau}$]
while $((s_0, s_1)$!= $(0, 0))$ **do**
  if $s_0 \equiv 0 \pmod 2$
    output $\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$
    $(s_0, s_1) \leftarrow (\mu\frac{s_0}{2} + s_1, -\frac{s_0}{2})$ [Divide by τ]
  else
    switch $(s_0 - 2\mu s_1 \pmod 8)$
      case 1 : $s_0 \leftarrow s_0 - 1$, **output** $\left(\begin{smallmatrix}0&0&0\\0&0&1\end{smallmatrix}\right)$
      case -1 : $s_0 \leftarrow s_0 + 1$, **output** $\left(\begin{smallmatrix}0&0&0\\0&0&\bar{1}\end{smallmatrix}\right)$
      case 3μ : $s_0 \leftarrow s_0 - \mu$, $s_1 \leftarrow s_1 + 1$, **output** $\left(\begin{smallmatrix}0&0&1\\0&0&0\end{smallmatrix}\right)$
      case -3μ : $s_0 \leftarrow s_0 + \mu$, $s_1 \leftarrow s_1 - 1$, **output** $\left(\begin{smallmatrix}0&0&\bar{1}\\0&0&0\end{smallmatrix}\right)$
    $(s_0, s_1) \leftarrow \left(\frac{1}{8}(-3\mu s_0 - 2s_1), \frac{1}{8}(s_0 - 2\mu s_1)\right)$ [Divide by τ^3]

proof follows the same arguments as in Solinas' paper [14] and in fact the length has a similar bound than that for the τ-NAF. The recoding is easy to implement and can be used with Algorithm 1, because all remarks following Algorithm 1 apply also here.

It must be noted that, in order for this recoding to be used efficiently, the quantity $s_0 + s_1\tau$ should be reduced modulo $\tau^n - 1$ as for the NAF and the width-w τ-NAF, as explained in [14] (even partial reduction is fine).

4.3 Optimality

In this section we prove that the wide-double-NAF minimizes the Hamming weight in its equivalence class. This provides also a second, but more complicated, construction of the form.

Theorem 3. *Let* $\mathbf{s}$ *be a* $(\bar{\tau}, 1)$*-double expansion. Then there exists a wide-double-NAF that is equivalent to* $\mathbf{s}$*. Its Hamming weight is not larger than that of* $\mathbf{s}$*.*

Proof. We allow arbitrary (rational) integer digits in $\mathbf{s}$ and prove the theorem by induction on

$$c(\mathbf{s}) := \sum_{j\geq 0} \left(\left|s_j^{(1)}\right| + \left|s_j^{(2)}\right|\right) \ .$$

By the proof of Theorem 1, we may replace $(s_j^{(i)})_{j\geq 0}$ by its τ-NAF $(s'^{(i)}_j)_{j\geq 0}$ for $i \in \{1, 2\}$ without increasing the costs c. Of course, we have $\mathbf{s} \equiv \mathbf{s}'$.

We easily check that for all $t_j^{(i)}$, we have

$$\begin{pmatrix} t_2^{(1)} & 0 & 1 \\ t_2^{(2)} & 0 & \mu \end{pmatrix} \equiv \begin{pmatrix} t_2^{(1)} & 0 & 0 \\ (\bar{\mu}+t_2^{(2)}) & 0 & 0 \end{pmatrix} , \qquad \begin{pmatrix} 0 & \bar{1} & 0 \\ t_2^{(2)} & 0 & 1 \end{pmatrix} \equiv \begin{pmatrix} 0 & 0 & 0 \\ t_2^{(2)} & 0 & \bar{1} \end{pmatrix} ,$$

$$\begin{pmatrix} 0 & 1 \\ 0 & \bar{\mu} \end{pmatrix} \equiv \begin{pmatrix} 0 & 0 \\ \bar{1} & 0 \end{pmatrix} , \qquad \begin{pmatrix} t_2^{(1)} & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \equiv \begin{pmatrix} t_2^{(1)} & 0 & 0 \\ 0 & 0 & \mu \end{pmatrix} ,$$

$$\begin{pmatrix} 0 & t_1^{(1)} & 0 \\ 1 & 0 & 1 \end{pmatrix} \equiv \begin{pmatrix} 0 & t_1^{(1)} & \bar{\mu} \\ 0 & 0 & 0 \end{pmatrix} , \qquad \begin{pmatrix} 0 & 0 & 1 \\ \mu & 0 & 0 \end{pmatrix} \equiv \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \bar{\mu} \end{pmatrix} , \tag{10}$$

$$\begin{pmatrix} t_5^{(1)} & t_4^{(1)} & t_3^{(1)} & 0 & 1 & 0 \\ t_5^{(2)} & t_4^{(2)} & 0 & \bar{1} & 0 & 1 \end{pmatrix} \equiv \begin{pmatrix} t_5^{(1)} & t_4^{(1)} & t_3^{(1)} & 0 & 0 & 0 \\ (\mu+t_5^{(2)}) & t_4^{(2)} & 0 & 0 & 0 & \bar{1} \end{pmatrix} .$$

We note that in all the above equivalences, the costs c decrease from the left hand side to the right hand side (even if, occasionally, digits with absolute value 2 may appear on the r.h.s). This means that if we find one of the left hand sides (or its negatives, of course) as subblocks in our double expansion $\mathbf{s}'$, we can replace this subblock by the corresponding right hand side and use the induction hypothesis to convert the resulting expansion to a wide-double-NAF not increasing the costs.

So we may assume that the left hand sides of (10) do not occur (at least in the rightmost digits). Furthermore, we have

$$\begin{pmatrix} t_4^{(1)} & t_3^{(1)} & 0 & 0 & 1 \\ t_4^{(2)} & t_3^{(2)} & 0 & \bar{1} & 0 \end{pmatrix} \equiv \begin{pmatrix} t_4^{(1)} & t_3^{(1)} & 0 & 0 & \bar{1} \\ (\mu+t_4^{(2)}) & t_3^{(2)} & 0 & 0 & 0 \end{pmatrix} , \qquad \begin{pmatrix} 0 & 1 & 0 & 1 \\ t_3^{(2)} & 0 & 0 & 0 \end{pmatrix} \equiv \begin{pmatrix} 0 & 0 & 0 & 0 \\ (\bar{1}+t_3^{(2)}) & 0 & 0 & \bar{\mu} \end{pmatrix} ,$$

$$\begin{pmatrix} t_3^{(1)} & 0 & 1 & 0 \\ t_3^{(2)} & 0 & 0 & 1 \end{pmatrix} \equiv \begin{pmatrix} t_3^{(1)} & 0 & 0 & \mu \\ (\bar{\mu}+t_3^{(2)}) & 0 & 0 & 0 \end{pmatrix} , \qquad \begin{pmatrix} t_3^{(1)} & 0 & 0 & 0 \\ 0 & \bar{1} & 0 & 1 \end{pmatrix} \equiv \begin{pmatrix} (\bar{1}+t_3^{(1)}) & 0 & 0 & \bar{\mu} \\ 0 & 0 & 0 & 0 \end{pmatrix} ,$$

$$\begin{pmatrix} t_3^{(1)} & 0 & 0 & 1 \\ 0 & \bar{\mu} & 0 & 0 \end{pmatrix} \equiv \begin{pmatrix} (\bar{\mu}+t_3^{(1)}) & 0 & 0 & 0 \\ 0 & 0 & 0 & \bar{\mu} \end{pmatrix} , \qquad \begin{pmatrix} 0 & \mu & 0 & 0 \\ t_3^{(2)} & 0 & 0 & 1 \end{pmatrix} \equiv \begin{pmatrix} 0 & 0 & 0 & \bar{\mu} \\ (\bar{\mu}+t_3^{(2)}) & 0 & 0 & 0 \end{pmatrix} , \tag{11}$$

$$\begin{pmatrix} t_4^{(1)} & 0 & \bar{1} & 0 & 1 \\ t_4^{(2)} & t_3^{(2)} & 0 & 0 & 0 \end{pmatrix} \equiv \begin{pmatrix} t_4^{(1)} & 0 & 0 & 0 & 0 \\ (\mu+t_4^{(2)}) & t_3^{(2)} & 0 & 0 & \bar{\mu} \end{pmatrix} , \qquad \begin{pmatrix} t_4^{(1)} & 0 & \bar{\mu} & 0 & 0 \\ t_4^{(2)} & t_3^{(2)} & 0 & 0 & 1 \end{pmatrix} \equiv \begin{pmatrix} t_4^{(1)} & 0 & 0 & 0 & \bar{\mu} \\ (1+t_4^{(2)}) & t_3^{(2)} & 0 & 0 & 0 \end{pmatrix} ,$$

$$\begin{pmatrix} t_6^{(1)} & t_5^{(1)} & t_4^{(1)} & 0 & \bar{1} & 0 & 1 \\ t_6^{(2)} & t_5^{(2)} & t_4^{(2)} & t_3^{(2)} & 0 & \bar{1} & 0 \end{pmatrix} \equiv \begin{pmatrix} t_6^{(1)} & t_5^{(1)} & t_4^{(1)} & 0 & 0 & 0 \\ (\bar{\mu}+t_6^{(2)}) & t_5^{(2)} & t_4^{(2)} & (\bar{1}+t_3^{(2)}) & 0 & 0 & \mu \end{pmatrix} .$$

Note that for every $\mathbf{s}'$ found above, the least significant columns of $\mathbf{s}'$ are found in the l.h.s. of exactly one of the equivalences (11). Thus we can replace them with the corresponding block in the r.h.s. to obtain a new expansion $\mathbf{s}''$. In each of these equivalences, the costs do not increase from left to right and the last three digits of the right hand side always form a block that is allowed in a wide-double-NAF. In particular $\mathbf{s}''$ has cost not larger than $\mathbf{s}'$ (and thus not larger than the cost of $\mathbf{s}$). Hence we can apply the induction hypothesis to $\mathbf{s}''$ with the last three digits removed. This proves the Theorem. (Note that after applying one of the replacements (11), patterns of the l.h.s.'s of (10) may appear again and these should be replaced with the corresponding r.h.s.'s too, should one desire to formulate a constructive version of this theorem.)

4.4 An Example

Let us consider the rational integer 195. If $a = 1$ in (1), then the τ-NAF of 195 is $\tau^{16} + \tau^{14} + \tau^{10} + \tau^7 - \tau^5 + \tau^2 - 1$ or

$$\mathsf{val}_\tau(10100010010\bar{1}00010\bar{1}) = 195 \ .$$

The weight is 7 and the ACS recoding has also weight 7 (in fact, no subsequence of the given τ-NAF can be simplified by the ACS method, hence the output is identical with the input).

However, Algorithm 2 gives the following wide-double-NAF

$$\begin{pmatrix} 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \bar{1}\ 0\ 0\ 0\ 0 \\ 0\ 0\ \bar{1}\ 0\ 0\ \bar{1}\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1 \end{pmatrix} \ ,$$

that has weight 5.

4.5 Analysis

We now analyze the wide-double-NAF using methods from average case analysis of algorithms, cf. for instance [12].

To calculate the asymptotic density of the wide-double-NAF it is sufficient to sum up the Hamming weights of all wide-double-NAF's of a certain length N and divide it by the number of all wide-double-NAF's of the same length. A more detailed analysis based on the τ-NAF or the unique expansion with digits $\{0, 1\}$ for the input statistics is beyond the scope of this paper, but the main term would agree.

We define $a_{M,N}$ to be the number of wide-double-NAF's of length N and Hamming weight M and consider the generating function

$$G(Y, Z) := \sum_{N \geq 0} \sum_{M \geq 0} a_{M,N} Y^M Z^N \ .$$

This function can be derived from our regular expression in (7) by labelling contributions to the Hamming weight by Y and to the length by Z and by transforming $(\cdots)^*$ to $(1 - (\cdots))^{-1}$. Thus we get

$$G(Y, Z) = \frac{1 + 4YZ + 4YZ^2}{1 - (Z + 4YZ^3)} \ .$$

Obviously, the number W_N of wide-double-NAF's of length N equals the coefficient of Z^N in

$$G(1, Z) = \frac{1 + 4Z + 4Z^2}{1 - Z - 4Z^3} = \frac{2}{1 - 2\,Z} - \frac{1}{1 + Z + 2\,Z^2} \ .$$

We obtain

$$W_N = 2^{N+1} + O(2^{N/2}) \ .$$

We differentiate $G(Y,Z)$ with respect to Y and set $Y=1$,

$$\left.\frac{\partial G(Y,Z)}{\partial Y}\right|_{Y=1} = \frac{1}{2(1-2Z)^2} + \frac{1}{4(1-2Z)} + \frac{-3-Z}{4(1+Z+2Z^2)^2} + \frac{Z}{4(1+Z+2Z^2)} \ ,$$

and extract the coefficient of Z^N to obtain the sum H_N of the Hamming weights of all wide-double-NAF's of length N as

$$H_N = \left(N+\frac{3}{2}\right)\cdot 2^{N-1} + O(N2^{N/2}) \ .$$

Dividing H_N by W_N, we proved

Theorem 4. *The expected Hamming weight of a wide-double-NAF of length N equals*

$$\frac{1}{4}N + \frac{3}{8} + O\left(\frac{N}{2^{N/2}}\right) \ .$$

Hence the wide-double-NAF achieves an average improvement of 25 % over the τ-and-add algorithm.

5 Final Remarks and Conclusions

In this paper we consider a few problems related to τ-adic expansions associated to Koblitz curves.

The first problem is the optimality of Solinas' τ-NAF. We give the first proof that the τ-NAF has minimal weight among all the τ-adic expansions with digit set $\{0,\pm 1\}$. In fact we prove a stronger result, namely that the τ-NAF has cost (sum of the absolute values of the digits) minimal among the costs of all τ-adic representations with arbitrary rational integer digits (Theorem 1).

Then we consider a result presented at PKC 2004. There, Avanzi, Ciet, and Sica [2] showed that one could perform a scalar multiplication on Koblitz curves by "inserting" a point halving in a τ-and-add scalar multiplications, thereby reducing the number of group additions required. They attained, under suitable conditions, a reduction of 14.3% of the number of group additions with respect to the plain τ-and-add method on average without increasing memory requirements. The method is thus just a faster drop-in replacement for the τ-and-add method. We improve on this result under the same assumptions they made, bringing the reduction to 25% on average. The corresponding expansion of the scalar is the *wide-double-NAF*: we construct it (cf. Subsection 4.2), we prove its uniqueness (Theorem 2) and a suitable minimality property (Theorem 3), and we carefully analyse its expected Hamming weight (Theorem 4).

A speed-up is achieved using the new recoding together with the scalar multiplication algorithm from [2] (cf. Algorithm 1). Due to the increased number of Frobenius applications, the speed-up in a real world implementations may be smaller

than 25%: in [2] an effective speedup of 12.5% was found on standard Koblitz curves over the field $\mathbb{F}_{2^{233}}$ using normal bases. By repeating the computations done in [2, § 4.2] using the new density $1/4$ in place of $2/7$ we expect our method to bring at least an improvement of 23% on average under the same conditions. Like the method in [2] it is a drop-in replacement for the τ-and-add method.

We also note that the decrease of group operations from $2/7n$ to $1/4n$ represents a reduction of 12.5%, i.e. our method can perform on average 12.5% faster than the previous combination of the Frobenius with the halving.

Acknowledgements. The authors wish to express their gratitude to the anonymous reviewers for their remarks and suggestions.

References

1. D. W. Ash, I. F. Blake and S. Vanstone. *Low complexity normal bases.* Discrete Applied Math. **25**, pp. 191–210, 1989.
2. R. M. Avanzi, M. Ciet, and F. Sica. *Faster Scalar Multiplication on Koblitz Curves combining Point Halving with the Frobenius Endomorphism.* Proceedings of PKC 2004, LNCS 2947, 28–40. Springer, 2004.
3. K. Fong, D. Hankerson, J. Lopez and A. Menezes. *Field inversion and point halving revisited.* IEEE Trans. on Computers **53** (8), pp. 1047–1059. August 2004.
4. D. Hankerson, J. Lopez-Hernandez, and A. Menezes. *Software Implementatin of Elliptic Curve Cryprography over Binary Fields.* In: *Proceedings of CHES 2000.* LNCS 1965, pp. 1–24. Springer, 2001.
5. E. W. Knudsen. *Elliptic Scalar Multiplication Using Point Halving.* In: *Proocedings of ASIACRYPT 1999*, LNCS 1716, pp. 135–149. Springer, 1999.
6. N. Koblitz. *Elliptic curve cryptosystems.* Mathematics of computation **48**, pp. 203–209, 1987.
7. N. Koblitz. *CM-curves with good cryptographic properties.* In: *Proceedings of CRYPTO 1991*, LNCS 576, pp. 279–287. Springer, 1991.
8. V. S. Miller. *Use of elliptic curves in cryptography.* In: *Proceedings of CRYPTO '85.* LNCS 218, pp. 417–426. Springer, 1986.
9. G. W. Reitwiesner. *Binary arithmetic.* Advances in Computers **1**, pp. 231–308, 1960.
10. R. Schroeppel. *Point halving wins big.* Talks at: (i) Midwest Arithmetical Geometry in Cryptography Workshop, November 17–19, 2000, University of Illinois at Urbana-Champaign; and (ii) ECC 2001 Workshop, October 29–31, 2001, University of Waterloo, Ontario, Canada.
11. R. Schroeppel. *Elliptic curve point ambiguity resolution apparatus and method.* International Application Number PCT/US00/31014, filed 9 November 2000.
12. R. Sedgewick and P. Flajolet. *An Introduction to the Analysis of Algorithms.* Addison-Wesley, 1996.
13. J. A. Solinas. *An improved algorithm for arithmetic on a family of elliptic curves.* In: *Proceedings of CRYPTO 1997*, LNCS 1294, pp. 357–371. Springer, 1997.
14. J. A. Solinas. *Efficient Arithmetic on Koblitz Curves.* Designs, Codes and Cryptography **19** (2/3), pp. 125–179, 2000.
15. IEEE Std 1363-2000. *IEEE Standard Specifications for Public-Key Cryptography.* IEEE Computer Society, August 29, 2000.
16. National Institute of Standards and Technology. *Digital Signature Standard.* FIPS Publication 186-2, February 2000.

SPA Resistant Left-to-Right Integer Recodings

Nicolas Thériault

Department of Combinatorics and Optimization, University of Waterloo
ntheriau@math.uwaterloo.ca

Abstract. We present two left-to-right integer recodings which can be used to perform scalar multiplication with a fixed sequence of operations. These recodings make it possible to have a simple power analysis resistant implementation of a group-based cryptosystem without using unified formulas or introducing dummy operations. This approach is very useful for groups in which the doubling step are less expensive than the addition step, for example with hyperelliptic curves over binary fields or elliptic curves with mixed coordinates.

1 Introduction

Side channel attacks are a constant threat to the implementations of a cryptosystem. This is particularly true for most discrete log based cryptosystems where the basic group operations are often easily distinguishable depending on the nature of their inputs. As a general practice, countermeasures must always be used against simple side channel analysis, even if using one-time keys.

In this paper, we look at the impact of integer recoding for cryptosystems based on the discrete logarithm problem in additive groups. We are particularly interested in groups where the doubling operation is significantly cheaper than the addition. Examples of such groups are hyperelliptic curves over binary fields [22, 23, 10, 14] (where additions are often more than twice as expensive as doublings) or elliptic curves with mixed coordinates [8]. For these groups, the standard countermeasures against SPA attacks are particularly disappointing as they remove most of the saving due to efficient implementations of the group operations. Another particularity of many of these additive groups (and which we take advantage of) is that the addition and the subtraction operations are almost identical.

We introduce the general situation of scalar multiplication for additive groups in Section 2. In Section 3 we describe some of the basic countermeasures to SPA attacks. We then present the most common forms of integer recoding in Section 4 and introduce our recodings in Section 5. Finally, we compare the efficiency of the different recodings in Section 6.

2 Scalar Multiplication

Many discrete log based public key cryptosystems are done on additive groups and required the multiplication of a group element D by a scalar e (the secret

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 345–358, 2006.

key). It is therefore very important to compute $[e]D$ as efficiently and as securely as possible. This is usually done through a variation of the double-and-add algorithm which relies on two basic group operations: Adding two distinct group elements (addition) and adding a group element to itself (doubling).

2.1 Double-and-Add Algorithms

For this paper, we are interested in the left-to-right version of the double-and-add algorithm (i.e. most significant bit first). Right-to-left double-and-add can also be used, but the left-to-right version is often more interesting, in particular when combined with integer recodings and applied to a fixed group element (using precomputations). For right-to-left recodings, one might also consider Yao's algorithm [28], although it does not take advantage of precomputations.

Given a n-bits integer $e = \sum_{j=0}^{n-1} e_j 2^j$ (with the e_j in $\{0,1\}$), let $f_{n-i} = \sum_{j=i}^{n-1} e_j 2^{j-(n-i)}$, i.e. the number formed by the $n-i$ most significant bits of the binary expansion of e. Then f_{n-i} can be obtained from f_{n-i-1} and e_i via the relation $f_{n-i} = 2f_{n-i-1} + e_i$. In terms of scalar multiplication, this becomes: $[f_{n-i}]D = [2]([f_{n-i-1}]D) + [e_i]D$. The left-to-right double-and-add algorithm follows easily from this relation and its general form proceeds as in Algorithm 1. This algorithm is written in its most general form to cover most of the cases encountered in this paper. If we consider only the "classical" double-and-add algorithm on the binary representation, there is no recoding step, no precomputation ($[1]D$ is already known) and the addition step when $r_j \neq 0$ is simply $D_0 + D_1$.

Algorithm 1. Generic double-and-add algorithm

Input: D, e	
Output: $[e]D$	
recode e as $\sum_{j=0}^{m} r_j 2^j$	recoding
precompute $[r]D$ for every digit $r \neq 0$	precomputations
$D_0 \leftarrow [r_m]D$	
for $j = m-1$ down to 0 do	
$D_0 \leftarrow [2]D_0$	doubling
if $r_j \neq 0$ then	
$D_0 \leftarrow D_0 + [r_j]D$	addition
return D_0	

3 Simple Side Channel Analysis Attacks

Power traces [13] and electromagnetic emissions of processors [1] can be used as sources of information for simple side channel analysis (we will refer to both as SPA attacks for simplicity). SPA attacks may exploit even small differences between the addition and the doubling operations on group elements to discover the sequence in which they are used in the double-and-add algorithm. If successful, this gives the binary expansion of e, hence the secret key. It is therefore essential to secure implementations of public key cryptosystems against this type of attack.

As a general rule, countermeasures against simple side channel attacks do not secure the encryption against differential side channel analysis (DPA). If DPA is a potential threat, i.e. if the scalar is used more than once, this problem can be resolved by combining SPA and DPA countermeasure when possible (see [3] for details). On the other hand, DPA countermeasures are useless if the encryption is insecure against simple side channel attacks, so SPA countermeasures should always be used.

3.1 Standard Countermeasures

There are two standard countermeasures against SPA attacks: Dummy operations and unified formulas. Both approach attempt to make the power traces of the two group operations (addition and doubling) indistinguishable.

The first approach consists in adding extra or "dummy" operations in the addition and doubling algorithms where the sequences of operations differ [9]. The result is an addition and a doubling formula which use the same sequence of operations, so they will appear identical to SPA attacks. Obviously this will increase the cost of the group operations (or at least the cheapest of the two), having a negative impact on the efficiency of the encryption algorithm.

Although this is a very simple countermeasure to implement, it is not always safe: If the secret key is used multiple times, dummy operations can be revealed by adaptive fault analysis [29, 30], and further countermeasures are required to prevent this attack.

The second approach consist in rewriting the two group operations into a unified formula. Since both operations will then use the same set of operations, the two operations will have the same power trace.

Unified formulas tend to be more costly to use than dummy operations, but they prevent adaptive fault analysis (but not DPA). The main disadvantage of unified formulas is that they are group specific and so far they have only been developed for elliptic curves [11, 15, 5, 4, 6].

Moreover, some of these formulas have been shown to be weak because some field multiplications are performed twice with the same inputs in the doubling formula but not in the addition formula, making the system potentially vulnerable [27] (the same can sometimes be said of dummy operations [17]).

3.2 Montgomery Ladders

Another countermeasure against SPA is the use of a Montgomery ladder [12]. The algorithm proceeds from left-to-right, computing two elements at each step: $[f_j]D$ and $[f_j + 1]D$, where f_j is the partial sum of the $n - j$ most significant bits of e, i.e. $f_j = \sum_{i=j}^{n} e_i 2^{i-j}$.

Since $f_j = 2f_{j+1} + e_j$, the pair $(f_j, f_j + 1)$ can be obtained from the pair $(f_{j+1}, f_{j+1} + 1)$ (computed at the previous step) using the rules:

e_j	f_j	$f_j + 1$
0	$2f_{j+1}$	$f_{j+1} + (f_{j+1} + 1)$
1	$f_{j+1} + (f_{j+1} + 1)$	$2(f_{j+1} + 1)$

which gives Algorithm 2 for scalar multiplication (where $D_0 = [f_j]D$ and $D_1 = [f_j + 1]D$). Since all the steps use the same set of operations the two group operations do not have to be secured against SPA attacks. As no dummy operations are introduced, the risk posed by adaptive fault analysis is minimal.

Algorithm 2. Montgomery ladder

Input: D, $e = \sum_{i=0}^{n} e_i 2^i$
Output: $[e]D$

$D_0 \leftarrow 0$; $D_1 \leftarrow D$
for $j = n$ down to 0 do
  if $e_j = 0$ then
    $D_1 \leftarrow D_0 + D_1$; $D_0 \leftarrow 2D_0$ $\quad e_j = 0$
  else
    $D_0 \leftarrow D_0 + D_1$; $D_1 \leftarrow 2D_1$ $\quad e_j = 1$
return D_0

One drawback of the Montgomery ladder is the high count of group operations since every step requires one doubling and one addition. Since at any given step the two group operations are independent from each other, it is sometimes possible to offset part of the high operation count by combining the them. For example, with elliptic curves in affine coordinates it is possible to combine the field inversions of the group addition and doubling into one field inversion and three field multiplication. Unfortunately, for most groups used in cryptographic applications this approach is unlikely to give enough savings to justify using a Montgomery ladder instead of other SPA countermeasures.

4 Integer Recoding

A common approach to improve the efficiency of the scalar multiplication is to use integer recoding to reduce the number of operations required in the double-and-add algorithm. By allowing integers other than 0 or 1 to be used in the expansion of e, it becomes possible to *recode* $e = \sum_{i=0}^{n} e_i 2^i$ as $e = \sum_{i=0}^{n'} r_i 2^i$.

The double-and-add algorithm will still work as in Algorithm 1, but the term added may now be different from D. If the *weight* (number of non-zero digits) of the recoding of e is smaller than the weight of its binary expansion, then Algorithm 1 will require fewer additions to compute $[e]D$ (and possibly fewer doublings if $n' < n$). The main difference is that unlike the double-and-add algorithm on the binary representation, the elements $[s]D$ must be precomputed for all the possible digits s.

Most recodings can be divided into two categories depending on the order in which the bits are processed. Right-to-left recodings, i.e. from the least significant bit to the most significant one, are more natural but they must be computed before the double-and-add algorithm and the storage required for the recoding is usually greater than that of the integer itself. Left-to-right recodings are computed from the most significant bits down to the least significant bit, hence the

recoding can be done at the same time as the left-to-right double-and-add multiplication, avoiding the need to record the new representation. This makes left-to-right recodings somewhat more interesting for implementations in restricted environments, especially if the group element is fixed (so the precomputed values $[s]D$ for the digits s can be reused for a number of scalar multiplications).

For a number of groups used in cryptography, and in particular for Elliptic and Jacobians of hyperelliptic curves, recodings can take advantage of symmetries. For these groups, the group subtraction is almost identical to the group addition, up to a few sign changes in the field operations. Since field additions and subtractions are indistinguishable under SPA attacks, the performance and security of the cryptosystem are unaffected if we subtract $[s]D$ instead of adding $[-s]D$, but the storage requirement for the double and add algorithm can be reduced. This makes it very interesting to use digit sets which are symmetric around 0 since only half of the points must be precomputed (for example those corresponding to the positive digits).

4.1 Recodings and SPA Attacks

In general, SPA attacks are much less effective on double-and-add algorithms using integer recodings than those using the binary representation directly. From the power trace of a double-and-add algorithm, it is possible to know which digits in the recoding are non-zero, but not their values.

If the recoding has a *density* (weight of the recoding divided by its length) which is too low or if it contains long sequences of zero digits, the attacker may be able to restrict the portion of the keyspace the secret key could be in. The size of the keyspace to consider may then become small enough for the attacker to find the key using other methods (for example Shanks' Baby-step Giant-step algorithm, Pollard's Rho algorithm, etc). When this is the case, the implementation of the double-and-add algorithm must also include a countermeasure against SPA attacks (see Section 3).

On the other hand, if the weight of the representation is high enough and the non-zero digits are distributed uniformly enough, the recoding is inherently secure and act as a SPA countermeasures. This is the idea behind the fixed recodings in Subsection 4.4 and Section 5.

4.2 w-NAF

The most commonly used recodings are the Non Adjacent Form (NAF) [25] and its extension the w-NAF [7, 26]. For this paper, we will denote the w-NAF as using the digit set $\{\pm 1, \pm 3, \ldots, \pm(2^w - 1)\} \cup \{0\}$ and such that any pairs of non-zero digits are separated by at least w zeros. This is also called the $(w-1)$-NAF and sometimes denoted NAF_{w-1}. The w-NAF recoding is computed from right to left and has average density $1/(w+2)$.

To use the negative digits, we consider sequences of up to w bits and a *carry* c_j (just as in a base 2 addition). To a sequence of w bits starting from the j-th bit of e, we associate the integer $s_j = \sum_{i=0}^{w} e_{i+j} 2^i$. Starting with $j = 0$ and $c_0 = 0$, each step of the recoding follows the rules

$e_j + c_j$	$s_j + c_j$	k	c_{j+k}	r_j	$r_{j+1}, \ldots, r_{j+k-1}$
0	–	1	0	0	–
2	–	1	1	0	–
1	$< 2^w$	$w+1$	0	$s_j + c_j$	0
1	$> 2^w$	$w+1$	1	$s_j + c_j - 2^{w+1}$	0

where the next bit to be encoded is the $(j+k)$-th bit of the binary representation.

Although the w-NAF gives a recoding of the smallest possible weight for the given digit set (see [2]), which is advantageous for the performance of the encryption, the key is weakened by the low density and by the knowledge of the variable positions of the non-zero digits. Since there are 2^w possible values for the non-zero digits and the recodings have an average density of $1/(w+2)$, there are (on average) $2^{wn/(w+2)}$ keys of n bits with a given sequence of doublings and additions. Compared to the 2^n possible keys of length n, we get a reduction by a factor of $2^{2n/(w+2)}$ in the number of possible keys. We can see that unless SPA countermeasures are used, the w-NAF is not intended for applications such as restricted environments which are susceptible of side channel attacks.

4.3 Minimal Weight Left-to-Right Recoding

Avanzi [2] and Muir and Stinson [18] developed left-to-right equivalents of the w-NAF (see also [19]). Although the recodings in [2] and [18] are not always identical, they differ only in some special cases and their outputs always have the same weight. These recodings give the same advantage as the w-NAF, i.e. they give a recoding of minimal weight for the digit set, but with the added bonus that they proceed from the most significant bit downward so they can be interleaved with the left-to-right scalar multiplication.

Let $v_{j,k} = s_{j-k} + e_{j-k} - e_j 2^k$ and let $t_{j,k}$ be the highest power of 2 dividing $v_{j,k}$, then the recoding step in [2] follows the rule:

$e_j - e_{j-1}$	k	$r_j, \ldots, r_{j-k+1}$	r_{j-k+t}
0	1	0	–
± 1	$\min\{w, j+1\}$	0	$v_{j,k}/2^{t_{j,k}}$

where the next recoding step is for the bit $j-k$ (and lower). We refer to [2] and [18] for the proof of correctness of the recoding process.

As was the case with the w-NAF, the group operations will also have to be secured against SPA attacks.

4.4 Fixed Right-to-Left Recoding

In [17], Möller introduced a new fixed right-to-left recoding. The idea consists in computing a 2^w-ary expansion of e, but in such a way that none of the digits are 0 (hence producing a "regular" or "fixed" expansion). The recoding we present here is from the extended version of [17].

Since a 2^w-ary recoding requires a set of at least 2^w digits to be able to represent every possible integer, the digit 0 is replaced by -2^w (to ensure a regular addition structure), while the digits $\{2^{w-1}+1, 2^{w-1}+2, \ldots, 2^w - 1\}$ are replaced

by $\{-(2^{w-1}-1), -(2^{w-1}-2), \ldots, -1\}$ (to take advantage of symmetries), giving the digit set $\{\pm 1, \pm 2, \pm 3, \ldots, 2^{w-1}-1\} \cup \{2^{w-1}, -2^w\}$.

As with the w-NAF, we need to introduce a carry to do the recoding but, to cover all the possible situations, it can take the values 0, 1 and 2. The recoding goes from right to left by blocks of w bits, starting with a carry of 0. Given $s_j = \sum_{i=0}^{w-1} e_{i+wj} 2^i$, the recoding steps follows the rule:

$s_j + c_j$	r_j	c_{j+1}
0	-2^w	1
2^w	-2^w	2
$2^w + 1$	1	1
$1, 2, 3, \ldots, 2^{w-1}$	$s_j + c_j$	0
$(2^{w-1}+1), \ldots, (2^w - 1)$	$s_j + c_j - 2^w$	1

Once the scalar is recoded (and stored), the scalar multiplication works very much like a left-to-right "2^w and add" algorithm on the recoding. Rather than computing $[2]D_0$ (where D_0 is the partial sum at the previous step of the scalar multiplication) and then adding $[e_j]D$, the algorithm computes $[2^w]D_0$ (by doubling w times) and then adds $[r_j]D$.

Since the sequence of doublings and additions is fixed and is the same for all integers of the same size, this recoding is resistant against SPA attacks and the fastest implementations of the group operations can be used even if they are very unbalanced.

A side effect of this approach is that even leading zero digits can (and will) be recoded as non-zero. The length of the recoding must then be decided beforehand – usually to fit the longest possible key – with the added bonus that short scalars are indistinguishable from longer ones.

5 Fixed Left-to-Right Recodings

The main disadvantage of Möller's recoding algorithm is that it is right-to-left, so it must be computed and stored before the scalar multiplication. To obtain a left-to-right recodings (which can be interleaved with the scalar multiplication) and to use symmetries (to save space and precomputations), we use digit sets which are symmetric around 0.

Since the recoding goes from the highest powers of 2^w down to the lowest, the carry will not behave as usual: Instead of delaying the addition of 2^w and replacing it by the addition of 1 at the next (higher) power of 2^w, the carry (if different from 0) will delay the subtraction of 1 and replace it by the subtraction of 2^w at the next (lower) power of 2^w. For simplicity, the values of the carry will still be denoted 0 and 1 as in the w-NAF, but with the understanding that it has the new meaning.

To simplify the notation, we define s_j as $\sum_{i=0}^{w-1} e_{i+wj} 2^i$: The coefficient of 2^{wj} in the 2^w-ary expansion (using the digit set $\{0, 1, \ldots, 2^w - 1\}$). As was the case with the fixed right-to-left recoding, the length of the representation must be decided beforehand.

5.1 Groups of Odd Order

The recoding presented in this section is equivalent to a recoding suggested by Martin Seysen (unpublished work), which is also described in [20], [21] and [16].

The digit set $\{\pm 1, \pm 3, \ldots, \pm(2^w - 1)\}$ is a quite natural choice: Since the carry produces a shift of $+1$ on s_j, this is the smallest symmetric set of integer not containing 0 for which all possible values of s_j and $s_j - 2^w$ (to take into account the previous carry) can be recoded using a carry of either 0 or 1. With this digit set, the general recoding step is described by the following rule:

s_j	r_j	c_{j-1}
even	$s_j + 1 - c_j 2^w$	1
odd	$s_j - c_j 2^w$	0

It is easy to verify that at every step $r_j = s_j - c_j 2^w + c_{j-1}$, so the final recoding is $\sum_{j=0}^{m} r_j 2^{wj} = e + c_{-1}$. A nice aspect of this rule is that for $j < m$ it can be rewritten as: $r_j = 1 - 2^w + \sum_{i=1}^{w} e_{i+wj} 2^i$ (with $c_{j-1} = 1 - e_{wj}$), making it very straightforward to implement and requires no conditional statement.

Since the recoding goes from left to right, the final recoding step takes place at the w least significant bits. We could use the same recoding system for the final step, but one must then decide what to do if there is a carry after that step (a "rightward" carry at the unit level would require a fractional expansion, which is incompatible with the scalar multiplication). One solution consists in taking the result obtained at the final step and apply the carry directly to it (without any extra doubling) instead of delaying it. But in the case of a SPA attack, this would reveal the final bit.

Although this problem cannot be fixed in general, it can be avoided in most cryptographic applications. From a cryptographic point of view, there is no disadvantage to consider that the order of the group used is a large prime. This is because the discrete logarithm problem in a group can be reduced to the discrete log problem in its subgroups using the Chinese Remainder Theorem [24]. We can therefore make the assumption that the group in which the scalar multiplication is done has odd order.

Under this condition, it is always possible to force the secret key to be an odd integer: If e is even, it can be replaced by $e' = e + \#G$ (since $[e']D = [e]D$). Since we can ensure the scalar is always odd, the left-to-right recoding using digits $\pm 1, \pm 3, \ldots, \pm(2^w - 1)$ will have a final carry c_{-1} equal to zero and the recoding will always terminate correctly. Interleaving the recoding and the scalar multiplication gives us Algorithm 3.

5.2 General Case

We now consider another digit set that could be used for the fixed left-to-right recoding, but this time without any restriction on the group order. The argument used here is by no means the only one possible and other digit sets could also give a valid recoding with the same properties.

Algorithm 3. fixed left-to-right (odd) scalar multiplication

Input: D, w, $e = 1 + \sum_{i=1}^{wm} e_i 2^i$ (odd)
Output: $[e]D$

precompute $[1]D, [3]D, \ldots, [2^w - 1]D$
$r_j \leftarrow 1 + \sum_{i=1}^{w-1} e_{i+wm} 2^i$ recoding
$D_0 \leftarrow [r_m]D$
for $j = m - 1$ down to 0 do
  for $k = 0$ to $w - 1$ do
    $D_0 \leftarrow [2]D_0$ w doublings
  $r_j \leftarrow 1 - 2^w + \sum_{i=1}^{w} e_{i+wj} 2^i$ recoding
  $D_0 \leftarrow D_0 + [r_j]D$ addition
return D_0

As the carry is done downward, we must be able to recode all the possible values of s_j and $s_j - 2^w$, i.e. all the integers in $[-2^w, 2^w - 1]$. Since the introduction of a carry of one to the next (lower) power of 2^w will increase the current coefficient by 1, the possible values (after the carry) are $-2^w, -(2^w - 1), \ldots, 2^w - 1, 2^w$, so the set of even integers $0, \pm 2, \pm 4, \pm 6, \ldots, \pm 2^w$ seems like a reasonable choice. However, we want to remove the possibility of a zero digit in the 2^w-ary expansion. Since the carry is either 0 or 1, the only possible choice for the recoding of 0 is 1 (with a new carry of 1). Similarly, -1 is also necessary since it cannot be recoded as 0 with a new carry of 1, so ± 1 must also be allowed as digits instead of 0. Bringing all this together, we obtain the digit set $\{\pm 1\} \cup \{\pm 2, \pm 4, \pm 6, \ldots, \pm 2^w\}$.

If with start with a carry of 0 for the leftmost bit recoded, the general recoding rule can be written as follows:

$s_j - c_j 2^w$	r_j	c_{j-1}
even, $\neq 0$	$s_j - c_j 2^w$	0
0	1	1
odd, $\neq -1$	$(s_j - c_j 2^w) + 1$	1
-1	-1	0

It is easy to verify that at every step of the recoding $r_j = s_j - c_j 2^w + c_{j-1}$, so that $\sum_{j=0}^{m} r_j 2^{wj} = \sum_{j=0}^{m} s_j 2^{wj} - 2^{(m+1)w} c_m + c_{-1} = e + c_{-1}$.

Remark: With the residue system $\{\pm 1\} \cup \{\pm 2, \pm 4, \pm 6, \ldots, \pm 2^w\}$, there are multiple choices for the recodings of -2 and 1:

- -2 can be recoded as -2 without a carry, or as -1 with a carry;
- 1 can be recoded as 1 without a carry, or as 2 with a carry.

The recodings rules given above were chosen for simplicity.

Once again, we must choose what to do with the final carry. We could apply the carry directly at the end of the computation, but this would reveal the last bit of the scalar (with the possible exception of a recoding of 0 or -1).

A better alternative consists in replacing the final recoding step so that the final step of the encryption always consists of *two* additions (with $r_0 + r'_0 = s_0 - c_0 2^w$):

$s_0 - c_0 2^w$	r_0	r'_0
even, $\neq -2$	$(s_0 - c_0 2^w) + 2$	-2
-2	-1	-1
odd, $\neq -1$	$(s_0 - c_0 2^w) + 1$	-1
-1	1	-2

Remark: With the exceptions of ± 1 (and -4 if $w = 2$), there exists multiple choices for the recodings of all the possible values of $s_0 - c_0 2^w$. The recodings rules given above were chosen for simplicity.

Note that this approach was not possible with the previous recoding. Since the parity of a sum of odd digits depends only on the number of additions, not which digits are added, there was no regular sum that could give both even and odd results.

The computation of the scalar multiplication proceeds as in Algorithm 1 (with w doublings between every two additions since we have a fixed 2^w-ary expansion, as in Subsection 4.4) except for the final step which becomes: $D_0 \leftarrow D_0 + [r_0]D + [r'_0]D$.

6 Performance Comparison

We can now summarize and compare the efficiency of the different scalar multiplication and recoding algorithms to get a better idea of which ones are more interesting depending on the situation. To compare equivalent security levels, we assume that SPA countermeasures (for example unified formulas) are used on the group operations in the cases where SPA attacks could reveal even partial information on the secret key.

6.1 Unrestricted Environment

We first consider the case of applications where there is no restriction on the memory used by the algorithm and where the group element is assumed fixed for every scalar multiplication while the scalar varies. Under these conditions, we can assume that the precomputations are already done when the double-and-add algorithm is used, so their cost does not have to be taken into account. To have a common basis for the comparison, we assume that the recodings all have the same (average) density of $1/t$, with the exception of the double-and-add algorithm on the binary representation (average density of $1/2$) and the Montgomery ladders (density of 1).

We express the costs as "group operations (on average) per bit of the scalar". We denote by r the cost (in normal group addition) of an optimized group doubling, and by c the cost of indistinguishable group operations (either using uniform formulas or dummy operations). Note that $c \geq 1$ and should ideally be very close to 1.

By memory, we mean the number of precomputed elements which must be in memory for the double-and-add algorithms, including $[1]D$. Since Montgomery

ladders do not require any precomputations but compute two group elements instead of one, we write its memory requirement as 1.

We get the following table:

method	section	w	cost	memory	direction
double-and-add	2.1	1	$\frac{3}{2}c$	1	left-to-right
Montgomery Ladder	3.2	1	$r+1$	1	left-to-right
w-NAF	4.2	$t-2$	$c(1+\frac{1}{t})$	2^{t-3}	right-to-left
minimal LtoR	4.3	$t-2$	$c(1+\frac{1}{t})$	2^{t-3}	left-to-right
Möller	4.4	t	$r+\frac{1}{t}$	$2^{t-1}+1$	right-to-left
fixed LtoR (odd order)	5.1	t	$r+\frac{1}{t}$	2^{t-1}	left-to-right
fixed LtoR (general)	5.2	t	$r+\frac{1}{t}$	$2^{t-1}+1$	left-to-right

We can see that for the same density, the three fixed recodings require four times as much memory and precomputations than the w-NAF (a little more in the case of general group orders and the right-to-left recoding). If $r < 1+\frac{c-1}{t}$, the three fixed recodings are more efficient, but if $r > 1+\frac{c-1}{t}$, the w-NAF and the minimal weight left-to-right recoding become more efficient.

6.2 Restricted Memory

In some applications (such as restricted environments and implementations where the secret key is used more than once but on different group elements), it is really unfair to compare recodings which require different number of precomputations. The easiest way to compare the different recodings in these situations is to assume that a fixed number of precomputations are done (here we assume either 2^t or 2^t+1) and compare the cost of the multiplications without taking into account the precomputation cost (which is the same for all the recodings, even thought they use different digit sets).

To make the comparisons uniform, we do not consider the double-and-add on the binary expansion and Montgomery ladders. Using the same notation as in the previous subsection, we find:

method	section	memory	average density	cost	direction
w-NAF	4.2	2^t	$\frac{1}{t+3}$	$c(1+\frac{1}{t+3})$	right-to-left
minimal LtoR	4.3	2^t	$\frac{1}{t+3}$	$c(1+\frac{1}{t+3})$	left-to-right
Möller	4.4	2^t+1	$\frac{1}{t+1}$	$r+\frac{1}{t+1}$	right-to-left
fixed LtoR	5.2	2^t+1	$\frac{1}{t+1}$	$r+\frac{1}{t+1}$	left-to-right
fixed LtoR	5.1	2^t	$\frac{1}{t+1}$	$r+\frac{1}{t+1}$	left-to-right

This time the comparisons are much more clearly delimited. If we let $\gamma = c\left(1+\frac{1}{t+3}\right)-\frac{1}{t+1}$, we get the following rules:

- The fixed left-to-right recodings are at least as efficient as Möller's fixed right-to-left recoding.
- If $r < \gamma$, the fixed recodings are faster than the w-NAF or the minimal weight left-to-right recoding.

- If $r > \gamma$, the w-NAF and the minimal weight left-to-right recoding are faster than the fixed recodings, even though SPA countermeasures must be added in the implementation of these algorithms.
- Although the recodings from Sections 4.4 and 5.2 require one more step of precomputation, the use of even integers as coefficients is often advantageous. If doublings are faster than additions, the total cost of the precomputations can be lower than for the other coefficient sets (to be precise, when $r < 1/(1+2^{1-t})$ since $2^{t-1}+1$ of the precomputations can be done by group doublings rather than group additions). These recodings may be more interesting than the recoding of Section 5.1 if having to store one more group element is an acceptable compromise.

7 Conclusion

We presented two integers recodings which are resistant to SPA attacks. These recodings are left-to-right so they can be interleaved with a left-to-right scalar multiplication, removing the need to store both the scalar and its recoding. In groups where the doubling operations can be implemented with significant savings compared to a group addition, these algorithms become faster than a w-NAF (or its left-to-right equivalent) which has been secured against SPA attacks. It should be kept in mind that these implementation do not ensure in any way the security against differential side channel analysis, so countermeasures against these attacks should also be used if the secret key is used more than once.

Acknowledgements

The author would like to thank Roberto Avanzi and Bodo Möller for their useful comments and remarks.

References

1. D. Agrawal, B. Archambeault, J.R. Rao, and P. Rohatgi. The em side-channel(s). In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 29–45. Springer–Verlag, 2003.
2. R.M. Avanzi. A note on the signed sliding window integer recoding and a left-to-right analogue. In H. Handschuh and M.A. Hasan, editors, *Selected Areas in Cryptography – SAC 2004*, volume 3357 of *LNCS*, pages 130–143. Springer–Verlag, 2005.
3. R.M. Avanzi. Side channel attacks on implementations of curve-based cryptographic primitives. Cryptology ePrint Archive, Report 2005/017, 2005. Available at: <http://eprint.iacr.org/>.
4. O. Billet and M. Joye. The jacobi model of an elliptic curve and side-channel analysis. In M. Fossorier, T. Høholdt, and A. Poli, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes – AAECC-15*, volume 2643 of *LNCS*, pages 34–42. Springer–Verlag, 2003.

5. É. Brier and M. Joye. Weierstraßelliptic curves and side-channel attacks. In D. Naccache and P. Paillier, editors, *Public Key Cryptography – PKC 2002*, volume 2274 of *LNCS*, pages 335–345. Springer–Verlag, 2002.
6. É. Brier, M. Joye, and I. Déchène. Unified point addition formulæfor elliptic curve cryptosystems. In N. Nedjah and L. de Macedo Mourelle, editors, *Embedded Cryptographic Hardware: Methodologies & Architectures*. Nova Science Publishers, 2004.
7. H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation. In *ICICS'97*, volume 1334 of *LNCS*, pages 282–290. Springer–Verlag, 1997.
8. H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In K. Ohta and D. Pei, editors, *Advances in Cryptology - ASIACRYPT'98*, volume 1514 of *LNCS*, pages 51–65. Springer–Verlag, 1998.
9. J.-S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99*, volume 1717 of *LNCS*, pages 292–302. Springer–Verlag, 1999.
10. C. Guyot, K. Kaveh, and V. Patankar. Explicit algorithm for the arithmetic on the hyperelliptic jacobians of genus 3. *J. Ramanujan Math. Soc.*, 19(2):75–115, 2004.
11. M. Joye and J.-J. Quisquater. Hessian elliptic curves and side-channel attacks. In Ç.K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *LNCS*, pages 402–410. Springer–Verlag, 2001.
12. M. Joye and S.-M. Yen. The montgomery powering ladder. In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 291–302. Springer–Verlag, 2003.
13. P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer–Verlag, 1999.
14. T. Lange and M. Stevens. Efficient doubling on genus two curves over binary fields. In H. Handschuh and M.A. Hasan, editors, *Selected Areas in Cryptography – SAC 2004*, volume 3357 of *LNCS*, pages 170–181. Springer–Verlag, 2005.
15. P.-Y. Liardet and N.P. Smart. Preventing spa/dpa in ecc systems using the jacobi form. In Ç.K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *LNCS*, pages 391–401. Springer–Verlag, 2001.
16. C. H. Lim. A new method for securing elliptic scalar multiplication against side-channel attacks. In H. Wang, J. Pieprzyk, and V. Varadharajan, editors, *Information Security and Privacy – ACISP 2004*, volume 3108 of *LNCS*, pages 289–300. Springer–Verlag, 2004.
17. B. Möller. Securing elliptic curve point multiplication against side-channel attacks. In G.I. Davida and Y. Frankel, editors, *Information Security: 4th International Conference – ISC 2001*, volume 2200 of *LNCS*, pages 324–334. Springer–Verlag, 2001. Extended version available at: <http://www.bmoeller.de/#ecc-sca>.
18. J. Muir and D. Stinson. New minimal weight representations for left-to-right window methods. CACR Technical Report, CORR 2004-19, 2004. Available at: <http://www.cacr.math.uwaterloo.ca/techreports/2004/corr2004-19.pdf>.
19. K. Okeya, K. Schmidt-Samoa, C. Spahn, and T. Takagi. Signed binary representations revisited. In M. Franklin, editor, *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *LNCS*, pages 123–139. Springer–Verlag, 2004.

20. K. Okeya and T. Takagi. The width-w naf method provides small memory and fast elliptic scalar multiplications secure against side channel attacks. In M. Joye, editor, *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *LNCS*, pages 328–343. Springer–Verlag, 2003.
21. K. Okeya, T. Takagi, and C. Vuillaume. On the exact flexibility of the flexible countermeasure against side channel attacks. In H. Wang, J. Pieprzyk, and V. Varadharajan, editors, *Information Security and Privacy – ACISP 2004*, volume 3108 of *LNCS*, pages 466–477. Springer–Verlag, 2004.
22. J. Pelzl, T. Wollinger, J. Guajardo, and C. Paar. Hyperelliptic curve cryptosystems: Closing the performance gap to elliptic curves. In *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *LNCS*, pages 351–365. Springer–Verlag, 2003.
23. J. Pelzl, T. Wollinger, and C. Paar. Low cost security: Explicit formulae for genus-4 hyperelliptic curves. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography – SAC 2003*, volume 3006 of *LNCS*, pages 1–16. Springer–Verlag, 2004.
24. S.C. Pohlig and M.E. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Trans. Information Theory*, 24(1):106–110, 1978.
25. G.W. Reitwiesner. Binary arithmetic. In *Advances in computers*, volume 1, pages 231–308. Academic Press, New York, 1960.
26. J.A. Solinas. An improved algorithm for arithmetic on a family of elliptic curves. In B.S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97*, volume 1294 of *LNCS*, pages 357–371. Springer–Verlag, 1997.
27. C.D. Walter. Simple power analysis of unified code for ecc double and add. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *LNCS*, pages 191–204. Springer–Verlag, 2004.
28. A.C.C. Yao. On the evaluation of powers. *SIAM J. Comput.*, 5(1):100–103, 1976.
29. S.-M. Yen and M. Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Trans. on Computers*, 49(9):967–970, Sept. 2000.
30. S.-M. Yen, S. Kim, S. Lim, and S. Moon. A countermeasure against one physical cryptanalysis may benefit another attack. In K. Kim, editor, *Information Security and Cryptology – ICISC 2001*, volume 2288 of *LNCS*, pages 414–427. Springer–Verlag, 2002.

Efficient FPGA-Based Karatsuba Multipliers for Polynomials over $\mathbb{F}_2$

Joachim von zur Gathen and Jamshid Shokrollahi

B-IT, Görresstr. 13, Universität Bonn, 53113 Bonn, Germany
gathen@bit.uni-bonn.de
jamshid@bit.uni-bonn.de

Abstract. We study different possibilities of implementing the Karatsuba multiplier for polynomials over $\mathbb{F}_2$ on FPGAs.

This is a core task for implementing finite fields of characteristic 2. Algorithmic and platform dependent optimizations yield efficient hardware designs. The resulting structure is hybrid in two different aspects. On the one hand, a combination of the classical and the Karatsuba methods decreases the number of bit operations. On the other hand, a mixture of sequential and combinational circuit design techniques includes pipelining and can be adapted flexibly to time-area constraints. The approach—both theory and implementation—can be viewed as a further step towards taming the machinery of fast algorithmics for hardware applications.

Keywords: Finite field arithmetic, fast multiplication, asymptotically fast algorithms, Karatsuba method, hardware, FPGA.

1 Introduction

Arithmetic in finite fields is a central algorithmic task in cryptography. There are two types of groups associated to such fields: their multiplicative group of invertible elements, and elliptic (or hyperelliptic) curves. These can then be used in group-based cryptography, relying on the difficulty of computing discrete logarithms. Here we focus on fields of characteristic 2. The most fundamental task in arithmetic is multiplication. In our case, this amounts to multiplication of polynomials over $\mathbb{F}_2$, followed by a reduction modulo the fixed polynomial defining the field extension. This reduction can itself be performed by using multiplication routines or by a small hardware circuit when the polynomial is sparse. A trinomial can be used in many cases, and it is conjectured that otherwise a pentanomial can be found (see [6]). As to the other arithmetic operations, addition is bitwise XORing of vectors, squaring a special case of multiplication (much simplified by using a normal basis), and inversion more expensive and usually kept to a minimum.

Classical methods to multiply two n-bit polynomials require $O(n^2)$ bit operations. The Karatsuba algorithm reduces this to $O(n^{\log_2 3})$, and fast Fourier transformations to $O(n \log n \operatorname{loglog} n)$. The Cantor multiplier with a cost of $O(n(\log n)^2(\operatorname{loglog} n)^3)$ is designed for fields of characteristic 2, but we do not

B. Preneel and S. Tavares (Eds.): SAC 2005, LNCS 3897, pp. 359–369, 2006.

study it here (see [3] and [4]). Traditional lore held that asymptotically fast methods are not suitable for hardware. We disprove this view in the present paper, continuing our work in [7].

Our methods are asymptotically good and thus efficient for large degrees. Sophisticated implementation strategies decrease the crossover points between different algorithms and make them efficient for practical applications. Much care is required for software implementations (see [5], chapter 8, and Shoup's NTL software). The Karatsuba method has the lowest crossover point with the classical algorithm.

In hardware, the methods used are either platform independent or platform dependent. The first group consists of algorithmic optimizations which reduce the total number of operations, whereas the second approach uses specific properties of implementation environments to achieve higher performance.

The Karatsuba algorithme, for multiplication of large integers, was introduced in [10]. This algorithm is based on a formula for multiplying two linear polynomials which uses only 3 multiplications and 4 additions, as compared to 4 multiplications and 1 addition in the classical formula. The extra number of additions disappears asymptotically. This method can be applied recursively to 2^m-bit polynomials, where m is an integer. Here we optimize and adapt the Karatsuba algorithm for hardware realization of cryptographic algorithms.

FPGAs provide useful implementation platforms for cryptographic algorithms both for prototyping where early error finding is possible, and as systems on chips where system parameters can easily be changed to satisfy evolving security requirements.

Efficient software implementations of Karatsuba multipliers using general purpose processors have been discussed thoroughly in the literature (see [12], [1], [11], [8], chapter 2, and [5], chapter 8), but hardware implementations have attracted less attention. The only works known to us are [9], [14], and our previous paper [7]. [9] and [14] suggest to use algorithms with $O(n^2)$ operations to multiply polynomials which contain a prime number of bits. Their proposed number of bit operations is by a constant factor smaller than the classical method but asymptotically larger than those for the Karatsuba method. [7] contains a hybrid implementation of the Karatsuba method which reduces the latency by pipelining and by mixing sequential and combinational circuits.

The present work is to our knowledge the first one which tries to decrease the resource usage of polynomial multipliers using both known algorithmic and platform dependent methods. We present the best choice of hybrid multiplication algorithms for polynomials with at most 128 bits, as long as the choice is restricted to three (recursive) methods, namely classical, Karatsuba, and a variant of Karatsuba for quadratic polynomials. The "best" refers to minimizing the area measure. This is an algorithmic and machine independent optimization. In an earlier implementation ([7]) we had designed a 240-bit multiplier on a XC2V6000-4FF1517-4 FPGA. We re-use this structure to illustrate a second type of optimization, which is machine-dependent. Our goal is a 240-bit

multiplier with small area-time cost. This measure may be thought as the time on a single-bit processor. We now put a single 30-bit multiplier on our FPGA and use three Karatsuba steps to get from $240 = 2^3 \cdot 30$ to 30 bits. This requires judicious application of multiplexer and adder circuitry, but the major computational cost still resides in the multiplier. $27 = 3^3$ small multiplications are required for one 240-bit product, and these inputs are fed into the single small multiplier in a pipelined fashion. This has the pleasant effect of keeping the total delay small and the area reduced, with correspondingly small propagation delays. Using this 240-bit multiplier we cover in particular the 233-bit polynomials proposed by NIST for elliptic curve cryptography in [13].

One reviewer wrote: *The idea of using such a generalization of Karatsuba's method is not new, but it is usually dismissed for operands of relatively small sizes because of lower performance in software implementations. The fact that some area on an FPGA is saved is an interesting and new remark: the kind of remark usually "obvious"* **after** *one has seen it, but that only few seem able to see in the first place.*

The structure of this paper is as follows. First the Karatsuba method and its cost are studied in Section 2. Section 3 is devoted to optimized hybrid Karatsuba implementations. Section 4 shows how a hybrid structure and pipelining improves resource usage in our circuit from [7]. Section 5 analyzes the effect of the number of recursion levels on the performance, and Section 6 concludes the paper.

2 The Karatsuba Algorithm

The three coefficients of the product $(a_1x + a_0)(b_1x + b_0) = a_1b_1x^2 + (a_1b_0 + a_0b_1)x + a_0b_0$ are "classically" computed with 4 multiplications and 1 addition from the four input coefficients a_1, a_0, b_1, and b_0. The following formula uses only 3 multiplications and 4 additions:

$$(a_1x + a_0)(b_1x + b_0) = a_1b_1x^2 + ((a_1 + a_0)(b_1 + b_0) - a_1b_1 - a_0b_0)x + a_0b_0. \quad (1)$$

We call this the 2-*segment Karatsuba method or* $\mathcal{K}_2$. Setting $m = \lceil n/2 \rceil$, two n-bit polynomials (thus of degrees less than n) can be rewritten and multiplied using the formula:

$$(f_1x^m + f_0)(g_1x^m + g_0) = h_2x^{2m} + h_1x^m + h_0,$$

where f_0, f_1, g_0, and g_1 are m-bit polynomials respectively. The polynomials h_0, h_1, and h_2 are computed by applying the Karatsuba algorithm to the polynomials f_0, f_1, g_0, and g_1 as single coefficients and adding coefficients of common powers of x together. This method can be applied recursively. The circuit to perform a single stage is shown in Figure 1.

The "Overlap circuit" adds common powers of x in the three generated products. For example if $n = 8$, then the input polynomials have degree at most 7, each of the polynomials f_0, f_1, g_0, and g_1 is 4 bits long and thus of degree at

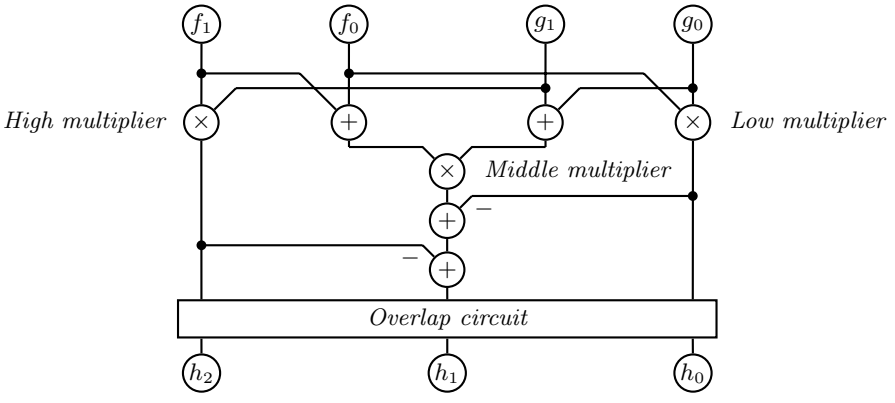


Fig. 1. The circuit to perform one level of the Karatsuba multiplication

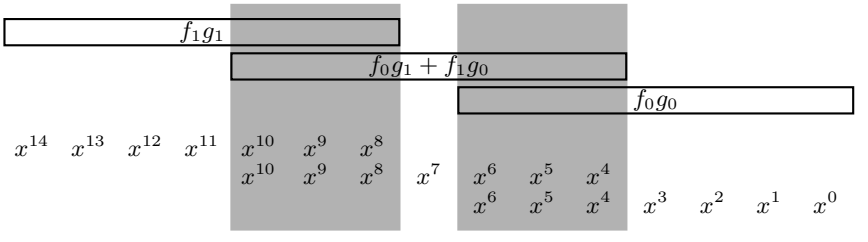


Fig. 2. The overlap circuit for the 8-bit Karatsuba multiplier

most 3, and their products will be of degree at most 6. The effect of the overlap module in this case is represented in Figure 2, where coefficients to be added together are shown in the same columns.

Figures 1 and 2 show that we need three recursive multiplication calls and some additions: $2m$ for input adders, $2(2m-1)$ for output adders, and $2(m-1)$ for the overlap module; where $m = \lceil n/2 \rceil$. If $\mathsf{M}_n^{(2)}$ is the total number of bit operations to multiply two n-bit polynomials, then

$$\mathsf{M}_n^{(2)} \leq 3\ \mathsf{M}_m^{(2)} + 8m - 4. \tag{2}$$

When n is a power of 2, with the initial values of $\mathsf{M}_1^{(2)} = 1$ we get:

$$\mathsf{M}_n^{(2)} \leq 7\ \lceil n^{\log_2 3} \rceil - 8n + 2. \tag{3}$$

The gain in Karatsuba's method is visually illustrated in Figure 8.2 of [5]. The delay of the circuit for $n \geq 2$ is at most

$$4\lceil \log_2 n \rceil \tag{4}$$

times the delay of a single gate. On the other hand, a classical multiplier for n-bit polynomials requires

$$2n^2 - 2n + 1 \tag{5}$$

gates and has a propagation delay of

$$1 + \lceil \log_2 n \rceil. \tag{6}$$

To multiply two quadratic polynomials, we use the following formula from [2] which we call 3-*segment Karatsuba or* $\mathcal{K}_3$. It uses 6 multiplications and 12 additions when used for fields of characteristic 2, compared to 9 multiplications and 4 additions in the classical method:

$$\begin{aligned}
&(a_2x^2 + a_1x + a_0)(b_2x^2 + b_1x + b_0) = \\
&a_2b_2x^4 + ((a_1 + a_2)(b_1 + b_2) - a_1b_1 - a_2b_2)x^3 \\
&+ ((a_2 + a_0)(b_2 + b_0) - a_0b_0 + a_1b_1 - a_2b_2)x^2 \\
&+ ((a_1 + a_0)(b_1 + b_0) - a_0b_0 - a_1b_1)x + a_0b_0.
\end{aligned} \tag{7}$$

Similar to (2) we can write the recursive costs of $\mathcal{K}_3$ as:

$$\mathsf{M}_n^{(3)} \leq 6\ \mathsf{M}_m^{(3)} + 22m - 10, \tag{8}$$

where $m = \lceil n/3 \rceil$.

Since $\log_2 3 \approx 1.5850 < 1.6309 \approx \log_3 6$, this approach is asymptotically inferior to the original Karatsuba method. One result of this paper is to determine the range of usefulness for this method (namely some $n \leq 81$) on our type of hardware.

3 Hybrid Design

For fast multiplication software, a judicious mixture of table look-up and classical, Karatsuba and even faster (FFT) algorithms must be used (see [5], chapter 8, and [8], chapter 2). The corresponding issues for hardware implementations have not been discussed in the literature, except that our previous paper [7] uses classical multipliers for polynomials with up to 40 bits.

We present a general methodology and execute it in the special case of a toolbox with these algorithms: classical, $\mathcal{K}_2$, and $\mathcal{K}_3$. The general idea is that we have a toolbox $\mathcal{A}$ of recursive multiplication algorithms. Each algorithm $A \in \mathcal{A}$ computes the product of two polynomials of degree less than n, for any n. The cost of A consists in some arithmetic operations plus recursive multiplications. For simplicity, we assume that the *optimal hybrid multiplication routine using* A is built from the bottom up. For each $n \geq 1$, we determine the best method for n-bit polynomials, starting with a single arithmetic operation (namely, a multiplication) for constant polynomials ($n = 1$). For $n \geq 2$, we compute the cost of applying each $A \in \mathcal{A}$ to n-bit polynomials, using the already computed optimal values for the recursive calls. We then enter into our table one of the algorithms with minimal cost.

We now execute this general approach on our toolbox $\mathcal{A} = \{\text{classical}, \mathcal{K}_2, \mathcal{K}_3\}$. The costs are given in (2) and (8). Whenever necessary, polynomials are padded with leading zeros. The results are shown in Table 1.

Table 1. The number of operations for the hybrid method for polynomial degrees below 128, and Karatsuba's algorithm according to [14]

length	hybrid			Karatsuba	length	hybrid			Karatsuba
	rec	cost	ratio			rec	cost	ratio	
3	C	13	0.404	19	66	2	4886	1.131	5402
4	C	25	0.492	33	67	2	4894	1.106	5675
5	C	41	0.567	61	68	2	4894	1.081	5812
6	2	59	0.611	77	69	2	4926	1.063	6091
7	C	85	0.689	110	70	2	4926	1.039	6231
8	2	103	0.676	127	71	2	4934	1.017	6374
9	3	134	0.730	175	72	2	4934	0.995	6041
10	2	159	0.733	219	73	2	5713	1.127	6737
11	C	221	0.875	257	74	2	5713	1.103	6883
12	2	221	0.763	275	75	2	5721	1.081	7032
13	2	307	0.933	346	76	2	5721	1.059	7107
14	2	307	0.830	382	77	2	5753	1.043	7262
15	3	346	0.838	421	78	2	5753	1.022	7340
16	2	369	0.807	441	79	2	5761	1.003	7421
17	2	470	0.934	572	80	2	5761	0.983	7381
18	2	470	0.853	593	81	3	6536	1.094	7777
19	2	553	0.921	707	82	2	6702	1.100	7935
20	2	553	0.849	733	83	2	6710	1.080	8096
21	3	654	0.930	817	84	2	6710	1.060	8177
22	2	747	0.986	855	85	2	7528	1.167	8344
23	2	755	0.929	896	86	2	7528	1.146	8428
24	2	755	0.869	917	87	2	7536	1.126	8515
25	3	992	1.070	1064	88	2	7536	1.106	8559
26	3	992	1.005	1138	89	2	7544	1.087	8738
27	3	992	0.947	1215	90	2	7544	1.068	8828
28	2	1029	0.927	1254	91	2	7699	1.071	8921
29	2	1154	0.984	1337	92	2	7699	1.053	8968
30	2	1154	0.932	1379	93	2	7731	1.039	9067
31	2	1231	0.944	1424	94	2	7731	1.022	9117
32	2	1231	0.898	1447	95	2	7739	1.006	9170
33	2	1542	1.071	1714	96	2	7739	0.989	9197
34	2	1542	1.021	1848	97	2	9904	1.245	9800
35	2	1550	0.981	1985	98	2	9904	1.225	10102
36	2	1550	0.938	2054	99	2	9912	1.207	10407
37	2	1807	1.047	2197	100	2	9912	1.188	10560
38	2	1807	1.003	2269	101	2	9944	1.173	10871
39	2	1815	0.967	2344	102	2	9944	1.155	11027
40	2	1815	0.929	2355	103	2	9952	1.138	11186
41	2	2126	1.047	2537	104	2	9952	1.121	11266
42	2	2126	1.007	2615	105	2	9984	1.107	11589
43	3	2396	1.094	2696	106	2	9984	1.091	11751
44	3	2396	1.055	2737	107	2	9992	1.075	11916
45	3	2396	1.018	2824	108	2	9992	1.060	11999
46	2	2445	1.003	2868	109	2	10357	1.082	12170
47	2	2453	0.973	2915	110	2	10357	1.067	12256
48	2	2453	0.941	2939	111	2	10365	1.053	12345
49	2	3172	1.177	3238	112	2	10365	1.038	12390
50	2	3172	1.140	3388	113	2	11522	1.137	12737
51	2	3180	1.108	3541	114	2	11522	1.122	12911
52	2	3180	1.074	3618	115	2	11530	1.107	13088
53	2	3188	1.045	3777	116	2	11530	1.092	13177
54	2	3188	1.014	3857	117	2	11562	1.080	13360
55	2	3307	1.022	3940	118	2	11562	1.066	13452
56	2	3307	0.993	3982	119	2	11570	1.052	13547
57	2	3690	1.078	4153	120	2	11570	1.038	13595
58	2	3690	1.048	4239	121	2	12295	1.089	13790
59	2	3698	1.022	4328	122	2	12295	1.075	13888
60	2	3698	0.996	4373	123	2	12303	1.062	13989
61	2	3937	1.033	4468	124	2	12303	1.048	14040
62	2	3937	1.006	4516	125	2	12335	1.038	14147
63	2	3945	0.983	4567	126	2	12335	1.025	14201
64	2	3945	0.959	4593	127	2	12343	1.013	14258
65	2	4886	1.159	5132	128	2	12343	1.000	14288

The first column gives the number n of bits, so that we deal with polynomials of degree up to $n-1$. The second column "rec" specifies the first recursive level, that is the algorithm from $\mathcal{A} = \{\text{classical}, \mathcal{K}_2, \mathcal{K}_3\}$ to be used, abbreviated as $\{C, 2, 3\}$. The column "cost" gives the total number of arithmetic operations. The next column states the "ratio" of practice to theory, namely $c \cdot \text{cost}/n^{\log_2 3}$, where the constant c is chosen so that the last entry is 1. The asymptotic regime visibly takes over already at the fairly small values that we consider. The final

column gives the cost of algorithm from [14], which is Karatsuba-based. We know of no other implementation that can be easily compared with ours.

For example, the entry $n = 41$ refers to polynomials of degree up to 40. The entry 2 in column "$\mathcal{A}$" says that $\mathcal{K}_2$ is to be employed at the top of the recursion. Since $m = \lceil 41/2 \rceil = 21$, (2) says that three pairs of 21-bit polynomials need to be multiplied, plus $8 \cdot 21 - 4 = 164$ operations. One has to look up the algorithm for 21 bits in the table. Continuing in this way, the prescription for 41 bits is:

n	41	21	7
algorithm	$\mathcal{K}_2$	$\mathcal{K}_3$	C
add	164	144	85

$$\text{total} = 164 + 3 \cdot (144 + 6 \cdot 85) = 2126.$$

In the recursive call of $\mathcal{K}_2$ at $n = 41$, the inputs are split into two pieces of 20 and 21 bits. It is tempting to single out one of the three recursive multiplications as a 20-bit operation, and indeed this view is taken in [14]. They pad input polynomials with enough zero coefficients and apply the Karatsuba method in a recursive manner. Operations involving a coefficient known to be zero are neglected. In our designs, we use three 21-bit multiplications, for a small loss in the operations count but a huge gain in modularity: we only implement a single 21-bit multiplier, thus simplifying the design and enabling pipelining. Section 4 exemplifies this (with 30 rather than 21 bits).

We note that designers of fast arithmetic software have used the general methodology sketched above, in particular formulating it as breakpoint between different algorithms. The classical algorithm can also be viewed recursively, which is used for some results in Table 2 below.

The goal of our hybrid design is to minimize the total arithmetic cost. The same methodology can, of course, also be applied to multi-objective applications, say minimizing A and AT. A concern with them would be to limit the number of table entries that are kept.

4 Hardware Structure

According to (4) and (6), the delay of a fully parallel combinational Karatsuba multiplier is almost 4 times that of a classical multiplier. It is the main disadvantage of the Karatsuba method for hardware implementations. In [7], we have suggested as solution a pipelined Karatsuba multiplier for 240-bit polynomials, shown in Figure 3.

The innermost part of the design is a combinational pipelined 40-bit classical multiplier equipped with 40-bit and 79-bit adders. The multiplier, these adders, and the overlap module, together with a control circuit, constitute a 120-bit multiplier. The algorithm is based on a modification of a Karatsuba formula for 3-segment polynomials which is similar to but slightly different from (7). (We were not aware of this better formula at that time.)

Another suitable control circuit performs the 2-segment Karatsuba method for 240 bits by means of a 120-bit recursion, 239-bit adders, and an overlap circuit.

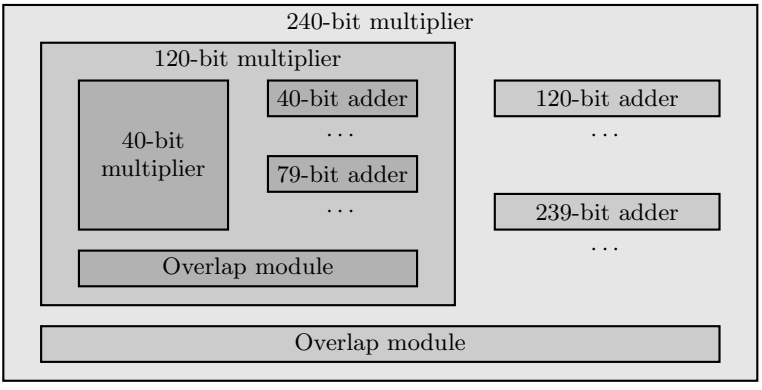


Fig. 3. The 240-bit multiplier in [7]

This multiplier can be seen as implementing the factorization $240 = 2 \cdot 3 \cdot 40$. Table 1 implies that it is usually best to apply the 2-segment Karatsuba, except for small inputs. Translating this into hardware reality, we now present a better design based on the factorization $240 = 2 \cdot 2 \cdot 2 \cdot 30$. The resulting structure is shown in Figure 4.

The 30-bit multiplier follows the recipe of Table 1. It is a combinational circuit without feedback and the design goal was to minimize its area. In general, k pipeline stages can perform n parallel multiplications in $n + k - 1$ instead of nk clock cycles without pipelining.

We have implemented our design, the structure of [7], and a purely classical implementation, on an XC2V6000-4FF1517-4 FPGA. The classical design has a classical 30-bit multiplier and applies the three classical recursion steps. The results after place and route are shown in Table 2. The second column shows the number of clock cycles for a multiplication. The third column represents the

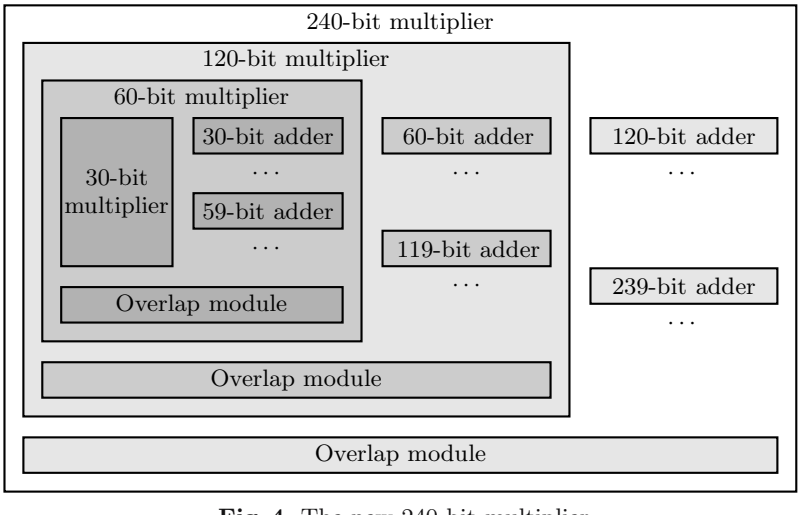


Fig. 4. The new 240-bit multiplier

Table 2. Time and area of different multipliers for 240-bit polynomials

Multiplier type	**Number of clock cycles**	**Number of slices**	**Multiplication time**	**AT** Slices $\times$ μs
classical	106	1328	$1.029\mu s$	1367
The circuit of [7] (Fig. 3)	54	1660	$0.655\mu s$	1087
Hybrid Karatsuba (Fig. 4)	55	1513	$0.670\mu s$	1014

area in terms of number of slices. This measure contains both logic elements, or LUTs, and flip-flops used for pipelining. The fourth column is the multiplication time as returned by the hardware synthesis tool. Finally the last column shows the product of area and time in order to compare the AT measures of our designs.

The synchronization is set so that the 30-bit multipliers require 1 and 4 clock cycles for classical and hybrid Karatsuba implementations, respectively. The new structure is smaller than the implementation in [7] but requires more area than the classical one. This drawback is due to the complicated structure of the Karatsuba method but is compensated by speed as seen in the time and AT measures. In the next section we further improve our structure by decreasing the number of recursions.

5 Hybrid Polynomial Multiplier with Few Recursions

In the recursive Karatsuba multiplier of [7], the core of the system, namely the combinational multipliers, is idle for about half of the time. To improve

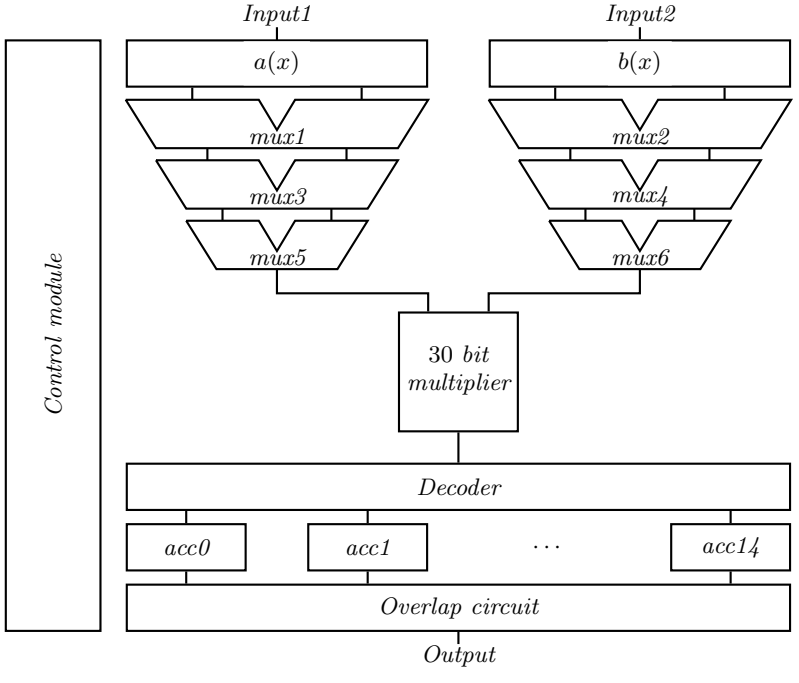


Fig. 5. The structure of the Karatsuba multiplier with fewer number of recursions

Table 3. Time and area of different 240-bit multipliers with reduced number of recursion levels

Multiplier type	**Number of clock cycles**	**Number of slices**	**Multiplication time**	**AT** Slices $\times$ μs
classical	56	1582	$0.523\mu s$	827
The circuit of [7](Fig. 3)	54	1660	$0.655\mu s$	1087
Hybrid Karatsuba (Fig. 5)	30	1480	$0.378\mu s$	559

resource usage, we reduce the communication overhead by decreasing the levels of recursion. In this new 240-bit multiplier, an 8-segment Karatsuba is applied at once to 30-bit polynomials. We computed symbolically the formulas describing three recursive levels of Karatsuba, and implemented these formulas directly.

The new circuit is shown in Figure 5. The multiplexers *mux1* to *mux6* are adders at the same time. Their inputs are 30-bit sections of the two original 240-bit polynomials which are added according to the Karatsuba rules. Now their 27 output pairs are pipelined as inputs into the 30-bit multiplier. The 27 corresponding 59-bit polynomials are subsequently combined according to the overlap rules to yield the final result. Time and space consumptions are shown in Table 3 and compared with the results of [7]. The columns are as in Table 2. We see that this design improves on the previous ones in all respects.

6 Conclusion

In this paper we have shown how combining algorithmic techniques with platform dependent strategies can be used to develop designs which are highly optimized for FPGAs. These modules have been considered as appropriate implementation targets for cryptographic purposes both as prototyping platforms and as system on chips.

We improved the structure proposed in [7] in both time and area aspects. The time has been improved by decreasing the number of recursion stages. To minimize the area we have further improved the results of [14], as witnessed in Table 1, by applying the Karatsuba method in a hybrid manner. The benefits of hybrid implementations are well known for software implementations, where the crossover points between subquadratic and classical methods depend on the available memory and processor word size. There seems to be no previous systematic investigation on how to apply these methods efficiently for hardware implementations. In this paper we have shown that a hybrid implementation mixing classical and two Karatsuba methods can result in significant area savings.

Comparisons with the work of [7] are shown in Table 3. The asymptotic methods are better than classical multipliers both with respect to time and area measures. An obvious open question is to optimize a larger class of recursive algorithms than our $\mathcal{K}_2$ and $\mathcal{K}_3$.

Acknowledgements

We thank Roberto Avanzi for various pointers to the literature and for pointing out to us the formula from [2].

References

1. Bailey, D.V., Paar, C.: Optimal extension fields for fast arithmetic in public-key algorithms. In Krawczyk, H., ed.: Advances in Cryptology: Proceedings of CRYPTO '98, Santa Barbara CA. Number 1462 in Lecture Notes in Computer Science, Springer-Verlag (1998) 472–485
2. Blahut, R.E.: Fast Algorithms for Digital Signal Processing. Addison-Wesley, Reading MA (1985)
3. Cantor, D.G.: On arithmetical algorithms over finite fields. Journal of Combinatorial Theory, Series A **50** (1989) 285–300
4. von zur Gathen, J., Gerhard, J.: Arithmetic and factorization of polynomials over $\mathbb{F}_2$. In Lakshman, Y.N., ed.: Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation ISSAC '96, Zürich, Switzerland, ACM Press (1996) 1–9 Technical report tr-rsfb-96-018, University of Paderborn, Germany, 1996, 43 pages. Final version in Mathematics of Computation.
5. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Second edn. Cambridge University Press, Cambridge, UK (2003) First edition 1999.
6. von zur Gathen, J., Nöcker, M.: Polynomial and normal bases for finite fields. Journal of Cryptology (2005) to appear.
7. Grabbe, C., Bednara, M., Shokrollahi, J., Teich, J., von zur Gathen, J.: FPGA designs of parallel high performance $GF(2^{233})$ multipliers. In: Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS-03). Volume II., Bangkok, Thailand (2003) 268–271
8. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer-Verlag (2003)
9. Jung, M., Madlener, F., Ernst, M., Huss, S.: A Reconfigurable Coprocessor for Finite Field Multiplication in $GF(2^n)$. In: Workshop on Cryptographic Hardware and Embedded Systems, Hamburg, IEEE (2002)
10. Karatsuba, A., Ofman, Y.: Multiplication of multidigit numbers on automata. Soviet Physics–Doklady **7** (1963) 595–596 translated from Doklady Akademii Nauk SSSR, Vol. 145, No. 2, pp. 293–294, July, 1962.
11. Koç, Ç.K., Erdem, S.S.: Improved Karatsuba-Ofman Multiplication in $GF(2^m)$. US Patent Application (2002)
12. Paar, C.: Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields. PhD thesis, Institute for Experimental Mathematics, University of Essen, Essen, Germany (1994)
13. U.S. Department of Commerce / National Institute of Standards and Technology: Digital Signature Standard (DSS). (2000) Federal Information Processings Standards Publication 186-2.
14. Weimerskirch, A., Paar, C.: Generalizations of the karatsuba algorithm for efficient implementations. Technical report, Ruhr-Universität-Bochum, Germany (2003)

Author Index