Josef Pieprzyk
Ahmad-Reza Sadeghi
Mark Manulis (Eds.)

LNCS 7712

Cryptology and Network Security

11th International Conference, CANS 2012
Darmstadt, Germany, December 2012
Proceedings

Lecture Notes in Computer Science 7712

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

Josef Pieprzyk Ahmad-Reza Sadeghi
Mark Manulis (Eds.)

Cryptology and Network Security

11th International Conference, CANS 2012
Darmstadt, Germany, December 12-14, 2012
Proceedings

Springer

Volume Editors

Josef Pieprzyk
Macquarie University
Department of Computing
Sydney, NSW 2109, Australia
E-mail: josef.pieprzyk@mq.edu.au

Ahmad-Reza Sadeghi
Technische Universität Darmstadt
System Security Lab.
64293 Darmstadt, Germany
E-mail: ahmad.sadeghi@trust.cased.de

Mark Manulis
University of Surrey
Department of Computing
Guildford, GU2 7XH, UK
E-mail: m.manulis@surrey.ac.uk

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-35403-8 e-ISBN 978-3-642-35404-5
DOI 10.1007/978-3-642-35404-5
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012953024

CR Subject Classification (1998): E.3, C.2, K.6.5, D.4.6, G.2.1, E.4

LNCS Sublibrary: SL 4 – Security and Cryptology

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

CANS 2012 was held at the Darmstadtium Congress Center in Darmstadt, Germany, during December 12–14, 2012. The conference was organized in cooperation with the International Association for Cryptologic Research (IACR).

The history of CANS started in 2001, when the first edition of the conference was organized in Taipei, followed by San Francisco (2002), Miami (2003), Xiamen (2005), Suzhou (2006), Singapore (2007), Hong Kong (2008), Kanazawa (2009), Kuala Lumpur (2010), and Sanya (2011). CANS 2012 was the 11th event in this series and it was the first time that the conference came to Europe.

Since 2005, CANS proceedings have been published by Springer in their Lecture Notes in Computer Science (LNCS) series. We thank Alfred Hofmann from Springer for his support in the publication of the CANS 2012 proceedings.

CANS 2012 received 99 submissions of which the Program Committee chose 22 papers to be included in the conference program. Each submitted paper got assigned to three reviewers. However, papers submitted by Program Committee members were reviewed by five referees. The double-blind review process consisted of two stages. In the first stage, papers were evaluated by reviewers and their comments very submitted to the EasyChair server. In the second stage, the papers were scrutinized in extensive anonymous discussions among the committee members. Some papers received up to 21 discussion comments. We hope that all good submissions that did not make it into the program of CANS 2012 will eventually be accepted elsewhere and that the papers that got accepted to the conference are interesting to the readers.

Special words of appreciation go to Nicolas Courtois, Sherman Chow, and Vincent Rijmen, who kindly agreed to shepherd three papers that were accepted to the conference. The authors of the accepted papers had two weeks for revision and preparation of final versions. The revised papers were not subject to editorial review and the authors bear full responsibility for their contents. The submission and review process was supported by EasyChair and we thank the EasyChair team for letting us use their server.

The paper *"A Simple Key-Recovery Attack on McOE-X"* by Florian Mendel, Bart Mennink, Vincent Rijmen and Elmar Tischhauser won the best paper award.

CANS 2012 also featured two invited talks

- *"Confined Guessing: Practical Signatures from Standard Assumptions"* by Dennis Hofheinz, Karlsruhe Institute of Technology, Germany.
- *"Cryptographic Failures and Successes"* by Bart Preneel, Katholieke Universiteit Leuven, Belgium.

There are many people who contributed to the success of CANS 2012. First, we would like to thank the authors of all papers (both accepted and rejected) for submitting their results to the conference. A special thanks goes to the members

of the Program Committee and the external referees who gave their time, expertise, and enthusiasm in order to ensure that each paper received a thorough and fair review. Last but not least, we thank Stanislav Bulygin, Heike Meissner, and Anette Mittenhuber for their support in the organization of the conference.

December 2012

Josef Pieprzyk
Ahmad-Reza Sadeghi
Mark Manulis

CANS 2012

The 11th Cryptology and Network Security Conference
Darmstadt, Germany
December 12–14, 2012

General Chair

Mark Manulis	University of Surrey, UK

Program Chair

Josef Pieprzyk	Macquarie University, Australia
Ahmad-Reza Sadegh	Technische Universität Darmstadt / Fraunhofer SIT, Germany

Program Committee

Michel Abdalla	ENS, France
Gildas Avoine	Université Catholique de Louvain, Belgium
Feng Bao	Institute for Infocomm Research, Singapore
Sébastien Canard	Orange Labs, France
Sherman Chow	University of Waterloo, Canada and Chinese University of Hong Kong, Hong Kong
Nicolas Courtois	University College London, UK
Emiliano De Cristofaro	PARC Research, USA
Reza Curtmola	New Jersey Institute of Technology, USA
George Danezis	Microsoft Research Cambridge, UK
Roberto Di Pietro	Università di Roma Tre, Italy
Juan Garay	AT&T Labs Research, USA
Philip Hawkes	Qualcomm, Australia
Amir Herzberg	Bar-Ilan University, Israel
Nick Hopper	University of Minnesota, USA
Stanisław Jarecki	University of California, Irvine, USA
Xuxian Jiang	North Carolina State University, USA
Seny Kamara	Microsoft Research, USA
Angelos Keromytis	Columbia University, USA
Svein Johan Knapskog	NTNU Trondheim, Norway
Benoît Libert	Université Catholique de Louvain, Belgium
Atsuko Miyaji	JAIST, Japan
Refik Molva	Eurecom, France
Fabian Monrose	University of North Carolina, USA

Steering Committee

External Reviewers

Gérard, Benoît
Gouget, Aline
Hermans, Jens
Hsiao, Hsu-Chun
Huang, Yan
Hubacek, Pavel
Isshiki, Toshiyuki
Jee, Kangkook
Jhawar, Mahavir
Joux, Antoine
Kemerlis, Vasileios P.
Khoo, Khoongming
Kolesnikov, Vladimir
Kontaxis, Georgios
Kügler, Dennis
Lampe, Rodolphe
Le, Hoi
Leontiadis, Iraklis
Lin, Changlu
Liu, Lei
Loftus, Jake
Lu, Jiqiang
Lyubashevsky, Vadim
Malozemoff, Alex
Martin, Benjamin
Mendel, Florian
Mennink, Bart
Mittal, Prateek
Mohaisen, Abedelaziz
Nachef, Valerie
Nikova, Svetla
Nithyanand, Rishab
Norcie, Gregory
Olivier, Pereira
Omote, Kazumasa
Önen, Melek
Pappas, Vasilis
Peters, Thomas
Petit, Christophe
Plaga, Rainer
Polychronakis, Michalis
Portokalidis, Georgios
Quaglia, Elizabeth
Rasmussen, Kasper Bonne
Regazzoni, Francesco
Reparaz, Oscar
Riva, Ben
Sadeghian, Saeed
Sarkar, Sumanta
Sarr, Augustin
Schwabe, Peter
Seyalioglu, Hakan
Shao, Jun
Stehlé, Damien
Tan, Syh-Yuan
Tartary, Christophe
Thoma, Achint
Traoré, Jacques
Trujillo-Rasua, Rolando
Tselekounis, Yiannis
Tuhin, Ashraful
Varıcı, Kerem
Vergnaud, Damien
Volte, Emmanuel
Wang, Pengwei
Wieschebrink, Christian
Yang, Guomin
Zhang, Haibin
Zhang, Tongjie
Zhang, Xin
Zhang, Yun
Zheng, Qingji
Zohner, Michael

Invited Talks
Confined Guessing: Practical Signatures from Standard Assumptions

Dennis Hofheinz

Karlsruhe Institute of Technology
email: Dennis.Hofheinz@kit.edu

Abstract. In the first part of the talk, we survey existing paradigms to construct digital signature schemes. We highlight the surprising difficulty to build practical schemes. Namely, from a theoretic point of view, digital signatures are equivalent to one-way functions, which in turn appear to be a weaker primitive than public-key encryption (PKE). However, while we know how to construct practical PKE schemes from standard complexity assumptions, it seems much harder to construct practical signature schemes.
In the second part of the talk, we put forward a new technique to construct very efficient and compact signature schemes. Our technique combines several instances of an only mildly secure signature scheme to obtain a fully secure scheme. Since the mild security notion we require is much easier to achieve than full security, we can combine our strategy with existing techniques to obtain a number of interesting new (and fully secure) signature schemes. Concretely, we obtain efficient and compact new signature schemes from the Computational Diffie-Hellman, RSA, and Short Integer Solutions assumptions. Each of the arising schemes provides significant improvements upon state-of-the-art schemes.

Cryptographic Failures and Successes

Bart Preneel

Katholieke Universiteit Leuven, ESAT/COSIC
Kasteelpark Arenberg 10 Bus 2446, B-3001 Leuven, Belgium
email: bart.preneel@esat.kuleuven.be

Abstract. This talk discusses a broad range of applications of cryptography and tries to make the balance of the achievements. Topics that will be covered include credit card payments (EMV), e-commerce (SSL/TLS and PKI), mobile communications (GSM and 3G) and identification technologies (eID and e-passport). We will also evaluate how cryptography can lead to new architectures, that result in distributed solutions for privacy-friendly metering, that have applications in insurance pricing, road pricing and smart electricity grids.

Table of Contents

Cryptanalysis

Network Security

Cryptographic Protocols

Encryption

S-Box Theory

Conditional Differential Cryptanalysis of Grain-128a

Michael Lehmann and Willi Meier

FHNW, Switzerland

Abstract. Grain-128a is a new version of the stream cipher Grain-128. To analyse the security of the cipher, we study the monomial structure and use high order differential attacks on both the new and old versions. The comparison of symbolic expressions suggests that Grain-128a is immune against dynamic cube attacks. Additionally, we find that it is also immune against differential attacks as the best attack we could find results in a bias at round 189 out of 256.

Keywords: stream ciphers, Grain-128, Grain-128a, conditional differential cryptanalysis.

1 Introduction

Grain is a family of lightweight stream ciphers that share the property of a very small hardware implementation. As any modern stream cipher, Grain allows for public initial vectors so that the initial state for keystream generation is produced by an initialization mechanism that depends on the secret key and on the initial vector. There are two versions, Grain v1 [9] with a 80-bit key, and Grain-128a [1] with a 128-bit key. The latter has built-in support for optional authentication. Grain v1 is a finalist in the eSTREAM portfolio of hardware oriented stream ciphers. Grain-128a is modelled on its predecessor Grain-128 [8], but uses slightly different non-linear functions with the aim to strengthen it against known attacks on Grain-128.

Grain v1 and Grain-128a share a very similar structure based on non-linear feedback shift registers (NFSR). In [10], conditional differential cryptanalysis, first introduced in [4], has been applied to such constructions. The idea is to control the propagation of differences by imposing conditions on the public variables, i.e. the initial vector (IV), of the cipher. Depending whether these conditions involve secret variables or not, key-recovery or distinguishing attacks can be mounted. The technique extends to higher order differential cryptanalysis. A different but related concept is the dynamic cube attack presented in [7]. Because of the higher complexity of the update functions of Grain v1, the attacks are not applicable to this cipher.

The aim of this paper is to compare the security of Grain-128a with that of Grain-128 with regard to higher order differential attacks, including conditional differential cryptanalysis. By studying the monomial structure in the initialisation mechanism it is argued that dynamic cube attacks will not be effective

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 1–11, 2012.

against Grain-128a as was the case for Grain-128. This is the first analysis of the security of Grain-128a in a chosen IV scenario.

The paper is organized as follows. Section 2 gives a brief summary of the cube attack as well as the dynamic cube attack and recalls the idea of conditional differential analysis. Section 3 gives a concise overview of the design of Grain-128a and the changes made with respect to its predecessor Grain-128. Section 4 provides a comparison of Grain-128a with its predecessor describing the impacts of the improvements made to the cipher. It shows the results achieved with higher order differential attacks in general and the conditional differential analysis in particular. In order to make a statement concerning the security of the cipher, these results are again compared to results on Grain-128.

2 Background

This section briefly depicts the two different variants of the cube attack, namely the static and the dynamic variants. The section is a summary of [6], [7] and [5], respectively. Initial work related to cube attacks is also [13]. Section 2.3 recalls relevant facts on conditional differential analysis in a concise manner. For more detailed information we refer to [10].

2.1 Cube Attack

For most stream ciphers, an output bit can be described as a master polynomial $p(k_1, \ldots, k_n, v_1, \ldots, v_m)$ over $\mathbb{F}_2$. Such a polynomial can be split and written as a sum of two polynomials:

$$p(k_1, \ldots, k_n, v_1, \ldots, v_m) = t_I * p_{S(I)} + q(k_1, \ldots, k_n, v_1, \ldots, v_m)$$

Where:

- t_I is called maxterm and is a product of certain IV bits, for example $v_1 v_2 v_5$
- $p_{S(I)}$ is called superpoly. It does not contain any variables of t_I
- $q(k_1, \ldots, k_n, v_1, \ldots, v_m)$ is the remainder polynomial. The summands of this polynomial miss at least one of the variables of t_I.

The maxterm t_I is defined through a subset of indices I (a so-called cube). In order to get the super polynomial $p_{S(I)}$, one assigns all possible values to the variables contained in I, evaluates the master polynomial and sums up the results. In this way, every summand missing k (k≥1) variables of the maxterm t_I is added exactly 2^k times and is therefore eventually eliminated with the modular reduction. The super polynomial $p_{S(I)}$, however, is part of the evaluated master polynomial if and only if all variables of t_I have value one. All other variables whose indices are not contained in I are assigned a certain value, usually zero. The idea of cube attacks is to find enough maxterms t_I whose super polynomial is linear and not a constant. This enables to recover the key through solving a system of linear equations.

2.2 Dynamic Cube Attack

The Dynamic Cube Attack is very similar to the static version. The difference is that certain variables which are not part of t_I are assigned a function of public and private variables instead of a constant value. Those functions are chosen in a way that the symbolic expressions of certain variables are simplified. The idea is to rewrite a polynomial P with three polynomials:

$$P = P_1 * P_2 + P_3$$

In order to simplify the polynomial P, one sets a linear term of P_1 in such a way that the whole polynomial P_1 is zero. This eliminates P_2 and simplifies P to P_3. An example is given in [7]. The attack was further improved and published in [5]. This attack breaks the full version of Grain-128 with a complexity of about 2^{90} and memory usage of 2^{63} bit.

2.3 Conditional Differential Analysis

The notion of a conditional differential characteristic has been introduced in [4] to improve differential attacks against DES. Similar ideas are used to accelerate the differential collision search in hash function cryptanalysis [14,15]. In [10,11] the principle has been applied to NFSR-based constructions and extended to higher order differential attacks.

The Basic Idea for NFSR-based Constructions. Let us consider the case of a synchronous stream cipher taking as its input an initial value (IV) and a key. We assume a scenario where the attacker can observe the keystream for many chosen IVs under the same secret key. The basic idea of conditional differential cryptanalysis is to control the propagation of a difference in the IV through the first few rounds of the initialization process. This is done by imposing specific conditions on certain bits of the IV. From these conditions, a sample of IV pairs is derived and experimentally tested for a bias in the resulting keystream differences. Conditions might be also imposed on the key which defines classes of weak keys.

An example for Grain-128a . Consider a difference in bit 69 of the IV. The difference does not affect the rounds 0 to 8. Then, at round 9, the value of the feedback is computed as $x_{17}k_{21}+x_{22}x_{29}+x_{51}k_{104}+x_{69}x_{88}+k_{11}+k_{21}k_{104}+k_{24}+k_{45}+k_{54}+k_{73}+k_{82}+k_{98}+1$, that is, the difference eventually propagates to s_{9+128} and b_{9+128}. Imposing the condition $x_{88} = 0$ we can prevent the difference from propagating. Similarly at round 27, h is computed as $x_{35}k_{39} + x_{40}x_{47} + x_{69}k_{122} + x_{87} + k_{29} + k_{39}k_{122} + k_{42} + k_{63} + k_{72} + k_{91} + k_{100} + k_{116} + 1$ and the propagation can be prevented by the condition $k_{122} = 0$. Testing a sample of 2^{16} randomly chosen IV pairs separated by a difference in bit 69, a significant bias can be detected at round 140. However, if the IVs satisfy the conditions $x_{88} = 0$ and the key satisfies $k_{122} = 0$, biases can be detected up to round 159.

Extension to Higher Orders. If the keystream bits are modeled as Boolean functions of the form $f : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}$, a first order attack evaluates

$$\Delta_v f(k,x) := f(k,x) + f(k,x+v)$$

for a fixed difference $v \in \{0,1\}^n$ many chosen $x \in \{0,1\}^n$. In [12], $\Delta_v f$ is called a first order derivative of f with respect to v. More generally, if $V \subset \{0,1\}^n$ is a linear subspace of dimension d,

$$\Delta_V f(k,x) := \sum_{v \in V} f(k,x+v)$$

is called the derivative of f with respect to V. The principle of conditional differential cryptanalysis extends to higher orders as follows. If $\{a_1, \ldots, a_d\}$ is a basis of V, conditions are derived for each b_i as a first order difference and merged to a total set of differences from which the sample is derived.

Notation and Terminology. In this paper, the subspaces V will be always generated by IVs of Hamming weight one and we write v_i, $0 \le i \le 95$ for the IV with a 1 at position i and 0s otherwise. Conditional differential cryptanalysis can be seen as a refinement of cube testers introduced in [3].

3 Grain-128a

In our analysis, the authentication mechanism will be ignored. Fig. 1 depicts an overview of the building blocks of the output generator, which is constructed using three main building blocks, namely an LFSR, an NFSR and a pre-output function. We denote by $s_i, s_{i+1}, \ldots, s_{i+127}$ the contents of the LFSR. Similarly, the content of the NFSR is denoted by $b_i, b_{i+1}, \ldots, b_{i+127}$. Together, the 256 memory elements in the two shift registers represent the state of the output generator.

The primitive feedback polynomial of the LFSR, denoted $f(x)$, is defined as

$$f(x) = 1 + x^{32} + x^{47} + x^{58} + x^{90} + x^{121} + x^{128}.$$

We also recall the corresponding update function of the LFSR as

$$s_{i+128} = s_i + s_{i+7} + s_{i+38} + s_{i+70} + s_{i+81} + s_{i+96}.$$

The nonlinear feedback polynomial of the NFSR, $g(x)$, is defined as (changes to the predecessor Grain-128 are in boldface)

$$\begin{aligned} g(x) = {} & 1 + x^{32} + x^{37} + x^{72} + x^{102} + x^{128} + x^{44}x^{60} \\ & + x^{61}x^{125} + x^{63}x^{67} + x^{69}x^{101} \\ & + x^{80}x^{88} + x^{110}x^{111} + x^{115}x^{117} \\ & + \boldsymbol{x^{46}x^{50}x^{58}} + \boldsymbol{x^{103}x^{104}x^{106}} + \boldsymbol{x^{33}x^{35}x^{36}x^{40}}. \end{aligned}$$

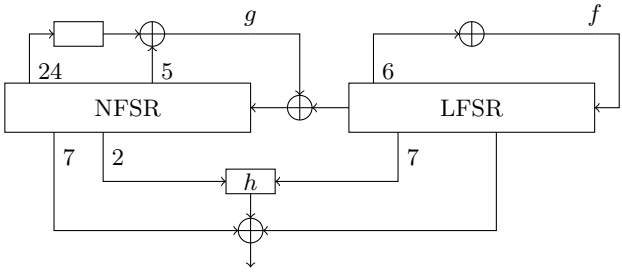


Fig. 1. An overview of the output generator

Again, recall the rule for updating the NFSR, with changes to Grain-128 in boldface.

$$\begin{aligned} b_{i+128} = s_i &+ b_i + b_{i+26} + b_{i+56} + b_{i+91} + b_{i+96} \\ &+ b_{i+3}b_{i+67} + b_{i+11}b_{i+13} + b_{i+17}b_{i+18} \\ &+ b_{i+27}b_{i+59} + b_{i+40}b_{i+48} + b_{i+61}b_{i+65} \\ &+ b_{i+68}b_{i+84} + \boldsymbol{b_{i+88}b_{i+92}b_{i+93}b_{i+95}} \\ &+ \boldsymbol{b_{i+22}b_{i+24}b_{i+25}} + \boldsymbol{b_{i+70}b_{i+78}b_{i+82}}. \end{aligned}$$

Note that the update rule contains the bit s_i which is not part of the feedback polynomial and is output from the LFSR, thus masking the input to the NFSR.

Nine state variables are taken as input to a Boolean function, $h(x)$: two bits come from the NFSR and seven from the LFSR. This function is defined as

$$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8$$

where the variables $x_0, \ldots, x_8$ correspond to, respectively, the state variables b_{i+12}, s_{i+8}, s_{i+13}, s_{i+20}, b_{i+95}, s_{i+42}, s_{i+60}, s_{i+79} and s_{i+94} (or s_{i+95} for Grain-128, respectively). The pre-output function is defined as

$$y_i = h(x) + s_{i+93} + \sum_{j \in \mathcal{A}} b_{i+j},$$

where $\mathcal{A} = \{2, 15, 36, 45, 64, 73, 89\}$.

Before keystream is generated the cipher must be initialized with the key and the IV. Denote the bits of the key as $k_i,\ 0 \leq i \leq 127$ and the IV bits $IV_i,\ 0 \leq i \leq 95$. The initialisation of the key and IV is done as follows. The 128 NFSR elements are loaded with the key bits, $b_i = k_i,\ 0 \leq i \leq 127$, and the first 96 LFSR elements are loaded with the IV bits, $s_i = IV_i,\ 0 \leq i \leq 95$. The last 32 bits of the LFSR are filled with ones and a zero, $s_i = 1,\ 96 \leq i \leq 126$, $s_{127} = 0$. Then, the cipher is clocked 256 times without producing any keystream. Instead, the output function is fed back and xored with the input, both to the LFSR and to the NFSR, see Fig. 2.

In the mode without authentication, all output bits are used directly as keystream. This mode of operation is the same as in Grain-128.

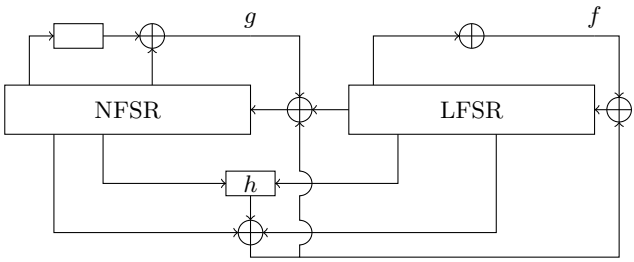


Fig. 2. The state initialization

4 Findings

The NFSR-update used by Grain-128 is merely of order two, whereas the one used by Grain-128a is of order four and contains two extra monomials of order three. Consequently, the symbolic expressions of Grain-128a grow faster. Fig. 3 depicts this fact using the data we collected.

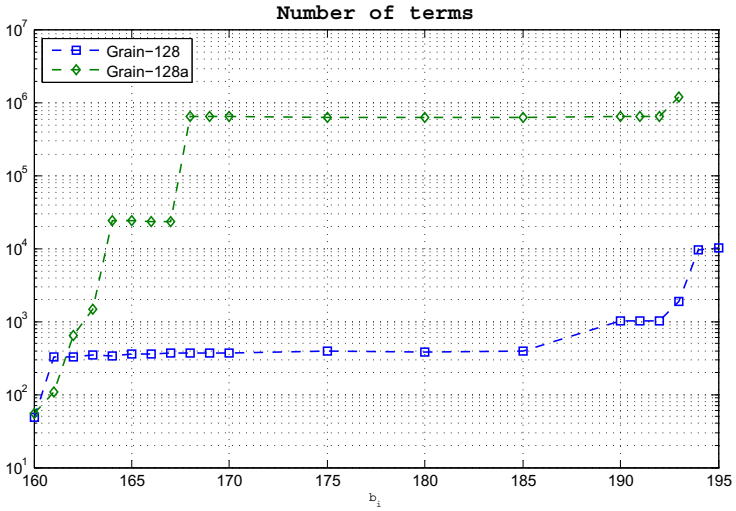


Fig. 3. Number of terms

To illustrate the difference relating to the order of the symbolic expressions, i.e. the order of monomials in variables b_i and s_i, and the number of terms of maximum order, Table 1 shows the order and number of terms of some b_i of Grain-128 and Grain-128a, respectively. Note that the two columns on the left both describe the number of monomials of order greater or equal the maximum order in the symbolic expression of Grain-128.

Table 1. Order of the symbolic expressions of the b_i for Grain-128 vs Grain-128a

	Order of symbolic expressions		**#terms ≥ max. order Grain-128**	
	Grain-128	**Grain-128a**	**Grain-128**	**Grain-128a**
b_{160}	3	4	1	7
b_{161}	6	7	7	4
b_{162}	5	8	35	153
b_{163}	5	9	35	768
b_{164}	5	11	35	20786
b_{165}	5	11	35	20757
b_{166}	5	11	35	20229
b_{167}	5	11	35	19701
b_{168}	5	13	35	597807
b_{169}	5	13	35	597807
b_{170}	5	13	35	597507
b_{175}	5	13	35	583133
b_{180}	5	13	28	577368
b_{185}	5	13	28	570672
b_{190}	5	13	28	583829
b_{191}	5	13	28	583514
b_{192}	5	13	28	583514
b_{193}	6	14	21	859311
b_{194}	9	*out of memory*	89	*out of memory*
b_{195}	8	*out of memory*	466	*out of memory*

Table 1 shows that after 36 rounds (b_{164}), the symbolic expression of Grain-128a is of order 11 as opposed to 5 in the case of Grain-128, but even more striking is the difference in the number of terms. After the same 36 rounds, the number of terms of order 5 or greater is 35 for Grain-128 compared to 20786 for Grain-128a, which is over 590 times more.

An interesting fact is that the order of the expressions of Grain-128 stays the same for rounds 34 - 64. The reason is that the term of highest order is part of the pre-output function and the LFSR is filled with ones during the initialisation. The seven monomials of maximum order 6 in Grain-128's b_{161} are the following:

$$\boldsymbol{s_{95}} b_{45} b_{12} b_{95} (b_3 b_{67} + b_{11} b_{13} + b_{17} b_{18} + b_{27} b_{59} + b_{40} b_{48} + b_{61} b_{65} + b_{68} b_{84})$$

For the next 31 rounds, the bit s_{i+95} will have the value one, hence reducing the order to 5 and keeping it on that same level during those rounds. The growth of the number of terms of maximum order has the same cause. As the terms of maximum order are determined through those of the pre-output function and the monomial of order 3 is only of order 2 during those 31 rounds, there are five monomials of maximum order 2 in the pre-output function, as opposed to one of maximum order 3 for b_{161}.

Grain-128a's monomial with maximum order lies in the NFSR update:

$$b_{i+88}b_{i+92}b_{i+93}b_{i+95}$$

The smallest index reaches 128 exactly four rounds later than the other indices, which results in the stall of the order from b_{164} up to b_{168}. After that, the order of this term is determined by the sub-terms, i.e. the monomials in the expanded expressions of the variables, of order 3 and 4. No more indices reach 128 before b_{193}. This b_{193}, however, contains the term $b_{153}b_{157}b_{158}b_{160}$, and b_{160}, in turn, contains the bit b_{128} (b_{i+96} in the NFSR update) which results in the observed order 14.

4.1 Higher Order Differential Analysis on Grain-128 and Grain-128a

In order to investigate the impacts of the higher order and the higher number of terms of the symbolic expressions, a higher order differential attack is conducted with random cubes of different dimensions. For each dimension, sums over 100 random cubes are calculated and in each case the last round with a significant bias is identified. The number of random IVs used per cube is $2^{12} = 4096$ and the significance level for the frequency test is 0.001. The result, i.e. the last rounds with a bias for each dimension, is shown as a Boxplot in the following Fig. 4. Note that the upper boxes are the results of the analysis of Grain-128, whereas the lower boxes correspond to Grain-128a.

Not only are the sums biased up to many more rounds for Grain-128 than for Grain-128a, but the dimension of the cube also has a much bigger influence.

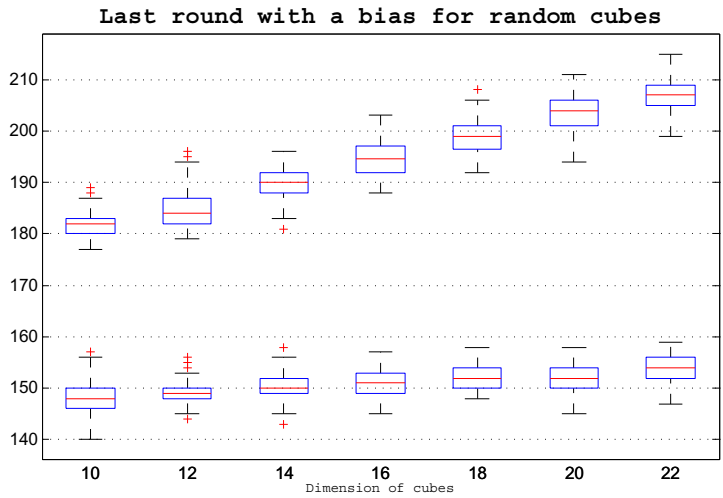


Fig. 4. Boxplot of last bias for random cubes

Cubes constructed by looking at the conditions of the cube bits and combining those with similar conditions, however, yield better results. For example, a random cube of dimension 20 results in a bias approximately up to round 152 without conditions (cf. Fig. 4) or up to round 160 with conditions, whereas the constructed cube of the same dimension results in a bias up until round 167.

4.2 Conditional Differential Analysis on Grain-128 and Grain-128a

To find the mentioned constructed cubes, we first symbolically sum over cubes of dimension one and examine the obtained symbolic expression of this sum over the cipher output z_i, thus getting the conditions of all public variables. The next step is to look for indices with the same conditions, as they seem to usually lead to good cubes. The reason is that the conditions generally do not occur at the same round and therefore one condition prevents that derivatives propagate at different rounds. Calculating the sum over v_{20}, for example, one finds the condition v_{27} at round 7 (z_7), whereas when calculating the sum over v_{34}, one finds the same condition at round 14 (z_{14}).

The following Table 2 lists the best results, i.e. the number of rounds attacked, on Grain-128 and Grain-128a for various cube dimensions, using 2048 random IVs and a significance level of 0.01:

Table 2. Best results for various cube dimensions on Grain-128 and Grain-128a

Cube dimension	12	16	20	24	28	33
Grain-128	207	215	219	225	231	236
Grain-128a	164	165	167	172	175	177

The best cube is of dimension 33 and results in a bias of approximately 0.463 at round 177. The public variables used as cube are the following:

$$v_1, v_2, v_3, v_{20}, v_{21}, v_{22}, v_{23}, v_{24}, v_{25}, v_{26}, v_{34}, v_{35}, v_{36}, v_{37}, v_{48}, v_{49}, v_{50},$$
$$v_{51}, v_{52}, v_{53}, v_{54}, v_{63}, v_{64}, v_{65}, v_{66}, v_{67}, v_{68}, v_{69}, v_{77}, v_{78}, v_{79}, v_{80}, v_{95}$$

If we apply the same cube to Grain-128, we find a bias of approximately 0.469 at round 236. In [2], the best cube found is of dimension 40 and results in a bias up to round 237. Furthermore, we find that the initialisation of Grain-128a is clearly much better as we find the last bias 59 rounds earlier with the same cube.

The results presented so far are achieved by imposing conditions only in the public variables. If we consider conditions in the private variables of Grain-128a as well, we get even better results in much less computing time. Evaluating the sum over the cube

$$v_{64}, v_{65}, v_{66}, v_{67}, v_{68}, v_{69}$$

of dimension 6, we find a significant bias at round 189. The conditions, i.e. the public and private variables set to a certain value, imposed here are the following:

$$v_{57}, v_{58}, v_{59}, v_{60}, v_{61}, v_{62}, v_{71}, v_{72}, v_{73}, v_{74}, v_{75}, v_{76}, v_{83}, v_{84}, v_{85}, v_{86}, v_{87}, v_{88}$$

$$k_{117}, k_{118}, k_{119}, k_{120}, k_{121}, k_{122}$$

Note that in order to attack the mentioned 189 rounds, all conditions have to be set to zero. This is the best attack we could find using conditional differential analysis. Based on the data collected and described in this section, we do not expect significantly better results to be found using this approach.

5 Conclusion

We evaluated the security of the stream cipher Grain-128a by comparing the monomial structure to its predecessor Grain-128. The significantly higher order of the symbolic expressions and the significantly higher number of monomials suggests that the dynamic cube attack, which breaks the predecessor Grain-128, is not applicable to Grain-128a. Additionally, we evaluated the security with respect to higher order differential attacks including conditional differential cryptanalysis. We used an automatic approach to find and analyse the conditions in terms of polynomial ideals. The attack of order 6 for 189 out of 256 rounds is the best attack we could find. Imposing conditions in only the public variables, the best attack found was of order 33 and attacked round 177. Consequently, Grain128a seems to have a comfortable security margin with respect to the approaches described in this paper.

Acknowledgements. We thank the reviewers of CANS 2012 for their helpful comments and Simon Knellwolf for his support and the contribution of the results with conditions in the private variables.

References

1. Ågren, M., Hell, M., Johansson, T., Meier, W.: Grain-128a: A New Version of Grain-128 with Optional Authentication. IJWMC 5(1), 48–59 (2011)
2. Aumasson, J.P., Dinur, I., Henzen, L., Meier, W., Shamir, A.: Efficient FPGA Implementations of High-Dimensional Cube Testers on the Stream Cipher Grain-128. Cryptology ePrint Archive, Report 2009/218 (2009), http://eprint.iacr.org/
3. Aumasson, J.-P., Dinur, I., Meier, W., Shamir, A.: Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 1–22. Springer, Heidelberg (2009)
4. Ben-Aroya, I., Biham, E.: Differential Cryptanalysis of Lucifer. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 187–199. Springer, Heidelberg (1994)
5. Dinur, I., Güneysu, T., Paar, C., Shamir, A., Zimmermann, R.: An Experimentally Verified Attack on Full Grain-128 Using Dedicated Reconfigurable Hardware. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 327–343. Springer, Heidelberg (2011)

6. Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009)
7. Dinur, I., Shamir, A.: Breaking Grain-128 with Dynamic Cube Attacks. Cryptology ePrint Archive, Report 2010/570 (2010)
8. Hell, M., Johansson, T., Maximov, A., Meier, W.: A Stream Cipher Proposal: Grain-128. In: ISIT, pp. 1614–1618 (2006)
9. Hell, M., Johansson, T., Meier, W.: Grain: A Stream Cipher for Constrained Environments. IJWMC 2(1), 86–93 (2007)
10. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 130–145. Springer, Heidelberg (2010)
11. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional Differential Cryptanalysis of Trivium and KATAN. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 200–212. Springer, Heidelberg (2012)
12. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Blahut, R.E., Costello, D.J., Maurer, U., Mittelholzer, T. (eds.) Communicationis and Cryptography: Two Sides of one Tapestry, pp. 227–233. Kluwer Academic Publishers (1994)
13. Vielhaber, M.: Breaking one.fivium by aida an algebraic iv differential attack. IACR Cryptology ePrint Archive 2007, 413 (2007)
14. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
15. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)

A Real-Time Key Recovery Attack on the Lightweight Stream Cipher A2U2*

Zhenqing Shi[1], Xiutao Feng[2], Dengguo Feng[1], and Chuankun Wu[3]

[1] Institute of Software, Chinese Academy of Sciences, Beijing, 100190, China
[2] Key Laboratory of Mathematics Mechanization, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, 100190, China
[3] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, 100195, China
zhenqingshi@gmail.com, fengxt@amss.ac.cn, feng@is.iscas.ac.cn, ckwu@iie.ac.cn

Abstract. The stream cipher A2U2 proposed by David et al. [7] is one of lightweight cipher primitives. In this paper we present a real-time key recovery attack on A2U2 under the known-plaintext-attack model, which only needs at most 210 consecutive ciphertext bits and its corresponding plaintext with the time complexity about $2^{24.7}$. Our result is much better than that of the attack proposed by M. Abdelraheem et al. in [9] whose complexity is $O(2^{49} \times C)$, where C is the complexity of solving a sparse quadratic equation system on 56 unknown key bits. Furthermore we provide a new approach to solving the above sparse quadratic equation system, which reduces the complexity C to a very small constant. Finally we do an entire experiment on a PC and recover all bits of a random key in a few seconds.

Keywords: Stream ciphers, A2U2, Lightweight ciphers, Key recovery attacks.

1 Introduction

Recently, Radio Frequency Identification (RFID) becomes more and more attractive [1]. For the security of RFID tag, cryptographic protection is required. The main problem is that the cryptographic primitives involved must be small and cheap enough to make the whole system practical. This limits many traditional cryptographic primitives like AES. So a new kind of cryptographic primitives called lightweight cipher primitives is proposed, for example PRESENT [2], KATAN/KTANTAN [3], Hummingbird [4], PRINT [5], Grain [6], and so on.

More recently, David et al. [7] proposed a new lightweight stream cipher called A2U2 in RFID'11. As the authors said, it is one of the most lightweight ciphers,

* This work was supported by the Natural Science Foundation of China (Grant No. 60833008, 60902024, 61121062), the 973 Program (Grant No.2007CB807902, 2011CB302401) and Foundation of President of the Chinese Academy of Sciences (Grant No. 50Y24103900).

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 12–22, 2012.

Table 1. Comparison of our attack and other known attacks on A2U2

Source	Attack Model	Time Complexity	Data(bits)
Our attack	known-plaintext-attack	$2^{24.7}$	≤ 210
M. Abdelraheem et al's attack [9]	known-plaintext-attack	$2^{49} \times C$	about 200
Q. Chai et al's attack [8]	chosen-plaintext-attack	–	638

and in one instance it only has 284 GE in hardware implementation. In [8], Q. Chai et al. proposed an ultra-efficient key recovery attack under the chosen-plaintext-attack model against A2U2, which requires two chosen plaintexts encrypted using the same key and initialization vector. In [9] M. Abdelraheem et al. presented a guess and determine attack under the known-plaintext-attack model against A2U2 with the time complexity $O(2^{49} \times C)$, where C is the complexity of solving a quadratic equation system on 56 unknown key bits. In this paper, we present another key recovery attack under the known-plaintext-attack model whose time complexity is about $2^{24.7}$, which is much better than that of the attack proposed by M. Abdelraheem et al. in [9]. Furthermore we present a new approach to solving the above sparse quadratic equation system, which reduces the complexity C to a very small constant. Our attack only needs at most 210 pairs of consecutive plaintext/ciphertext bits, and can recover all key bits on a PC in a few seconds. The comparison of our attack and other known attacks on A2U2 is showed in Table 1. It should be pointed that, comparing to Q. Chai et al's attack [8] under the chosen-plaintext-attack model, our attack under the known-plaintext-attack model is a stronger attack more likely to occur.

The rest of the paper is organized as follows: in Section 2 we recall the A2U2 algorithm briefly, and in Section 3 we give some basic properties of A2U2. In Section 4 we describe all the details of our attack method on A2U2, and finally conclude the paper in Section 5.

2 Description of A2U2

In this section we recall the A2U2 algorithm briefly and all the details can be found in [7,8]. Here we use different notations from that in [7,8], which are more suitable for our cryptanalysis.

A2U2 has a key size of 61 bits and is composed of four blocks: a counter (7 bits), two NFSRs (17 bits and 9 bits, resp.), a key register (56 bits), and a filter function. The structure of A2U2 is illustrated in Fig.1. In the rest of the paper, we denote an XOR operation by"+", and an AND operation by "·".

2.1 The Counter

The counter is a 7-stage Linear Feedback Shift Register (LFSR), and its feedback function $F(X) = X^7 + X^4 + 1$ is a primitive polynomial over the binary field F_2 (whose period is $2^7 - 1$). Below we denote the state of the LFSR at time i by a binary vector $(t_{i+6}, ..., t_i) \in F_2^7$, $i \geq 0$.

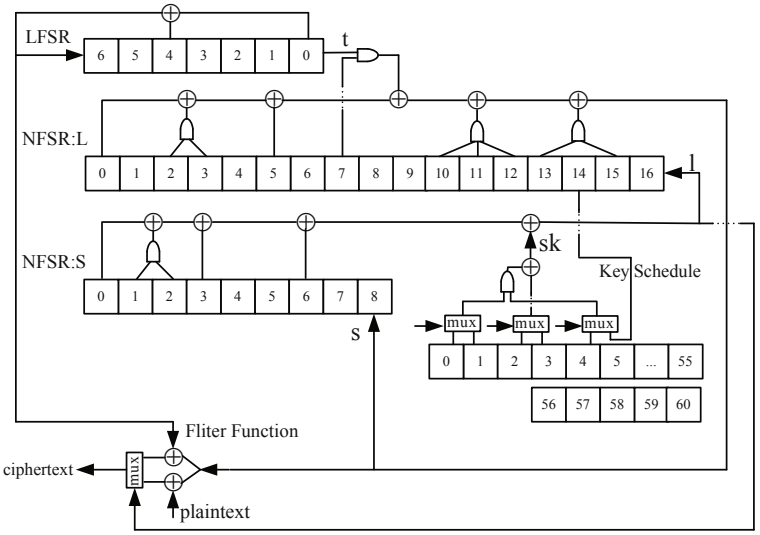


Fig. 1. The structure of the Stream Cipher A2U2

2.2 The NFSRs L and S

The design of the nonlinear parts of A2U2 is highly inspired by the block cipher KATAN [3], which introduces a new combination of two NFSRs, namely L and S, where the feedback function of each NFSR provides a feedback to the other. The states of the NFSRs L and S at time i are denoted by a binary vector $(l_{i+16}, ..., l_i)$ and $(s_{i+8}, ..., s_i)$ respectively. At time i, the feedback functions of L and S are defined by the nonlinear Boolean functions given in (1) and (2) respectively:

$$\begin{aligned} s_{i+9} &= l_i + l_{i+2} \cdot l_{i+3} + l_{i+5} + l_{i+7} \cdot t_i + \\ &\quad l_{i+10} \cdot l_{i+11} \cdot l_{i+12} + l_{i+13} \cdot l_{i+15}, \qquad (1) \\ l_{i+17} &= s_i + s_{i+1} \cdot s_{i+2} + s_{i+3} + s_{i+6} + sk_i, \qquad (2) \end{aligned}$$

where sk_i is the subkey bit generated by the key schedule.

2.3 The Key Schedule

The key of A2U2 has 61 bits, and the front 56 bits among them are used to generate the subkey and the last 5 bits are reserved for the counter initialization. At time i, five of 56 key bits are combined with three bits of the counter and one bit of the NFSR L to generate the subkey sk_i as follows:

$$\begin{aligned} sk_i &= \mathrm{mux}(k_{5i \bmod 56}, k_{(5i+1) \bmod 56}, t_{i+1}) \cdot \\ &\quad \mathrm{mux}(k_{(5i+4) \bmod 56}, l_{i+14}, t_{i+5}) + \\ &\quad \mathrm{mux}(k_{(5i+2) \bmod 56}, k_{(5i+3) \bmod 56}, t_{i+3}), \qquad (3) \end{aligned}$$

where mux() is a multiplexer, that is, for any given $x, y, z \in F_2$, $\text{mux}(x, y, z) = x$ if $z = 0$, otherwise $\text{mux}(x, y, z) = y$.

2.4 The Filter Function

The final building block of A2U2 is a filter function, named the shrinking filter in reference to the clock controlled generator design of the Shrinking Generator [10]. Without loss of generality, we suppose the cipher begins to output ciphertext at time 0. The filter is defined by (4):

$$c_i = \text{mux}(s_{i+8} + t_{i+6}, s_{i+8} + p_{f(i)}, l_{i+16}), \tag{4}$$

where p_i, c_i are the plaintext bit and ciphertext bit at time i respectively, and $f(i) = \sum_{j=0}^{i-1} l_{j+16}$ for $i \geq 1$, $f(0) = 0$.

2.5 Initialization

The initial value of the counter and two NFSRs are determined by the secret key and two 32-bit pseudo-random numbers generated by the tag and the reader respectively, see [7] for the details. After the state registers of the counter and two NFSRs are filled up, the counter and two NFSRs will clock without any output until the counter has the state of all 1s.

3 Some Properties of A2U2

In this section we give some basic properties of A2U2 which will be used in the next section.

3.1 On the States of the Counter

Note that after the initialization the counter has the state of all 1s, since the counter simply works by self and the non-zero sequences generated by the counter are m-sequences of period 127, so we can know the state of the counter at any time after the cipher begins to output. A fragment of the sequences generated by the counter after the initialization is listed in Table 2, namely, $(t_0, t_1, \cdots, t_{127})$.

Table 2. One period of the sequence$\{t_i\}_{i\geq 0}$

1	1	1	1	1	1	1	0	0	0	1	1	1	0	1	1	0	0	0	1	0	1	0	0	1	0	1	1	1	1	1	0	1	0	1	0	1	0	0	0	0	1	0
1	1	0	1	1	1	1	0	0	1	1	1	0	0	1	0	1	0	1	1	0	0	1	1	0	0	0	0	0	1	1	0	1	1	0	1	0	1	1	1	0	1	0
0	0	1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	0	1	1	1	0	0	0	0		

3.2 On Two NFSRs

By (1) we have

$$\begin{aligned} l_i &= s_{i+9} + l_{i+2} \cdot l_{i+3} + l_{i+5} + l_{i+7} \cdot t_i + \\ &\quad l_{i+10} \cdot l_{i+11} \cdot l_{i+12} + l_{i+13} \cdot l_{i+15}, \end{aligned} \tag{5}$$

which shows that when $l_{i+j}(j = 2, 3, 5, 7, 10, 11, 12, 13, 15)$ and s_{i+9} are known, we can deduce l_i. This property makes it possible to deduce the previous state backward by the current state.

What's more, by (2) we have

$$sk_i = l_{i+1+16} + s_i + s_{i+1} \cdot s_{i+2} + s_{i+3} + s_{i+6}, \tag{6}$$

which shows that we can recover the subkey when the state of two NFSRs L and S are known.

3.3 On the Key Schedule

By (3), when $sk_i, t_{i+j}(j = 1, 3, 5)$ and l_{i+14} are known, we can collect an equation on the unknown key bit variables. When $t_{i+5} = 1$, it is easy to check that the collected equation is linear, and furthermore, if $l_{i+14} = 0$, $k_{(5i+2) \bmod 56}$ or $k_{(5i+3) \bmod 56}$ will be obtained directly from the collect equation. When the number of the collected equations is large enough, all 56 key bit variables can be solved out uniquely from those equations.

3.4 On the Filter Function

By (4) we have

$$s_{i+8} = \mathrm{mux}(c_i + t_{i+6}, c_i + p_{f(i)}, l_{i+16}). \tag{7}$$

So, if we know $f(i)$ and l_{i+16}, we can deduce s_{i+8} under the known-plaintext-attack model.

4 Real-Time Attack on A2U2

In this section we present a real-time attack on A2U2 under the known-plaintext-attack model. We suppose that an attacker captured a fragment of ciphertext $c_0, ..., c_n$ from the starting time 0, and he had known their corresponding plaintext $p_0, ..., p_{n'}$, where n and n' are two positive integers such that $n' \leq n$, and n is required to be large enough to execute our attack successfully. Our attack is divided into the following three phases:

4.1 Recover l_i $(0 \leq i \leq n)$ and s_i $(8 \leq i \leq n+8)$

At first we guess the values of l_i $(n \leq i \leq n+16)$ and $f(n)$. Because all t_i are known in A2U2's work stage, we first deduce s_{n+8} by (7), and then l_{n-1} by (5). Since $f(i) = f(i+1) - l_{i+16}$ for $i \geq 0$, repeating the above procedure for $i = n-1, n-2, \cdots, 0$, we can recover all the states of two NFSRs L and S. More precisely, the above procedure can be described as follows:

For $i = n-1, ..., 0$, repeat the following 3 steps:

1. $f(i) = f(i+1) - l_{i+16}$;
2. $s_{i+8} = \text{mux}(c_i + t_{i+6}, c_i + p_{f(i)}, l_{i+16})$;
3. $l_i = s_{i+1+8} + l_{i+2} \cdot l_{i+3} + l_{i+5} + l_{i+7} \cdot t_i + l_{i+10} \cdot l_{i+11} \cdot l_{i+12} + l_{i+13} \cdot l_{i+15}$.

The above whole process is described in Algorithm 1.

Algorithm 1. Recover $l_i (0 \leq i \leq n)$ and $s_i (8 \leq i \leq n+8)$

Input: l_i $(n \leq i \leq n+16)$ and $f(n)$.

Output: l_i $(0 \leq i \leq n)$ and s_i $(8 \leq i \leq n+8)$, or error information.

1: **if** $f(n) > n$ or $f(n) < 0$ **then**
2: Return 0 (error information);
3: **else** $\{0 \leq f(n) \leq n\}$
4: **if** $l_{n+16} = 0$ **then**
5: $s_{n+8} = c_n + t_{n+6}$;
6: **else** $\{l_{n+16} = 1\}$
7: $s_{n+8} = c_n + p_{f(n)}$;
8: **end if**
9: **end if**
10: $f(n-1) = f(n) - l_{n+15}$;
11: **for** $i = n-1$ to 0 **do**
12: **if** $f(i) > i$ or $f(i) < 0$ **then**
13: Return 0 (error information);
14: **else** $\{0 \leq f(i) \leq i\}$
15: **if** $l_{i+16} = 0$ **then**
16: $s_{i+8} = c_i + t_{i+6}$;
17: **else** $\{l_{i+16} = 1\}$
18: $s_{i+8} = c_i + p_{f(i)}$;
19: **end if**
20: $f(i) = f(i+1) - l_{i+15}$;
21: $l_i = s_{i+1+8} + l_{i+2} \cdot l_{i+3} + l_{i+5} + l_{i+7} \cdot t_i + l_{i+10} \cdot l_{i+11} \cdot l_{i+12} + l_{i+13} \cdot l_{i+15}$;
22: **end if**
23: **end for**
24: Output l_i $(0 \leq i \leq n)$ and s_i $(8 \leq i \leq n+8)$.

Remark 1. *At Line 1 and Line 12 of Algorithm 1, when we get $f(i) > i$ or $f(i) < 0$, it shows that the guessed values of l_i $(n \leq i \leq n+16)$ or $f(n)$ are incorrect. Then we will try another possible values of l_i $(n \leq i \leq n+16)$ or $f(n)$.*

Remark 2. *By the definition of $f(\cdot)$, it follows that two sequences $\{l_i\}_{i\geq 0}$ and $\{f(i)\}_{i\geq 0}$ are one-to-one. The further experiments show that for each state l_i $(i = n, n+1, ..., n+16)$, only a few values of $f(n)$ can survive in Algorithm 2.*

Algorithm 2. Recover $f(n)$

Input: $l_i (n \leq i \leq n+16)$
Output: $\{f(n)\}$

```
1: A = ∅.
2: for 0 ≤ f(n) ≤ n do
3:    Invoke Algorithm 1;
4:    if Algorithm 1 outputs l_i's and s_i's then
5:       A = A ∪ {f(n)};
6:    end if
7: end for
8: Return A.
```

Remark 3. *Suppose the value of each register of the NFSR L is uniformly distributed, then $E(f(i)) = \frac{i}{2}$, where $E(\cdot)$ is the expectation function. So we should first guess $f(n) = \lfloor \frac{n}{2} \rfloor$ in practice, if it is incorrect, then guess $f(n) = \lfloor \frac{n}{2} \rfloor \pm 1, \lfloor \frac{n}{2} \rfloor \pm 2, ...$ until we get the correct one.*

4.2 Recover the Key k_i $(i = 0, 1, ..., 55)$

In the previous section, we have recovered the internal states of the NFSRs L and S, that is, l_i $(i = 0, 1, ..., n+16)$, s_i $(i = 8, 9, ..., n+8)$. By (6), we further deduce all subkeys sk_i $(i = 8, , 9, ..., n-1)$. So we can collect $n-8$ equations of 56 key bit variables by (3):

$$\begin{aligned}
k_{40} \cdot k_{44} + k_{43} &= sk_8 \\
k_{46} \cdot l_{23} + k_{48} &= sk_9 \\
k_{51} \cdot l_{24} + k_{52} &= sk_{10} \\
k_0 \cdot k_3 + k_2 &= sk_{11} \\
k_4 \cdot k_8 + k_7 &= sk_{12} \\
k_{10} \cdot k_{13} + k_{11} &= sk_{13} \\
k_{15} \cdot l_{28} + k_{16} &= sk_{14} \\
k_{19} \cdot k_{23} + k_{21} &= sk_{15} \\
&... \\
\end{aligned}$$

In [8] Q. Chai et al. proposed a method for recovering k_i $(i = 0, 1, ..., 55)$: first collect 56 linearly independent linear equations and then get k_i $(i = 0, 1, ..., 55)$ by solving this linear equation system. Based on their testing experiments, they claimed that 56 linearly independent equations about k_i $(i = 0, 1, ..., 55)$ were got with probability 1 when $n \geq 512$.

Here we present a new approach to getting k_i $(i = 0, 1, ..., 55)$. It is noticed that when $t_{i+5} = 1$, we get a linear equation of the form $k_x \cdot l_y + k_z = sk_w$, where x, y, z and w are positive integers. By the properties of the m-sequence $\{ t_i \}_{i \geq 0}$, it is known that about half of t_is are equal to one, so about half of the above equation system are linear. In particular, when $l_y = 0$, we can get k_z directly. Next we take all the known k_z into the rest of equations, and we will get some new k_z's further. Repeat the above process until we can not get new k_z any more. In this process, each equation can be used at most one time. And once some equation is used, one of the following two situations will occur: (1) get some new k_z's directly from this equation; (2) when the k_z's in this equation are all known, we can use this equation to determine our guess, that is: if this equation does not hold, it shows that our guess is incorrect.

In order to determine how many candidate keys we can recover by means of our new approach and how many key bits we can recover for each candidate key, an experiment is done as follows:

Experiment

1. Choose a pair of a random key $(k_0, k_1, ..., k_{60})$ and a random plaintext string $(p_0, p_1, ..., p_{n'})$;
2. Encrypt the plaintext $(p_0, p_1, ..., p_{n'})$ with the key $k_0, k_1, ..., k_{60}$ and get the ciphertext $(c_0, c_1, ..., c_n)$;
3. Use the plaintext $(p_0, p_1, ..., p_{n'})$ and its corresponding ciphertext $(c_0, c_1, ..., c_n)$ to recover k_is by means of our new approach mentioned above;
4. Count how many candidate keys we could get, and how many key bits we could recover among all k_is for each candidate key.

In practice the length n of the ciphertext is a key parameter to determine the number of candidate keys and the number of key bits for each candidate key in average. In order to demonstrate the relation of the parameter n and the number of candidate keys in average, we did the following three tests:

Test 1. For each $n(n = 120, 125, 130, ..., 210)$(where the step is 5.), we did the above experiment 2000 times, where the key $(k_0, k_1, ..., k_{60})$ and the plaintext $(p_0, p_1, ..., p_{n'})$ were taken randomly each time, and counted the number of candidate keys and the number of key bits for each candidate key in average, which were listed in Table 3. From Table 3, it is seen that there are more than one candidate keys recovered when n is small, and there is about one candidate key recovered when n is no less than 205.

Test 2. For each $n(n = 203, 204, 205, ..., 212)$ (where the step is 1.), we did the same experiment as **Test 1** 10000 times, and the results were listed in Table 4.

Test 3. For a fixed random key, we did the above experiment 200000 times for each n $(n = 203, 204, ..., 212)$, where the plaintext were taken randomly. We got the results similarly to those of Table 4.

By Tests 1, 2 and 3, in order to get exactly one candidate key, we should take $n \geq 205$ in practice.

Table 3. The number of valid recovered k_i's for $n = 120, 125, ..., 210$ in average

n	the average num. of candidate keys	the average num. of valid recovered k_i's
120	1.0540	55.401919
125	1.0360	55.420974
130	1.0270	55.557312
135	1.0240	55.601034
140	1.0195	55.777562
145	1.0170	55.805874
150	1.0155	55.829585
155	1.0150	55.834000
160	1.0125	55.870905
165	1.0105	55.908321
170	1.0100	55.899753
175	1.0080	55.935500
180	1.0050	55.928000
185	1.0045	55.963147
190	1.0030	55.968531
195	1.0020	55.988072
200	1.0005	55.999000
205	1.0000	56.000000
210	1.0000	56.000000

4.3 Recover the Key $k_i (i = 56, 57..., 60)$

After the k_i $(i = 0, 1, ..., 55)$ are recovered, we will go on to recover k_i $(i = 56, 57..., 60)$. There are three approaches to recovering k_i $(i = 56, 57..., 60)$: one in [8], two in [9]. Here we adopt the approach in [8] where the correct k_i $(i = 56, 57..., 60)$ will lead to the same values of l_i $(0 \leq i \leq n)$ and s_i $(8 \leq i \leq n + 8)$ as recovered above. And the time complexity is 2^5.

4.4 Time Complexity of Our Attack

Our attack is subdivided into three phases. In Phase 1 (i.e., Section 4.1) we guess the values of $l_{n+i} (i = 0, 1, ..., 16)$ (17 bits) and $f(n) (0 \leq f(n) \leq n)$. So its time complexity is $O(2^{17} \times (n + 1))$. Since only a few values of $f(n)$ can survive for each state $l_{n+i} (i = 0, 1, ..., 16)$, the time complexity in Phase 2 (i.e., Section 4.2) is $O(2^{17} \times K \times C)$, where K is the number of $f(n)$ survived in Algorithm 2 such that $K \ll n + 1$, and C is the time complexity of solving the key bits

Table 4. The number of valid recovered k_i's for $n = 203, 204, ..., 212$ in average

n	the average num. of candidate keys	the average num. of valid recovered k_i's
203	1.0002	55.99920016
204	1.0001	55.99960004
205	1.0000	56.00000000
206	1.0000	56.00000000
207	1.0000	56.00000000
208	1.0000	56.00000000
209	1.0000	56.00000000
210	1.0000	56.00000000
211	1.0000	56.00000000
212	1.0000	56.00000000

k_i $(0 \leq i \leq 55)$ from the equation system. In Phase 3 we recover the key k_i $(56 \leq i \leq 60)$ with time complexity 2^5. Summing the above time complexity of all phases, we get the total time complexity is $O(2^{17} \times (n+1) + 2^{17} \times K \times C + 2^5)$.

In our attack we choose $n = 210$. This is because almost all $k_i (i = 0, ..., 55)$ can be obtained directly in this case by Tables 3 and 4. Thus C is not large, and we have $2^{17} \times K \times C < 2^{17} \times (n + 1)$. So the time complexity of our attack is about $2^{17} \times 210 \approx 2^{24.7}$, which is very low and can be done successfully on a PC in a few seconds.

5 Conclusion

In this paper we presented a real-time key recovery attack under the known-plaintext-attack model against the stream cipher A2U2, which only needs at most 210 consecutive ciphertext bits and its corresponding plaintext to recover all key bits with time complexity $2^{24.7}$.

Acknowledgement. The authors gratefully acknowledge the anonymous referees, whose comments helped to improve the presentation.

References

1. Finkenzeller, K.: Introduction. In: RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication, 3rd edn., ch. 1 (2010)
2. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)

3. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
4. Engels, D., Fan, X., Gong, G., Hu, H., Smith, E.M.: Hummingbird: Ultra-Lightweight Cryptography for Resource-Constrained Devices. In: Sion, R., Curtmola, R., Dietrich, S., Kiayias, A., Miret, J.M., Sako, K., Sebé, F. (eds.) FC 2010 Workshops. LNCS, vol. 6054, pp. 3–18. Springer, Heidelberg (2010)
5. Knudsen, L., Leander, G., Poschmann, A., Robshaw, M.J.B.: PRINTcipher: A Block Cipher for IC-Printing. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 16–32. Springer, Heidelberg (2010)
6. Hell, M., Johansson, T., Meier, W.: Grain: a stream cipher for constrained environments. International Journal of Wireless and Mobile Computing 2(1), 86–93 (2007)
7. David, M., Ranasinghe, D.C., Larsen, T.: A2U2: a stream cipher for printed electronics RFID tags. In: IEEE International Conference on RFID 2011, pp. 173–183 (2011)
8. Chai, Q., Fan, X., Gong, G.: An Ultra-Efficient Key Recovery Attack on the Lightweight Stream Cipher A2U2, IACR Cryptology ePrint Archive, p. 247 (2011)
9. Abdelraheem, M.A., Borghoff, J., Zenner, E., David, M.: Cryptanalysis of the Light-Weight Cipher A2U2. In: Chen, L. (ed.) Cryptography and Coding 2011. LNCS, vol. 7089, pp. 375–390. Springer, Heidelberg (2011)
10. Coppersmith, D., Krawczyk, H., Mansour, Y.: The Shrinking Generator. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 22–39. Springer, Heidelberg (1994)

A Simple Key-Recovery Attack on McOE-X

Florian Mendel[1,2], Bart Mennink[1], Vincent Rijmen[1], and Elmar Tischhauser[1]

[1] Katholieke Universiteit Leuven, ESAT/COSIC and IBBT, Belgium
[2] Graz University of Technology, IAIK, Austria

Abstract. In this paper, we present a key-recovery attack on the online authenticated encryption scheme McOE-X proposed by Fleischmann et al. at FSE 2012. The attack is based on the observation that in McOE-X the key is changed for every block of message that is encrypted in a deterministic way. This allows an adversary to recover the key by using a standard time-memory trade-off strategy. On its best setting the attack has a complexity as low as $2 \cdot 2^{n/2}$, while this should be 2^n for a good scheme. Taking AES-128 as an example this would result in an attack with complexity of 2^{65}.

Keywords: authenticated encryption, McOE-X, key-recovery attack.

1 Introduction

Motivation. Authenticated encryption is an important part in information security. Whenever two parties communicate over a network an authenticated encryption algorithm might be used to provide both privacy and authentication of the data. In most applications, there is not much value in keeping the data secret if they are not authenticated. Authentication of data is often of more value than their confidentiality.

Authenticated encryption can be generically constructed by combining an encryption scheme and a MAC. In [3], Bellare and Namprempre analyzed the three generic compositions of these two primitives: MAC-then-Encrypt (MtE), Encrypt-then-MAC (EtM), and Encrypt-and-MAC (E&M). They showed that the strongest notion of security for authenticated encryption can only be achieved by the EtM approach. However, schemes built from generic composition have some disadvantages. Besides that two different algorithms with two different keys are needed, the message needs to be processed twice, making the scheme impractical for some applications. Therefore, ISO/IEC specifies, next to the generic composition EtM, five authenticated encryption modes for block ciphers, namely OCB, SIV (Key Wrap), CCM, EAX, and GCM. Most of them are much faster than any solution which uses generic composition. All of them are proven to be secure against nonce-respecting adversaries assuming that the underlying block cipher is ideal. However, as pointed out in [8,9] all these schemes, excluding SIV, are vulnerable to nonce-reusing adversaries. SIV has been explicitly designed to resist nonce-reuse attacks, but it has the disadvantage that it is inherently

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 23–31, 2012.

offline. For encryption one must either keep the entire message in memory or read the message twice.

Therefore, Fleischmann et al. proposed a new family of authenticated encryption schemes in [8,9] that are on the one hand secure against nonce-reusing adversaries and on the other hand are online. The construction extends the online encryption scheme TC3 by Rogaway and Zhang [16] to a provable secure nonce-reuse resistant online authenticated encryption scheme. The family consists of three members: McOE-X, McOE-D and McOE-G.

Our Contribution. In this paper, we present a key-recovery attack on McOE-X. The basic idea of the attack is very simple. Since in McOE-X the key is changed for every block of message that is encrypted, an adversary can recover the key by keeping the message input of some block cipher operation fixed and using a time-memory trade-off strategy. In its best setting the attack has a complexity as low as $2 \cdot 2^{n/2}$ with similar memory requirements, while this should be 2^n in the ideal case. Our attack allows a free trade-off between memory (precomputation) and time (online phase), and as such can be tailored to different attack scenarios. In all variants, it is significantly more efficient than Hellman's generic time-memory trade-off.

Note that this is close to the security bound of the McOE family. In more detail, Fleischmann et al. provide a formal security proof, which guarantees CCA3 indistinguishability up to about $2^{n/2}$. Since our key-recovery attack on McOE-X matches this bound (and from a theoretical point of view it even invalidates the proof), we took a detailed look at their security proof and identified a severe mistake that causes this gap: at a high level, Fleischmann et al. use ideal cipher results as if they were standard model results. These issues with the proof can, however, be resolved by explicitly considering the ideal cipher model.

Outline. The remainder of the paper is organized as follows. Section 2 describes the generic McOE construction and in particular McOE-X. In Section 3, we recall the security claims of the McOE construction respectively McOE-X. We present our key-recovery attack on McOE-X in Section 4 and discuss its relation to the security proof of McOE-X in Section 5. Finally, we discuss how McOE-X might be fixed in Section 6.

2 The McOE Family

The McOE construction is a new family of online authenticated encryption schemes recently proposed by Fleischmann et al. [8,9]. It consists of three members: McOE-X, McOE-D and McOE-G. The general structure follows the online permutation approach described by Bellare et al. in [1] and is based on the Tweak Chain Hash construction [12] that is adapted from the Matyas-Meyer-Oseas construction. To be more precise, the construction itself is built on the online encryption scheme TC3 recently proposed by Rogaway and Zhang in [16]

that is based on a tweakable block cipher $\tilde{E}$. With an additional overhead of only two invocations of the tweakable block cipher $\tilde{E}$ the authors extend it to an online authenticated encryption scheme that is also secure against nonce-reusing adversaries. Both the encryption/authentication and the decryption/verification operations are described in Figure 1.

Encryption/Authentication $\mathcal{E}(K, V, M)$

1: *Partition* M *into* $M_1 \cdots M_L$
2: $U \leftarrow 0^n$
3: $\tau \leftarrow \tilde{E}(K, U, V)$
4: $U \leftarrow \tau \oplus V$
5: **for** $i = 1$ to L **do**
$C_i \leftarrow \tilde{E}(K, U, M_i)$
$U \leftarrow M_i \oplus C_i$
6: $T \leftarrow \tilde{E}(K, U, \tau)$
7: $C \leftarrow C_1 \cdots C_L$
8: **return** (C, T)

Decryption/Verification $\mathcal{D}(K, V, C, T)$

1: *Partition* C *into* $C_1 \cdots C_L$
2: $U \leftarrow 0^n$
3: $\tau \leftarrow \tilde{E}(K, U, V)$
4: $U \leftarrow \tau \oplus V$
5: **for** $i = 1$ to L **do**
$M_i \leftarrow \tilde{E}^{-1}(K, U, C_i)$
$U \leftarrow M_i \oplus C_i$
6: $T' \leftarrow \tilde{E}(K, U, \tau)$
7: $M \leftarrow M_1 \cdots M_m$
8: **if** $T = T'$ **return** M
else return FAIL

Fig. 1. Encryption/Authentication and Decryption/Verification operation of the McOE construction, where $\tilde{E}(K, U, \cdot)$ is a tweakable block cipher with key K and tweak U. Furthermore, M denotes the message, C denotes the ciphertext, V denotes the nonce and T is the authentication tag.

Additionally, Fleischmann et al. proposed a second scheme to provide length preservation using tag-splitting. The concept of tag-splitting is very similar to ciphertext stealing. We refer to the specification [9] for a detailed description of this method, since we do not need it for the attack described in this paper.

The generic construction of the McOE family with and without tag-splitting is depicted in Figure 2, where $\tilde{E}$ denotes a n-bit tweakable block cipher and V is a nonce. Due to the current lack of a dedicated n-bit block cipher with an n-bit tweak, Fleischmann et al. proposed three different constructions to convert an ordinary block cipher into a tweakable block cipher resulting in the three members of the McOE family: McOE-X, McOE-D and McOE-G.

2.1 McOE-X

In this instance of the McOE family the tweak U (i.e. the chaining value) is xored to the key K to turn the block cipher E into a tweakable block cipher

$$\tilde{E}(K, U, \cdot) := E(K \oplus U, \cdot) \tag{1}$$

As noted by the designers for McOE-X related-key security is needed for the block cipher E. However, this requirement is not needed for the other two instances of the McOE family.

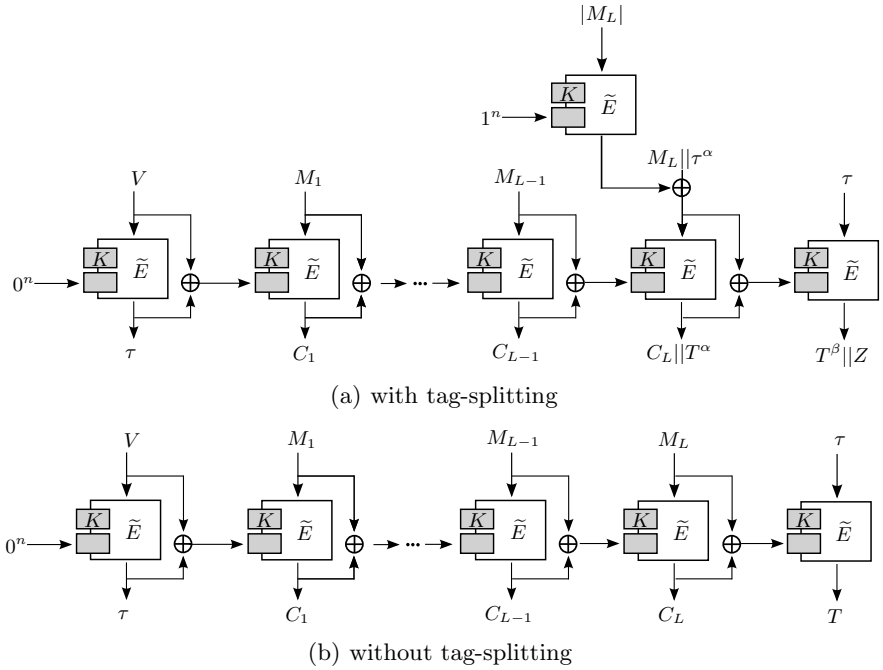


Fig. 2. Outline of the generic McOE construction (a) with and (b) without tag-splitting [9]

2.2 Other Members of the McOE Family

In addition to McOE-X, Fleischmann et al. proposed two other members of the McOE family not requiring related-key security, McOE-D and McOE-G. The first uses two block cipher invocations per message block to update the chaining value similar to the TCH-CBC construction as described in [5]. The second is based on the HCBC2 construction described in [2] and uses a universal hash function to update the chaining value. For a detailed description of the two schemes we refer to the specification [9].

3 Security of the Schemes

Fleischmann et al. [9] analyze their McOE schemes with respect to CCA3 security, a security notion for authenticated cryptosystems proposed in [15]. We informally describe the security definitions, referring to [9] for a more formal treatment. For an authenticated cryptosystem $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, where $\mathcal{K}$ denotes the key derivation function, we denote by $\mathbf{Adv}_{\Pi}^{\mathrm{CCA3}}(q, \ell, t)$ the CCA3 security of Π against any nonce-reusing adversary A, where q denotes the number of total queries an adversary A is allowed to ask to $\mathcal{E}$ and $\mathcal{D}$, ℓ the total length in blocks of the queries, and t the running time of A.

They derive the following result for the McOE schemes without tag-splitting.

Theorem 1 (Thm. 2 of [9]). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a McOE scheme based on a tweakable block cipher $\tilde{E}$. Then,*

$$\boldsymbol{Adv}_{\Pi}^{CCA3}(q, \ell, t) \leq \frac{3(q+\ell)(q+\ell+1) + 4q + 3\ell}{2^n - (q+\ell)} + 3\delta.$$

Here, $\delta = \boldsymbol{Adv}_{\tilde{E}}^{IND}(q+\ell, O(t))$ denotes the advantage of distinguishing $\tilde{E}$ from an ideal tweakable block cipher, where $q+\ell$ denotes the number of queries an adversary A is allowed to ask to $\tilde{E}$ and $\tilde{E}^{-1}$ and $O(t)$ the running time of A.

A significantly worse bound is obtained for McOE with tag-splitting. We refer to [9] for more details.

For the McOE-X construction without tag-spitting, they generalize this result as follows. Note that in this case δ refers to the related-key advantage of distinguishing E from an ideal block cipher.

Theorem 2 (Thm. 4 of [9]). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a McOE-X scheme based on a block cipher E. Then,*

$$\boldsymbol{Adv}_{\Pi}^{CCA3}(q, \ell, t) \leq \frac{3(q+\ell)(q+\ell+1) + 4q + 3\ell}{2^n - (q+\ell)} + 3\delta.$$

Here, $\delta = \boldsymbol{Adv}_{E}^{RK}(2q+\ell, O(t))$ denotes the related-key advantage of distinguishing E from an ideal block cipher, where $2q+\ell$ denotes the number of queries an adversary A is allowed to ask to E and E^{-1} and $O(t)$ the running time of A.

Again, the bound is slightly worse in case of tag-splitting and we refer to [9] for more details.

4 Our Attack on McOE-X

In this section, we propose our simple key-recovery attack on McOE-X. The attack consists of two phases: an offline (precomputation) phase and an online phase. It is a chosen plaintext attack and in its best setting it has a complexity as low as $2 \cdot 2^{n/2}$ with similar memory requirements.

4.1 Basic Attack

The basic idea of our attack can be explained as follows. The McOE-X mode changes the key for every block of plaintext that is encrypted. By keeping the plaintext input of some block cipher operation fixed, the adversary can exploit a basic time-memory trade-off strategy.

Let $E(k, x)$ denote the raw block cipher encryption operation with key k and plaintext x, and denote by K the target key we want to recover. Since the nonce plays no role in our attack, we omit it from the notation. The attack goes as follows.

Offline Phase (Precomputation)

1. Choose an arbitrary value a.
2. Repeat r times:
   (a) Choose a new value for k.
   (b) Compute $b = E(k, a)$ and save the pair (b, k) in a list L_1.

Online Phase

Repeat $2^n/r$ times:

1. Choose a new value for x and set $m = x\|a$.
2. Ask for the ciphertext/tag pair $(c, T) = \text{McOE-X}(K, m)$, with $c = C_1\|C_2$, and save the pair $(x \oplus C_1, C_2)$ in a list L_2.

Every match between a b-value in the list L_1 and a C_2-value in the list L_2 gives a candidate key $K = k \oplus x \oplus C_1$. We have set the number of iterations such that the expected number of matches between the two lists equals 1. Since the expected number of false alarms is small, we can state that the algorithm finds the correct key with a total complexity of approximately $r + 2^n/r$ encryptions.

Note that the queries in the online phase can be grouped. The adversary can ask for the encryption of $m = x\|a\|a\|\ldots\|a$ and save in L_2 the pairs $(x \oplus C_1, C_2), (a \oplus C_2, C_3), (a \oplus C_3, C_4), \ldots$. In this way the total number of block cipher encryptions is reduced.

Obviously by choosing $r = 2^{n/2}$ the attack has the best overall complexity, considering both the offline and the online phase, resulting in a final attack complexity of about $2 \cdot 2^{n/2}$ and similar memory requirements. We want to note that in the online phase of the attack we do not need to store the values in a list L_2 which reduces the memory requirements of the attack.

Sometimes an attacker wants to recover more than only a single key. In these cases only the second phase of the attack has to be repeated, while the precomputation phase has to be done only once. In such settings, in particular if the number of attacked keys is large, other values of r might result in a better overall complexity. In Table 1 we give the complexities and memory requirements for different choices of r.

Table 1. Complexities and memory requirements for both phases of the attack with different choices of r

$\log_2(r)$	offline phase	online phase	memory	total
$n/4$	$2^{n/4}$	$2^{3n/4}$	$2^{n/4}$	$2^{3n/4}$
$n/3$	$2^{n/3}$	$2^{2n/3}$	$2^{n/3}$	$2^{2n/3}$
$n/2$	$2^{n/2}$	$2^{n/2}$	$2^{n/2}$	$2 \cdot 2^{n/2}$
$2n/3$	$2^{2n/3}$	$2^{n/3}$	$2^{2n/3}$	$2^{2n/3}$
$3n/4$	$2^{3n/4}$	$2^{n/4}$	$2^{3n/4}$	$2^{3n/4}$

4.2 A Memory-Less Variant of the Attack

In practice, there is a profound imbalance between the cost of storage and the cost of computations. Hence, the high memory requirements of the attack could be seen as the bottleneck of the attack. It is therefore important to note that the attack with $r = 2^{n/2}$ can be implemented with negligible memory requirements and only a small increase in runtime by using a memory-less variant of the meet-in-the-middle attack introduced by Quisquater and Delescaille [14].

4.3 Comparison to Hellman's TMTO Attack

In [10] Hellman described a generic cryptanalytic TMTO attack on DES. Even though the attack was specifically designed for DES, it is applicable to any block cipher. For a block cipher with a key size of n bits and a precomputation with time complexity of about 2^n, Hellman's method has an (online) time complexity of $T = 2^{2n/3}$ and memory requirements of $M = 2^{2n/3}$. In more detail, it allows a time/memory trade-off curve of $M \cdot \sqrt{T} = 2^n$. Since we are only interested in attacks with $T \leq 2^n$ (faster than brute force), M has to be at least $2^{n/2}$. We want to note that the attack described in this paper is on a much better time/memory trade-off curve, i.e. $M \cdot T = 2^n$, and does not require a 2^n precomputation.

5 Relation of the Attack to the Security Proof

Fleischmann et al. [9] derive a security proof for McOE, which they also generalize to McOE-X. They derive security up to approximately $2^{n/2}$ queries (see Thm. 2). Although we want to stress that our attack is a key-recovery attack, which is much stronger than a distinguishability attack, it does not seem to directly invalidate the security proof of [9]. Yet, it turns out to expose a critical weakness in the security proof.

In short, the proof is technically invalid due to the fact that the authors (implicitly) consider security in the standard model. The security advantage is expressed in terms of parameters q, ℓ, and t. A critical observation is that q *only* denotes the number of queries made by the adversary to the full evaluation of McOE-X ($\mathcal{E}$ or $\mathcal{D}$), and in fact, the queries made in the offline phase of our attack *do not count* as queries. The adversary is considered to have free access to the underlying block cipher E and this offline phase only influences the variable t.

In this respect, for our attack we have parameters $q = 2^n/r$, $\ell = 2 \cdot 2^n/r$, and $t \approx r + 2 \cdot 2^n/r$. Now, considering the security claims of [9] for McOE-X in more detail (see Thm. 2 of this work), we see that the first part of the bound is independent of t.[1] As the authors claim, this part of the bound is determined by the event that a collision for the keyed compression function $f(K, U, M) = E(K \oplus U, M) \oplus M$ occurs, and the bound is obtained by applying the results of Black et al. [6,7] for the PGV compression functions [13]. However,

[1] When we apply our attack for $r = 2^n$, this part of the bound misleadingly suggests an almost zero advantage.

the authors oversee that these results *do not apply*: Fleischmann et al. consider the standard model where the underlying block cipher E is freely accessible by the adversary, while the results of Black et al. hold in the ideal model, where E is an idealized block cipher to which the adversary has query access only. Note that in our attack, the success probability of a collision (between L_1 and L_2) increases if more offline computation is done: we could for example choose $r = 2^{3n/4}$ and recover the key with $q = 2^{n/4}$ queries (see Table 1). In fact, (contrived) examples are known where the results of Black et al. do not apply when the PGV compression functions are instantiated with a CCA secure block cipher [11]. In [4], Biryukov et al. present an attack on the Davies-Meyer compression function $f(U, M) = E(M, U) \oplus U$ when instantiated with the AES block cipher.

In order to restore the proof of Thm. 1 as given in [9], one needs to consider the ideal cipher model for $\tilde{E}$. This means that an adversary has query access to $\tilde{E}$ and $\tilde{E}^{-1}$ (next to the query access to $\mathcal{E}$ and $\mathcal{D}$). In this way, the results of Black et al. *do* apply. Additionally, the second part of the bound of Thm. 1 gets superfluous: an ideal cipher is obviously perfectly indistinguishable from an ideal cipher, and hence $\delta = 0$. The same remarks apply to Thm. 2. Note that in this model, the evaluations in the first phase of our attack are counted as queries, and the attack corresponds to parameters $q = r + 2^n/r$ and $\ell = r + 2 \cdot 2^n/r$.

6 How to Fix McOE-X

As an alternative to McOE-X one can always use McOE-D or McOE-G which are not vulnerable to the attack presented in this paper. However, both constructions have some drawbacks. In McOE-D two block cipher invocations are needed per message block processed and in McOE-G a universal hash function is used to update the chaining value.

The main problem of McOE-X construction is that the tweak U (i.e. the chaining value) of n bits is xored to the key K of also n bits to turn the block cipher E into a tweakable block cipher. This allows generic TMTO attacks on the construction with complexity as low as $2 \cdot 2^{n/2}$ in its best setting as described in Section 4. For instance in the case of AES-128 this could be as low as 2^{65}. One option to fix the construction with still using only a single block cipher invocations per message block processed is to use a block cipher with a key input of $2n$ bits instead of n bits.

$$\tilde{E}(K, U, \cdot) := E(K \| U, \cdot) \tag{2}$$

For instance AES-256 seems to be natural choice and the performance overhead compared to AES-128 is not so large, only about 40%.

Acknowledgments. This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy) and by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. In addition, this work was supported by the Research Fund KU Leuven, OT/08/027.

References

1. Bellare, M., Boldyreva, A., Knudsen, L.R., Namprempre, C.: Online Ciphers and the Hash-CBC Construction. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 292–309. Springer, Heidelberg (2001)
2. Bellare, M., Boldyreva, A., Knudsen, L.R., Namprempre, C.: On-Line Ciphers and the Hash-CBC Constructions. Cryptology ePrint Archive, Report 2007/197 (2007)
3. Bellare, M., Namprempre, C.: Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. J. Cryptology 21(4), 469–491 (2008)
4. Biryukov, A., Khovratovich, D., Nikolić, I.: Distinguisher and Related-Key Attack on the Full AES-256. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 231–249. Springer, Heidelberg (2009)
5. Black, J., Cochran, M., Shrimpton, T.: On the Impossibility of Highly-Efficient Blockcipher-Based Hash Functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 526–541. Springer, Heidelberg (2005)
6. Black, J., Rogaway, P., Shrimpton, T.: Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 320–335. Springer, Heidelberg (2002)
7. Black, J., Rogaway, P., Shrimpton, T., Stam, M.: An Analysis of the Blockcipher-Based Hash Functions from PGV. J. Cryptology 23(4), 519–545 (2010)
8. Fleischmann, E., Forler, C., Lucks, S.: McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 196–215. Springer, Heidelberg (2012)
9. Fleischmann, E., Forler, C., Lucks, S., Wenzel, J.: McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes (extended version). Cryptology ePrint Archive, Report 2011/644 (2011)
10. Hellman, M.E.: A cryptanalytic time-memory trade-off. IEEE Transactions on Information Theory 26(4), 401–406 (1980)
11. Hirose, S.: Secure Block Ciphers Are Not Sufficient for One-Way Hash Functions in the Preneel-Govaerts-Vandewalle Model. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 339–352. Springer, Heidelberg (2003)
12. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable Block Ciphers. J. Cryptology 24(3), 588–613 (2011)
13. Preneel, B., Govaerts, R., Vandewalle, J.: Hash Functions Based on Block Ciphers: A Synthetic Approach. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 368–378. Springer, Heidelberg (1994)
14. Quisquater, J.-J., Delescaille, J.-P.: How Easy Is Collision Search. New Results and Applications to DES. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 408–413. Springer, Heidelberg (1990)
15. Rogaway, P., Shrimpton, T.: Deterministic Authenticated-Encryption: A Provable-Security Treatment of the Key-Wrap Problem. Cryptology ePrint Archive, Report 2006/221 (2006)
16. Rogaway, P., Zhang, H.: Online Ciphers from Tweakable Blockciphers. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 237–249. Springer, Heidelberg (2011)

Cryptanalysis of a Lattice-Knapsack Mixed Public Key Cryptosystem

Jun Xu[1,2,3], Lei Hu[1], Siwei Sun[1], and Ping Wang[4,5,6]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[2] University of Chinese Academy of Sciences, Beijing 100049, China
[3] School of Mathematical Science, Anhui University, Hefei 230601, Anhui, China
[4] Tian Jin Zhong Wei Aerospace Data System Technology Co., Ltd
[5] Space Star Technology Co., Ltd
[6] Institute No.503 of the fifth Research Academy, China Aerospace Science and Technology Corporation. Beijing 100086, China
{jxu,hu,swsun}@is.ac.cn

Abstract. Recently, a lattice based public key cryptosystem mixed with a knapsack was presented in the CANS 2011 conference. In this paper, we propose two message recovery attacks on this cryptosystem. The first one is a broadcast attack: a single message of m bits can be recovered if it is encrypted for $\lceil \frac{m+1}{2} \rceil$ recipients. The second attack is a multiple transmission attack in which a message can be recovered with a probability of $(1-2^{-l})^m$ if it is encrypted under a same public key for $l = \lceil \log_2 m + 2 \rceil$ times using different random numbers. The multiple transmission attack can be further improved with a linearization technique to that only $\lceil \frac{\log_2 m+1}{2} \rceil$ times of encryptions are required to recover the message. An open problem related to the message recovery attack using only one cipehertext is discussed.

Keywords: Public Key Cryptosystem, Lattice, Knapsack, Linearization.

1 Introduction

Ever since the discovery of the quantum algorithm [26] which can factor integers and compute discrete logarithms in polynomial time and hence can break RSA, DSA, and ECDSA efficiently, the necessity for new designs of public key cryptosystems immune to quantum algorithm attacks has become strong and urgent. Several new designs based on computational hard problems resistant to quantum algorithm attacks have been extensively studied, including hash based, code based, multivariate polynomial equation based and lattice based cryptography.

The first lattice based cryptosystem was proposed by Ajtai and Dwork [1] in 1997. However, to avoid Nguyen and Stern's heuristic attack [20], implementations of the Ajtai-Dwork cryptosystem would require very large keys, making

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 32–42, 2012.

it far from being practical. To increase the efficiency of the Ajtai-Dwork cryptosytem, Cai and Cusick [6] constructed a new cryptosystem by mixing the Ajtai-Dwork crytosystem with a knapsack. Unfortunately, Pan and Deng [23] presented an iterative method to recover the message encrypted by the Cai-Cusick cryptosystem under a ciphertext-only scenario.

With several known attacks in mind, Pan et al [24] proposed a new lattice-based public key cryptosystem which mixes with a knapsack in its design in the CANS 2011 conference. Unfortunately, it does not enjoy any security proof and is slower than state-of-the-art lattice-based encryption schemes (e.g., [11, 18, 27]), although its security was carefully evaluated against some lattice based attacks in [24]. One of such lattice based attacks, for example, is some attack similarly as that in [15–17] against NTRU [14] targeting at its cyclic structure, but the underlying lattice in this new cryptosystem has no special cyclic structure like in NTRU. Furthermore, the new cryptosystem hides the knapsack structure behind linear combinations. This strategy is very different from those knapsack based public key cryptosystems which hide their trapdoors by transforming a superincreasing knapsack into a generic one. Therefore, the existing successful attacks [21] against knapsack based cryptosystems seem to not be applicable to the new system.

In this paper we propose two feasible attacks on the cryptosystem of Pan et al [24]. The first one is a broadcast attack, it assumes a single message is encrypted by the sender directed for several recipients with different public keys, the message can be recovered by solving a system of nonlinear equations via linearization technique. The second one is a multiple transmission attack in which a single message is encrypted under the same public key for several times using different random vectors. In this situation, the message can be easier to recover.

The rest of this paper is organized as follows. Section 2 gives a description of the cryptosystem of Pan et al. Section 3 presents a broadcast attack and a multiple transmission attack on the crytosystem. In section 4, we further improve the multiple transmission attack by linearization technique. Section 5 is devoted to discuss an open problem related to the message recovery attack using only one cipehertext. The last section is a conclusion.

2 Description of the Cryptosystem of Pan et al.

In this section we describe the cryptosystem recently proposed by Pan et al [24]. For detailed information on its design rationale, please refer to their paper [24].

This cryptosystem is parameterized by a security parameter m. Let $n = 2m$.

Key Generation: Randomly choose a superincreasing sequence $\{N_1 = 1, N_2, \cdots, N_n\}$ and a permutation τ on $\{1, 2, \cdots, n\}$ such that $\tau^{-1}(1) \leq m$. For each $1 \leq i \leq m$, represent $N_{\tau(i+m)}$ as a linear combination of $\{N_{\tau(1)}, \cdots, N_{\tau(m)}\}$ with integer coefficients, namely,

$$N_{\tau(i+m)} = \sum_{j=1}^{m} b_{i,j} N_{\tau(j)}, \quad i = 1, 2, \cdots, m.$$

The absolute values of the coefficients $b_{i,j}$ $(1 \leq i, j \leq m)$ should be made reasonably small [24] and this can be done by employing Babai's nearest plane algorithm [3].

Use the above integer coefficients to form an $m \times n$ matrix

$$A = \begin{bmatrix} 1 & 0 & \cdots & 0 & b_{1,1} & b_{2,1} & \cdots & b_{m,1} \\ 0 & 1 & \cdots & 0 & b_{1,2} & b_{2,2} & \cdots & b_{m,2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & b_{1,m} & b_{2,m} & \cdots & b_{m,m} \end{bmatrix},$$

and let

$$l_{i,1} = \sum_{\substack{j=1,\cdots,n \\ A_{i,j}<0}} A_{i,j}, \quad i = 1, \cdots, m$$

$$l_{i,2} = \sum_{\substack{j=1,\cdots,n \\ A_{i,j}>0}} A_{i,j}, \quad i = 1, \cdots, m$$

$$q = \max_{i=1,\cdots,m} \{l_{i,2} - l_{i,1}\}.$$

Finally, randomly choose a permutation σ on $\{1, \cdots, n\}$ such that the matrix $S = [A_{\sigma(1)}, A_{\sigma(2)}, \cdots, A_{\sigma(m)}]$ is an invertible matrix over the field $\mathbb{F}_p$, where p is the smallest prime such that $p > q$ and A_j denotes the j-th column of A.

Public Key: The smallest prime p such that $p > q$, and $H = S^{-1}[A_{\sigma(m+1)}, \cdots, A_{\sigma(n)}]$.

Here the entries of the matrices are regarded as integers in the interval $[0, p-1]$ and identically regarded as elements of the field $\mathbb{F}_p$, and the matrices are operated modulo p.

Private Key: $N_2, \cdots, N_n, S, \tau, \sigma, l_{i,1}, l_{i,2}$ $(i = 1, \cdots, m)$.

Encryption: For any message $t \in \{0,1\}^m$, randomly choose a vector $r \in \{0,1\}^m$ and the ciphertext c is computed as follows

$$c = Ht + r \mod p.$$

Note that all vectors are written as column vectors in this paper as in [24].

Decryption: Let $v = (r^T, t^T)^T = (v_1, \cdots, v_n)^T$, and $d = Sc \mod p$. The message t and the random number r are recovered by solving the following superincreasing knapsack problem with unknown coefficients $x_i \in \{0, 1\}$

$$(N_{\tau(1)}, \cdots, N_{\tau(m)})\tilde{d} = \sum_{i}^{n} x_i N_i,$$

where $\tilde{d} = d \mod p$ with its entries $l_{i,1} \leq \tilde{d}_i \leq l_{i,2}$. It can be shown that $x_i = v_{\sigma^{-1}(\tau^{-1}(i))}$.

3 Two Attacks on the Cryptosystem of Pan et al.

In this section we present two attacks against the cryptosystem of Pan et al, one is a broadcast attack and the other is a multiple transmission attack, they can recover the encrypted messages with practical complexities, without any knowledge on the structure of the public and secret keys.

3.1 Broadcast Attack

The first broadcast attack was introduced in 1988 by Håstad in [13] to compromise the security of the RSA cryptosystem with low public key exponents. Broadcast attacks against lattice based encryption schemes have already been proposed by Plantard and Susilo [22]. In a broadcast attack, it is assumed that a single message is encrypted by the sender several times for multiple recipients with different public keys.

Below we will show that Pan et al's cryptosystem is insecure under this attack scenario. To be more specific, if the number of recipients exceeds $l := \lceil (m+1)/2 \rceil$, an attacker can derive the corresponding plaintext from the l ciphertexts.

Let $H = (h_{ij})_{m \times m}$. From $c = Ht + r \mod p$, we have

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} - \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1m} \\ h_{21} & h_{22} & \cdots & h_{2m} \\ & \cdots & \cdots & \\ h_{m1} & h_{m2} & \cdots & h_{mm} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_m \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix} \mod p$$

Since $r_i \in \{0, 1\}$ for $1 \leq i \leq m$, we have

$$\left(c_i - \sum_{j=1}^{m} h_{ij} t_j\right) \cdot \left(c_i - 1 - \sum_{j=1}^{m} h_{ij} t_j\right) = 0 \mod p$$

Obviously, this results in m quadratic equations over $\mathbb{F}_p$ on m variables $t_1, \cdots, t_m$. These variables are binary, but we do not know how to exploit this feature to solve them out from a single encryption (see the discussion in Section 5). In a broadcast scenario, assume a same message $t = (t_1, \cdots, t_m)^T$ is encrypted under

$l = \lceil (m+1)/2 \rceil$ different public keys, say $H^{(k)} = (h_{ij}^{(k)})_{m\times m}$, $1 \le k \le l$, into the ciphertexts $(c_1^{(k)}, \cdots, c_m^{(k)})^T$, $1 \le k \le l$, then

$$\begin{cases} \left(c_i^{(1)} - \sum_{j=1}^{m} h_{ij}^{(1)} t_j\right)\left(c_i^{(1)} - 1 - \sum_{j=1}^{m} h_{ij}^{(1)} t_j\right) = 0, \quad 1 \le i \le m \\ \cdots \\ \left(c_i^{(l)} - \sum_{j=1}^{m} h_{ij}^{(l)} t_j\right)\left(c_i^{(l)} - 1 - \sum_{j=1}^{m} h_{ij}^{(l)} t_j\right) = 0, \quad 1 \le i \le m \end{cases} \tag{1}$$

This is a system of $m\lceil (m+1)/2 \rceil \ge m(m+1)/2$ quadratic polynomial equations in $t_1, \cdots, t_m$. Utilizing the known linearization technique [2, 4, 9] and regarding all quadratic monomials $t_i t_j$ as new variables, the system (1) become a linear system in $m(m+1)/2$ unknowns ($t_1, \cdots, t_m$ and the $\binom{m}{2}$ new variables), it can be solved in time complexity $(\frac{m^2}{2})^\omega$ by a naive or advanced linear equation solving method, where $\omega = 3$ for a naive Gaussian elimination and $\omega = 2.376$ for the method of Coppersmith and Winograd [7].

Complexity and Experimental Result: To demonstrate the feasibility of the broadcast attack, we have implemented the attack in the Magma computer algebra system on a PC with Intel(R) Core(TM) Quad CPU (2.83GHz, 3.25GB RAM, Windows XP). For $m = 100$, which is one of the parameter suggested by the designers [24], we can recover the unknown message in no more than one hour. We chose $p = 271$, our Magma experiment successfully obtained the corresponding message within 46 minutes assuming $l = 51$.

For other suggested parameters: $m = 200, 300, 400, 500$, our Magma experiment failed to recover the message due to un-optimized programming and large memory consumption, but in all these cases, the time complexity of the attack is still in an acceptable range since even for $m = 500$ and by using a naive method, it is roughly $(\frac{500^2}{2})^\omega \approx 2^{50.79}$, which can be done on a minicomputer like DELL PowerEdge 7250 [10] and it will output a desired result within a week for an optimized memory management.

3.2 Multiple Transmission Attack

As observed in [14], multiple NTRU encryptions of a single message under a single public key may compromise the security of the encrypted message of NTRU, and this method used to recover the secret message is named as Multiple Transmission Attack [17]. In this section, we show that a similar vulnerability exists in Pan et al's cryptosystem, and we propose an efficient attack targeting at this vulnerability.

Assume a single message t is encrypted using the same public key H for l different but uniformly and independent random m-dimensional vectors in the encryption. Let

$$c^{(j)} = Ht + r^{(j)} \mod p, \quad 1 \le j \le l$$

be the ciphertext under the random m-dimensional vectors $r^{(j)}$, and use them as columns to form $n \times l$ matrices C and R:

$$C = \begin{bmatrix} c_1^{(1)} & c_1^{(2)} & \cdots & c_1^{(l)} \\ c_2^{(1)} & c_2^{(2)} & \cdots & c_2^{(l)} \\ \vdots & \vdots & & \vdots \\ c_n^{(1)} & c_n^{(2)} & \cdots & c_n^{(l)} \end{bmatrix}, R = \begin{bmatrix} r_1^{(1)} & r_1^{(2)} & \cdots & r_1^{(l)} \\ r_2^{(1)} & r_2^{(2)} & \cdots & r_2^{(l)} \\ \vdots & \vdots & & \vdots \\ r_n^{(1)} & r_n^{(2)} & \cdots & r_n^{(l)} \end{bmatrix}.$$

Let u_k and $r_k^{(j)}$ be the k-th entries of Ht and $r^{(j)}$, respectively. It is clear that the j-th entry of the k-th row $(c_k^{(1)}, c_k^{(2)}, \cdots, c_k^{(l)})$ of C is equal to $u_k + r_k^{(j)}$, which is either u_k or $u_k + 1 \mod p$.

We determine each row of R as follows. We have assumed that $p \geq 3$. Thus, for any two elements in $\{0, 1, \cdots, p-1\}$ which differ by 1 modulo p, we define their cyclic minimum as $\min\{a, a+1\} = a$ if $a \in \{0, 1, \cdots, p-2\}$ and $\min\{p-1, 0\} = p-1$. For a fixed $1 \leq k \leq l$, since the entries $c_k^{(1)}, c_k^{(2)}, \cdots, c_k^{(l)}$ either take two values which differ 1 modulo p or take the same value, we can always find a value $\widetilde{u_k}$ as $\min\{c_k^{(1)}, c_k^{(2)}, \cdots, c_k^{(l)}\}$. $\widetilde{u_k}$ will be equal to u_k in almost cases except when $(r_k^{(1)}, r_k^{(2)}, \cdots, r_k^{(l)}) = (1, \cdots, 1)$. Therefore, we can obtain correctly the value of u_k with a probability of $1 - 2^{-l}$. Assuming the uniformness and independence of random vectors $r^{(j)}$, we correctly get the vector Ht with a probability of $(1-2^{-l})^m$. Consequently, we can successfully recover the message t by computing $\tilde{t} = H^{-1}\tilde{u}$ with a probability of $(1-2^{-l})^m$.

Let $l \approx \log_2 m + 2$, we have

$$(1-2^{-l})^m \approx ((1 + \frac{1}{2^l - 1})^{-2^l})^{2^{-2}} \longrightarrow e^{-1/4} \approx 0.78, \ when \ m \to \infty,$$

which means that roughly $l = \lceil \log_2 m \rceil + 2$ encryptions can be used to probably successfully recover the message t. This number $(\lceil \log_2 m \rceil + 2)$ of needed times of encryptions is greatly less than the corresponding number $(\lceil (m+1)/2 \rceil)$ in a broadcast attack. The concrete value of the probability $(1-2^{-l})^m$ for different pairs of m and l are listed in Table 1, which are verified in our experiment on multiple transmission attacks.

Table 1. Success probability of the multiple transmission attack

m \ l	7	8	9	10	11	12	13	14
100	0.4542	0.6649	0.8237	0.9034	0.9510	0.9746	0.9885	0.9935
200	0.2047	0.4583	0.6742	0.8203	0.9038	0.9525	0.9753	0.9886
300	0.0956	0.3080	0.5654	0.7461	0.8640	0.9281	0.9643	0.9841
400	0.0461	0.2109	0.4596	0.6753	0.8231	0.9051	0.9476	0.9766
500	0.0220	0.1413	0.3731	0.6235	0.7874	0.8901	0.9414	0.9709

4 Improved Multiple Transmission Attack

In this section we show that the number of transmissions needed to successfully recover the message in a multiple transmission attack can be halved with the help of linearization technique.

Suppose a single message t is encrypted using the same public key H for l times and let

$$\begin{bmatrix} c_1^{(k)} \\ c_2^{(k)} \\ \vdots \\ c_m^{(k)} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1m} \\ h_{21} & h_{22} & \cdots & h_{2m} \\ \vdots & \vdots & & \vdots \\ h_{m1} & h_{m2} & \cdots & h_{mm} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_m \end{bmatrix} + \begin{bmatrix} r_1^{(k)} \\ r_2^{(k)} \\ \vdots \\ r_m^{(k)} \end{bmatrix} \mod p$$

be the k-th ciphertext, where $k \in \{1, 2, \cdots, l\}$.

According to Section 3.2, each row of R can be determined correctly with a probability $1 - 2^{-l}$, and can be correctly determined with a probability $1 - 2 \cdot 2^{-l}$ since we can correctly determine u_k when $c_k^{(1)}, c_k^{(2)}, \cdots, c_k^{(l)}$ are not the same, namely when $(r_k^{(1)}, r_k^{(2)}, \cdots, r_k^{(l)}) \neq (1, \cdots, 1)$ and $(0, \cdots, 0)$. Thus, about $(1 - 2^{1-l})m$ rows of R can be determined surely. Without loss of generality, we assume the remaining undetermined rows of R are the last $\rho = \lceil 2^{1-l} m \rceil$ ones.

Now from the first ciphertext and

$$\begin{bmatrix} c_1^{(1)} \\ c_2^{(1)} \\ \vdots \\ c_{m-\rho}^{(1)} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1m} \\ h_{21} & h_{22} & \cdots & h_{2m} \\ \vdots & \vdots & & \vdots \\ h_{m-\rho,1} & h_{m-\rho,2} & \cdots & h_{m-\rho,m} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_m \end{bmatrix} + \begin{bmatrix} r_1^{(1)} \\ r_2^{(1)} \\ \vdots \\ r_{m-\rho}^{(1)} \end{bmatrix} \mod p,$$

we can linearly represent $m - \rho$ unknowns t_j by other ρ ones. Without loss of generality, we assume the ρ unknowns are $t_1, t_2, \cdots, t_\rho$ and the linear relations are

$$\begin{cases} t_{\rho+1} = L_{\rho+1}(t_1, t_2, \cdots, t_\rho) \\ \quad \vdots \\ t_m = L_m(t_1, t_2, \cdots, t_\rho) \end{cases}$$

Then from binary property of t_i, $i = 1, \cdots, m$, a system of quadratic polynomial equations with ρ unknowns can be built as follows:

$$\begin{cases} t_1^2 = t_1 \\ \quad \vdots \\ t_\rho^2 = t_\rho \\ L_{\rho+1}^2(t_1, t_2, \cdots, t_\rho) = L_{\rho+1}(t_1, t_2, \cdots, t_\rho) \\ \quad \vdots \\ L_m^2(t_1, t_2, \cdots, t_\rho) = L_m(t_1, t_2, \cdots, t_\rho) \end{cases} \tag{2}$$

It can be successfully solved by linearization technique provided $m \geq \binom{\rho}{2} + 2\rho$, or equivalently, provided $l \geq \lceil \frac{\log_2 m+1}{2} \rceil$. Thus, about $\lceil \frac{\log_2 m+1}{2} \rceil$ encryptions can be used to recover the secret message, which is about one half the number of the method proposed in Section 3.2. As an example, for $m = 500$, this number is 5, we need only 5 encryptions to recover the secret message.

5 Discussion and Open Problem

The broadcast and multiple transmission attacks presented in the previous two sections do not use the inherited information about the structure of the public key H except that it is invertible. A natural question to ask is that in the case of only assuming H is an invertible matrix, whether it is possible to recover the encrypted message with only one ciphertext, namely is there a ciphertext only attack on the cryptosystem in this case? Mathematically, this problem can be formulated as follows:

Problem: Let $n = 2m$, A be an $m \times n$ matrix of rank m over $\mathbb{F}_p$. Given $c \in \mathbb{F}_p^m$, find a binary vector $x \in \{0,1\}^n$ (if any) such that $c = Ax \mod p$.

For most such matrices A, this problem has at most one solution for a general c, this is a requirement for cryptographic decryption scenario. We may further assume any m columns of A are linearly independent. Then, how to find the solution x efficiently?

The above problem can be viewed as a syndrome decoding of a linear code with parity check matrix A where the error noise vector is limited to be binary. A slightly general but essentially equivalent limitation is that each component of the error noise vector only takes two distinct values and the two values for different components may be different. This case can be translated to the case of binary noise vectors via an affine transformation. Here for the syndrome decoding, we do not limit the number of the errors (namely the Hamming weight) of the noise but limit the component of the noise to be binary. On the contrary in classical coding theory, the Hamming weight of the noise is limited but its component values are not restricted.

Another point of view is to look at the above problem as a knapsack problem on a vector space, that is, a subset sum problem on the column vectors of A.

In the encryption scheme of Pan et al [24], the corresponding A is chosen as (H, I), where I is identity matrix. This scheme is like the problem of Learning With Errors (LWE) [25]. It can be treated as the problem to solve a specific system of $2m$ quadratic equations over $\mathbb{F}_p$ in m variables of the form:

$$\begin{cases} x_1^2 - x_1 = 0 \\ \quad \vdots \\ x_m^2 - x_m = 0 \\ (c_1 - h_{11}x_1 - \cdots - h_{1m}x_m)^2 - (c_1 - h_{11}x_1 - \cdots - h_{1m}x_m) = 0 \\ \quad \vdots \\ (c_m - h_{m1}x_1 - \cdots - h_{mm}x_m)^2 - (c_m - h_{m1}x_1 - \cdots - h_{mm}x_m) = 0 \end{cases} \tag{3}$$

We do not know how to efficiently solve (if possibly) this seemingly very specific nonlinear system for large m. We have tried several methods such as XL [9], Fix-XL [4] and ElimLin [8] to solve (3), but failed to achieve an efficient solving method. We think this highly structured system of equations or knapsack problem on a vector space is an interesting pursuing topic and it may be a computational hard problem.

Finally, we point out that the cryptosystem of Pan et al is not secure under chosen plaintext attacks (CPA-secure) [12]. It is obvious that one can tell the difference between two messages $(0, \cdots, 0)^T$ and $(1, 0, \cdots, 0)^T$ by simply checking whether the ciphertext is close to the all zero vector or to the first column of H. However, this does not mean that this kind of observation can be directly utilized to launch an above-mentioned narrow-sense ciphertext-only attack, i.e., recovering a message given its only one ciphertext.

6 Conclusion

In this paper, we proposed two efficient attacks on a recently proposed cryptosystem that mixes the lattice and knapsack ideas in its design rational. Both attacks are capable of recovering the encrypted messages in practical time complexities under broadcast-like attack modes. The vulnerability of the new design is clearly due to the characteristic that the random vectors in its encryption process are chosen from a very limited set, namely the binary vectors.

We did not propose a ciphertext-only attack on this new cryptosystem, this type of attacks is equivalent to solve a knapsack problem on a vector space, which is an interesting research topic.

Acknowledgements. The authors would like to thank anonymous reviewers for their helpful comments and suggestions. The work of this paper was supported by the National Key Basic Research Program of China (2013CB834203), the National Natural Science Foundation of China (Grants 61070172 and 10990011), the Strategic Priority Research Program of Chinese Academy of Sciences under Grant XDA06010702, and the State Key Laboratory of Information Security, Chinese Academy of Sciences.

References

1. Ajtai, M., Dwork, C.: A public-key cryptosystem with worst-case/average-case equivalence. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pp. 284–293 (1997)
2. Arora, S., Ge, R.: New Algorithms for Learning in Presence of Errors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 403–415. Springer, Heidelberg (2011)
3. Babai, L.: On Lovász lattice reduction and the nearest lattice point problem. Combinatorica 6, 1–13 (1986)
4. Bard, G.V.: Algebraic Cryptanalysis. Springer, Heidelberg (2001) ISBN 978-0-387-88756-2

5. Bosma, W., Cannon, J., Playoust, C.: The Magma Algebra System I: The user language. Journal of Symbolic Computation 24, 235–265 (1997)
6. Cai, J.-Y., Cusick, T.W.: A Lattice-Based Public-Key Cryptosystem. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 219–233. Springer, Heidelberg (1999)
7. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progression. Journal of Symbolic Computation 9, 251–280 (1990)
8. Courtois, N.T., Bard, G.V.: Algebraic Cryptanalysis of the Data Encryption Standard. In: Galbraith, S.D. (ed.) Cryptography and Coding 2007. LNCS, vol. 4887, pp. 152–169. Springer, Heidelberg (2007)
9. Courtois, N.T., Klimov, A., Patarin, J., Shamir, A.: Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 392–407. Springer, Heidelberg (2000)
10. Ding, J., Hu, L., Nie, X., Li, J., Wagner, J.: High Order Linearization Equation (HOLE) Attack on Multivariate Public Key Cryptosystems. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 233–248. Springer, Heidelberg (2007)
11. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, pp. 197–206. ACM Press (2008) ISBN 978-1-60558-047-0
12. Goldwasser, S., Micali, S.: Probabilistic Encryption. J. Computer and System Sciences 28, 270–299 (1983)
13. Håstad, J.: Solving simultaneous modular equations of low degree. SIAM J. Comput. 17, 336–341 (1988)
14. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: A Ring-Based Public Key Cryptosystem. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (1998)
15. Howgrave-Graham, N.: A Hybrid Lattice-Reduction and Meet-in-the-Middle Attack Against NTRU. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 150–169. Springer, Heidelberg (2007)
16. Howgrave-Graham, N., Silverman, J.H.: A Meet-In-The-Meddle Attack on an NTRU Private Key. Technical report, http://www.ntru.com/cryptolab/technotes.htm#004
17. Howgrave-Graham, N., Silverman, J.H.: Implementation Notes for NTRU PKCS Multiple Transmissions. Technical report, http://www.ntru.com/cryptolab/technotes.htm#006
18. Lyubashevsky, V., Peikert, C., Regev, O.: On Ideal Lattices and Learning with Errors over Rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010)
19. May, A., Silverman, J.H.: Dimension Reduction Methods for Convolution Modular Lattices. In: Silverman, J.H. (ed.) CaLC 2001. LNCS, vol. 2146, pp. 110–125. Springer, Heidelberg (2001)
20. Nguyên, P.Q., Stern, J.: Cryptanalysis of the Ajtai-Dwork Cryptosystem. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 223–242. Springer, Heidelberg (1998)
21. Odlyzko, A.M.: The rise and fall of knapsack cryptosystems. Cryptology and Computational Number Theory 42, 75–88 (1990)
22. Plantard, T., Susilo, W.: Broadcast Attacks against Lattice-Based Cryptosystems. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 456–472. Springer, Heidelberg (2009)

23. Pan, Y., Deng, Y.: A Ciphertext-Only Attack Against the Cai-Cusick Lattice-Based Public-Key Cryptosystem. IEEE Transactions on Information Theory 57, 1780–1785 (2011)
24. Pan, Y., Deng, Y., Jiang, Y., Tu, Z.: A New Lattice-Based Public-Key Cryptosystem Mixed with a Knapsack. In: Lin, D., Tsudik, G., Wang, X. (eds.) CANS 2011. LNCS, vol. 7092, pp. 126–137. Springer, Heidelberg (2011)
25. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: The 37th Annual ACM Symposium on Theory of Computing, pp. 84–93. ACM Press (2004) ISBN 1-58113-960-8
26. Shor, P.: Algorithms for Quantum Computation: Discrete Logrithms and Factoring. In: The 35th Annual Symposium on Foundations of Computer Science, pp. 124–134. IEEE Computer Science Press, Santa Fe (1994)
27. Stehlé, D., Steinfeld, R.: Making NTRU as Secure as Worst-Case Problems over Ideal Lattices. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 27–47. Springer, Heidelberg (2011)

Biclique Cryptanalysis of TWINE

Mustafa Çoban[1,2], Ferhat Karakoç[1,3], and Özkan Boztaş[1,4]

[1] TÜBİTAK BİLGEM UEKAE, 41470, Gebze, Kocaeli, Turkey
{mustafacoban,ferhatk,ozkan}@uekae.tubitak.gov.tr
[2] Sakarya University, Mathematics Department, Sakarya, Turkey
[3] Istanbul Technical University, Computer Engineering Department, Istanbul, Turkey
[4] Middle East Technical University, Institute of Applied Mathematics, Cryptography Department, Ankara, Turkey

Abstract. TWINE is a lightweight block cipher firstly proposed at ECRYPT Workshop on Lightweight Cryptography 2011 and then presented at the Conference on Selected Areas in Cryptography 2012. The cipher consists of 36 rounds and has two versions TWINE-80 and TWINE-128 supporting key lengths of 80 and 128 bits, respectively. The block length of the two versions is 64-bit. In this paper, we present the first single-key attacks on both the versions of the cipher. In these attacks, we use the recently developed biclique technique. The complexities of the attacks on TWINE-80 and TWINE-128 are $2^{79.10}$ and $2^{126.82}$ respectively and the data requirement for the two attacks is 2^{60}.

Keywords: TWINE, lightweight block cipher, biclique cryptanalysis, meet-in-the-middle attack.

1 Introduction

The needs for security and privacy issues in resource constraint platforms such as RFID tags and sensor nodes give rise to design lightweight cryptographic algorithms. Some of the lightweight algorithms recently proposed are HIGHT [6], PRESENT [2], KATAN/KTANTAN [3], PRINTCIPHER [7], KLEIN [4], LED [5], Piccolo [9], and TWINE [10].

In this paper, we give our cryptanalytic results on TWINE. TWINE supports two key lengths, 80 and 128 bits. For each key length, encryption functions are the same but the key schedules are different. Corresponding to the key lengths, we denote TWINE-80 and TWINE-128. To the best of our knowledge, the most powerful attack is the impossible differential attacks against 23-round TWINE-80 and 24-round TWINE-128 with time complexities of $2^{76.88}$ and $2^{115.10}$ encryptions, respectively [10]. In this paper, we present attacks on the full TWINE-80 and TWINE-128. In these attacks, we use the biclique technique [1]. The complexities of the attacks on TWINE-80 and TWINE-128 are $2^{79.10}$ and $2^{126.82}$, respectively.

The organization of the paper is as follows. In Section 2 we give the notation which we use throughout the paper and a short description of TWINE algorithm.

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 43–55, 2012.

We overview the biclique technique in Section 3. In Section 4 and 5 we present the attacks on TWINE-80 and TWINE-128, respectively. We conclude the paper in Section 6.

2 Notation and a Short Description of TWINE

2.1 Notation

Throughout the paper, we use the following notations:

A	: a bit string
$A(i)$	: i-th nibble of A. The left most nibble is $A(0)$.
$A(i, j, ..., k)$	: concatenation of i, j, ... , k-th nibbles of A.
$A(i - j)$	: concatenation of i, $(i + 1)$, ..., j-th nibbles of A where $i \leq j$.
$A[i]$	: i-th bit of A. The left most bit of A is $A[0]$.
$A[i, j, ..., k]$	: concatenation of i, j, ..., k-th bits of A.
$A[i - j]$	: concatenation of i, $(i + 1)$, ..., j-th bits of A where $i \leq j$.
$A <<< i$	: i-bit cyclic left shift of A.
$A\|\|B$	: concatenation of A and B.
RK_i	: 32-bit round key used in the i-th round where $1 \leq i \leq 36$.
K^i	: k-bit value calculated after i iterations in the key schedule where k is the key length of the cipher.
X_i	: the output of the i-th round where X_0 is the plaintext and X_{36} is the ciphertext.

2.2 TWINE

TWINE is a block cipher supporting two key lengths, 80 and 128 bits. The global structure of TWINE algorithm is a variant of Type 2 generalized Feistel structure [12] with 16 4-bit sub-blocks. Each version of the algorithm has the same round function depicted in Figure 1 and consists of 36 rounds.

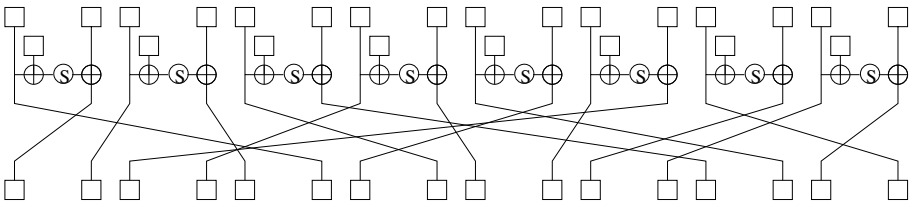


Fig. 1. One round of TWINE

In the round function, the key addition is applied before the S-box operation as seen in the figure and then the permutation is performed. In the last round the permutation does not exist.

The two versions of TWINE have key schedules which consist of S-box, round constant addition, $CON^i = CON^i_H || CON^i_L$, and permutation operations.

The key schedule of TWINE-80 generates 36 32-bit round keys from the 80-bit master key as follows:

1. $K^0 = K$
2. $RK_1 = K^0(1, 3, 4, 6, 13, 14, 15, 16)$
3. for i=1,2,...,35 do the followings
4. – $K^i = K^{i-1}$
   – $K^i(1) = K^i(1) \oplus S[K^i(0)]$
   – $K^i(4) = K^i(4) \oplus S[K^i(16)]$
   – $K^i(7) = K^i(7) \oplus (0||CON^i_H)$
   – $K^i(19) = K^i(19) \oplus (0||CON^i_L)$
   – $K^i(0, 1, 2, 3) = K^i(0, 1, 2, 3) <<< 4$
   – $K^i = K^i <<< 16$
   – $RK_{i+1} = K^i(1, 3, 4, 6, 13, 14, 15, 16)$

TWINE-128 has the following key schedule which generates 36 32-bit round keys from the 128-bit master key.

1. $K^0 = K$
2. $RK_1 = K^0(2, 3, 12, 15, 17, 18, 28, 31)$
3. for i=1,2,...,35 do the followings
4. – $K^i = K^{i-1}$
   – $K^i(1) = K^i(1) \oplus S[K^i(0)]$
   – $K^i(4) = K^i(4) \oplus S[K^i(16)]$
   – $K^i(23) = K^i(23) \oplus S[K^i(30)]$
   – $K^i(7) = K^i(7) \oplus (0||CON^i_H)$
   – $K^i(19) = K^i(19) \oplus (0||CON^i_L)$
   – $K^i(0, 1, 2, 3) = K^i(0, 1, 2, 3) <<< 4$
   – $K^i = K^i <<< 16$
   – $RK_{i+1} = K^i(2, 3, 12, 15, 17, 18, 28, 31)$

Our attacks are independent from the S-box and constants so we skip the details. For a complete description of the algorithm one can refer to [10].

3 An Overview of the Biclique Cryptanalysis Technique

In this section, we give an overview of the biclique technique on block ciphers proposed in [1]. In the biclique attack, firstly the key space is divided into 2^{k-2d} subspaces in which there exists 2^{2d} keys where k and d is the key length and the dimension of the biclique, respectively. Then, for all the key subspaces the two steps, biclique construction and meet-in-the-middle attack, are applied. To perform the two steps, firstly the cipher E is considered as a composition of three parts f, g, and h where $E = h \circ g \circ f$. Then, a biclique is constructed on the first or last parts (in the attack on TWINE-80 and TWINE-128 we construct the bicliques on the first and last parts as done in [11] and [1], respectively). After that, the meet-in-the-middle attack is applied on the remaining parts.

In this section we give the attack idea in the case that the biclique is constructed in the first part. The attack idea is similar for the other case.

A d dimensional biclique is a 3-tuple $(\{P_i\}, \{S_j\}, \{K_{i,j}\})$ such that $f_{K_{i,j}}(P_i) = S_j$, $\forall i, j \in \{0, 1, ..., 2^d - 1\}$. Two methods to construct a biclique are given in [1]. In this work, we use one of the methods called independent related-key differentials. In this method, first S_0 is calculated from a chosen P_0 under the key $K_{0,0}$. Then, S_j and P_i values are calculated from P_0 and S_0 using $K_{0,j} = K_{0,0} \oplus \Delta_j^K$ and $K_{i,0} = K_{0,0} \oplus \nabla_i^K$, respectively. Lets the differential trails in forward and backward directions called as Δ_j and ∇_i, respectively. If the trails do not have common active non-linear operations such as S-boxes then the probability of the equation $f_{K_{i,j}}(P_i) = S_j$ where $K_{i,j} = K_{0,0} \oplus \nabla_i^K \oplus \Delta_j^K$ is 1 as proved in [1].

The second step is the meet-in-the-middle attack on $h \circ g$. In this step first $C_i = E_K(P_i)$ values are obtained from the encryption oracle. After that a partial matching is searched at some portion v between g and h. This matching step has two sub-steps also. Firstly, in the forward and backward directions the internal values which affect the value of v and does not depend on the value of i and j respectively calculated using $K_{0,j}$ and $K_{i,0}$. Then the remaining internal values which affect the values of v are calculated using $K_{i,j}$ for all i and j. The keys $K_{i,j}$ are selected as candidate keys which lead to a matching on v. The number of candidates will be approximately 2^{2d-m} in a subspace where m is the bit length of v. The total number of key candidates will be $2^{k-2d} \times 2^{2d-m} = 2^{k-m}$. At the end approximately 2^{k-m} encryptions using $\lceil \frac{k-m}{n} \rceil$ plaintext-ciphertext pairs are performed to eliminate all the wrong candidates.

4 Biclique Cryptanalysis of TWINE-80

In this section, we present a biclique attack on the full TWINE-80. Firstly, we divide the key space into 2^{72} subspaces of 2^8 keys each. Then for each key subspace we construct a biclique on the first 8 rounds and by using this biclique we apply the meet-in-the-middle attack on the last 28 rounds of the cipher.

4.1 Key Partitioning

The base keys of the key subspaces are of the form $K_{0,0} = (* * * * \mid * * * 0 \mid * * * * \mid 0 * * * \mid * * * *)$, where two nibbles are fixed to zero and the remaining 18 nibbles determine the subspace. The 2^8 keys $\{K_{i,j}\}$ in the subspaces are taken as follows

$$K_{i,j} = K_{0,0} \bigoplus (0000|000i|0000|j000|0000), i, j \in \{0, 1, \ldots, 2^4 - 1\}.$$

4.2 Biclique Construction on 8 Rounds

First of all, S_0 is calculated from a randomly chosen P_0 as $S_0 = f_{K_{0,0}}(P_0)$ where f is the first 8 rounds of the cipher. Then a biclique is constructed using the following two sets of 2^4 related-key differentials with respect to the base computation $P_0 \xrightarrow{K_{0,0}} S_0$.

1. Δ_j-differentials over f. Each related-key differential in the first set maps input difference $\Delta P = 0$ to an output difference $\Delta_j = \Delta S = S_0 \bigoplus S_j$ under the key difference $\Delta_j^K = (0000|0000|0000|j000|0000)$.

$$0 \xrightarrow[f]{\Delta_j^K} \Delta_j.$$

2. ∇_i-differentials over f^{-1}. Each related-key differentials in the second set maps input difference $\Delta S = 0$ to an output difference $\nabla_i = \nabla P = P_0 \bigoplus P_i$ under the key difference $\nabla_i^K = (0000|000i|0000|0000|0000)$.

$$0 \xrightarrow[f^{-1}]{\nabla_i^K} \nabla_i.$$

Δ_j and ∇_i differentials are given in Figure 2.

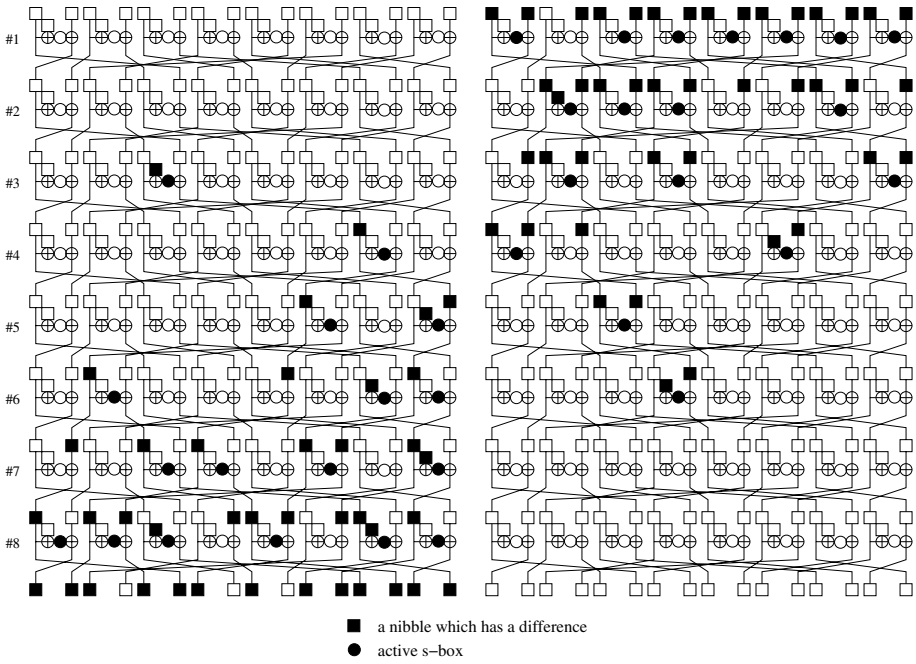


Fig. 2. Δ_j and ∇_i differential trails for TWINE-80 on the left and right respectively

As seen in the figure the differential trails do not share any active S-box.

Thus

$$\nabla_i \xrightarrow[f]{\Delta_j^K \oplus \nabla_i^K} \Delta_j, \forall i, j \in \{0, 1, \ldots, 2^4 - 1\}.$$

As a result, the triple $(\{P_i\}, \{S_j\}, \{K_{i,j}\})$ with the definition

$$P_i = P_0 \bigoplus \nabla_i,$$
$$S_j = S_0 \bigoplus \Delta_j,$$
$$K_{i,j} = K_{0,0} \bigoplus \Delta_j^K \bigoplus \nabla_i^K$$

is an 8-round biclique of dimension 4.

Note that there is no difference in the 2-nd nibble of the plaintext in the biclique. Thus we can use the plaintexts whose 2-nd nibble is a fixed value. This reduces the data requirement of the attack to 2^{60}.

4.3 The Meet-in-the-Middle Step

By the biclique construction, 2^4 plaintexts P_i and 2^4 intermediate states S_j are available with corresponding $K_{i,j}$'s. First of all, we obtain C_i in the chosen plaintext scenario. Then, we check if there is some i, j such that

$$C_i \xrightarrow[g^{-1} \circ h^{-1}]{K_{i,j}} S_j. \tag{1}$$

where g and h as the composition of the rounds from the beginning of the 9-th round to the end of the 19-th round and from the beginning of the 20-th round to the end of the 36-th round respectively. For each one of the 2^{72} key subspaces, the complexity of this stage is 2^8. Hence, the overall complexity of the attack will be near exhaustive search, but we can reduce this complexity applying the attack given in Algorithm 1. In the algorithm the nibble $X_{19}(3)$ is taken as the matching variable v. To meet on v, we do partial calculations in forward direction starting from S_j and in backward direction starting from C_i.

4.4 The Attack Complexity

The computational complexity of the attack is composed of several parts. In the biclique construction step, for each of the 2^{72} key subspaces we perform 2^4 8-round encryptions to compute 2^4 intermediate states S_j and $2^4 - 1$ 8-round decryptions to compute 2^4 P_i's. Therefore, $\frac{2^4 \times 8 + (2^4 - 1) \times 8}{36} \approx 2^{2.78}$ encryptions are performed to construct a biclique. In the meet-in-the-middle attack given in Algorithm 1, the total number of S-boxes calculated for 2^4 and 2^8 different keys are 30 and 127 as seen in Figure 3 as gray and black S-boxes, respectively. Since the average number of remaining keys after the condition in Step 11 is $2^8 \times 2^{-4} = 2^4$, we perform 2^4 encryption operations in Step 12. Thus the total number of operations in the algorithm is $\frac{2^4 \times 30 + 2^8 \times 127}{36 \times 8} + 2^4 \approx 2^{7.03}$ encryptions under the assumption that the time needed for one S-box look-up is equivalent to the running time of $\frac{1}{8}$ round function and $\frac{1}{8 \times 36}$ encryption function. Therefore, $2^{2.78} + 2^{7.03} \approx 2^{7.10}$ encryptions are performed for each key subspace. As a result, the overall complexity of the attack on the full TWINE-80 is approximately $2^{79.10}$ encryptions with 2^8 memory.

Algorithm 1.

1: S_j and C_i's are given.
2: **for** j in 0,1,...15 **do**
3: Calculate the nibbles colored in gray in the forward direction in Figure 3 and $S[X_{10}(2) \oplus RK_{10}(1)]$, $S[X_{11}(12) \oplus RK_{11}(6)]$, $S[X_{12}(2) \oplus RK_{12}(1)]$, $S[X_{12}(6) \oplus RK_{12}(3)]$, $S[X_{12}(2) \oplus RK_{12}(1)]$, $S[X_{13}(12) \oplus RK_{13}(6)]$ using $K_{0,j}$ and S_j.
4: **for** i in 0,1,...15 **do**
5: Calculate the nibbles colored in black in the forward direction in Figure 3 using $K_{i,j}$ and the values calculated in step 3.
6: Store $X_{19}(3)$ in the $(16 \times i + j)$-th cell of a table called A.
7: **end for**
8: **end for**
9: **for** i, j in 0,1,...15 **do**
10: Calculate the nibbles colored in black in the backward direction in Figure 3 using C_i and $K_{i,j}$.
11: **if** The calculated value of $X_{19}(3)$ is equal to the value in the $(16 \times i + j)$-th cell of A **then**
12: **if** $K_{i,j}$ satisfies another plaintext-ciphertext pair **then**
13: Output $K_{i,j}$ as the right key
14: **end if**
15: **end if**
16: **end for**

5 Biclique Cryptanalysis of TWINE-128

In this section, we introduce a biclique attack on the full TWINE-128. In this attack we divide the key space considering K^{32}. The attack steps are similar to the attack steps given in the previous section. Firstly, we divide the key space into 2^{120} subspaces of 2^8 keys each and then for each key subspace we construct a biclique on the last 11 rounds and by using this biclique we perform the meet-in-the-middle attack on the first 25 rounds.

5.1 Key Partitioning

The key subspaces are enumerated by 2^{120} base keys of the form $K^{32}_{0,0} = (****|****|****|****|0***|**0*|****|****)$. The 2^8 keys $\{K^{32}_{i,j}\}$ in a subspace are taken as follows $K^{32}_{i,j} = K^{32}_{0,0} \bigoplus (0000|0000|0000|0000|i000|00j0|0000|0000), i, j \in \{0, 1, \ldots, 2^4 - 1\}$.

5.2 Contructing a Biclique

First of all, S_0 is calculated from a randomly chosen C_0 as $S_0 = h^{-1}_{K^{32}_{0,0}}(C_0)$ where h is the last 11 rounds of the cipher. Then a biclique is constructed using the following two sets of 2^4 related-key differentials with respect to the base computation $C_0 \xrightarrow[h^{-1}]{K^{32}_{0,0}} S_0$.

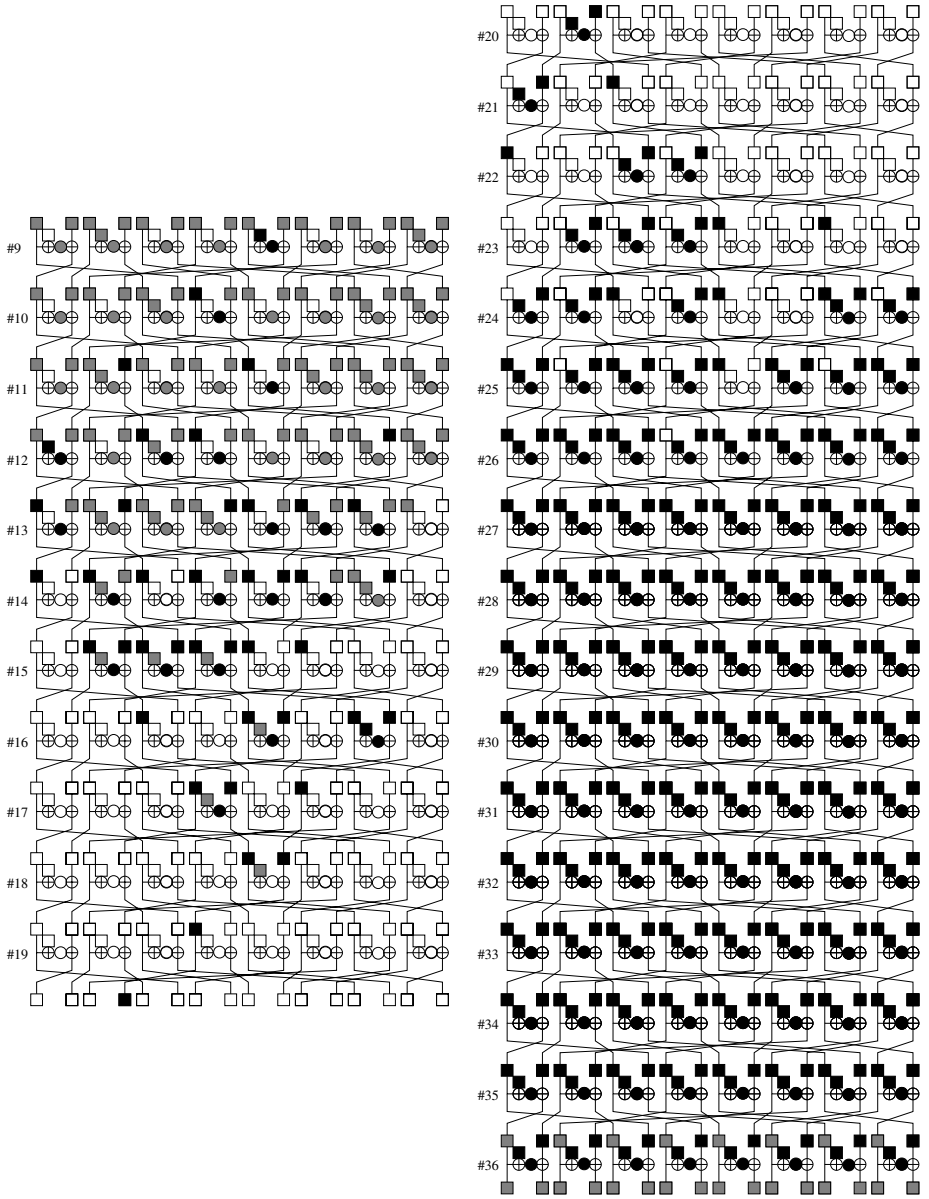


Fig. 3. Meet-in-the-middle step for TWINE-80

1. Δ_j-differentials over h^{-1}. Each related-key differential in the first set maps input difference $\Delta C = 0$ to an output difference $\Delta_j = \Delta S = S_0 \bigoplus S_j$ under the key difference $\Delta_j^K = (0000|0000|0000|0000|0000|00j0|0000|0000)$.

$$0 \xrightarrow[h^{-1}]{\Delta_j^K} \Delta_j.$$

2. ∇_i-differentials over h. Each related-key differentials in the second set maps input difference $\Delta S = 0$ to an output difference $\nabla_i = \nabla C = C_0 \bigoplus C_i$ under the key difference $\nabla_i^K = (0000|0000|0000|0000|i000|0000|0000|0000)$.

$$0 \xrightarrow[h]{\nabla_i^K} \nabla_i.$$

Δ_j and ∇_i differentials are given in Figure 4.

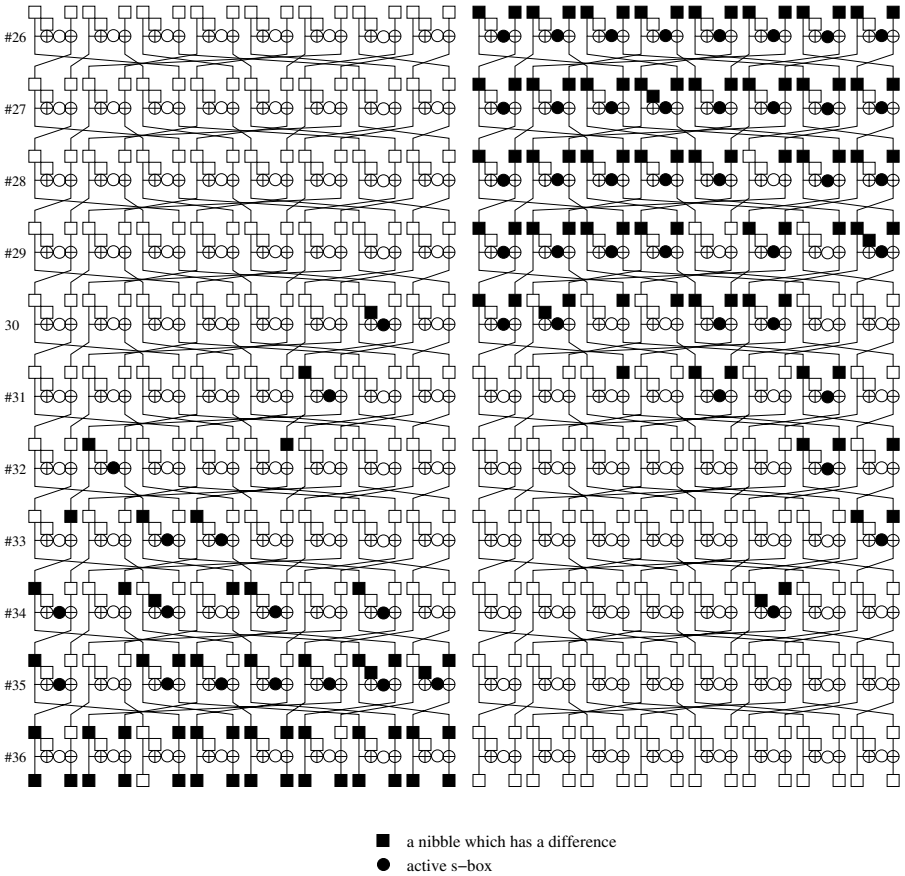


Fig. 4. ∇_i and Δ_j differential trails for TWINE-128 on the left and right respectively

As seen in the figure the differential trails do not share any active S-box.

Thus

$$\nabla_j \xrightarrow[h]{\Delta_j^K \bigoplus \nabla_i^K} \Delta_i, \forall i, j \in \{0, 1, \ldots, 2^4 - 1\}.$$

As a result, the triple $(\{S_j\}, \{C_i\}, \{K_{i,j}\})$ with the definition

$$\begin{aligned} C_i &= C_0 \bigoplus \nabla_i, \\ S_j &= S_0 \bigoplus \Delta_j, \\ K_{i,j} &= K_{0,0} \bigoplus \Delta_j^K \bigoplus \nabla_i^K \end{aligned}$$

is an 11-round biclique of dimension 4.

Note that there is no difference in the 7-th nibble of the ciphertext in the biclique. Thus we can use the ciphertexts whose 7-th nibble is a fixed value. This reduces the data requirement of the attack to 2^{60}.

5.3 The Meet-in-the-Middle Step

The attack is very similar to that in TWINE-80. In this attack we choose the subcipher f from the beginning of the 1-st round to the end of the 6-th round, and the subcipher g from the beginning of the 7-th round to the end of the 25-th round. The nibble $X_6(11)$ is taken as the matching variable v. The meet-in-the-middle step is given in Algorithm 2.

Algorithm 2.

1: P_i and S_j's are given.
2: **for** i in 0,1,...15 **do**
3: Calculate the nibbles colored in gray in the forward direction in Figure 5 using $K_{i,0}^{32}$ and P_i.
4: **for** j in 0,1,...15 **do**
5: Calculate the nibbles colored in black in the forward direction in Figure 5 using $K_{i,j}^{32}$ and the values calculated in step 3.
6: Store $X_6(11)$ in the $(16 \times i + j)$-th cell of a table called A.
7: **end for**
8: **end for**
9: **for** j in 0,1,...15 **do**
10: Calculate the nibbles colored in gray in the backward direction in Figure 5 and $S[X_{23}(15) \oplus RK_{23}(6)]$, $S[X_{22}(11) \oplus RK_{22}(7)]$, $S[X_{21}(7) \oplus RK_{21}(2)]$, $S[X_{21}(3) \oplus RK_{21}(3)]$, $S[X_{20}(3) \oplus RK_{20}(3)]$, $S[X_{20}(15) \oplus RK_{20}(6)]$ using $K_{0,j}^{32}$ and S_j.
11: **for** i in 0,1,...15 **do**
12: Calculate the nibbles colored in black in the backward direction in Figure 5 using S_j, $K_{i,j}^{32}$ and the values calculated in step 10.
13: **if** The calculated value of $X_6(11)$ is equal to the value in the $(16 \times i + j)$-th cell of A **then**
14: **if** $K_{i,j}^{32}$ satisfies another plaintext-ciphertext pair **then**
15: Output $K_{i,j}^{32}$ as the right key
16: **end if**
17: **end if**
18: **end for**
19: **end for**

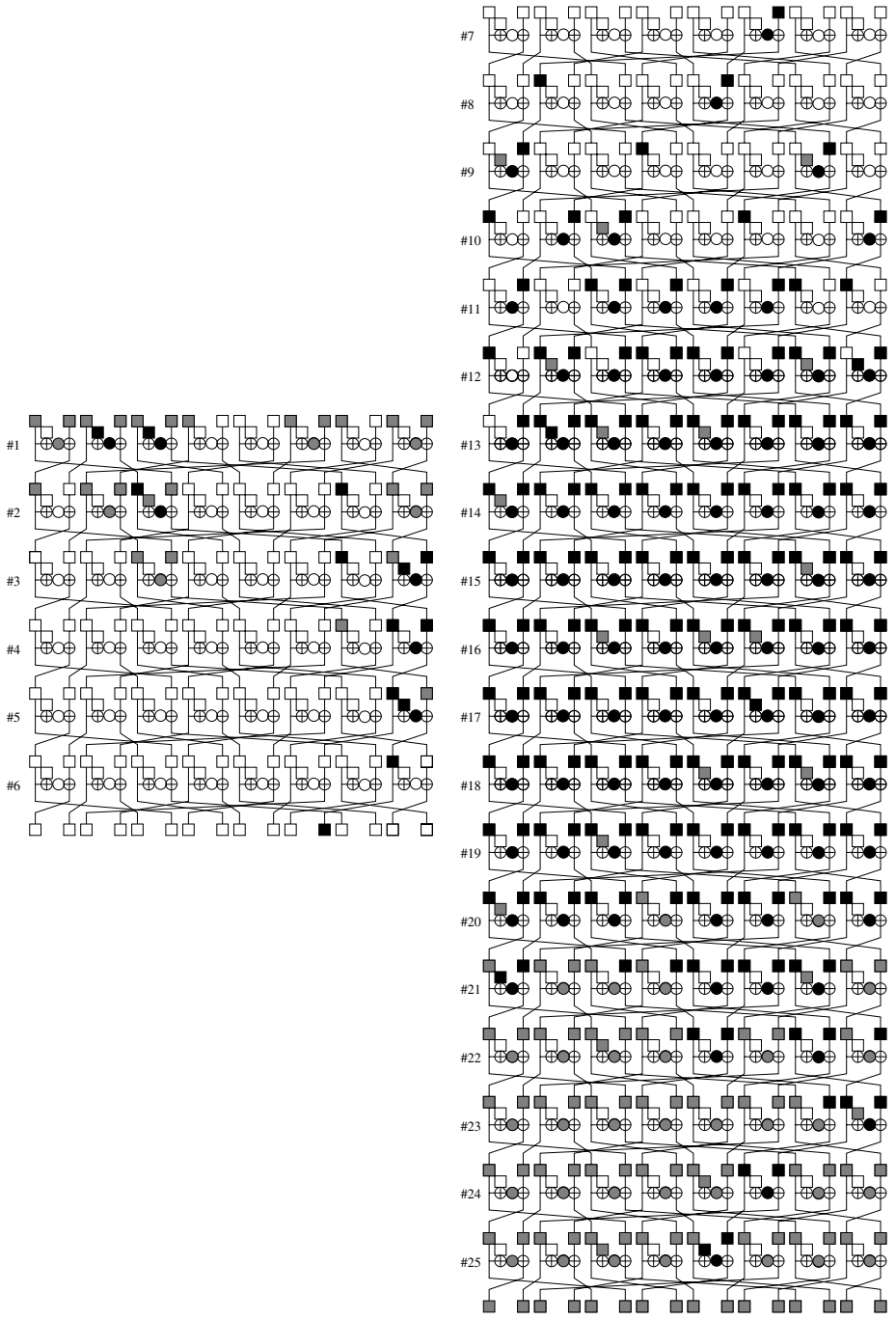


Fig. 5. Meet-in-the-middle step for TWINE-128

5.4 The Attack Complexity

For each key subspace, to construct a biclique $\frac{2^4\times 11+(2^4-1)\times 11}{36} \approx 2^{3.24}$ encryptions are performed. Also, in the meet-in-the-middle step $\frac{2^8\times 96+2^4\times 39}{36\times 8} + 2^4 \approx 2^{6.69}$ encryptions are needed. Thus, $2^{3.24} + 2^{6.69} \approx 2^{6.82}$ encryptions are performed for each key subspace. As a result, the overall complexity of the biclique attack is approximately $2^{126.82}$ encryptions with 2^8 memory.

6 Conclusion

In this paper, we present the first single-key attacks on the full TWINE-80 and TWINE-128 by using recently developed biclique attack technique. In the both attacks 2^{60} data and 2^8 memory are required and the time complexities of the attacks on TWINE-80 and TWINE-128 are $2^{79.10}$ and $2^{126.82}$ encryption operations, respectively.

References

1. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique Cryptanalysis of the Full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 344–371. Springer, Heidelberg (2011)
2. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
3. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
4. Gong, Z., Nikova, S., Law, Y.W.: KLEIN: A New Family of Lightweight Block Ciphers. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. LNCS, vol. 7055, pp. 1–18. Springer, Heidelberg (2012)
5. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The led block cipher. In: Preneel, Takagi (eds.) [8], pp. 326–341
6. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B.-S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: HIGHT: A New Block Cipher Suitable for Low-Resource Device. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)
7. Knudsen, L., Leander, G., Poschmann, A., Robshaw, M.J.B.: PRINTcipher: A Block Cipher for IC-Printing. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 16–32. Springer, Heidelberg (2010)
8. Preneel, B., Takagi, T. (eds.): CHES 2011. LNCS, vol. 6917. Springer, Heidelberg (2011)
9. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An ultra-lightweight blockcipher. In: Preneel, Takagi (eds.) [8], pp. 342–357

10. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: Twine: A lightweight, versatile block cipher. In: Proceedings of ECRYPT Workshop on Lightweight Cryptography (2011), http://www.uclouvain.be/
11. Chen, S.Z., Xu, T.M.: Biclique attack of the full aria-256. IACR Cryptology ePrint Archive, 2012:11 (2012)
12. Zheng, Y., Matsumoto, T., Imai, H.: On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 461–480. Springer, Heidelberg (1990)

Differential and Linear Attacks on the Full WIDEA-n Block Ciphers (under Weak Keys)

Jorge Nakahara Jr.

Université Libre de Bruxelles (ULB), Dept. of Computer Science, Belgium
jorge.nakahara@ulb.ac.be

Abstract. We report on differential and linear analysis of the full 8.5-round WIDEA-n ciphers for $n \in \{4, 8\}$, under weak-key assumptions. The novelty in our attacks include the use of differential and linear relation patterns that allow to bypass the diffusion provided by MDS codes altogether. Therefore, we can attack only a single IDEA instance out of n copies, effectively using a narrow trail for the propagation of differences and masks across WIDEA-n. In fact, the higher the value of n, the better the attacks become. Our analyses apply both to particular MDS matrices, such as the one used in AES, as well as general MDS matrices. Our attacks exploit fixed points of MDS matrices. We also observed a curious interaction between certain differential/linear patterns and the coefficients of MDS matrices for non-trivial fixed points. This interaction may serve as an instructive design criterion for block cipher designs such as WIDEA-n. The authors of WIDEA-n suggested a compression function construction using WIDEA-8 in Davies-Meyer mode. In this setting, the weaknesses identified in this paper can lead to free-start collisions and even actual collisions depending on the output transformation of the hash function.

Keywords: wide-block cipher, cryptanalysis, WIDEA-n, free-start collisions.

1 Introduction

In [6], Junod and Macchetti presented a Wide-block version of IDEA cipher [7] called WIDEA-n, combining n instances of the 8.5-round IDEA cipher joined by an $n \times n$ matrix derived from a Maximum Distance Separable (MDS) code. Their approach not only led to improved performance, due to the bit-slicing technique allowing parallel instances of IDEA to be evaluated altogether, but also showed wide-block cipher variants operating on bit strings whose size is a multiple of 64 bits (the original block size of IDEA).

The contributions of this paper include

- the first differential and linear distinguishers of the full 8.5-round WIDEA-n, for $n \in \{4, 8\}$, under weak-key assumptions. Actually, our attacks would hold even if n was allowed to be much larger than 8. **Weak-key assumptions mean that user keys in our attacks lead, through the key schedule,**

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 56–71, 2012.

to round subkeys which are either 0 or 1 in specific positions along differential or linear trails [2]. In this way, both the user key and some subkeys are weak. For this reason, we refer equally to weak-key and weak-subkeys assumptions.
- differential and linear distinguishers that apply both in a secret-key and in a hash/compression function settings, assuming WIDEA-n becomes the compression function in Davies-Meyer (DM) mode [8].
- we show how to bypass the MDS matrices using trivial and non-trivial fixed points. This procedure is possible due to carefully chosen differences and linear masks that lead to trivial differences and masks at the input to the MA/MAD-boxes. Therefore, avoiding these diffusion components in every round of WIDEA-n, for any n, means that we restrict the propagation of differences and masks to one single IDEA instance, out of n. The larger n is, the better the attacks become. Previous analyses using fixed points include [1,3,11], but the latter worked on fixed points for an entire block. As far as we aware of, this paper presents the first use of fixed points (in differential and linear settings) that bypass MDS codes in block cipher designs such as WIDEA-n.
- we show how and why some matrices from MDS codes can rather help the cryptanalysis of WIDEA-n, depending on where they are placed in a block cipher design such as WIDEA-n, and even depending on the exclusive-or sum of its coefficients. See Sect. 4.2.

This paper is organized as follows: Sect. 2 briefly details WIDEA-n; Sect. 3 describes the key schedule algorithms of WIDEA-n; Sect. 4 describes differential attacks on WIDEA-n; Sect. 5 details linear attacks on WIDEA-n. Sect. 6 describes attacks on WIDEA-n used in compression function constructions. Sect. 7 discusses weak keys. We conclude in Sect. 8.

2 The WIDEA-n Block Ciphers

WIDEA-n, $n \in \{4, 8\}$, stands for two Wide-block variants of the IDEA cipher [7] operating on $64n$-bit blocks. The rationale is to join n instances of the IDEA cipher using an $n \times n$ matrix derived from an MDS code, placed inside the MA-box (Multiplication-Addition) in each round of each IDEA instance (see Fig. 1). Thus, the original MA-box in IDEA becomes a so called MAD-box (Multiply-Add-Diffuse) [6] in WIDEA-n. The key size is $128n$ bits and WIDEA-n iterates 8.5 rounds. Fig. 1 depicts one round of WIDEA-4 cipher. For WIDEA-8, there are eight copies of IDEA side-by-side, connected by an 8×8 MDS matrix (Fig. 2). The MDS matrix in WIDEA-4 is the one used in the AES cipher [4]:

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}. \tag{1}$$

In WIDEA-4, the matrix (1) is multiplied on the right by a 4×1 vector $(a, b, c, d)^t$ containing part of the internal state of four IDEA instances, where t denotes vector transpose. This matrix product operation will be denoted $\mathrm{MDS}(a, b, c, d)^t$.

The MDS matrix in WIDEA-8 comes from the W cipher in the Whirlpool hash function [5], and the semantics for matrix multiplication is the same as explained for (1):

$$\begin{pmatrix} 1&1&4&1&8&5&2&9 \\ 9&1&1&4&1&8&5&2 \\ 2&9&1&1&4&1&8&5 \\ 5&2&9&1&1&4&1&8 \\ 8&5&2&9&1&1&4&1 \\ 1&8&5&2&9&1&1&4 \\ 4&1&8&5&2&9&1&1 \\ 1&4&1&8&5&2&9&1 \end{pmatrix}. \tag{2}$$

In both (1) and (2), matrix coefficients and operations are performed over $\mathrm{GF}(2^{16}) = \mathrm{GF}(2)[\mathrm{x}]/(p(x))$, where $p(x) = x^{16} + x^5 + x^3 + x^2 + 1$ is an irreducible polynomial over GF(2).

We briefly describe an MA-box (Fig. 1): let the input to the i-th MA-box of the i-th IDEA instance, for $1 \leq i \leq n$, be denoted (p_i, q_i), its output be (r_i, s_i) and $(Z_{5,i-1}, Z_{6,i-1})$ be the round subkeys in it. We ignore the superscripts since they are irrelevant in this setting. The three group operations in IDEA and WIDEA-n are: $\oplus$ denote bitwise exclusive-or, $\boxplus$ denote addition modulo 2^{16} and $\odot$ denote multiplication in $\mathrm{GF}(2^{16}+1)$, with $0 \equiv 2^{16}$. Then, $s_i = (p_i \odot Z_{5,i-1} \boxplus q_i) \odot Z_{6,i-1}$ and $r_i = p_i \odot Z_{5,i-1} \boxplus s_i$, where $\odot$ has higher precedence than $\boxplus$.

Now, for the MAD-box (Fig. 2): the MDS matrix in WIDEA-n is placed inside the original MA-box of IDEA after $p_i \odot Z_{5,i-1} \boxplus q_i$. This way, a single MAD-box of WIDEA-n has output (r', s') such that

- $s' = MDS(p_i \odot Z_{5,i-1} \boxplus q_i) \odot Z_{6,i-1}$, and $r' = p_i \odot Z_{5,i-1} \boxplus s'$, where $1 \leq i \leq n$ and $MDS(p_i \odot Z_{5,i-1} \boxplus q_i)$ stands for the multiplication of an MDS matrix (1) or (2) by an $n \times 1$ vector containing the n values $p_i \odot Z_{5,i-1} \boxplus q_i$, for $1 \leq i \leq n$.
- every single output tuple (r', s') depends on all $(p_i, q_i, Z_{5,i-1})$, for $1 \leq i \leq n$, but not on $Z_{6,i-1}$.
- the placement of the MDS matrix also implies that its dependence on $(p_i, q_i, Z_{5,i-1})$, for $1 \leq i \leq n$, is spread to both (r', s') in all n IDEA instances in every round.
- the MDS matrix is preceded by $\boxplus$ and followed by $\odot$, while inside the matrix computation there is a combination of xor and multiplication in $\mathrm{GF}(2^{16})$. Except for the repeated xor in the matrix product, no other operation is repeated twice in a row in the MAD-box.
- since the half-round containing the MAD-box is an involution there is no need to compute the inverse MDS matrix for decryption.

To allow a compact representation for analysis, and taking into account the 3-dimensional structure of WIDEA-4, we denote the internal state of WIDEA-4 by the 4×4 matrix:

$$\begin{pmatrix} a_{12} & a_{13} & a_{14} & a_{15} \\ a_8 & a_9 & a_{10} & a_{11} \\ a_4 & a_5 & a_6 & a_7 \\ a_0 & a_1 & a_2 & a_3 \end{pmatrix}, \tag{3}$$

where each a_i, for $0 \leq i \leq 15$, is a 16-bit word and the numbering follows from Fig. 1, where $(a_{4(j-1)}, a_{4(j-1)+1}, a_{4(j-1)+2}, a_{4(j-1)+3})$ represent the state of the j-th IDEA instance for $1 \leq j \leq 4$. The MAD-boxes of the four IDEA instances are connected to each other via the 4×4 MDS matrix (1). Analogously, we denote the internal state of WIDEA-8 by the 8×4 matrix:

$$\begin{pmatrix} a_{28} & a_{29} & a_{30} & a_{31} \\ a_{24} & a_{25} & a_{26} & a_{27} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{16} & a_{17} & a_{18} & a_{19} \\ a_{12} & a_{13} & a_{14} & a_{15} \\ a_8 & a_9 & a_{10} & a_{11} \\ a_4 & a_5 & a_6 & a_7 \\ a_0 & a_1 & a_2 & a_3 \end{pmatrix}, \tag{4}$$

where each a_i, for $0 \leq i \leq 31$, is a 16-bit word. Word numbering follows Fig. 2 where $(a_{4(j-1)}, a_{4(j-1)+1}, a_{4(j-1)+2}, a_{4(j-1)+3})$ represents the state of the j-th IDEA instance, $1 \leq j \leq 8$. The MAD-boxes of the eight IDEA instances are connected by a single 8×8 MDS matrix (2) in every round.

3 The Key Schedule of WIDEA-n

Let Z_i, for $0 \leq i \leq 51$, denote the round subkeys used in 8.5-round WIDEA-n, $n \in \{4, 8\}$. The key schedule algorithm of WIDEA-4 is as follows [6]: due to the 4-way parallelism, each subkey has 64 bits. Thus, each subkey Z_i can be split into four slices $Z_{i,0}, \ldots, Z_{i,3}$ (see Fig. 1). Let K_i, for $0 \leq i \leq 7$, denote the eight 64-bit words representing the user key. The round subkeys are computed as follows:

- $Z_i = K_i$, for $0 \leq i \leq 7$.
- $Z_i = ((((Z_{i-1} \oplus Z_{i-8}) \overset{16}{\boxplus} Z_{i-5}) \overset{16}{\lll} 5) \lll 24) \oplus C_{i/8-1}$, for $8 \leq i \leq 51$, $i \equiv 0 \bmod 8$.
- $Z_i = ((((Z_{i-1} \oplus Z_{i-8}) \overset{16}{\boxplus} Z_{i-5}) \overset{16}{\lll} 5) \lll 24)$, for $8 \leq i \leq 51$, $i \not\equiv 0 \bmod 8$.

where operations superscripted with '16' indicate that the operation is actually carried out over 16-bit slices of Z_i. Othewise, the operation is carried out across 64-bit words, such as the bitwise left-rotation $\lll 24$. Following [6], $C_0, \ldots, C_5$ are constants inserted every eight rounds. This design using nonlinear feedback shift registers was inspired on the key schedule of MESH ciphers [9].

The key schedule algorithm of WIDEA-8 [6] follows an 8-way parallelism. Each 128-bit subkey Z_i can be split into eight slices $Z_{i,0}, \ldots, Z_{i,7}$ (see Fig. 2).

Let K_i, for $0 \leq i \leq 7$, denote the eight 128-bit words representing the user key. The round subkeys are computed exactly as for WIDEA-4, except that the subkeys and constants $C_{i/8-1}$ are 128 bits long.

4 Differential Cryptanalysis of WIDEA-n

For a differential analysis, we start with Table 2 (in the appendix) that lists exhaustively all one-round characteristics of IDEA [2] using wordwise difference $\delta = 8000_x$. The subscript x indicates hexadecimal value. This difference propagates across $\oplus$ and $\boxplus$ with certainty and for any subkey value because the only active difference is in the most significant bit position. The arrows indicate difference propagation across one-round or across an MA/MAD-box in the encryption direction, depending on the context. All these characteristics hold with probability 1 under weak-key assumptions. Thus, the main purpose of weak keys is that they cause weak subkeys in specific positions inside WIDEA-n which allow straightforward propagation of differences (and bit masks). The third column in Table 2 shows the difference propagation inside the MA-box and consequently, if the MA-box is differentially active or not. An MA/MAD-box is differentially active if its input difference is nonzero. It is passive, otherwise.

We choose characteristics based on two criteria: (i) minimize the number of weak-key assumptions per round, and (ii) choose iterative difference patterns. Under these two conditions, the best choices include the 3-round characteristic

$$(0, 0, \delta, \delta) \rightarrow (0, \delta, \delta, 0) \rightarrow (0, \delta, 0, \delta) \rightarrow (0, 0, \delta, \delta), \tag{5}$$

with four weak-subkey assumptions[1]: $Z_{6(j-1)+3}$, $Z_{6(j-1)+4}$, Z_{6j+4}, $Z_{6(j+1)+3}$ starting from round j, for $j \geq 1$. All rotations of (5), for instance, starting from $(0, \delta, \delta, 0)$ instead of $(0, 0, \delta, \delta)$, result in equivalent characteristics.

Another relevant choice is the 1-round iterative characteristic

$$(\delta, \delta, \delta, \delta) \rightarrow (\delta, \delta, \delta, \delta), \tag{6}$$

with two weak-subkey assumptions: $Z_{6(j-1)}, Z_{6(j-1)+3}$ starting from round j.

We next describe attacks on WIDEA-n that bypass all the MDS matrices across 8.5-round WIDEA-n.

4.1 Differential Attack on One IDEA Instance Only

We use (6), a 1-round iterative characteristic whose differential trail does not include any MA-box, that is, all MA-boxes are passive. See Table 2. Extending it to WIDEA-4, one IDEA instance will follow the differential pattern (6), while

[1] We adapted the original terminology $Z_i^{(j)}$ that represents the i-subkey of the j-th round in [7] to the notation Z_l in WIDEA-n as described in Sect. 3, where $l = 6(j-1) + i - 1$, since there are six subkeys per round in IDEA.

the other three IDEA instances will have zero input difference. This means that all MAD-boxes will be passive. In other words, we exploit the (trivial) fixed point

$$MDS(0,0,0,0)^t = (0,0,0,0)^t,$$

where the superscript t denotes the transpose operation. In other words, if all weak-subkey assumption are satisfied, then the differential trail (6) concatenated with itself will propagate across a single 8.5-round IDEA instance in WIDEA-n, instead of (nonzero) differences spreading to all n IDEA instances, effectively bypassing the MDS diffusion layer in every round. This attack holds independent of which MDS matrix is used. Note that our attack does not contradict the branch number of the MDS matrix [4], but rather exploit a (trivial) fixed point.

Thus, we have the following 1-round iterative characteristic, using (6) in only a single IDEA instance in WIDEA-4:

$$\begin{pmatrix} 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ \delta\,\delta\,\delta\,\delta \end{pmatrix} \rightarrow \begin{pmatrix} 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ \delta\,\delta\,\delta\,\delta \end{pmatrix}, \tag{7}$$

following the state representation (3). Without loss of generality, we picked the last row of the state as the active IDEA instance. The same reasoning applies if we had picked any of the other IDEA instances (but the weak key would not be the same). Thus, the fifteen most significant bits (MSB) of the following eighteen subkeys should be zero across 8.5 rounds of one IDEA instance: $Z_{6(j-1)}$, $Z_{6(j-1)+3}$, for $1 \leq j \leq 9$. Under these conditions, the output difference pattern in (7) will appear as ciphertext difference with certainty. If the key schedule of WIDEA-4 behaves as a random mapping and the weak-subkey conditions hold independently, then in WIDEA-4's key space of 2^{512} keys, the weak subkeys would represent a class of about $2^{512-15\cdot 18} = 2^{242}$ keys. The reasoning is that each 16-bit weak subkey can be either 0 or 1, and we need eighteen of them to be weak. For WIDEA-8, the same reasoning applies, but the weak-key class size is estimated at $2^{1024-15\cdot 18} = 2^{754}$.

In order to avoid this distinguishing attack using (7), more than $512/15 = 34$ weak-subkey conditions would be required, since each weak subkey implies the fifteen MSBs to be zero. This means WIDEA-4 would need more than $34/2 = 17$ rounds, which means more than double the original number of rounds, since each round requires two weak subkeys. But, the resulting performance would hardly be acceptable.

A key-recovery attack on the full 8.5-round WIDEA-4, using (7), can obtain the subkeys $Z_{48,0}$ and $Z_{51,0}$ of the last half-round. In this case, only sixteen subkeys need be weak, which imply a weak-key class of about $2^{512-15\cdot 16} = 2^{272}$ keys. The output difference after eight rounds, restricted to one IDEA instance, is $(\delta, \delta, \delta, \delta)$. For the additive subkeys, the δ difference propagates across to the ciphertext, but not for $Z_{48,0}$ and $Z_{51,0}$. This means a 16-bit condition for each 16-bit subkey piece. Thus, one chosen pair of texts is enough. Decrypting two multiplications in a half-round in one IDEA instance is equivalent to $\frac{1}{17\cdot 2\cdot 4}$ of

a full WIDEA-4, that is, the cost becomes $\frac{2^{32}}{17 \cdot 8} \approx 2^{25}$ WIDEA-4 encryptions. There are 17 half-rounds in 8.5-round IDEA. Memory cost is negligible. Time complexity for WIDEA-8 becomes 2^{24} encryptions since WIDEA-8 contains eight copies of IDEA. Moreover, the weak-key class size becomes $2^{1024-15 \cdot 16} = 2^{784}$. Recovering subkeys from the other IDEA instances is not possible because of the zero differences in (7). If we shift the $(\delta, \delta, \delta, \delta)$ pattern to another row of the state in (7), then we would need another key that has weak subkeys in that same part of the state. We leave the issue of a full key-recovery attack as an open problem.

4.2 Differential Attack on All IDEA Instances

Let us analyse WIDEA-4. If we use the 3-round iterative linear relation (5), then there are active MAD-boxes along the differential trail. See Table 2. This means we need to exploit another fixed point of the AES MDS matrix (1):

$$MDS(\delta, \delta, \delta, \delta)^t = (\delta, \delta, \delta, \delta)^t.$$

When a MAD-box is active, we have to attack all four copies in WIDEA-4 at once so that the same value δ appears inside each MAD-box. This means that the input to the active MDS matrices is $(\delta, \delta, \delta, \delta)$. Applying it to the matrix in (1) results in $2 \cdot \delta \oplus 3 \cdot \delta \oplus \delta \oplus \delta = (2 \oplus 3 \oplus 1 \oplus 1) \cdot \delta = \delta$ in all four rows since the MDS matrix in AES is circulant. In other words, this fixed point exploits the fact that the exclusive-or of the coefficients in a line (or column) of the AES MDS matrix xor to 1. This is a new and surprising interaction between the differential pattern $(\delta, \delta, \delta, \delta)$ and the coefficients of the AES MDS matrix.

This attack does not contradict the branch number of the MDS matrix [4], but rather exploit a non-trivial fixed point. Note that this property does not hold for the 8×8 MDS matrix in (2) since in the latter, the exclusive-or sum of the coefficients in a line or column is 3. A consequence of this finding is an additional criterion for block cipher designs that employ matrices from MDS codes in the way they are used in WIDEA-4: carefully select the coefficients in these matrices in order to avoid fixed-point (differences).

We arrive at the following 3-round iterative characteristic:

$$\begin{pmatrix} 0 & \delta & 0 & \delta \\ 0 & \delta & 0 & \delta \\ 0 & \delta & 0 & \delta \\ 0 & \delta & 0 & \delta \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 0 & \delta & \delta \\ 0 & 0 & \delta & \delta \\ 0 & 0 & \delta & \delta \\ 0 & 0 & \delta & \delta \end{pmatrix} \rightarrow \begin{pmatrix} 0 & \delta & \delta & 0 \\ 0 & \delta & \delta & 0 \\ 0 & \delta & \delta & 0 \\ 0 & \delta & \delta & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & \delta & 0 & \delta \\ 0 & \delta & 0 & \delta \\ 0 & \delta & 0 & \delta \\ 0 & \delta & 0 & \delta \end{pmatrix}. \tag{8}$$

Thus, we exploit a combined symmetry in the AES MDS matrix, the differential pattern and the four identical copies of IDEA in WIDEA-4. Across 8.5 rounds, we need the following eleven 64-bit subkeys to be weak: Z_3, Z_9, Z_{10}, Z_{16}, Z_{21}, Z_{27}, Z_{28}, Z_{34}, Z_{39}, Z_{45} and Z_{46}, across all four IDEA instances. These weak subkeys imply conditions on $11 \cdot 4 \cdot 15 = 660 > 512$ bits. In a key space of 2^{512} keys we do not expect any weak-key class to satify all these conditions. It is a negative result, though. Analogous conclusions hold for WIDEA-8.

5 Linear Cryptanalysis of WIDEA-n

For a linear analysis, we listed exhaustively all one-round linear relations of IDEA [2] in Table 3 (in the appendix). All these linear relations hold with maximum bias 2^{-1} under weak-subkey assumptions.

The linear relations with the best profile for an attack on WIDEA-n have the same patterns as the characteristics used in Sect. 4, with γ instead of δ. From Table 3 we choose linear relations that: (i) minimize the number of weak-subkey assumptions per round, and (ii) are iterative. Under these two conditions, the best choices include the 3-round relation

$$(0, \gamma, \gamma, 0) \to (\gamma, 0, \gamma, 0) \to (\gamma, \gamma, 0, 0) \to (0, \gamma, \gamma, 0), \tag{9}$$

with only four weak-subkey assumptions: $Z_{6(j-1)+4}, Z_{6j}, Z_{6(j+1)}, Z_{6(j+1)+4}$ starting from round j. All rotations of (9), for instance, starting from $(\gamma, 0, \gamma, 0)$ instead of $(0, \gamma, \gamma, 0)$, also fulfill the same criteria.

Another relevant choice is the 1-round iterative characteristic

$$(\gamma, \gamma, \gamma, \gamma) \to (\gamma, \gamma, \gamma, \gamma), \tag{10}$$

with only two weak-subkey assumptions per round: $Z_{6(j-1)}, Z_{6(j-1)+3}$ starting from round j.

If we use linear relation (10) in a single IDEA instance in WIDEA-4, we arrive at the following 3-round iterative linear relation:

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \gamma & \gamma & \gamma & \gamma \end{pmatrix} \to \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \gamma & \gamma & \gamma & \gamma \end{pmatrix}. \tag{11}$$

Similar to (7), there are no active MAD-boxes along (11) because all approximations at the input to the MAD-boxes are trivial: $(0, 0, 0, 0)$. Thus, once again, we exploit the fixed-point relation $\mathrm{MDS}(0, 0, 0, 0)^t = (0, 0, 0, 0)^t$. Concatenating (11) with itself across 8.5 rounds, this linear relation holds with maximum bias 2^{-1} as long as the following eighteen subkeys are weak: $Z_{6(j-1)}, Z_{6(j-1)+3}$ for $0 \le j \le 9$. The weak-key class for this linear relation has size $2^{512-15 \cdot 18} = 2^{242}$ keys. Using relation (9) instead of (10) would require all IDEA instance to be attacked at once. This means using the fixed point $\mathrm{MDS}(\gamma, \gamma, \gamma, \gamma)^t = (\gamma, \gamma, \gamma, \gamma)^t$, which leads to the same problem as in Sect. 4.2: there are too many weak-subkey conditions because some MAD-boxes become active.

The linear relation (11) can be translated into $P \cdot \Gamma = E_K(P) \cdot \Gamma$, where $\Gamma = (\gamma, \gamma, \gamma, \gamma)$. This linear relation can be used to distinguish the full WIDEA-4 from a random permutation. A equally interesting consequence would be its implications in a hash function context. We discuss it in Sect. 6.

Applying (11) to WIDEA-8 gives even better results, since the later has key size of 1024 bits, and the same weak-subkey conditions lead to an estimated weak-key class of $2^{1024-15 \cdot 18} = 2^{754}$ keys.

A (partial) key-recovery attack on the full 8.5-round WIDEA-4, using (11), can recover subkeys $Z_{48,0}$ and $Z_{51,0}$. In this case, we attack the last half-round and only sixteen subkeys need be weak: $Z_{6(j-1)}, Z_{6(j-1)+3}$ for $0 \leq j \leq 8$, which implies a weak key class of about $2^{512-15\cdot 16} = 2^{272}$ keys. Using Matsui's estimation for a high-success rate attack, $8(2^{-1})^{-2} = 32$ known plaintexts are enough. The effort is equivalent to 2^{32} multiplications per subkey, which is equivalent to fraction of $\frac{1}{17\cdot 2\cdot 4}$ of a full WIDEA-4 computation, or $2^{32}/(17\cdot 2\cdot 4) \approx 2^{25}$ WIDEA-4 encryptions. The memory needed is 32 counters. Recovery of the remaining subkeys has the same problems as in the key-recovery attack in Sect. 4.1. The time complexity for WIDEA-8 becomes 2^{24} WIDEA-8 encryptions since there are eight IDEA instances, Also, the weak-key class size is $2^{1024-15\cdot 16} = 2^{784}$.

6 WIDEA-n in Davies-Meyer Mode

In [6], the authors suggested to use WIDEA-n as a compression function in Davies-Meyer (DM) mode, since the key size is double the block size[2]. The hash digest could range from 224 bits up to 512 bits, as in the SHA-2 hash function family [10] by truncation of the last chaining variable. The DM mode for a compression function construction is as follows [8]: the i-th chaining value is

$$H_i = H_{i-1} \oplus E_{m_i}(H_{i-1}), \tag{12}$$

where $H_0 = IV$ is the initial value, m_i is the i-th message block and $E_x(y)$ is a block cipher with key x and plaintext y. In particular, E is WIDEA-n, $|m_i|$ is $128n$ bits, $|H_i|$ is $64n$ bits.

The issue of weak subkeys in WIDEA-n is even more relevant in a hash function setting. In this case, the message to be hashed becomes the key input and can be chosen by the adversary.

We point to the following consequences from the results in the previous sections when WIDEA-n is used in DM mode:

- semi free-start collision: suppose we can set H_{i-1} with difference (7) for WIDEA-4. If m_i is a weak key that leads to weak subkeys as required in Sect. 4.1, then $H_{i-1} = E_{m_i}(H_{i-1})$, that is, H_i contains only zero word differences according to (12). It is a semi free-start collision because only the chaining variable has nonzero difference [8]. The same reasoning applies to WIDEA-8, using the same difference (7), but extended to a 1024-bit state. Note that this attack is independent of the MDS matrix used.
- truncation: suppose the output transformation in a (hypothetical) hash function using WIDEA-n in the compression function simply truncates the output to the least significant 192 bits for WIDEA-4 (3), or to the least significant 448 bits for WIDEA-8 (4). In both cases, we assume that at least 64 bits are cut off from the last chaining variable. Alternatively, more bits

[2] The number of rounds was increased from 8.5 to 10.5 to provide some security margin.

can be dropped, but 64 bits is enough for our attack. Suppose we have a differential trail like (7) in H_{i-1} but with weak subkeys up to the 8th round. These trails reduce the number of required weak subkeys to sixteen instead of eighteen (increasing the weak-key class size), but the input and output difference patterns are not the same. This fact implies that in DM mode, the exclusive-or between H_{i-1} and $E_{m_i}(H_{i-1})$ will not vanish. But, on the other hand, the nonzero difference words are isolated in a single 64-bit piece of the state. If that 64-bit piece is in the most significant part of the state, it will be truncated and we have a collision since the rest of the state has only zero word difference. In this way, we use the output transformation of the hash function to an attacker's advantage, if we can control the difference to remain in the part of the state that is going to be truncated[3].

– a linear relation such as

$$\begin{pmatrix} 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \\ \gamma\ \gamma\ \gamma\ \gamma \end{pmatrix} \rightarrow \begin{pmatrix} 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \\ \gamma\ \gamma\ \gamma\ \gamma \end{pmatrix}, \tag{13}$$

for WIDEA-4 = E, using relation (10), imply the linear relation $H_{i-1} \cdot \Gamma_1 = E_{m_i}(H_{i-1}) \cdot \Gamma_1$, where Γ_1 is any of the masks in a state in (13). Applying this iterative relation to a single compression function in DM mode, leads to

$$H_i \cdot \Gamma_1 = 0, \tag{14}$$

that is, the linear relation does not depend on H_{i-1} due to feedforward in the DM mode and the splitting of the Γ_1 mask. Relation (14) could be used to distinguish the compression function using WIDEA-4 in DM mode from a random function. Note that (14) depends only on the output H_i, and could be applied to the hash digest only if the masked bits are not truncated. This linear relation have implications for WIDEA-n in applications such as pseudorandom number generation, since the masked bit would leak information in the output bitstream.

Similar reasoning applies for WIDEA-8 in place of WIDEA-4 since the reasoning concerns one single IDEA instance (out of n).

Another example of collision using only two text blocks is in DES using the complementation property: suppose two texts P and $\overline{P}$ encrypted under an arbitrary key K and its bitwise complement $\overline{K}$. The corresponding ciphertexts are $C = \mathrm{DES}_K(P)$ and $\overline{C} = \mathrm{DES}_{\overline{K}}(\overline{P})$. In DM mode, $\Delta H_{i-1} = \Delta P = P \oplus \overline{P} = ffffffffffffffff_x$ and $\Delta H_i = \Delta H_{i-1} \oplus DES_K(P) \oplus$

[3] This collision has to happen in the last message block hashed. If the message length is say at most $2^{128} - 1$ bits, then the last 128 bits are reserved for the message length. Assume the first 64 bits are variable, so we can control the difference in it. For WIDEA-4, the remaining 256-64-128=64 bits are padding. For WIDEA-8, the remaining 1024-64-128=832 bits are padding. So, this is feasible, since we only need nonzero difference in the most significant 64 bits, while the rest of the state has zero difference.

$DES_{\overline{K}}(\overline{P}) = ffffffffffffffff_x \oplus ffffffffffffffff_x = 0$, which is a free-start collision (we have nonzero difference in both the key and the plaintext, which corresponds to message and chaining variable). This is yet another example of how a weakness in the key schedule turns into a weakness in a hash function setting, jeopardizing potential applications of a block cipher in a hash function setting.

7 Weak Keys

An important question to address is whether weak keys exist in WIDEA-n that can generate the weak subkeys required in the attacks in Sect. 4, 5 and 6. Recalling the key schedule of WIDEA-4 in Sect. 3 and taking, for instance, the pattern (7) repeated over six rounds, requires that the most significant fifteen bits (corresponding to the first IDEA instance in Fig. 1) of the following subkeys to be zero: Z_0, Z_3, Z_6, Z_9, Z_{12}, Z_{15}, Z_{18}, Z_{21}, Z_{24}, Z_{27}, Z_{30} and Z_{33}. Satisfying Z_0, Z_3 and Z_6 is straightforward since they are part of the user key. There are nine subkey conditions left. Writing down the corresponding equations in the key schedule we got the following, where the variables in boldface are either 0 or 1. So far, we did not find any contradiction. That is, even though we could not yet find a 512-bit user key that leads to the eleven weak subkeys, we have found no reason these nine equations (15)–(23) cannot be satisfied.

$$\mathbf{Z}_9 = (((Z_8 \oplus Z_1) \overset{16}{\boxplus} Z_4) \overset{16}{\lll} 5) \lll 24, \tag{15}$$

$$\mathbf{Z}_{12} = (((Z_{11} \oplus Z_4) \overset{16}{\boxplus} Z_7) \overset{16}{\lll} 5) \lll 24, \tag{16}$$

$$\mathbf{Z}_{15} = (((Z_{14} \oplus Z_7) \overset{16}{\boxplus} Z_{10}) \overset{16}{\lll} 5) \lll 24, \tag{17}$$

$$\mathbf{Z}_{18} = (((Z_{17} \oplus Z_{10}) \overset{16}{\boxplus} Z_{13}) \overset{16}{\lll} 5) \lll 24, \tag{18}$$

$$\mathbf{Z}_{21} = (((Z_{20} \oplus Z_{13}) \overset{16}{\boxplus} Z_{16}) \overset{16}{\lll} 5) \lll 24, \tag{19}$$

$$\mathbf{Z}_{24} = ((((Z_{23} \oplus Z_{16}) \overset{16}{\boxplus} Z_{19}) \overset{16}{\lll} 5) \lll 24) \oplus C_2, \tag{20}$$

$$\mathbf{Z}_{27} = (((Z_{26} \oplus Z_{19}) \overset{16}{\boxplus} Z_{22}) \overset{16}{\lll} 5) \lll 24, \tag{21}$$

$$\mathbf{Z}_{30} = (((Z_{29} \oplus Z_{22}) \overset{16}{\boxplus} Z_{25}) \overset{16}{\lll} 5) \lll 24, \tag{22}$$

$$\mathbf{Z}_{33} = (((Z_{32} \oplus Z_{25}) \overset{16}{\boxplus} Z_{28}) \overset{16}{\lll} 5) \lll 24. \tag{23}$$

Assuming that the key schedule behaves as a random mapping, generating (approximately) uniformly distributed subkeys, we expect each subkey to have equal chance to assume a value in the range $[0, \ldots, 2^{64} - 1]$ for WIDEA-4, or $[0, \ldots, 2^{128} - 1]$ for WIDEA-8. Therefore, we assume each 16-bit subkey for each IDEA instance to have approximately the same chance to have values in the range $[0, \ldots, 2^{16} - 1]$. For the particular values 0 and 1 the chance is 2^{-16} for each. Under these assumptions, we estimated the weak-key classes in Sect. 4, 5 and 6.

In order to have some experimental evidence of the presence of weak subkeys, we searched for them in mini-versions of the WIDEA-4 key schedule. Recall that in WIDEA-4, the subkeys are 64-bit wide since each one of them has to key four IDEA instances at once. As such, the search effort is too big even though we are looking for a weak subkey value in a 16-bit piece of the 64-bit WIDEA-4 subkey. Taking into account that the key schedule operates wordwise, we shrank the word size from 16 to 4 bits. So, for instance, equations such as (15) would be modified to $Z_9 = (((Z_8 \oplus Z_1) \overset{4}{\boxplus} Z_4) \overset{4}{\lll} 1) \lll 6$, where the rotation amounts 1 and 6 were chosen to match the reduced word sizes. Attack simulations on such reduced scale equations shows weak 4-bit weak subkey values to appear for the first IDEA instance, as expected for the attacks in Sect. 4, 5 and 6. The same behavior was observed when the word size was reduced to 5 bits, leading to equations such as $Z_9 = (((Z_8 \oplus Z_1) \overset{5}{\boxplus} Z_4) \overset{5}{\lll} 2) \lll 7$. These experiments provide evidence that weak subkey values can and do appear in critical places in differential and linear trails, which gives some evidence for the propagation of differential and linear patterns.

8 Conclusions

This paper described the first differential and linear analyses of the full WIDEA-n ciphers [6], for $n \in \{4, 8\}$ under weak-key assumptions, both in the block cipher and in the hash function settings. Table 1 summarizes our attack complexities for WIDEA-n.

We exploited iterative differential characteristics and iterative linear relations that bypassed the MDS matrices in WIDEA-n by carefully choosing trails that input trivial differences or relations, such as $(0, 0, 0, 0)^t$, or symmetric ones such as $(\delta, \delta, \delta, \delta)^t$ to the MAD-boxes. The rationale is to exploit fixed points for the MDS matrix for these particular differences and masks. This effectively means that we found and exploited **narrow differential and linear trails**. This phenomenon was observed for the AES MDS matrix, for which $\text{MDS}(\delta, \delta, \delta, \delta)^t = (\delta, \delta, \delta, \delta)^t$. This simple observation allowed us to bypass all diffusion layers connecting the four IDEA instances in WIDEA-4 because the exclusive-or sum of the coefficients in the MDS matrix of AES equals one. This result does not hold for the MDS matrix used in WIDEA-8. Our attacks exploit structural weaknesses due to the way MDS matrices are placed inside WIDEA-n to connect n IDEA instances. These attacks do not appply to the AES cipher [4].

Other attacks, that hold for any MDS matrix in WIDEA-n, exploit iterative differential (and linear) patterns that avoid the MAD-boxes altogether in every round. Such patterns cause zero input differences or zero input masks into each MAD-box, and thus, exploit the all-zero fixed point: $\text{MDS}(0, 0, 0, 0)^t = (0, 0, 0, 0)^t$. This approach is much more effective than the previous one because: (i) there are many fewer weak-subkey restrictions, (ii) it applies to any MDS matrix, (iii) we only attack one IDEA instance instead of n, which reduces considerably the attack complexity and increases the weak-key class size for the attack. The larger the value of n, the better the attacks become.

The implications of weak differential and linear attacks are not restricted to the block cipher setting. In [6], the authors suggested to use WIDEA-n in a compression function in Davies-Meyer mode. Our attacks lead to semi free-start collisions (depending on truncation of the final hash digest) or distinguishing attacks on the compression function using WIDEA-n in Davies-Meyer mode.

Even though the weak-key classes correspond to a small fraction of the key space, their existence implies that WIDEA-n are not ideal ciphers, and as such cannot be used in cryptographic constructions that require tight security, in the same way as IDEA [12].

Table 1. Attack complexities for the full 8.5-round WIDEA-n with $n \in \{4, 8\}$

cipher	attack type	complexity			# weak keys (‡)	comment
		data	time	memory		
WIDEA-4	DC (distinguishing)	2 CP	2	negl.	2^{242}	see (7)
	DC (key recovery)*	2 CP	2^{25}	negl.	2^{272}	see (7)
	LC (distinguishing)	32 KP	32	negl.	2^{242}	see (11)
	LC (key recovery)*	32 KP	2^{25}	negl.	2^{272}	see (11)
WIDEA-8	DC (distinguishing)	2 CP	2	negl.	2^{754}	see (7)
	DC (key recovery)*	2 CP	2^{24}	negl.	2^{784}	see (7)
	LC (distinguishing)	32 KP	32	negl.	2^{754}	see (11)
	LC (key recovery)*	32 KP	2^{24}	negl.	2^{784}	see (11)

CP: chosen plaintext; CC: chosen ciphertext; KP: known plaintext; *: partial key recovery; ‡: estimated

Open problems include: (i) how to recover the full (512- or 1024-bit) key of WIDEA-n; (ii) find weak keys that through the key schedule algorithms lead to weak round subkeys fitting in the requirements of our differential and linear distinguishers.

References

1. Courtois, N.: Algebraic complexity reduction and cryptanalysis of GOST, IACR ePrint archive 2011/626 (2011)
2. Daemen, J., Govaerts, R., Vandewalle, J.: Weak Keys for IDEA. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 224–231. Springer, Heidelberg (1994)
3. Dinur, I., Dunkelman, O., Shamir, A.: Improved attacks on full GOST, IACR ePrint archive, 2011/558 (2011)
4. FIPS197: Advanced Encryption Standard (AES), FIPS PUB 197 Federal Information Processing Standard Publication 197, U.S. Department of Commerce (2001)
5. ISO: Information Technology – Security Techniques – Hash functions – Part 3: Dedicated hash functions. ISO/IEC 10118-3:2004, International Organization for Standardization (2004)
6. Junod, P., Macchetti, M.: Revisiting the IDEA Philosophy. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 277–295. Springer, Heidelberg (2009)

7. Lai, X., Massey, J.L., Murphy, S.: Markov Ciphers and Differential Cryptanalysis. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 17–38. Springer, Heidelberg (1991)
8. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1997)
9. Nakahara Jr., J., Rijmen, V., Preneel, B., Vandewalle, J.: The MESH Block Ciphers. In: Chae, K., Yung, M. (eds.) WISA 2003. LNCS, vol. 2908, pp. 458–473. Springer, Heidelberg (2004)
10. SHS: Secure Hash Standard, Federal Information Processing Standards, FIPS PUB 180-3 (October 2008)
11. Vaudenay, S.: Related-key attack against triple encryption based on fixed points. In: SECRYPT 2011, pp. 59–67. SciTPress (2011)
12. Wei, L., Peyrin, T., Sokolowski, P., Ling, S., Pieprzyk, J., Wang, H.: On the (in)security of IDEA in various hashing modes. IACR ePrint archive, 2012/264 (2012)

A Appendix

Table 2. One-round characteristics of IDEA using xor differences and $\delta = 8000_x$

1-round characteristics	weak subkeys j-th round	diff. in MA-box
$(0,0,0,\delta) \to (\delta,\delta,\delta,0)$	$Z_{6(j-1)+3}, Z_{6(j-1)+5}$	$(0,\delta) \to (\delta,\delta)$
$(0,0,\delta,0) \to (\delta,0,0,0)$	$Z_{6(j-1)+4}, Z_{6(j-1)+5}$	$(\delta,0) \to (0,\delta)$
$(0,0,\delta,\delta) \to (0,\delta,\delta,0)$	$Z_{6(j-1)+3}, Z_{6(j-1)+4}$	$(\delta,\delta) \to (\delta,0)$
$(0,\delta,0,0) \to (\delta,\delta,0,\delta)$	$Z_{6(j-1)+5}$	$(0,\delta) \to (\delta,\delta)$
$(0,\delta,0,\delta) \to (0,0,\delta,\delta)$	$Z_{6(j-1)+3}$	**(0,0)→(0,0)**
$(0,\delta,\delta,0) \to (0,\delta,0,\delta)$	$Z_{6(j-1)+4}$	$(\delta,\delta) \to (\delta,0)$
$(0,\delta,\delta,\delta) \to (\delta,0,\delta,\delta)$	$Z_{6(j-1)+3}, Z_{6(j-1)+4}, Z_{6(j-1)+5}$	$(\delta,0) \to (0,\delta)$
$(\delta,0,0,0) \to (0,\delta,0,0)$	$Z_{6(j-1)}, Z_{6(j-1)+4}, Z_{6(j-1)+5}$	$(\delta,0) \to (0,\delta)$
$(\delta,0,0,\delta) \to (\delta,0,\delta,0)$	$Z_{6(j-1)}, Z_{6(j-1)+3}, Z_{6(j-1)+4}$	$(\delta,\delta) \to (\delta,0)$
$(\delta,0,\delta,0) \to (\delta,\delta,0,0)$	$Z_{6(j-1)}$	**(0,0)→(0,0)**
$(\delta,0,\delta,\delta) \to (0,0,\delta,0)$	$Z_{6(j-1)}, Z_{6(j-1)+3}, Z_{6(j-1)+5}$	$(0,\delta) \to (\delta,\delta)$
$(\delta,\delta,0,0) \to (\delta,0,0,\delta)$	$Z_{6(j-1)}, Z_{6(j-1)+4}$	$(\delta,\delta) \to (\delta,0)$
$(\delta,\delta,0,\delta) \to (0,\delta,\delta,\delta)$	$Z_{6(j-1)}, Z_{6(j-1)+3}, Z_{6(j-1)+4}, Z_{6(j-1)+5}$	$(\delta,0) \to (0,\delta)$
$(\delta,\delta,\delta,0) \to (0,0,0,\delta)$	$Z_{6(j-1)}, Z_{6(j-1)+5}$	$(0,\delta) \to (\delta,\delta)$
$(\delta,\delta,\delta,\delta) \to (\delta,\delta,\delta,\delta)$	$Z_{6(j-1)}, Z_{6(j-1)+3}$	**(0,0)→(0,0)**

Table 3. One-round linear relations of IDEA with $\gamma = 1$

1-round linear relations	weak subkeys j-th round	masks in MA-box
$(0,0,0,\gamma) \to (0,0,\gamma,0)$	$Z_{6(j-1)+3}, Z_{6(j-1)+5}$	$(0,\gamma) \to (\gamma,0)$
$(0,0,\gamma,0) \to (\gamma,0,\gamma,\gamma)$	$Z_{6(j-1)+4}, Z_{6(j-1)+5}$	$(\gamma,\gamma) \to (0,\gamma)$
$(0,0,\gamma,\gamma) \to (\gamma,0,0,\gamma)$	$Z_{6(j-1)+3}, Z_{6(j-1)+4}$	$(\gamma,0) \to (\gamma,\gamma)$
$(0,\gamma,0,0) \to (0,0,0,\gamma)$	$Z_{6(j-1)+5}$	$(0,\gamma) \to (\gamma,1)$
$(0,\gamma,0,\gamma) \to (0,0,\gamma,\gamma)$	$Z_{6(j-1)+3}$	**(0,0)→(0,0)**
$(0,\gamma,\gamma,0) \to (\gamma,0,\gamma,0)$	$Z_{6(j-1)+4}$	$(\gamma,0) \to (\gamma,\gamma)$
$(0,\gamma,\gamma,\gamma) \to (\gamma,0,0,0)$	$Z_{6(j-1)+3}, Z_{6(j-1)+4}, Z_{6(j-1)+5}$	$(\gamma,\gamma) \to (0,\gamma)$
$(\gamma,0,0,0) \to (0,\gamma,\gamma,\gamma)$	$Z_{6(j-1)}, Z_{6(j-1)+4}, Z_{6(j-1)+5}$	$(\gamma,\gamma) \to (0,\gamma)$
$(\gamma,0,0,\gamma) \to (0,\gamma,0,\gamma)$	$Z_{6(j-1)}, Z_{6(j-1)+3}, Z_{6(j-1)+4}$	$(\gamma,0) \to (\gamma,\gamma)$
$(\gamma,0,\gamma,0) \to (\gamma,\gamma,0,0)$	$Z_{6(j-1)}$	**(0,0)→(0,0)**
$(\gamma,0,\gamma,\gamma) \to (\gamma,\gamma,\gamma,0)$	$Z_{6(j-1)}, Z_{6(j-1)+3}, Z_{6(j-1)+5}$	$(0,\gamma) \to (\gamma,0)$
$(\gamma,\gamma,0,0) \to (0,\gamma,\gamma,0)$	$Z_{6(j-1)}, Z_{6(j-1)+4}$	$(\gamma,0) \to (\gamma,\gamma)$
$(\gamma,\gamma,0,\gamma) \to (0,\gamma,0,0)$	$Z_{6(j-1)}, Z_{6(j-1)+3}, Z_{6(j-1)+4}, Z_{6(j-1)+5}$	$(\gamma,\gamma) \to (0,\gamma)$
$(\gamma,\gamma,\gamma,0) \to (\gamma,\gamma,0,\gamma)$	$Z_{6(j-1)}, Z_{6(j-1)+5}$	$(0,\gamma) \to (\gamma,0)$
$(\gamma,\gamma,\gamma,\gamma) \to (\gamma,\gamma,\gamma,\gamma)$	$Z_{6(j-1)}, Z_{6(j-1)+3}$	**(0,0)→(0,0)**

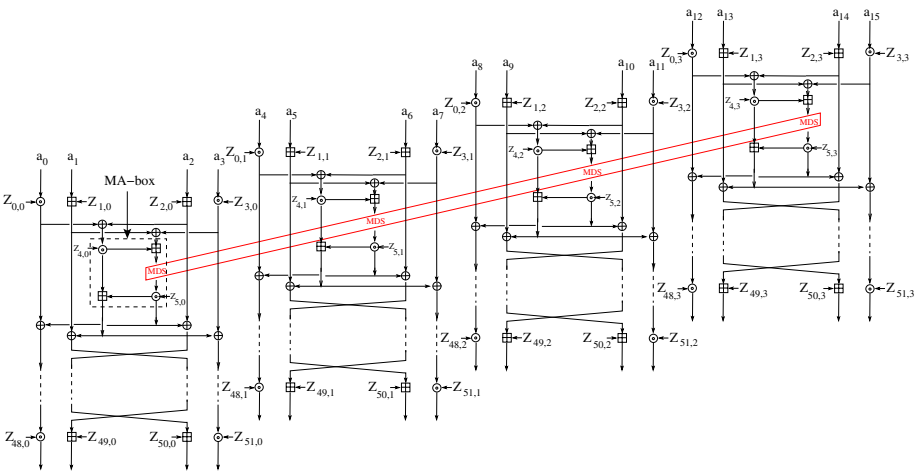


Fig. 1. Computational graph of the WIDEA-4 block cipher

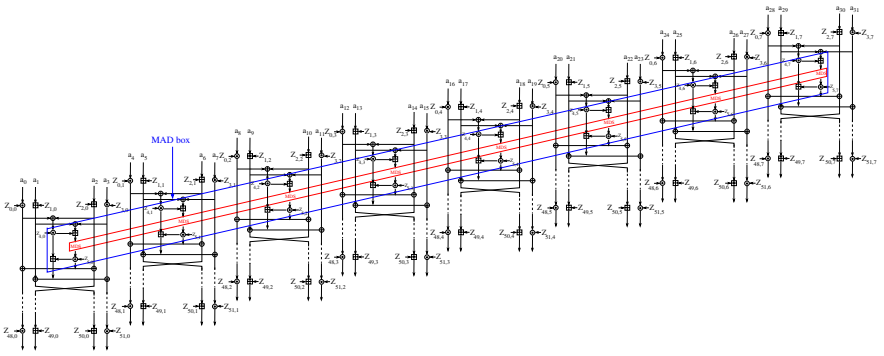


Fig. 2. Computational graph of the WIDEA-8 block cipher

Improved Linear Analysis on Block Cipher MULTI2*

Yi Lu, Liping Ding, and Yongji Wang

National Engineering Research Center of Fundamental Software,
Institute of Software, Chinese Academy of Sciences, Beijing, China

Abstract. Developed by Hitachi, MULTI2 is a block cipher used mainly to secure the multimedia content. It was registered in ISO/IEC 9979 and was patented in US and Japan. MULTI2 uses the Feistel structure and operates on the 64-bit blocks. The encryption key has 256 bits.

This paper studies the linear analysis on MULTI2. We give a detailed bias analysis on MULTI2 round functions. For the first time formal proofs on their bias properties are given. This allows to find a new 4-round bias 2^{-2}. Previously, the best 4-round bias $2^{-5.7}$ was proposed. Using our results on the MULTI2 round functions, we propose the linear attacks on r-round MUTLI2 to recover the encryption key. Our linear attack can recover the 256-bit encryption key in time 2^{46}, $2^{60.4}$, $2^{83.8}$, $2^{91.7}$, $2^{123.4}$, $2^{123.2}$ of r-round encryptions for $r = 8, 12, 16, 20, 24, 28$ respectively. Further, we can recover the 32-bit sub-key in last round much faster than the whole encryption key recovery, i.e., in time 2^{37} for $r = 8, 12, 16, 20, 24$. Note that previously, the best linear key-recovery attack was a 20-round attack with time $2^{93.4}$ (of 20-round encryptions) and data $2^{39.2}$. As ISO register recommends to use at least 32 rounds, our attacks remain to be theoretical and do not threaten security for the practical use currently.

Keywords: Block Cipher, Feistel Structure, MULTI2, Linear Analysis, Bias.

1 Introduction

MULTI2 is a block cipher developed by Hitachi. It has been used mainly to secure the multimedia content. It was registered in ISO/IEC 9979 in 1994 and was patented in US and Japan [7]. MULTI2 is the only cipher specified in Japanese standard ARIB [3] for conditional access systems. ARIB is the basic standard of the recent ISDB (Integrated Services Digital Broadcasting), which is the Japanese standard for digital television and digital radio. MULTI2 adopts the Feistel structure and operates on the 64-bit blocks. The encryption key has

* This work is supported by the National Science and Technology Major Project under Grant No. 2010ZX01036-001-002 & 2010ZX01037-001-002, the Knowledge Innovation Key Directional Program of Chinese Academy of Sciences under Grant No. KGCX2-YW-125 & KGCX2-YW-174, and the National Natural Science Foundation of China under Grant No. 61170072.

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 72–85, 2012.

Table 1. Our linear key-recovery attacks on r-round MULTI2, and the time unit is one r-round encryption

key-recovery attack	r	time	data
[2] linear attack	20	$2^{93.4}$	$2^{39.2}$
[2] guess-and-determine attack	any r	$2^{185.4}$	3
[2] related-key slide attack (strong assumptions on key relations)	$r = 0 \pmod 8$	$2^{128}/r$	2^{33}
this paper	8	2^{46}	2^{17}
this paper	12	$2^{60.4}$	2^{33}
this paper	16	$2^{83.8}$	2^{49}
this paper	20	$2^{91.7}$	2^{63}
this paper	24	$2^{123.4}$	2^{63}
this paper	28	$2^{123.2}$	2^{63}

256 bits. The ISO register recommends to use at least 32 rounds. For a survey on backgrounds and the attacks on MULTI2, we refer to [2].

This paper gives the linear analysis on MULTI2. Our main contributions are the following. First, we give a detailed bias analysis on MULTI2 round functions. In particular, we give formal proofs on the bias properties *for the first time*. This allows to discover new interesting results. Secondly, we give a new 4-round bias 2^{-2}. Previously, the best 4-round bias $2^{-5.7}$ was proposed in [2]. Thirdly, using our results on the MULTI2 round functions, we propose the linear attacks on r-round MUTLI2 to recover the 256-bit encryption key for $r = 8, 12, 16, 20, 24, 28$. In Table 1, we give our attack results and compare with the best known key-recovery attacks [2]. Note that to recover the 32-bit sub-key in the last round we need (time, data): $(2^{37}, 2^{12.4})$ for $r = 8$, $(2^{37}, 2^{23.8})$ for $r = 12$, $(2^{37}, 2^{35.2})$ for $r = 16$, $(2^{37}, 2^{46.6})$ for $r = 20$, and $(2^{37}, 2^{58})$ for $r = 24$ respectively, which are much faster than the whole key recovery. Previously, the best linear key-recovery attack was a 20-round attack [2] with time $2^{93.4}$ and data $2^{39.2}$. The rest of the paper is organized as following. In Sect. 2, we give preliminaries on MULTI2 and review the related work. In Sect. 3, we present detailed analysis on MULTI2 round functions with formal proofs. In Sect. 4, we give a new 4-round distinguisher. This is the best known 4-round linear attack. We give the key-recovery attacks on MULTI2 in Sect. 5. Finally, we conclude in Sect. 6.

2 Preliminaries on MULTI2

MULTI2 (Multi-Media Encryption Algorithm 2) is a standard balanced Feistel structured block cipher [10]. The block length is 64 bits and each half branch has 32 bits. The ISO register entry recommends at least 32 rounds. The round functions are $\pi_1, \pi_2, \pi_3, \pi_4$ repeatedly every 4 rounds. The basic operations are XOR (denoted by $\oplus$), modulo 2^{32} addition (denoted by $+$) and subtraction (denoted by $-$), bit-wise left rotation (denoted by $\lll$) and logical OR (denoted by OR). We give detailed description and analysis on $\pi_1, \pi_2, \pi_3, \pi_4$ in Sect. 3.

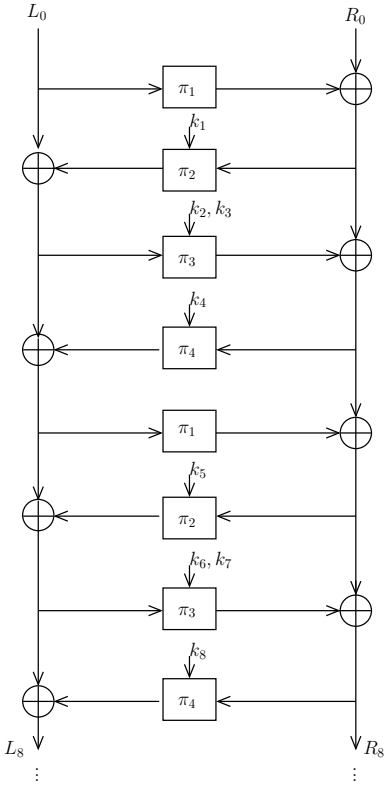


Fig. 1. MULTI2 encryption diagram

The 256-bit encryption key consists of $k_1, k_2, \ldots, k_8$ of 32 bits each. The eight sub-keys are derived from a 256-bit system key and a 64-bit data key (see [2] for details). In this paper, we restrict ourselves to the encryption key. Note that by the key schedule, we always have

$$k_4 = k_2 \oplus k_3, \tag{1}$$

$$k_8 = k_6 \oplus k_7. \tag{2}$$

Thus, the 256-bit encryption key only consists of $32 \times 6 = 192$ unknown key bits. Note that Round 1 through Round 4 use sub-keys $k_1, \ldots, k_4$ and Round 5 through Round 8 use sub-keys $k_5, \ldots, k_8$. The eight sub-keys are used repeatedly every 8 rounds. Throughout the paper, let the 32-bit L_0, R_0 be the left half and right half of the plaintext respectively. We let L_n, R_n denote the left half and right half of the n-round MULTI2 output. Figure 1 shows MULTI2 encryption diagram.

Previously, [1,2] studied linear analysis on MULTI2. Based on the 4-round bias $2^{-5.7}$ in [2], the best linear key-recovery attack [2] was proposed for 20-round MULTI2 to recover the encryption key. For this 20-round attack, [2] proposes to

recover $k_1, \ldots, k_4$ with $32 \times 3 = 96$ unknown bits[1] all together using $2^{39.2}$ data, which needs $2^{93.4}$ computations[2] of 20-round encryptions.

3 Detailed Analysis on MULTI2 Round Functions

We define the bias (also called imbalance [6, page 14]) of a binary random variable X by

$$\text{bias}(X) \stackrel{\text{def}}{=} |\Pr(X = 0) - \Pr(X = 1)|. \tag{3}$$

In this section, we discuss the bias properties for each of MULTI2 round functions in details. We not only give the detailed experimental results, but more importantly, we present formal proofs for our important results *for the first time*.

The round function π_1 is an identity function without key mixing, $\pi_1(L) \stackrel{\text{def}}{=} L$, which directly outputs the left half input to the right half. It is trivial to see that regardless of the distribution of L, we always have the bias 1 for $\alpha \cdot (L \oplus \pi_1(L))$ for any 32-bit α. Below, we discuss the bias properties for the other round functions.

3.1 Analysis of π_2

The function π_2 takes two inputs of 32 bits each, R and k_i, and it is defined by

$$\pi_2(R, k_i) = (x \lll 4) \oplus x, \tag{4}$$

and

$$x = ((R + k_i) \lll 1) + R + k_i - 1. \tag{5}$$

Previously, [2] reported the bias[3] 1 for $0\text{x}aaaaaaaa \cdot \pi_2(R, k_i)$ by experiments. Below, we give the complete results with formal proofs on the bias properties of π_2. Note that x as defined in (5) is not uniformly distributed assuming R, k_i are random with uniform distribution. We first analyze a simplified case, in which we let x come from a truly random source rather than generated by R, k_i.

Property 1. Given 32-bit x, let $\pi_2'(x) = (x \lll 4) \oplus x$. We assume that x is random with uniform distribution. Then, the bias for $\alpha \cdot \pi_2'(x)$ is 1, if α is in the set $\mathcal{E}_1$ defined below

```
{0x11111111, 0x22222222, 0x33333333,
 0x44444444, 0x55555555, 0x66666666,
 0x77777777, 0x88888888, 0x99999999,
 0xaaaaaaaa, 0xbbbbbbbb, 0xcccccccc,
 0xdddddddd, 0xeeeeeeee, 0xffffffff};
```

and the bias for $\alpha \cdot \pi_2'(x)$ is 0, if $\alpha \notin \mathcal{E}_1 \cup \{0\}$.

[1] By Eq.(1), $k_1, \ldots, k_4$ only contain 96 unknown bits.

[2] Throughout the paper, we adopt the conventional time unit of one r-round encryption for time estimate of the r-round attack; this makes easy to compare with the time complexity $O(2^L)$ of exhaustive search, where L denotes the key size.

[3] Note that our definition of bias slightly differs from [2] and our bias is always twice in quantity of that in [2]. For comparison, throughout the paper, we always cite the reported bias results from [2] by adjusting with our definition, which is twice of [2].

Proof. For the first half of the results, we show a more general result, i.e., the bias for $\alpha \cdot \pi_2'(x)$ is 1 *with any distribution* of x if $\alpha \in \mathcal{E}_1$. We demonstrate with the input mask $\alpha = 0x11111111$ assuming x complies with any distribution as follows. On one hand, we have $\alpha \cdot (x \lll 4) = x[28] \oplus x[0] \oplus x[4] \oplus x[8] \oplus x[12] \oplus x[16] \oplus x[20] \oplus x[24]$, where $x[0]$ denotes the Least Significant Bit (LSB) of x. On the other hand, we have $\alpha \cdot x = x[0] \oplus x[4] \oplus x[8] \oplus x[12] \oplus x[16] \oplus x[20] \oplus x[24] \oplus x[28]$. Thus, we always have $\alpha \cdot \pi_2'(x) \equiv 1$, for any x. Consequently, $\alpha \cdot \pi_2'(x)$ has bias 1 for any distribution of x. Our proof can be easily extended to other α's in $\mathcal{E}_1$. Meanwhile, we note the trivial case with $\alpha = 0$, in which the bias is 1.

To prove the second half of the results, first, we want to show that if $\alpha \notin \mathcal{E}_1 \cup \{0\}$, we have $\alpha \cdot \pi_2'(x) = b \cdot x$ for a nonzero 32-bit b. Let $y = \pi_2'(x)$. Let $y[31]$ and $y[0]$ denote the MSB and the LSB of y respectively. We have $(y[31], \ldots, y[0]) = (x[31] \oplus x[27], x[30] \oplus x[26], x[29] \oplus x[25], x[28] \oplus x[24], \ldots, x[7] \oplus x[3], x[6] \oplus x[2], x[5] \oplus x[1], x[4] \oplus x[0], x[3] \oplus x[31], x[2] \oplus x[30], x[1] \oplus x[29], x[0] \oplus x[28])$. And $\alpha \cdot \pi_2'(x) = \alpha \cdot y$ is linear in $x = (x[31], \ldots, x[0])$. So, there exists a 32-bit b, such that $\alpha \cdot \pi_2'(x) = b \cdot x$ for all x. We now prove $b \neq 0$ if $\alpha \notin \mathcal{E}_1 \cup \{0\}$ by contradiction. Suppose $b = 0$, we have $\alpha \cdot \pi_2'(x) = 0$, for all x. By linear algebra, we can check that the solution set to $\alpha \cdot (y[31], \ldots, y[0])^t = \mathbf{0}$ for all x (where α is the unknown variable and $\mathbf{0}$ denotes 32-bit zero vector) is $\mathcal{E}_1 \cup \{0\}$, because the basis for the kernel is $\{0x11111111, 0x22222222, 0x44444444, 0x88888888\}$. This contradicts the assumption $\alpha \notin \mathcal{E}_1 \cup \{0\}$. Hence, we know if $\alpha \notin \mathcal{E}_1 \cup \{0\}$, then we must have $\alpha \cdot \pi_2'(x) = b \cdot x$ for a nonzero 32-bit b,

Secondly, assuming x is uniformly distributed over $GF(2)^{32}$, we know that $x[31], \ldots, x[0]$ are 32 i.i.d. balanced binary random variables[4]. So, if $b \neq 0$, the bit $b \cdot x$ is balanced too. Consequently, we know if $\alpha \notin \mathcal{E}_1 \cup \{0\}$, the bit $\alpha \cdot \pi_2'(x)$ is balanced. □

From the first half of above proof, we can immediately get the following result for the case, in which we let x be defined in (5) by R, k_i rather than come from a truly random source.

Property 2. Assuming that R, k_i are random with uniform distribution, the bias for $\alpha \cdot \pi_2(R, k_i)$ is 1 when $\alpha \in \mathcal{E}_1$.

Further, our computation shows that the second largest bias $2^{-6.2}$ is obtained when α is in the set $\mathcal{E}_2$ defined below

```
{0x0f0f0f0f, 0x1e1e1e1e, 0x2d2d2d2d, 0x3c3c3c3c,
 0x4b4b4b4b, 0x5a5a5a5a, 0x69696969, 0x78787878,
 0x87878787, 0x96969696, 0xa5a5a5a5, 0xb4b4b4b4,
 0xc3c3c3c3, 0xd2d2d2d2, 0xe1e1e1e1, 0xf0f0f0f0}.
```

[4] We call a binary random variable *balanced* if it is uniformly distributed. Note that the bias for a balanced binary random variable is 0.

3.2 Analysis of π_3

The function π_3 takes three inputs of 32 bits each, L and k_i and k_j, and it is defined by

$$\pi_3(L, k_i, k_j) = (x \lll 16) \oplus (x \text{ OR } L), \tag{6}$$

where OR denotes the logical OR, and

$$x = (((y \lll 8) \oplus y + k_j) \lll 1) - ((y \lll 8) \oplus y + k_j),$$

and $y = ((L + k_i) \lll 2) + L + k_i + 1$.

Note that the input space of π_3 has 2^{96} possibilities. Generally speaking, a good π_3 should make it impossible to obtain bias characteristics, due to impossibility of exhaustive enumeration. Nevertheless, we point out that π_3 was not well designed actually. Our experiments already confirm that the largest bias 2^{-2} for $\alpha \cdot \pi_3(L, k_i, k_j)$ is achieved when α is in the set $\mathcal{E}_3$ defined below

```
{0x10001,   0x20002,    0x40004,    0x80008,
0x100010,   0x200020,   0x400040,   0x800080,
0x1000100,  0x2000200,  0x4000400,  0x8000800,
0x10001000, 0x20002000, 0x40004000, 0x80008000}.
```

Property 3. Assuming that L, k_i, k_j are random with uniform distribution, the bias for $\alpha \cdot \pi_3(L, k_i, k_j)$ is 2^{-2} when $\alpha \in \mathcal{E}_3$.

Proof. We let the function $\pi_3'(L, u)$ have two inputs of 32 bits,

$$\pi_3'(L, u) = (x \lll 16) \oplus (x \text{ OR } L), \tag{7}$$

where

$$x = (u \lll 1) - u. \tag{8}$$

According to definition of π_3, it is easy to see that assuming L, u are independent and uniformly distributed, the output distribution of $\pi_3'(L, u)$ is identical to that of $\pi_3(L, k_i, k_j)$ with L, k_i, k_j being independent and uniformly distributed. So, we just need to show that when L, u are independent and uniformly distributed, the bias for $\alpha \cdot \pi_3'(L, u)$ is 2^{-2} when $\alpha \in \mathcal{E}_3$. For $\alpha =$ 0x10001, we look at this bit

$$\beta = x[0] \oplus x[16] \oplus (x[0] \text{ OR } L[0]) \oplus (x[16] \text{ OR } L[16]). \tag{9}$$

According to the two bits $L[0], L[16]$, we have four cases.

Case One: $L[0] = L[16] = 0$. We have $\beta_1 \equiv 0$, regardless of the values of $x[0], x[16]$.

Case Two: $L[0] = L[16] = 1$. We have $\beta_2 = x[0] \oplus x[16]$.

Case Three: $L[0] = 1, L[16] = 0$. We have $\beta_3 = \overline{x[0]}$ (i.e., the complement of $x[0]$).

Case Four: $L[0] = 0, L[16] = 1$. We have $\beta_4 = \overline{x[16]}$.

The bias for $\beta_2, \beta_3, \beta_4$ can be obtained, because we can easily compute the bias for x in (8) when u is uniformly distributed. Computation shows that the bias for $\beta_2, \beta_3, \beta_4$ is zero. As L is uniformly distributed, we have the bias for $\beta = (1+0+0+0)/4 = 2^{-2}$. For other α's in $\mathcal{E}_3$, we can adjust Eq.(9) and our proof follows similarly with the four cases. □

Interestingly, with regards to the bias for $\alpha \cdot (L \oplus \pi_3(L, k_i, k_j))$, we have the same results.

Property 4. Assuming that L, k_i, k_j are random with uniform distribution, the bias for $\alpha \cdot (L \oplus \pi_3(L, k_i, k_j))$ is 2^{-2} when $\alpha \in \mathcal{E}_3$.

Proof. Recall assuming L, u are independent and uniformly distributed, the output distribution of $\pi_3'(L, u)$ is identical to that of $\pi_3(L, k_i, k_j)$ with L, k_i, k_j being independent and uniformly distributed. Further, given L, the distribution of $\pi_3'(L, u)$ (with uniformly distributed u) is identical to that of $\pi_3(L, k_i, k_j)$ with uniformly distributed k_i, k_j. Thus, we have the output distribution of $\pi_3'(L, u) \oplus L$ is identical to that of $\pi_3(L, k_i, k_j) \oplus L$ with uniformly distributed inputs.

Therefore, we just need to show that when L, u are independent and uniformly distributed, the bias for $\alpha \cdot (L \oplus \pi_3'(L, u))$ is 2^{-2} when $\alpha \in \mathcal{E}_3$ to complete the proof. For $\alpha =$ 0x10001, we look at this bit

$$\beta' = x[0] \oplus x[16] \oplus (x[0] \text{ OR } L[0]) \oplus (x[16] \text{ OR } L[16]) \oplus L[0] \oplus L[16]. \quad (10)$$

According to the two bits $L[0], L[16]$, we have four cases.

Case One: $L[0] = L[16] = 0$. We have $\beta_1' \equiv 0$, regardless of the values of $x[0], x[16]$.

Case Two: $L[0] = L[16] = 1$. We have $\beta_2' = x[0] \oplus x[16]$.

Case Three: $L[0] = 1, L[16] = 0$. We have $\beta_3' = x[0]$.

Case Four: $L[0] = 0, L[16] = 1$. We have $\beta_4' = x[16]$.

Our computation shows that the bias for $\beta_2', \beta_3', \beta_4'$ is all zero, when u is uniformly distributed. As L is uniformly distributed, we have the bias for $\beta' = (1+0+0+0)/4 = 2^{-2}$. For other α's in $\mathcal{E}_3$, we can adjust Eq.(10) and our proof follows similarly with the four cases. □

Note that we can easily extend Property 3 and Property 4 to other symmetric mask patterns similarly. For illustration, the results below are useful for our attacks later (and proofs are omitted).

Property 5. Assuming that L, k_i, k_j are random with uniform distribution, the bias for $\alpha \cdot \pi_3(L, k_i, k_j)$ and $\alpha \cdot (L \oplus \pi_3(L, k_i, k_j))$ both are 2^{-8} when α is in the set $\mathcal{E}_4$ defined by

$$\mathcal{E}_4 = \{0\text{x}11111111, 0\text{x}22222222, 0\text{x}44444444, 0\text{x}88888888\}. \quad (11)$$

Note that previously, [2] reported the estimated bias $2^{-7.6}$ with $\alpha =$ 0x88888888 (which is in $\mathcal{E}_4$) by experiments. From that, [2] estimates $\alpha \cdot (R_0 \oplus R_4)$ has the bias $2^{-7.6}$, when $\alpha =$ 0x88888888. By Property 5, it is easy to see that $\alpha \cdot (R_0 \oplus R_4)$ has the *precise* bias of 2^{-8} when $\alpha \in \mathcal{E}_4$.

3.3 Analysis of π_4

The function π_4 takes two inputs of 32 bits each, R and k_i, and it is defined by

$$\pi_4(R, k_i) = (x \lll 2) + x + 1, \tag{12}$$

and $x = R + k_i$. The largest bias[5] for $\alpha \cdot \pi_4(R, k_i)$ is $2^{-5.7}$ with $\alpha = 0x\mathit{aaaaaaaa}$. Based on that, [2] proposes the best 4-round linear distinguisher, i.e., $\alpha \cdot (L_0 \oplus L_4)$ has the bias $2^{-5.7}$ when $\alpha = 0x\mathit{aaaaaaaa}$. Moreover, there exist 81 biases in total, which are all larger than 2^{-7}. For completeness, we list them in Table 2 and the largest one is underlined.

Table 2. Complete list of all the biases $\alpha \cdot \pi_4(R, k_i)$ which are larger than 2^{-7}

α	bias	α	bias	α	bias	α	bias
0x55555557	$2^{-6.2}$	0x55555575	$2^{-6.2}$	0x55555755	$2^{-6.2}$	0x55557555	$2^{-6.2}$
0x55575555	$2^{-6.2}$	0x55755555	$2^{-6.2}$	0x57555555	$2^{-6.2}$	0x75555555	$2^{-6.2}$
$\underline{0x\mathit{aaaaaaaa}}$	$\underline{2^{-5.7}}$	$0x\mathit{aaaaaaae}$	$2^{-6.6}$	$0x\mathit{aaaaaaaf}$	$2^{-6.6}$	$0x\mathit{aaaaaabe}$	$2^{-6.6}$
$0x\mathit{aaaaaaea}$	$2^{-6.6}$	$0x\mathit{aaaaaaeb}$	$2^{-6.6}$	$0x\mathit{aaaaaafa}$	$2^{-6.6}$	$0x\mathit{aaaaabae}$	$2^{-6.6}$
$0x\mathit{aaaaabea}$	$2^{-6.6}$	$0x\mathit{aaaaaeaa}$	$2^{-6.6}$	$0x\mathit{aaaaaeab}$	$2^{-6.6}$	$0x\mathit{aaaaaeba}$	$2^{-6.6}$
$0x\mathit{aaaaafaa}$	$2^{-6.6}$	$0x\mathit{aaaabaae}$	$2^{-6.6}$	$0x\mathit{aaaabaea}$	$2^{-6.6}$	$0x\mathit{aaaabeaa}$	$2^{-6.6}$
$0x\mathit{aaaaeaaa}$	$2^{-6.6}$	$0x\mathit{aaaaeaab}$	$2^{-6.6}$	$0x\mathit{aaaaeaba}$	$2^{-6.6}$	$0x\mathit{aaaaebaa}$	$2^{-6.6}$
$0x\mathit{aaaafaaa}$	$2^{-6.6}$	$0x\mathit{aaabaaae}$	$2^{-6.6}$	$0x\mathit{aaabaaea}$	$2^{-6.6}$	$0x\mathit{aaabaeaa}$	$2^{-6.6}$
$0x\mathit{aaabeaaa}$	$2^{-6.6}$	$0x\mathit{aaaeaaaa}$	$2^{-6.6}$	$0x\mathit{aaaeaaab}$	$2^{-6.6}$	$0x\mathit{aaaeaaba}$	$2^{-6.6}$
$0x\mathit{aaaeabaa}$	$2^{-6.6}$	$0x\mathit{aaaebaaa}$	$2^{-6.6}$	$0x\mathit{aaafaaaa}$	$2^{-6.6}$	$0x\mathit{aabaaaae}$	$2^{-6.6}$
$0x\mathit{aabaaaea}$	$2^{-6.6}$	$0x\mathit{aabaaeaa}$	$2^{-6.6}$	$0x\mathit{aabaeaaa}$	$2^{-6.6}$	$0x\mathit{aabeaaaa}$	$2^{-6.6}$
$0x\mathit{aaeaaaaa}$	$2^{-6.6}$	$0x\mathit{aaeaaaab}$	$2^{-6.6}$	$0x\mathit{aaeaaaba}$	$2^{-6.6}$	$0x\mathit{aaeaabaa}$	$2^{-6.6}$
$0x\mathit{aaeabaaa}$	$2^{-6.6}$	$0x\mathit{aaebaaaa}$	$2^{-6.6}$	$0x\mathit{aafaaaaa}$	$2^{-6.6}$	$0x\mathit{abaaaaae}$	$2^{-6.6}$
$0x\mathit{abaaaaea}$	$2^{-6.6}$	$0x\mathit{abaaaeaa}$	$2^{-6.6}$	$0x\mathit{abaaeaaa}$	$2^{-6.6}$	$0x\mathit{abaeaaaa}$	$2^{-6.6}$
$0x\mathit{abeaaaaa}$	$2^{-6.6}$	$0x\mathit{aeaaaaaa}$	$2^{-6.6}$	$0x\mathit{aeaaaaab}$	$2^{-6.6}$	$0x\mathit{aeaaaaba}$	$2^{-6.6}$
$0x\mathit{aeaaabaa}$	$2^{-6.6}$	$0x\mathit{aeaabaaa}$	$2^{-6.6}$	$0x\mathit{aeabaaaa}$	$2^{-6.6}$	$0x\mathit{aebaaaaa}$	$2^{-6.6}$
$0x\mathit{afaaaaaa}$	$2^{-6.6}$	$0x\mathit{baaaaaae}$	$2^{-6.6}$	$0x\mathit{baaaaaea}$	$2^{-6.6}$	$0x\mathit{baaaaeaa}$	$2^{-6.6}$
$0x\mathit{baaaeaaa}$	$2^{-6.6}$	$0x\mathit{baaeaaaa}$	$2^{-6.6}$	$0x\mathit{baeaaaaa}$	$2^{-6.6}$	$0x\mathit{beaaaaaa}$	$2^{-6.6}$
$0x\mathit{eaaaaaaa}$	$2^{-6.2}$	$0x\mathit{eaaaaaab}$	$2^{-6.2}$	$0x\mathit{eaaaaaba}$	$2^{-6.2}$	$0x\mathit{eaaaabaa}$	$2^{-6.2}$
$0x\mathit{eaaabaaa}$	$2^{-6.2}$	$0x\mathit{eaabaaaa}$	$2^{-6.2}$	$0x\mathit{eabaaaaa}$	$2^{-6.2}$	$0x\mathit{ebaaaaaa}$	$2^{-6.2}$
$0x\mathit{faaaaaaa}$	$2^{-6.2}$						

4 New 4-Round Distinguisher on MULTI2

In this section, we discover a much larger bias (i.e., 2^{-2}) for 4-round MULTI2 and give an improved 4-round distinguishing attack. Given the 32-bit (nonzero) α, we study the bias (see Figure 2),

$$\alpha \cdot L_0 \oplus \alpha \cdot R_0 \oplus \alpha \cdot R_4. \tag{13}$$

[5] Note that this bias $2^{-5.7}$ was reported in [2] without mentioning this is the largest.

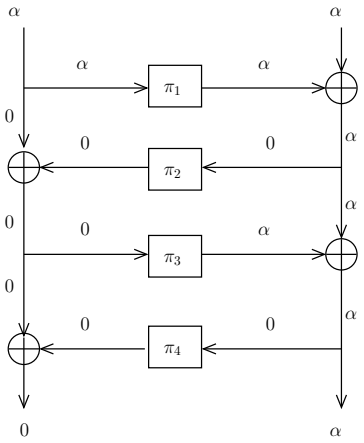


Fig. 2. The 4-round linear trail

Note that we have $R_0 \oplus R_4 = \pi_1(L_0) \oplus \pi_3(x, k_2, k_3) = L_0 \oplus \pi_3(x, k_2, k_3)$, where $x = L_0 \oplus \pi_2(L_0 \oplus R_0, k_1)$. So, (13) is equal to $\alpha \cdot \pi_3(x, k_2, k_3)$. By Property 3 in Sect. 3.2, we know when $\alpha \in \mathcal{E}_3$, (13) produces the bias $d = 2^{-2}$. This is much larger than the best 4-round bias $2^{-5.7}$ in [2]. As a known plaintext attack, the number N of known plaintexts required in linear cryptanalysis [8] is proportional to d^{-2}, where d is the bias for the linear relation. If N is taken as $2 \cdot d^{-2}$, the attack will be successful with very high probability. So, our new bias implies the 4-round MULTI2 distinguisher with $O(2^5)$ samples. In contrast, based on the known bias $2^{-5.7}$, the 4-round distinguisher in [2] would need $O(2^{12.4})$ samples. Note that though our 4-round bias cannot be iterated, i.e., the input mask (α, α) is not equal to the output mask $(0, \alpha)$ (see Figure 2), we will see in the next section that it is useful for our key-recovery attacks.

5 Improved Key-Recovery Attacks

Given a Feistel-structured block cipher, in linear cryptanalysis, we have a biased linear relation for k rounds, with the input masks β, α and output masks β', α' (for the left and right half respectively),

$$\beta \cdot L_0 \oplus \alpha \cdot R_0 \approx \beta' \cdot L_k \oplus \alpha' \cdot R_k \tag{14}$$

holds with bias d, i.e., the bit $\beta \cdot L_0 \oplus \alpha \cdot R_0 \oplus \beta' \cdot L_k \oplus \alpha' \cdot R_k$ has bias d. Usually, we look for the biased linear relation with $\beta = \beta'$ and $\alpha = \alpha'$ so that we can iterate the linear relation (14) n times to obtain a biased linear relation for nk rounds. That is, we have $\beta \cdot L_0 \oplus \alpha \cdot R_0 \approx \beta \cdot L_{nk} \oplus \alpha \cdot R_{nk}$ holds with the reduced bias d^n by Piling-up lemma [8]. The biased linear relation is known to be useful to make key-recovery attacks on $(nk+1)$ rounds by Matsui's algorithm 2 [8]. Its basic idea [8] is to use this nk-round bias to construct a nk-round distinguisher, which obtains

the nk-round outputs from the $(nk+1)$-round ciphertexts by exhaustively trying all possible ℓ-bit sub-keys in last round. Assuming that only the correct sub-key would yield a biased nk-round linear relation (with bias d^n), the distinguisher works with $O(d^{-2n})$ pairs of known plaintexts and $(nk+1)$-round ciphertexts, which allows to get this sub-key. Let N be the data complexity of the attack. The time complexity of the original Matsui's algorithm 2 [8] is $O(2^\ell N)$. For a survey on the improved Matsui's algorithm 2 [9] and recent optimization techniques, we refer to [5].

5.1 Our 8-Round Key-Recovery Attack

Given the 32-bit L_0, R_0, L_8, R_8, we aim to recover k_1, k_2, ..., k_8. From [2], we know $\alpha \cdot (L_0 \oplus L_4)$ has the bias $d = 2^{-5.7}$, when $\alpha = 0x\mathit{aaaaaaaa}$. By Property 2 in Sect. 3.1, we know $\alpha \cdot \pi_2(R, k_i)$ has bias 1 with $\alpha = 0x\mathit{aaaaaaaa}$. The 8-round linear trail (see Figure 3, Appendix) allows to deduce an important property, that is, $\alpha \cdot (L_0 \oplus L_7)$ (with $\alpha = 0x\mathit{aaaaaaaa}$) has exactly the same bias d as $\alpha \cdot (L_0 \oplus L_4)$. Therefore, using Matsui's algorithm 2 [8], we recover k_8, which is used to compute π_4 at Round 8 with the data complexity $2 \cdot d^{-2} = 2^{12.4}$. We need time $2^{12.4} \times 2^{32} = 2^{44.4}$.

Now, starting from the recovered L_7, R_7, for all possible (k'_5, k'_6, k'_7), when we decrypt back from the end of Round 7 to the end of Round 4, we obtain the corresponding L'_4. We claim that $\alpha \cdot (L_0 \oplus L'_4)$ has the same bias d as $\alpha \cdot (L_0 \oplus L_7)$ with $\alpha = 0x\mathit{aaaaaaaa}$. We briefly explain the reason as follows. Let L'_i denote the left half at Round i obtained with the sub-key candidate (k'_5, k'_6, k'_7) by going backward from Round 7. The Feistel structure allows to have both $L_7 = L'_6$ and $L'_6 = L'_5 \oplus \pi_2(R'_6, k'_5)$ hold true always. From proofs of Property 1 in Sect. 3.1, we know $\alpha \cdot \pi_2(R'_6, k'_5)$ with $\alpha = 0x\mathit{aaaaaaaa}$ has bias 1 regardless of the inputs R'_6, k'_5. Thus, we can deduce $\alpha \cdot L_7 = \alpha \cdot L'_5 = \alpha \cdot L'_4$ holds true always for all (k'_5, k'_6, k'_7). Consequently, we have the equality $\alpha \cdot (L_0 \oplus L_7) = \alpha \cdot (L_0 \oplus L'_4)$ with $\alpha = 0x\mathit{aaaaaaaa}$ holds true always, which completes our proof.

This important result implies that to recover k_5, k_6, k_7, k_8, we *cannot* directly apply the key-recovery attack idea [2], which is based on the linear distinguisher. This is because the correct key and all the wrong keys of the form (k'_5, k'_6, k'_7, k_8) all produce the same sequence (of same bias d), and we cannot distinguish the correct key (k_5, k_6, k_7) from the wrong keys (k'_5, k'_6, k'_7). In fact, we can only obtain the correct k_8. This serves as a counter-example of the wrong key randomization hypothesis, which is also discussed in [4].

To solve this problem, we proceed as follows to recover the other sub-keys. Recall the 4-round bias as mentioned in Sect. 3.2, i.e., $\alpha' \cdot (R_0 \oplus R_4)$ has the bias $d' = 2^{-8}$, when $\alpha' = 0x88888888$. Similarly as done before, we can show that $\alpha' \cdot (R_0 \oplus L_7 \oplus R_6)$ has the same bias d' as $\alpha' \cdot (R_0 \oplus R_4)$ (see Figure 4, Appendix). By Eq.(2), k_6, k_7 only have 32 bits unknown, given k_8. To recover k_6, k_7, which are used to compute π_3 at Round 7, we need to have data $2 \cdot d'^{-2} = 2^{17}$ and time $2^{32} \times 2^{17} = 2^{49}$.

After recovering k_6, k_7, we obtain L_6, R_6. We use $\alpha'' \cdot (L_0 \oplus R_0 \oplus R_4)$ with $\alpha'' = 0x10001$ and the bias $d'' = 2^{-2}$, as introduced in Sect. 4. We get k_5, which

Table 3. Our 8-round attack details to recover k_5, k_6, k_7, k_8

step	sub-keys	bias used	bias	time	data
No.1	k_8	0x$aaaaaaaa \cdot (L_0 \oplus L_7)$	$2^{-5.7}$	$2^{44.4}$	$2^{12.4}$
No.2	k_6, k_7	0x88888888 $\cdot (R_0 \oplus L_7 \oplus R_6)$	2^{-8}	2^{49}	2^{17}
No.3	k_5	0x10001 $\cdot (L_0 \oplus R_0 \oplus R_4)$	2^{-2}	2^{37}	2^5

is involved in π_2 at Round 5 with data $2 \cdot d''^{-2} = 2^5$ and time $2^{32} \times 2^5 = 2^{37}$. We give our results on recovering $k_5, \ldots, k_8$ in Table 3. Similarly, we can recover $k_1, \ldots, k_4$ with obviously much less data and time costs. In total, with time $O(2^{49})$ of one-round encryption (i.e., $O(2^{46})$ of 8-round encryptions), and data $O(2^{17})$, we recover $k_1, \ldots, k_8$.

5.2 Key-Recovery Attacks on Higher Rounds

Our 12-Round Attack: We can recover k_4 (in Round 12) and k_2, k_3 (in Round 11) as we do to get k_6, k_7, k_8 on the previous 8-round attack. To recover k_1, we note that the 4-round bias we used in above 8-round attack (as introduced in Sect. 4) cannot be iterated for higher rounds. However, we can still use this 4-round bias to get k_1 as follows. We peel off the first and last 4 rounds of the 12-round MULTI2 by guessing k_1, based on $\alpha'' \cdot (L_4 \oplus R_4 \oplus R_8)$ with $\alpha'' =$ 0x10001 and the bias $d'' = 2^{-2}$. Once we obtain the keys k_1, k_2, k_3, k_4, we can easily recover the keys $k_5, \ldots, k_8$ for the middle 4 rounds. Following the discussions of our 8-round attack, we know the data complexity for the r-round attack, is determined by the bias d' for $\alpha' \cdot (R_0 \oplus R_{r-4})$ with $\alpha' =$ 0x88888888, based on which we recover k_2, k_3 (in Round 11). For $r = 12$, we have $d' = (2^{-8})^2 = 2^{-16}$. The data complexity of the 12-round attack is thus calculated by $2 \times d'^{-2} = 2^{33}$. The time complexity of the attack is dominated by the time to get k_2, k_3 (in Round 11). By (11), k_2, k_3 have 32 bits unknown, given k_4. So, by the improved Matsui's algorithm 2 [9], the time complexity of the attack is $2^{32} \times 2^{32} = 2^{64}$ of one-round encryption, i.e., $O(2^{60.4})$ of 12-round encryptions.

Our 16-Round Attack: We recover k_6, k_7, k_8 (used in Round 15 and Round 16) similarly as before. For k_5 (in Round 14), we note that the previous trick used in the 12-round attack does not work, because the first 4 rounds and last 4 rounds of the 16-round MULTI2 use different sub-keys. Hence, we consider to use $\alpha \cdot (L_0 \oplus L_8)$ (with $\alpha =$ 0x$aaaaaaaa$ and the bias $d = 2^{-5.7 \times 2} = 2^{-11.4}$) and to recover k_4 (Round 12) and k_5 (Round 14) together. This step will take data $2 \times d^{-2} = 2^{23.8}$ and time $2^{23.8} \times 2^{64} = 2^{87.8}$. After that, the attack follows the same procedure as the 12-round attack above. For the data complexity, we note that to recover k_6, k_7 (in Round 15), we use $\alpha' \cdot (R_0 \oplus R_{12})$ with $\alpha' =$ 0x88888888 and bias $d' = (2^{-8})^3 = 2^{-24}$. The data complexity is thus calculated as $2 \times d'^{-2} = 2^{49}$. Clearly,

the time complexity of the attack is dominated[6] by the step to get k_4, k_5 together, i.e., $2^{87.8}$ of one-round encryption, that is, $O(2^{83.8})$ of 16-round encryptions.

Our 20-Round Attack: Note that the first and last 4 rounds both use k_1, k_2, k_3, k_4. First, we obtain k_4 (in Round 20) by using $\alpha \cdot (L_0 \oplus L_{16})$ with $\alpha = 0xaaaaaaaa$ and bias $d = 2^{-5.7\times4} = 2^{-22.8}$. This step takes time $O(2^{64})$ of one round encryptions by improved Matsui's algorithm 2 and data $2d^{-2} = 2^{46.6}$. Next, we want to get t k_2, k_3 with 32 unknown bits for Round 19. We take the previous approach and use $\alpha' \cdot (R_0 \oplus R_{16})$ with $\alpha' = 0x88888888$ and bias $d' = 2^{-32}$. By Property 5 in Sect. 3.2, we know when $\alpha' \in \mathcal{E}_4$, the bias for $\alpha' \cdot (R_0 \oplus R_{16})$ is the same d'. This allows to get k_2, k_3 with data $2d'^{-2} \times \frac{1}{4} = 2^{63}$ by using the multi-bias approach. Similarly as our 16-round attack, we recover k_1 (Round 18) and k_8 (Round 16) of 64 bits together. We need the data $2 \times (2^{-5.7\times3})^{-2} = 2^{35.2}$ and time $2^{64} \times 2^{32}$, i.e., $O(2^{96})$, by the improved Matsui's algorithm 2 [9]. We then use our 16-round attack to recover the remaining keys. The total data complexity of the attack is 2^{63}. We need total time 2^{96} of one-round encryption, i.e., $O(2^{91.7})$ of 20-round encryptions.

Our 24-Round Attack: We get k_8 in Round 24 using $\alpha \cdot (L_0 \oplus L_{20})$ with bias $d = 2^{-5.7\times5} = 2^{-28.5}$. We need data $2d^{-2} = 2^{58}$ and time 2^{64} of one-round encryption (i.e., $2^{59.4}$ of 24-round encryptions) to obtain k_8. To recover k_6, k_7 in Round 23, the previous approach, which is based on $\alpha' \cdot (R_0 \oplus R_{20})$ with $\alpha' = 0x88888888$ and the bias $d' = (2^{-8})^5 = 2^{-40}$, would need data amount higher than the maximum 2^{64}. So, we propose to recover many keys together. Due to the same reason as explained in Sect. 5.1, we *cannot* recover k_5, k_6, k_7 together by simply going backwards three rounds from the end of Round 23. However, by going backwards four rounds from the end of Round 23, we can recover k_4, k_5, k_6, k_7 together based on $\alpha \cdot (L_0 \oplus L_{19})$ with $\alpha = 0xaaaaaaaa$ and the bias $d = 2^{-5.7\times4} = 2^{-22.8}$. With data $2d^{-2} = 2^{46.6}$, we get $k_4, \ldots, k_7$ with 96 unknown bits. We need time $2^{96} \times 2^{32} = 2^{128}$ one-round encryptions by the improved Matsuis algorithm 2 [9]. After that, the attack degrades to the previous 20-round attack, which works with data 2^{63}. In total, the time cost is $2^{128}/24 = 2^{123.4}$ 24-round encryptions and the data cost is 2^{63}.

Our 28-Round Attack: If we use $\alpha \cdot (L_0 \oplus L_{24})$ with $\alpha = 0xaaaaaaaa$ and the bias $d = 2^{-5.7\times6} = 2^{-34.2}$ to get k_4 in Round 28, we need data $2d^{-2} = 2^{68.4}$, which is above the maximum 2^{64}. So, in our fist step, we apply the linear attack idea [2] on 20-round MULTI2, which uses the fact that the first and last 4 rounds use same sub-keys $k_1, \ldots, k_4$. So, we use $\alpha \cdot (L_4 \oplus L_{24})$ with the bias $d = 2^{-5.7\times5} = 2^{-28.5}$. With data $2d^{-2} = 2^{57}$, we recover $k_1, \ldots, k_4$ with 96 unknown bits. This step takes time $2^{96} \times 2^{32} = 2^{128}$ one-round encryptions by the improved Matsuis algorithm 2. After that, we peel off the first and last 4 rounds and the attack degrades to the previous 20-round attack, which works

[6] Note that to recover k_6, k_7 (in Round 15) with 32 unknown bits, by improved Matsui's algorithm 2 [9], we need time $2^{32} \times 2^{32}$, i.e., $O(2^{64})$.

with data 2^{63} and time 2^{96} one-round encryptions. In total, our attack works with data 2^{63} and time 2^{128} one-round encryptions, i.e., $2^{123.2}$ 28-round encryptions. Finally, we give our results in Table 1 and compare with the best known key-recovery attacks [2] on MULTI2. It is interesting to see that for our r-round attacks, the time complexities (calculated in units of one-round encryption) are the same for $r = 24, 28$; further, the data complexities are the same for $r = 20, 24, 28$. Meanwhile, following our discussions, recall that we can recover the last round sub-key of 32 bits with (time, data): $(2^{44.4}, 2^{12.4})$ for $r = 8$, $(2^{55.8}, 2^{23.8})$ for $r = 12$, $(2^{64}, 2^{35.2})$ for $r = 16$, $(2^{64}, 2^{46.6})$ for $r = 20$, and $(2^{64}, 2^{58})$ for $r = 24$ respectively. By optimized Matsui's algorithm 2 [5], we can recover the last round sub-key much faster than the whole 256-bit key recovery, i.e., in time $32 \times 2^{32} = 2^{37}$ for $r = 8, 12, 16, 20, 24$.

6 Conclusion

This paper studies the linear analysis on block cipher MULTI2. We give formal proofs on bias properties of MULTI2 round functions *for the first time*, which allows to discover new interesting results. Our linear attacks on r-round MUTLI2 recover the 256-bit encryption key in time 2^{46} (for $r = 8$), $2^{60.4}$ (for $r = 12$), $2^{83.8}$ (for $r = 16$), $2^{91.7}$ (for $r = 20$), $2^{123.4}$ (for $r = 24$), $2^{123.2}$ (for $r = 28$) of r-round encryptions respectively. As ISO register recommends to use at least 32 rounds, our attacks remain to be theoretical and do not threaten security for the practical use currently.

Acknowledgments. This paper was inspired by private communications with Serge Vaudenay. We gratefully thank Vincent Rijmen and the anonymous reviewers for many helpful and valuable comments.

References

1. Aoki, K., Kurokawa, K.: A study on linear cryptanalysis of MULTI2. In: The 1995 Symposium on Cryptography and Information Security, SCIS 1995 (1995) (in Japanese)
2. Aumasson, J.-P., Nakahara Jr., J., Sepehrdad, P.: Cryptanalysis of the ISDB Scrambling Algorithm (MULTI2). In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 296–307. Springer, Heidelberg (2009)
3. ARIB. STD B25 v.5.0 (2007), http://www.arib.or.jp
4. Bogdanov, A., Tischhauser, E.: On the wrong key randomization hypothesis in Matsui's algorithm 2 (submitted, 2012), https://lirias.kuleuven.be/handle/123456789/333158
5. Collard, B., Standaert, F.-X., Quisquater, J.-J.: Improving the Time Complexity of Matsui's Linear Cryptanalysis. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 77–88. Springer, Heidelberg (2007)
6. Harpes, C., Massey, J.L.: Partitioning Cryptanalysis. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 13–27. Springer, Heidelberg (1997)

7. Hitachi, Japanese laid-open patent application No. H1-276189 (1998)
8. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
9. Matsui, M.: The First Experimental Cryptanalysis of the Data Encryption Standard. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 1–11. Springer, Heidelberg (1994)
10. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1996)

Appendix

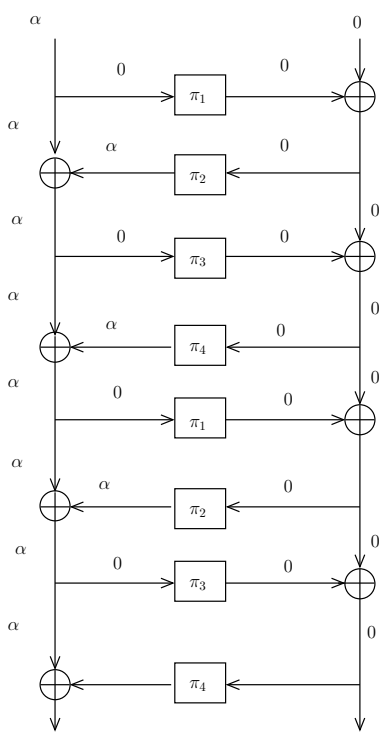


Fig. 3. Linear trail of our 8-round attack to recover k_8 with $\alpha = 0x\mathit{aaaaaaaa}$

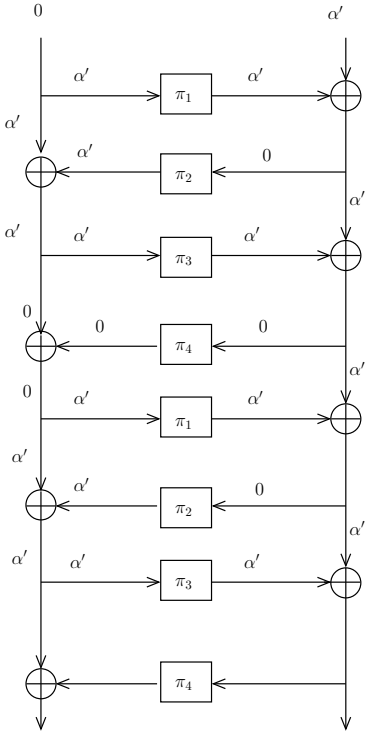


Fig. 4. Linear trail of our 8-round attack to recover k_6, k_7 with $\alpha' = 0x88888888$

Fixed Points of Special Type and Cryptanalysis of Full GOST*

Orhun Kara[1] and Ferhat Karakoç[1,2]

[1] TÜBİTAK BİLGEM UEKAE
National Research Institute of Electronics and Cryptology
Gebze 41470 Kocaeli/Turkey
{orhun.kara,ferhat.karakoc}@tubitak.gov.tr
[2] Istanbul Technical University, Computer Engineering Department, 34469, Maslak, Istanbul/Turkey

Abstract. GOST, the Russian encryption standard, is a block cipher of 64-bit block and 256-bit key size and consists of 32 rounds. In this work, we show that the probability that the GOST permutations produced through random keys have at least one fixed point and exactly two fixed points of special type are twice and five times more than those of random permutations respectively. We utilize this property of GOST to mount a new reflection attack on full GOST.

The reflection property on GOST was defined and exploited to mount an attack on the full cipher by Kara [7] which was successful only for one out of 2^{32} keys. This property has been further studied by Courtois [1], Dinur et al. [5] and Isobe [6]. Isobe mounted an attack that works for any key with a time complexity of 2^{225} [6]. Isobe's attack was improved by Dinur et al. reducing the time complexity to 2^{192} using the whole codebook [5]. They introduce a new version of the meet-in-the-middle technique which they call "2-dimensional meet in the middle (2DMITM)" attack. Their attack is based on applying 2DMITM attack on 8-round GOST 2^{64} times. In this work, we mount an attack with time complexity of 2^{129} using 2^{32} chosen plaintexts instead of the whole codebook utilizing the 2DMITM attack. The main advantages of our attack is that we mount the 2DMITM attack on 8-round GOST only twice. On the other hand, our attack works only for the weak key set of 2^{192} keys, which indicates that the security level of full GOST is equivalent to 129 bits for these keys. In addition, we have computed the success rates of Kara attack in [7] and our attack. We have verified our calculations experimentally.

Keywords: block cipher, self similarity, reflection attack, GOST, fixed point, Feistel network.

1 Introduction

GOST, the Russian encryption standard, is a Feistel network of 64-bit block and 256-bit key size and the number of rounds is 32 [8]. It has a relatively simple key

* We would like to thank Nicolas Courtois for proposing this title to us.

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 86–97, 2012.

schedule. The key is divided into 8 parts and each part is used as a subkey, first 3 times in a direct order and then in a reversed order. This leads to a self-similarity property of the cipher which we exploit to mount a reflection attack.

Several attacks have been published recently on full GOST. The first reflection attack on full GOST exploits extending the special fixed points of the first 8 rounds that occur only for a subset of the key space, to the whole cipher [7]. The extension of these fixed points comes from the extremely simple key schedule of GOST. This attack works for 2^{224} keys with a complexity of 2^{192} encryptions by using 2^{32} chosen plaintexts. The biased distribution of fixed points through the rounds was examined in the attack which was simply called "the reflection property". The reflection property has been further exploited to mount several other attacks on GOST [1,5,6].

Recently, Isobe mounts an attack on the full cipher which works for the whole key space using the reflection property [6]. The complexity and data requirements of the attack are 2^{225} and 2^{32} respectively. Then Dinur et al. proposed an attack on the full cipher with a complexity of 2^{192} and a need of 2^{64} data [5]. In their attack, two input/output pairs for 8-round GOST are guessed using the whole codebook since each guess is correct with a probability of 2^{-64}.These two input/output pairs allow them to eliminate the possible keys to 2^{128} candidates with a cost of 2^{128} GOST encryptions with 2^{36} memory by mounting a meet in the middle type attack which they call "2-dimensional meet in the middle (2DMITM)" attack. Then, they use other pairs to check their guesses. Thus, the total time complexity of their attack is 2^{192} since the 2DMITM attack is performed 2^{64} times. The 2DMITM technique has been further improved and generalized by Zhu and Gong [9].

Courtois has proposed many different attacks on full GOST including some attacks with Misztal [4,3,2,1]. Most of his attacks are collected in [1] where he mounted several reflection and other self-similarity attacks on full GOST, some of them works on weak keys. Courtois and Misztal mounted differential attack on full GOST with a complexity of 2^{226} GOST encryptions using the whole codebook [2]. Then, Courtois himself has further improved the time complexity of a differential attack on GOST to 2^{178} [3].

In this work, we show that the probabilities of having at least one fixed point and exactly two fixed points of a special type (whose left and right parts are equal) for GOST are twice and five times more than the corresponding probabilities of random permutations respectively. Moreover, we mount an attack on full GOST with a complexity of 2^{129} encryptions using two input/output pairs for 8 rounds obtained by exploiting this non-randomness property.

In our improved attack, we make use of only 2^{32} chosen plaintext/ciphertext pairs instead of the whole codebook to construct the two pairs. If there are two fixed points having equal halves then we mount the key recovery attack. One important advantage of our attack is that two input/output pairs of 8-round GOST are known to cryptanalyst with very high probability. Therefore, we perform the 2DMITM attack typically not more than twice. However, the attack works for only 2^{192} keys. As a result, we have shown that GOST provides only 129

bit security for the weak keys. In addition, the 2DMITM attack is not performed if there isn't a pair of suitable fixed points. The complexity of the attack is only to examine the 2^{32} texts in this case. Therefore our attack gives nice results in the following realistic scenario proposed in [1]: Assume there are many distinct keys used and the attacker tries to recover some of them. Then the complexity of our attack to recover one key is $2.5 \cdot 2^{129} + (2^{64} - 2.5)2^{32} \approx 2^{130.3}$. Because, among 2^{64} keys, there is roughly one weak key and the attack is performed about 2.5 times. On the other hand, if a random number generator providing keys to GOST, produces weak keys deliberately then the security margin of GOST declines dramatically.

We have calculated the success rate of the Kara attack on full GOST which is left out in [7] and the success rate of our attack. We prove that the probability of having two fixed points with equal halves is around $5 \cdot 2^{-65}$ and we can recover the key with a probability of 40% if such two fixed points occur. We have validated these calculations by computer simulations.

The paper is organized as follows. We give a short description of GOST and the first reflection attack on it given in [7] in Section 2. The improved attack is stated in Section 3. Then, the success rates of both the first reflection attack and the improved attack are computed in Section 4 and the experimental results are depicted in Section 5. We conclude the paper in the last section with an open question.

2 A Brief Description of GOST and the First Reflection Attack

GOST, the Russian encryption standard [8], is a 32-round Feistel network with 64-bit block and 256-bit key length. It has a simple key schedule: 256 bit key is divided into eight 32 bit words $k_0, ..., k_7$ and the sequence of round keys is given as $k_0, ..., k_7, k_0, ..., k_7, k_0, ..., k_7, k_7, k_6, ..., k_1, k_0$. We do not consider details of the round function. We only assume that it is bijective. Denote the first eight rounds of GOST as $F_K[1, 8]$. Note that $F_K[1, 8]$ ends with a swap operation. Then, the GOST encryption function is given as $E_K(x) = F_K[8, 1] \circ S \circ F_K^3[1, 8](x)$ where S is the swap operation of the Feistel network and $F_K[8, 1]$ is the inverse of $F_K[1, 8]$.

The first attempt to propose a key recovery attack on full GOST was the reflection attack [7]. We briefly describe this attack in this section. Assume there exists (x, x) for $x \in GF(2)^{32}$ such that (x, x) is a fixed point $F_K[1, 8]$. Note that (x, x) is also a fixed point of the swap operation S. Then, (x, x) will be a fixed point of the encryption function E_K. This observation leads to the following attack. Encrypt all 2^{32} plaintexts whose left and right halves are equal and collect the fixed points in a set, say U_E. If U_E is empty, then the attack is not applicable. Otherwise, for any (x, x) in U_E solve the equation $F_K[1, 8](x, x) = (x, x)$ for K. Guessing $k_0, k_1, ..., k_5$, one can construct a two-round Feistel network with unknown keys k_6 and k_7 and an input/output pair given as $(F_K[1, 6](x, x), (x, x))$. Then, solving the system for k_6 and k_7 is straightforward since the round functions

F_{k_6} and F_{k_7} are bijective and their outputs are known. By taking the inverses of F_{k_6} and F_{k_7}, obtain the inputs and then k_6 and k_7. Consequently, obtain 2^{192} candidates for the key by solving $F_K[1,8](x,x) = (x,x)$. Then one can recover the correct key by searching over all the candidates by roughly 2^{192} encryptions. However, it is most likely that U_E is empty if there exists no fixed point of $F_K[1,8]$ with the equal halves. On the other hand, the expected number of fixed points is one and the probability that any arbitrary value is a fixed point of S is 2^{-32}. Hence, the number of keys satisfying that $\exists\ x$ such that $F_K[1,8](x,x) = (x,x)$ is roughly 2^{224}.

3 Improved Attack on Full GOST

We improve the Kara attack on full GOST given in [7] by exploiting the additional fixed points and using the attack idea given by Dinur et al. in [5]. Dinur et al. showed that for a given two input/output pairs for 8 rounds, the key space is diminished to 2^{128} with 2^{128} GOST encryptions using 2^{36} memory and the right key can be recovered by searching these 2^{128} by using some other plaintext/ciphertext pairs for full GOST. The total complexity is 2^{192} since the probability of finding two correct input/output pairs for the 8-round GOST is 2^{-64}. One input/output pair produces just one guess for two pairs of 8-round GOST and hence they use the whole codebook to produce a right pair overall. We give our observation finding the two pairs in Theorem 1 with much less data complexity.

Theorem 1. *Assume that* $\exists(x,x)$ *and* (y,y) *such that* $F_K[1,8](x,x) = (y,y)$ *and* $F_K[1,8](y,y) = (x,x)$ *where* x *and* y *are 32-bit values. Then* $E_K(x,x) = (x,x)$ *and* $E_K(y,y) = (y,y)$.

Proof. Remember that $E_K(x,x) = F_K[8,1] \circ S \circ F_K^3[1,8](x,x)$. Then $E_K(x,x) = F_K[8,1] \circ S \circ F_K^2[1,8](y,y) = F_K[8,1] \circ S \circ F_K[1,8](x,x) = F_K[8,1] \circ S(y,y) = F_K[8,1](y,y) = (x,x)$. The proof for the equality $E_K(y,y) = (y,y)$ is similar.

Let us note that a more generalized property than the property given in Theorem 1 has been studied in [1] to obtain four input/output pairs for 8-round GOST. The idea is based on forming two points of order two where one is the output of the other for 8-round GOST. The probability that a given pair satisfies this property is 2^{-127}. Hence, it is expected to have one such pair among all the pairs produced from the whole codebook. The nice property that these points provide is that the required pair gives two more input/output pairs for 8-round GOST. These pairs are formed by the completion of two points with their corresponding ciphertexts. The main difference which gives us an advantage in reducing the complexity of our attack is that four input/output pairs are not known to the attacker in [1]. She has to try each pair as a candidate. However, in the case given in Theorem 1, the attacker most probably knows which plaintext/ciphertext pairs give two input/output pairs for 8-round GOST. We give these pairs a special name since we use it through the paper.

Definition 1. *Let us call that a given pair* $((x,x),(y,y))$ *satisfies Event-1 if* $F_K[1,8](x,x) = (y,y)$ *and* $F_K[1,8](y,y) = (x,x)$*; and similarly* $((x,x),(y,y))$ *satisfies Event-2 if the points* (x,x) *and* (y,y) *are fixed points of* $F_K[1,8]$.

Let us recall that similar to Theorem 1, we can state that a fixed point (x,x) of F_K is also a fixed point for E_K. This is because (x,x) is also a fixed point of both F_K^{-1} and the swap operation.

The probability of Event-1 for a pair $((x,x),(y,y))$ where $x \neq y$ is $\frac{1}{2^{64}(2^{64}-1)}$. For a fixed key, the probability that there exists one pair satisfying Event-1 is roughly 2^{-65} since there are approximately 2^{63} such pairs. Thus, only about $2^{256-65} = 2^{191}$ keys will produce pairs satisfying Event-1 that means we can find two pairs for 8 rounds for 2^{191} keys. We expect roughly 2^{191} more keys that satisfy Event-2 (none of them do not satisfy Event-1 most probably). So, there are approximately 2^{192} weak keys in total.

The attack works as follows. Get the encryptions of 2^{32} possible (x,x)'s checking fix points. If there are two fixed points then these are most probably caused due to either Event-1 or Event-2. Each event provides two input/output pairs for 8-round GOST. The attacker checks if full GOST has two fixed points of form (x,x) and (y,y), and if so, he applies the 2DMITM attack by Dinur et al. for the resulting input/output pairs for 8 rounds which are $((x,x),(y,y))$ and $((y,y),(x,x))$ with time 2^{128} and 2^{36} of memory. This produces 2^{128} candidates for the key which are then filtered by checking with 2-3 additional plaintext/ciphertext pairs for full GOST. If this is not successful then one needs to run the 2DMITM attack once more this time with the input/output pairs $((x,x),(x,x))$ and $((y,y),(y,y))$ for 8-round GOST. Thus the number of keys which subject to this attack and the time complexity of the attack will be 2^{192} and 2^{129} respectively.

The probability that full GOST has two fixed points of the form (x,x) is approximately $5 \cdot 2^{-65}$ whereas the probability that a random permutation of 64-bit block length has two fixed points of the form (x,x) is approximately 2^{-65}. The detailed calculations are given in Section 4.

4 Success Rates of the Attacks

In this section we give the success rates of both the Kara attack in [7] and our improved attack. By the success rate we mean the ratio of the number of the successful key recoveries among all the key recovery attempts. Hence, it is the probability that F_K has a fixed point of the form (x,x) when full GOST has a fixed point of the form (x,x) for the Kara attack and the probability of having a pair $((x,x),(y,y))$, where $x \neq y$, satisfying Event-1 or Event-2 when (x,x) and (y,y) are fixed points of full GOST for the improved attack.

4.1 Success Rate of the First Reflection Attack on Full GOST

The success rate of the attack on full GOST is left out in [7]. Indeed, if the function $F_K[1,8]$ has a fixed point of the form (x,x) where the former half of

the input is equal to its latter half, it is definitely a fixed point of the encryption function E_K. Nevertheless, the attack makes use of the opposite direction of the statement. That is, the attack is mounted if there is a fixed point of E_K, assuming that it is (probably) a fixed point for also $F_K[1,8]$. In this section, we examine this direction and find the probability that any fixed point of E_K is also a fixed point of $F_K[1,8]$. Let us remark that the attack is not successful for those fixed points of E_K which are not fixed points of $F_K[1,8]$.

Let the set of the fixed points of the form (x,x) of $F_K[1,8]$ be U_F. Recall that U_E is the set of the fixed points of E_K of the form (x,x). Then, the success rate, $\Pr(S)$, of the attack is given as the conditional probability that U_F is nonempty given that U_E is nonempty. That is, $\Pr(S) = \Pr(U_F \neq \emptyset | U_E \neq \emptyset)$. The following statement gives the success rate explicitly.

Theorem 2. *Assume the encryption function E_K behaves randomly when the function F_K has no fixed point of the form (x,x). Then, the probability that U_F is nonempty given that U_E is nonempty is*

$$\Pr(S) = \Pr(U_F \neq \emptyset | U_E \neq \emptyset) = \frac{1}{1+(1-2^{-64})^{2^{32}}} \approx \frac{1}{2-2^{-32}}.$$

Proof. We have $\Pr(S) = \Pr(U_F \neq \emptyset | U_E \neq \emptyset)$ which leads to $\Pr(S) = \frac{\Pr(U_F \neq \emptyset)}{\Pr(U_E \neq \emptyset)}$ since U_F is a subset of U_E. On the other hand $\Pr(U_F \neq \emptyset) = 1-(1-2^{-64})^{2^{32}}$ and $\Pr(U_E \neq \emptyset)$ is given as $\Pr(U_F \neq \emptyset) + \Pr(U_F = \emptyset)\Pr(U_E \neq \emptyset)|U_F = \emptyset)$. Assuming that E_K is a random function when F_K has no fixed point of the form (x,x), we have $\Pr(U_E \neq \emptyset)|U_F = \emptyset) = 1-(1-2^{-64})^{2^{32}}$. Hence, we calculate $\Pr(U_E \neq \emptyset)$ as $1-(1-2^{-64})^{2^{32}} + (1-2^{-64})^{2^{32}}(1-(1-2^{-64})^{2^{32}})$ which yields to the probability

$$\Pr(S) = \Pr(U_F \neq \emptyset | U_E \neq \emptyset) = \frac{1-(1-2^{-64})^{2^{32}}}{(1-(1-2^{-64})^{2^{32}})(1+(1-2^{-64})^{2^{32}})}.$$

For the Kara attack, the key recovery attempt is successful if U_F is nonempty and hence the probability given in Theorem 2 gives the success rate of the key recovery part of the attack. Let us remark that the success rate is very close to one half since the value $(1-2^{-64})^{2^{32}}$ is roughly $1-2^{-32}$. One interesting result is that assuming that the encryption permutation E_K behaves randomly when the function F_K has no fixed point of the form (x,x), then E_K does not behave as a random permutation because the probability that U_E is not empty is twice as large as the probability for a random permutation. Note that the probability that a random permutation has at least one fixed point of the form (x,x) is $1-(1-2^{-64})^{2^{32}}$. The following corollary states this phenomena formally.

Corollary 1.

$$\Pr(U_E \neq \emptyset) = (1-(1-2^{-64})^{2^{32}})(1+(1-2^{-64})^{2^{32}}) \approx 2(1-(1-2^{-64})^{2^{32}}).$$

which is roughly 2^{-31}.

4.2 Success Rate of Improved Attack

Similarly, we compute the success rate of the improved attack given in Section 3. The key recovery part of the attack is performed if there exists (at least) two symmetric fixed points for the encryption function. The attack is mounted by assuming that the fixed points are due to the existence of (x, x) and (y, y) such that $F_K[1, 8](x, x) = (y, y)$ and $F_K[1, 8](y, y) = (x, x)$ (Event-1) or the existence of two fixed points of $F_K[1, 8]$ (Event-2). However, there is a probability that two fixed points may occur by chance also. That is, E_K may have two fixed points of the symmetric form whereas neither Event-1 nor Event-2 happened. In this case the key recovery attack will not be successful.

Recall that we simple call the success rate of the attack as the ratio of the attempts where the keys are recovered over all the attempts. Hence, it is the ratio of the probability of having a pair $((x, x), (y, y))$ which satisfies Event-1 or Event-2 over the probability of having two fixed points of the form (x, x) for the encryption function E_K. Because we attempt to recover the key when there are two symmetric fixed points for full GOST and we get exactly two input/output pairs for 8-round GOST when these fixed points satisfy Event-1 or Event-2. We derive this ratio in the following statements. We also show that the probability of having two symmetric fixed points for E_K is five times more then the probability of having two symmetric fixed points for a random permutation of the same size.

Lemma 1. *The probability that a given pair* $((x, x), (y, y))$*, where* $x \neq y$*, satisfies Event-1 or Event-2 is*

$$\frac{1}{2^{63}(2^{64} - 1)}.$$

Proof. Let us recall that Event-1 is the event that $F_K[1, 8](x, x) = (y, y)$ and $F_K[1, 8](y, y) = (x, x)$. The probability that a given pair $(x, x), (y, y)$ satisfies Event-1 where $x \neq y$ is

$$\frac{1}{2^{64}(2^{64} - 1)}$$

since the probability that $F_K[1, 8](x, x) = (y, y)$ is 2^{-64} and the probability that $F_K[1, 8](y, y) = (x, x)$ provided that $F_K[1, 8](x, x) = (y, y)$ is $(2^{64} - 1)^{-1}$. Similarly, the probability that the pair $(x, x), (y, y)$ satisfies Event-2 is

$$\frac{1}{2^{64}(2^{64} - 1)}.$$

On the other hand the pair $((x, x), (y, y))$ cannot satisfy both Event-1 and Event-2 simultaneously since $x \neq y$. Hence, the probability that $((x, x), (y, y))$ satisfies Event-1 or Event-2 is

$$2 \cdot \frac{1}{2^{64}(2^{64} - 1)} = \frac{1}{2^{63}(2^{64} - 1)}.$$

The following lemma gives the probability that any given two symmetric points are the fixed points for full GOST. This probability is about five times more than the probability for the random case.

Lemma 2. *For a given pair* $((x,x),(y,y))$ *where* $x \neq y$*, the probability that* $E_K(x,x) = (x,x)$ *and* $E_K(y,y) = (y,y)$ *is given as*

$$\frac{2 + 2(1-2^{-64}) + (1-2^{-64})^2}{2^{64}(2^{64}-1)} - \frac{2(1-2^{-64}) + (1-2^{-64})^2}{2^{127}(2^{64}-1)^2}.$$

Proof. The probability that $E_K(x,x) = (x,x)$ and $E_K(y,y) = (y,y)$ can be evaluated when $((x,x),(y,y))$ satisfies Event-1 or Event-2 and when $((x,x),(y,y))$ satisfies neither Event-1 nor Event-2. Hence the probability is given as

$$1 \cdot \frac{1}{2^{63}(2^{64}-1)} + \frac{2(1-2^{-64}) + (1-2^{-64})^2}{2^{64}(2^{64}-1)} \cdot \left(1 - \frac{1}{2^{63}(2^{64}-1)}\right)$$

since both (x,x) and (y,y) are the fixed points of E_K when $((x,x),(y,y))$ satisfies Event-1 or Event-2 and the probability that both (x,x) and (y,y) are the fixed points of E_K is

$$\frac{2(1-2^{-64}) + (1-2^{-64})^2}{2^{64}(2^{64}-1)}$$

otherwise. Let us recall that if neither Event-1 nor Event-2 happened then we have either (x,x) is a fixed point of F_K and (y,y) is not a fixed point of F_K or vice versa ((x,x) is not a fixed point of F_K and (y,y) is a fixed point of F_K) or none of them are fixed points of F_K. The probability that they both are fixed points of E_K in all three cases is

$$\frac{1-2^{-64}}{2^{64}(2^{64}-1)} + \frac{1-2^{-64}}{2^{64}(2^{64}-1)} + \frac{(1-2^{-64})^2}{2^{64}(2^{64}-1)} = \frac{2(1-2^{-64}) + (1-2^{-64})^2}{2^{64}(2^{64}-1)}.$$

The probability of satisfying Event-1 or Event-2 is taken from Lemma 1. Then, if we add all the probabilities we obtain the probability that both the points are the fixed points of E_K as

$$\frac{2 + 2(1-2^{-64}) + (1-2^{-64})^2}{2^{64}(2^{64}-1)} - \frac{2(1-2^{-64}) + (1-2^{-64})^2}{2^{127}(2^{64}-1)^2}.$$

The following theorem is the main statement of this section. It shows the nonrandom selection process of GOST permutations in terms of having two symmetric fixed points and gives the success rate of the improved attack.

Theorem 3. *For a randomly chosen key* K*, the encryption function* E_K *of GOST has the following properties:*

- *The probability that* E_K *has two fixed points of the form* (x,x) *is given as*

$$2^{31}(2^{32}-1) \cdot \left(\frac{2 + 2(1-2^{-64}) + (1-2^{-64})^2}{2^{64}(2^{64}-1)} - \frac{2(1-2^{-64}) + (1-2^{-64})^2}{2^{127}(2^{64}-1)^2}\right) \cdot \left(1 - \frac{2 + 2(1-2^{-64}) + (1-2^{-64})^2}{2^{64}(2^{64}-1)} + \frac{2(1-2^{-64}) + (1-2^{-64})^2}{2^{127}(2^{64}-1)^2}\right)^{2^{31}(2^{32}-1)-1}$$

  which is approximately $5 \cdot 2^{-65}$.

– *Assume there are two fixed points of E_K of the form (x, x) for some K. Then the probability that these fixed points satisfy either Event-1 or Event-2 is*

$$\left(1 + \frac{2^{64}-1}{2^{64}} + \frac{(1-2^{-64})^2}{2} - \frac{1}{2^{127}} - \frac{(1-2^{-64})^2}{2^{128}}\right)^{-1} \approx 0.40.$$

Proof. We prove the statements as we have itemized them.

– The probability that a given pair $((x, x), (y, y))$ forms two fixed points of E_K where $x \neq y$ is given by Lemma 2 as

$$\frac{2 + 2(1-2^{-64}) + (1-2^{-64})^2}{2^{64}(2^{64}-1)} - \frac{2(1-2^{-64}) + (1-2^{-64})^2}{2^{127}(2^{64}-1)^2}.$$

On the other hand, for a fixed key, there are $\binom{2^{32}}{2}\binom{}{1} = 2^{31}(2^{32}-1)$ pairs $((x, x), (y, y))$ that can be produced from the symmetric points. We expect just one pair to satisfy the fixed point condition. Hence the probability is derived.

For the approximation, we have $2^{31}(2^{32}-1) \approx 2^{63}$,

$$\frac{2 + 2(1-2^{-64}) + (1-2^{-64})^2}{2^{64}(2^{64}-1)} - \frac{2(1-2^{-64}) + (1-2^{-64})^2}{2^{127}(2^{64}-1)^2} \approx \frac{5}{2^{128}}$$

and

$$\begin{aligned}(1 - &\frac{2 + 2(1-2^{-64}) + (1-2^{-64})^2}{2^{64}(2^{64}-1)} + \frac{2(1-2^{-64}) + (1-2^{-64})^2}{2^{127}(2^{64}-1)^2})^{2^{31}(2^{32}-1)-1} \\ &\approx (1 - \frac{5}{2^{128}})^{2^{63}} \approx \exp(-5/2^{65}) \approx 1 - \frac{5}{2^{65}} \approx 1\end{aligned}$$

and hence if we combine all these approximations we have the probability approximately $5 \cdot 2^{-65}$.

– The probability that a given pair $((x, x), (y, y))$ satisfies Event-1 or Event-2 given that the pair $((x, x), (y, y))$ forms two fixed points for E_K is given as the ratio of the probability that a given pair $((x, x), (y, y))$ satisfies Event-1 or Event-2 over the probability that the pair forms two fixed points of E_K since if the pair $((x, x), (y, y))$ satisfies Event-1 or Event-2 then both (x, x) and (y, y) are the fixed points of E_K. On the other hand, the ratio is given as the inverse of the ratio

$$2^{63}(2^{64}-1)\left(\frac{2 + 2(1-2^{-64}) + (1-2^{-64})^2}{2^{64}(2^{64}-1)} - \frac{2(1-2^{-64}) + (1-2^{-64})^2}{2^{127}(2^{64}-1)^2}\right)$$

which is equal to

$$1 + \frac{2^{64}-1}{2^{64}} + \frac{(1-2^{-64})^2}{2} - \frac{1}{2^{127}} - \frac{(1-2^{-64})^2}{2^{128}}.$$

Let us remark that this ratio is close to 2.5 since $\frac{2^{64}-1}{2^{64}} \approx 1$, $\frac{(1-2^{-64})^2}{2} \approx 0.5$ and $\frac{1}{2^{127}} + \frac{(1-2^{-64})^2}{2^{128}} \approx 0$. Hence the probability is approximately 0.40.

Let us remark that the probability that two symmetric fixed points of E_K satisfy Event-1 or Event-2 is equivalent to success rate of the key recovery part of the improved attack. Hence the success rate of the improved attack is approximately 40%. Also remark that the probability that E_K has two fixed points of the form (x, x) is roughly $5 \cdot 2^{-65}$ which is 5 times more then that of random permutations. Note that the probability that a random permutation has two fixed points among 2^{32} points of the form (x, x) is approximately 2^{-65} (see Appendix). Hence we derive a distinguisher for the selection process of random permutations through the GOST encryption by random keys.

5 Experimental Results

In this section we show the experimental results for both the Kara attack and the improved attack.

We have computed the success rates and the probabilities of having symmetric fixed points for minimized versions of GOST with block sizes of 16, 20, 24 and 28-bit lengths for the Kara attack and 12, 16, and 20-bit lengths for the improved attack. The number of rounds is fixed to 32 for any block length and we use $n \times 8$-bit key for the n-bit block length. The key is divided into 8 equal parts $k_0, ..., k_7$ and incorporated into the round function as for the original GOST function. The experimental results verify our statements in Theorem 2 and Theorem 3.

Table 1. The expected # of fixed points and weak keys are 3200 and 1600 respectively

Block length	# of keys	# of keys with symmetric fixed points of GOST	# of weak keys	Success rate
16 bits	100×2^{12}	3153	1556	0.493
20 bits	100×2^{14}	3255	1618	0.497
24 bits	100×2^{16}	3193	1598	0.501
28 bits	100×2^{18}	3229	1588	0.492

The expected number of fixed points of the form (x, x) for the encryption is fixed to 3200 and the expected number of weak keys is fixed to 1600 for the first experiment. The experimental results verify our statements for the Kara attack as depicted in Table 1. We have seen that the probability of having at least one symmetric fixed point for full GOST is roughly $2^{1-n/2}$ (the third column divided by the second column) and the probability that it is caused by having fixed point in 8-round GOST is around 50% where n is the block length.

We perform another set of experiments for the improved attack. The results are depicted in Table 2 which go along with the theoretical statements. We scan $2^{21}, 2^{25}$ and 2^{28} keys randomly for 12, 16 and 20-bit block lengths respectively. We count the number of keys which produce two fixed points of the form (x, x) for the encryption function E_K and keys which produce pairs $((x, x), (y, y))$ satisfying Event-1 or Event-2 which give weak keys. We have seen that the

Table 2. The expected number of keys which have two fixed points

Block length	# of keys	# of keys with two symmetric fixed points of GOST	# of weak keys	Success rate
12-bit	2^{21}	1228	526	0.428 %
16-bit	2^{25}	1312	533	0.406 %
20-bit	2^{28}	675	267	0.395 %

probability that GOST has two symmetric fixed points is around $5 \cdot 2^{1-n}$ (the third column divided by the second column) and the success rate is roughly 40%.

6 Conclusion

We show that the probabilities that a randomly chosen GOST permutation has a fixed point and two fixed points of special type are twice and five times more than those of a random permutation respectively. This allowed us to propose a new attack in which the reflection property introduced by Kara occurs twice. Our attack has a time complexity of 2^{129} encryptions and requires only 2^{32} chosen plaintexts in order to break GOST for a subset of the key space of size approximately 2^{192}. Let us remark that 2^{32} data is used to identify if the key is weak. One can note that given 2^{32} of data we can see if the key is weak very efficiently.

The open question is how to efficiently enumerate the set of weak keys. If it is possible to produce weak keys by a polynomial time algorithm then an intentionally weak protocol having a random number generator which provides weak keys to GOST can decrease the security margin of GOST to 129 bits even though the key length is 256 bits. In addition, we have calculated the success probabilities of the attacks given in [7] and our attack. We have validated these calculations by computer simulations.

Acknowledgments. We would like to thank the anonymous reviewers for their invaluable comments. In particular, we greatly appreciate the interest shown by Nicolas Courtois in our work. The quality of the presentation of the work has been improved by his detailed comments.

This work is supported by the project COGSA.

References

1. Courtois, N.T.: Algebraic Complexity Reduction and Cryptanalysis of GOST. IACR Cryptology ePrint Archive, 2011:626 (2011)
2. Courtois, N.T., Misztal, M.: Differential Cryptanalysis of GOST. IACR Cryptology ePrint Archive, 2011:312 (2011)
3. Courtois, N.T.: A Differential Attack on Full GOST. IACR Cryptology ePrint Archive, 2012:138 (2012)

4. Courtois, N.T.: Security Evaluation of GOST 28147-89 in View of International Standardisation. Cryptologia 36(1), 2–13 (2012)
5. Dinur, I., Dunkelman, O., Shamir, A.: Improved Attacks on Full GOST. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 9–28. Springer, Heidelberg (2012)
6. Isobe, T.: A Single-Key Attack on the Full GOST Block Cipher. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 290–305. Springer, Heidelberg (2011)
7. Kara, O.: Reflection Cryptanalysis of Some Ciphers. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 294–307. Springer, Heidelberg (2008)
8. Zabotin, I.A., Glazkov, G.P., Isaeva, V.B.: Cryptographic Protection for Information Processing Systems. Cryptographic Transformation Algorithm. Government Standard of the USSR, GOST 28147-89 (1989)
9. Zhu, B., Gong, G.: Multidimensional Meet-in-the-Middle Attack and Its Applications to GOST, KTANTAN and Hummingbird-2. Cryptology ePrint Archive, Report 2011/619 (2011), http://eprint.iacr.org/

A The Probability of Having Two Symmetric Fixed Points of Random Permutations

In this section we show the probability that a random permutation of 64-bit block length has two fixed points.

Theorem 4. *The probability that a random permutation of 64-bit block length has two fixed points of the form* (x, x) *is given as*

$$2^{31}(2^{32}-1)\cdot\left(\frac{1}{2^{64}(2^{64}-1)}\right)\left(1-\frac{1}{2^{64}(2^{64}-1)}\right)^{2^{31}(2^{32}-1)-1}$$

which is approximately 2^{-65}.

Proof. For a random permutation of 64-bit block length, the probability that given two distinct symmetric points are fixed points is given as

$$\frac{1}{2^{64}(2^{64}-1)}$$

and hence among $2^{31}(2^{32}-1)$ pairs, the probability that just one pair forms fixed points is

$$2^{31}(2^{32}-1)\cdot\left(\frac{1}{2^{64}(2^{64}-1)}\right)\left(1-\frac{1}{2^{64}(2^{64}-1)}\right)^{2^{31}(2^{32}-1)-1}.$$

For the approximation we can deduce similarly that $\frac{1}{2^{64}(2^{64}-1)} \approx 2^{-128}$ and

$$(1-\frac{1}{2^{64}(2^{64}-1)})^{2^{31}(2^{32}-1)-1} \approx \exp(-2^{-65}) \approx 1-\frac{1}{2^{65}} \approx 1$$

and hence the probability will be approximately 2^{-65}.

Attacking Animated CAPTCHAs via Character Extraction

Vu Duc Nguyen[1], Yang-Wai Chow[2], and Willy Susilo[1,⋆]

[1] Centre for Computer and Information Security Research
[2] Advanced Multimedia Research Laboratory
School of Computer Science and Software Engineering, University of Wollongong, Australia
{dvn108,caseyc,wsusilo}@uow.edu.au

Abstract. It is widely accepted that one of the principles in state-of-the-art text-based CAPTCHA design, requires that a robust CAPTCHA scheme be segmentation-resistant. This paper establishes the fact that the segmentation-resistant principle does not only apply to traditional single image CAPTCHAs, but is very much relevant to the design of animated CAPTCHAs. In this paper, we show that animated CAPTCHAs not designed with this principle in mind can be easily be broken using simple techniques to extract individual characters from the animation frames. We present our experimental results on attacking 13 existing animated CAPTCHAs.

Keywords: Animated CAPTCHA, automated attack, character extraction, segmentation resistant.

1 Introduction

Since its inception, CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) have become a vital part of protecting online services against automated abuse. CAPTCHAs are essentially automated challenge-response tests to distinguish between human users and automated computer programs, or bots [15]. Over the years, many diverse CAPTCHA designs have been proposed and implemented on the Internet. At the same time, many techniques have been developed to successfully break various CAPTCHAs.

The success of these CAPTCHA breaking techniques are due in part to a variety of flaws present in the design of various CAPTCHA schemes. It has been suggested that the development of good CAPTCHA schemes is hampered by the fact that the current collective understanding of CAPTCHAs is rather limited [18]. As such, a number of researchers have attempted to identify some of these design flaws in order to provide a better understanding of how to develop more robust CAPTCHAs.

⋆ This work is supported by ARC Future Fellowship FT0991397.

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 98–113, 2012.

One of the well established and widely accepted design principles for text-based CAPTCHAs is the need for a robust CAPTCHA to be segmentation-resistant [2]. This is because text-based CAPTCHAs typically consist of two challenges; a segmentation challenge, followed by a recognition challenge [6]. Segmentation refers to the identification and separation of a sequence of characters into individual characters. Research has shown that computers can outperform humans when it comes to character recognition [7]. In that respect, if a computer program can reduce a CAPTCHA challenge to the problem of recognizing individual characters, it is essentially broken. A number of CAPTCHAs have been shown to be vulnerable to segmentation attacks [16,17]. Furthermore, it has been suggested that relying on segmentation-resistance alone does not provide reliable defense against automated attacks [5].

In light of the fact that many existing CAPTCHAs have already been broken, a number of CAPTCHA developers have resorted to using different CAPTCHA design paradigms. Animated CAPTCHAs is an example of a paradigm, which relies on the mechanism of distributing the information required to solve the CAPTCHA challenge over multiple animation frames. The addition of the time dimension is assumed to provide better security over traditional single image CAPTCHAs. Cui et al. [9] dubbed this the 'zero knowledge per frame principle' because the information required to solve an animated CAPTCHA is not completely contained within a single image.

In this paper, we examine the security of animated text-based CAPTCHAs and show that the segmentation-resistant principle is still one of the design principles that apply to this type of CAPTCHA. In particular, we demonstrate how simple techniques can be used to easily extract characters from across the animation frames in order to break animated CAPTCHAs that are not designed to be segmentation-resistant.

Our Contribution. Animated text-based CAPTCHAs are assumed to provide better security over traditional single image text-based CAPTCHAs because important information can be spread over multiple animation frames, rather than being contained within a single image. We demonstrate how the use of simple techniques to extract characters from animation frames can be performed on a number of existing animated CAPTCHAs. The adopted techniques are a generalization of those used in Nguyen et al. [13] and essentially reduce the CAPTCHA challenge to a single image, which can easily be solved using existing OCR (Optical Character Recognition) programs. Our results show that we can attack these animated CAPTCHAs with a high degree of success. The purpose of our work is to highlight design flaws in existing schemes, which should be avoided in future animated CAPTCHA designs.

2 Related Work

The robustness of CAPTCHA schemes has been the topic of much scrutiny. Over the years, many techniques for breaking CAPTCHAs have been developed to exploit certain design flaws in various CAPTCHA schemes. Using these techniques,

a number of researchers have demonstrated that many existing CAPTCHA schemes are vulnerable to automated attacks.

One of the earliest efforts in breaking text-based CAPTCHAs was put forward by Mori and Malik [11]. In their work, they described the techniques they used to successfully break the EZ-Gimpy CAPTCHA 92% of the time, along with the Gimpy CAPTCHA at a success rate of 33%. The approach that they developed is based on matching the shape contexts of characters using a database of known objects. Since then, other techniques have also been develop to break the Gimpy family of CAPTCHAs. For example, Moy et al. [12] used a distortion estimation technique to break EZ-Gimpy and Gimpy-r.

In the work by Chellapilla et al. [7,8], they demonstrated that machine learning algorithms can successfully be used to break a variety of CAPTCHA schemes. In particular, they showed that computers can perform better than humans at the task of recognizing characters. Based on this, they postulated that since text-based CAPTCHAs mainly consist of a segmentation challenge and a recognition challenge, a secure CAPTCHA scheme must be designed to be segmentation-resistant [6]. In other words, once a computer program can adequately reduce a CAPTCHA challenge to the problem of recognizing individual characters, it is essentially broken.

As such, it is widely accepted that a robust CAPTCHA scheme must be designed to be segmentation-resistant [2]. However, Yan and Ahmad [17] have shown that even carefully designed CAPTCHAs may be vulnerable to segmentation attacks. In their work, they successfully broke a Microsoft CAPTCHA by developing a low-cost attack for segmenting this CAPTCHA scheme.They have also demonstrated that simple pattern recognition algorithms can be used to exploit flaws and design errors in CAPTCHA schemes, thus making them susceptible to simple attacks like counting the number of pixels to identify individual characters [16].

In a systematic study regarding the strengths and weaknesses of text-based CAPTCHAs, Bursztein et al. [5] observed that the segmentation-resistant principle alone is not enough to guarantee that a CAPTCHA scheme is secure against automated attacks. In other recent work, Li et al. [10] demonstrated the use of image processing and pattern recognition algorithms, such as k-means clustering, digital image in-painting, character recognition based on cross-correlation, etc. to successfully break a variety of e-Banking CAPTCHAs. The popular Google reCAPTCHA has also been broken using a holistic approach of recognizing shape contexts of entire words [3].

3 Our Approach

3.1 Extracting Characters

A number of animated CAPTCHAs merely rely on the addition of the time dimension to increase the security of the CAPTCHA scheme. These animated

CAPTCHA schemes are designed to spread the information required to solve the CAPTCHA challenge over multiple animation frames. This is unlike traditional single image CAPTCHAs where all relevant information must to be presented in a single image. As such, the CAPTCHA challenge in many animated CAPTCHAs is obscured in individual frames where the text characters may only be partially visible, joined together, overlapped with extra characters at random, and so on.

To attack this purported security mechanism, our strategy is to extract relevant information from the multiple animation frames into a single image which contains all the characters in the CAPTCHA challenge. Once this information is extracted, the animated CAPTCHA is in effect converted into a single image containing all the individual characters. From here, the standard techniques that are used to break traditional single image CAPTCHAs can easily be applied to this resulting image. Due to the fact that a number of animated CAPTCHAs are not designed to be segmentation-resistant, the relevant characters can be extracted using the two straightforward methods described below.

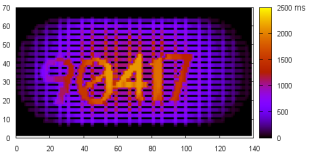
Pixel Delay Map (PDM). In various animated CAPTCHA schemes, the text characters required to solve the CAPTCHA challenge often appear at certain fixed locations for longer periods of time. This does not necessarily mean that the characters themselves are not moving from frame to frame, it simply means that their movement tends to pause or delay at specific locations long enough to get a human user's attention. In that manner, the noise elements and other peripheral components can be distinguished from the main text using what we call a Pixel Delay Map (PDM).

A PDM is an image that has the same dimensions as the animated CAPTCHA's frame dimensions. It is constructed by accumulating the lengths of time where each pixel's color is distinct from the background. The background color is easily determined for animated CAPTCHA schemes that use plain colored backgrounds, because this can automatically be obtained by finding the most frequently used color in each animation frame. In other schemes that use textured backgrounds, for usability purposes the main text in the challenge is most often displayed in a different color or luminance value from the background colors. Using a PDM, we can obtain the total duration in which the pixels are displayed in a color that is distinct from the background. This allows us to identify text characters in animated CAPTCHAs that appear, or pause, at certain locations for longer periods of time.

Figure 1 illustrates an example of character extraction using the PDM approach. Figure 1(a) shows six frames from an animated CAPTCHA[1], which were taken at different times during the animation with time increasing from left to right. This looks like a moving search light that reveals different sections of the challenge at different times. The PDM that was constructed from the animation frames is shown in Figure 1(b). The x and y axes represent the pixel positions, whereas the different colors represent the duration in which each pixel's color

[1] This is an example of an Animierte CAPTCHA.

(a) Several frames from an animated CAPTCHA challenge



(b) PDM of the challenge

(c) Extracted from the PDM

(d) Post-processed

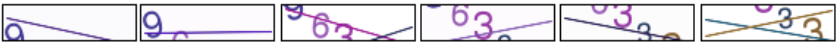
Fig. 1. Example of character extraction using the PDM method

was distinct from the background. This was computed in terms of milliseconds (ms), as depicted in the color scale on the right of Figure 1(b). The extracted binarized image shown in Figure 1(c) was obtained by thresholding the PDM, and the final image after a post-processing process to remove the thin lines is shown in Figure 1(d).

Catching Line (CL). Another observed approach commonly used in animated CAPTCHA schemes features text characters changing within a number of vertical areas over the animation frames. In these cases, the text characters are usually moved or scaled vertically within their respective columns. The reason for restricting text characters to their allocated columns, is so that the user can solve the CAPTCHA challenge by determining the characters in the correct order. Since the text characters usually move or scale vertically, this means that in most cases a 'good' image of a character can be obtained when the character reaches a certain height. This is because the entire character typically becomes legible as it is fully displayed at that height. For this, we define a virtual Catching Line (CL) to 'catch' a character (i.e. get an image of the character) when the character is displayed at a particular height. The CL can either be defined at a fixed position for all the characters in a CAPTCHA challenge or can adaptively be determined for each character based on the maximum height that each character's pixels reaches in the animation frames.

An example of how the CL method can be applied is depicted in Figure 2. Figure 2(a) shows several frames from an animated CAPTCHA challenge[2]. In this animated CAPTCHA, the text characters are constantly moving up from frame to frame. Figure 2(b) in turn shows the same frames after pre-processing

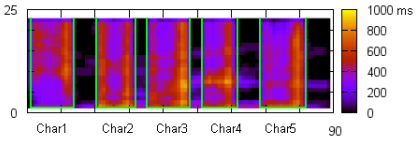
[2] An animated CAPTCHA from the AmourAngels website.



(a) Several frames from an animated CAPTCHA challenge

(b) Frames after noise removal

(c) Virtual catching line and the 'caught' characters (in red)



(d) Moving areas identified using PDM

(e) 'Caught' characters

Fig. 2. Example of character extraction using the CL method

to remove noise. An illustration of the virtual Catching Line (CL) that was used to 'catch' the characters is shown in Figure 2(c). The CL was set to a position $\frac{1}{10}$th from the top of the frame. To facilitate character extraction, the movement areas of each character can be identified using a PDM, as depicted in Figure 2(d). Figure 2(e) shows an image containing the extracted characters.

3.2 Character Recognition

Once the individual characters have been extracted into a single image, character recognition can be performed by passing the image through any good OCR program. In some cases, a pre-processing stage may be required to remove noise or to increase the legibility of characters in the image before passing it through the OCR software. This is discussed in the section to follow.

For our work, we used the ABBYY FineReader 11 Professional Edition [1], which is widely recognized as one of the best currently available OCR programs. The ABBYY FineReader uses a machine learning approach that can be trained from a training set of character samples. To improve the character recognition accuracy for some animated CAPTCHA schemes, we created a training set to be used in conjunction with the ABBYY FineReaders's existing embedded training database. In addition, certain CAPTCHA schemes only used a selection of uppercase letters and/or digits. In such cases, the input language for the OCR could be restricted to this subset.

4 Attacking Animated CAPTCHAs

The previously described PDM and CL methods were used to attack a number of existing animated CAPTCHA schemes. These schemes are listed in Table 1,

Table 1. Animated CAPTCHA schemes

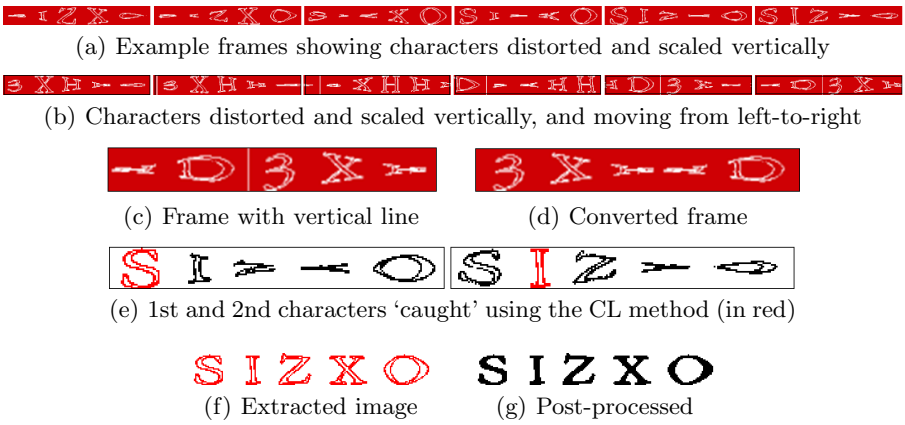
	CAPTCHA/Website Name	URL
1	SiteBlackBox	http://www.siteblackbox.com/captchaService.php
2	Animierte CAPTCHA	http://www.animierte-captcha.de
3	Sandbox	http://sandbox.palmnet.me.uk/gifcaptcha/index.php
4	CharitelBilling	http://charitelbilling.com/login.php
5	iCaptcha	http://www.icaptcha.com
6	Atlantis	http://www.atlantis-caps.com/eng/7-3-contacts.php
7	AmourAngels	http://members.amourangels.com/cgi-bin/login.cgi
8	SnapPages	http://snappages.com/register
9	Bayu	http://bayu.freelancer.web.id/blogfiles/captchaGIF
10	BulletDrive	http://www.bulletdrive.com/register.php
11	CAPTCHANIM	http://www.captchanim.cs.technion.ac.il
12	Dracon CAPTCHA	http://dracon.biz/captcha/
13	KillBot Professional	http://www.notonebit.com/projects/killbot

along with their respective URLs. Some of these are animated CAPTCHAs used on the respective websites, whereas others are commercial CAPTCHA schemes developed by CAPTCHA service providers. Please note that since there are too many different CAPTCHA schemes to adequately describe in this paper, we encourage interested readers to visit the respective websites for further details.

It has been observed that in general, an automated CAPTCHA attacking process may consist of five generic stages: pre-processing, segmentation, post-segmentation, recognition, and post-processing [5]. Of the animated CAPTCHA schemes listed in Table 1, the CAPTCHA schemes on the following websites: SiteBlackBox, Animierte, CharitelBilling, iCaptcha, Bayu and BulletDrive could be broken by directly applying the PDM method, followed by character recognition. The animated CAPTCHA scheme used by the SnapPages website on the other hand, could be broken by directly applying the CL method followed by character recognition. In other cases, some pre-processing and/or post-processing had to be performed before and after character extraction to improve the accuracy of the attack. The specific details for these cases are described below.

CAPTCHANIM. CAPTCHANIM was developed by a researcher at the Technion, Israel Institute of Technology. There are two versions of CAPTCHANIM, namely, the GateKeeper and the DataProtector. The main difference between the two versions is that GateKeeper is constructed using random characters, whereas DataProctector is to be constructed from a user specified string (the purpose of which to protect data against web-crawlers). As both versions are essentially built from the same concept, we only performed our attack on the GateKeeper version.

The GateKeeper itself has two variations. In the first variation, shown in Figure 3(a), characters are distorted and scaled vertically in different animation



(a) Example frames showing characters distorted and scaled vertically

(b) Characters distorted and scaled vertically, and moving from left-to-right

(c) Frame with vertical line

(d) Converted frame

(e) 1st and 2nd characters 'caught' using the CL method (in red)

(f) Extracted image

(g) Post-processed

Fig. 3. Example of character extraction using the CL method to attack CAPTCHAIM

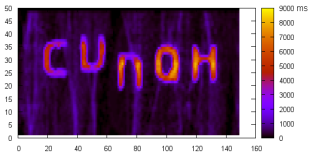
frames. In the second variation, not only are characters distorted and scaled vertically, they also move horizontally in and out of the frames either from left-to-right or from right-to-left. In order for a user to identify the correct character sequence, the start of the sequence is marked with a vertical line. Figure 3(b) shows a number of frames depicting this, where the characters are moving from left-to-right.

The presence of the vertical line marking the start of the character sequence in the second GateKeeper variation, means that it can easily be converted to be similar to the first variation. This can be done by simply swapping the contents on the left side of the vertical line with the contents of the right, for each frame. By doing so, the character sequence will always start on the left and no longer moves in the horizontal direction between animation frames. An example of an original frame is shown in Figure 3(c), whereas Figure 3(d) shows the same frame after swapping left and right sides.

The CL method can be used to attack CAPTCHANIM by extracting characters whenever they reach their highest position. The position of the catching line can automatically be determined for each character based on the maximum height reached by a character's pixels. Figure 3(d) shows example frames in which the first two characters were 'caught'. In addition, a pre-processing step was used before passing the extracted image to the OCR program, in order to fill in the hollow characters in the extracted single image. This was done by performing a boundary-fill in the area within each characters' boundaries. Figure 3(d) shows an example of an extracted image, whereas Figure 3(e) shows the results after the boundary-fill process.

In our experiments, we broke both GateKeeper variations using samples collected from their website. While the developers of CAPTCHANIM allow users to request various customizable features (e.g. number of character, image size, font style, colors, etc.) for their animated CAPTCHA, these are essentially built

(a) Example frames from a Dracon CAPTCHA challenge



(b) PDM of the challenge

(c) Extracted image

Fig. 4. Example of character extraction using the PDM method for a Dracon CAPTCHA

from the same underlying concepts and the same method described here can be used to attack them.

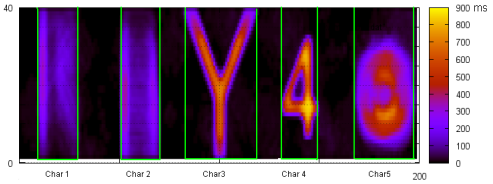
Dracon CAPTCHA. Dracon CAPTCHAs are animated visual Flash CAPTCHAs. The developers state on their website that their CAPTCHA scheme addresses OCR attacks. For our study, we collected samples for Dracon CAPTCHA v2.0 via screen recording and were able to attack these using the PDM method. We observe that other versions of the Dracon CAPTCHA are built using similar concepts and can thus be attacked similarly. As can be seen from the example frames provided in Figure 4(a), Dracon CAPTCHA v2.0 use five characters, which may be letters or numbers, at fixed locations which fade and blur at various times over the animation frames. In addition, the animation frames contain random falling bars in the foreground and small text characters in the background.

While the Dracon CAPTCHAs use noisy backgrounds, the main text characters typically appear in a different color, usually with a brighter luminance value, compared to the rest of the CAPTCHA. As such, the PDM method can be used to identify the text characters that fade in and remain displayed for a period of time, before blurring and fading out. This can be seen in the PDM shown in Figure 4(b). Figure 4(c) in turn, shows an image of the extracted characters obtained by thresholding the PDM.

KillBot Professional. There are a number of versions of the KillBot CAPTCHA. While the static version is free, the KillBot Professional version is a commercial animated CAPTCHA. The developers state on their website that KillBot Professional is used by the United States Federal Government.

From the samples collected from their website, it was observed that the CAPTCHA challenges consist of five characters amidst a noisy foreground and/or background. Of these characters, some may be moving vertically up or down, others may blur, scale and fade in and out from view, yet others may rotate. The noise on the other hand, typically consist of small random characters moving horizontally, and/or colored circles that scale whilst fading in and out (like rain drops).

(a) Example frames from a KillBot Professional CAPTCHA challenge



(b) Vertical segmentation using the PDM

(c) 1st and 2nd characters (in red) extracted

(d) 3rd character extracted

(e) Extracted image

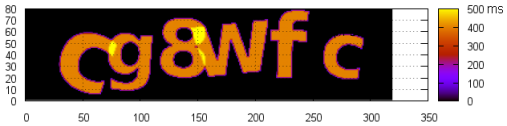
Fig. 5. Example of character extraction for a KillBot Professional CAPTCHA

One of the characteristics of the noise is that it is very often displayed in colors that are lighter than the main text characters.

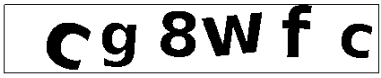
Example frames from a KillBot Professional CAPTCHA are shown in Figure 5(a). A number of different effects can be employed for KillBot Professional. The samples that we collected were for the version that was freely available on their website. Nevertheless, we note that the other versions provide different noise characteristics whilst the challenge characters are presented in a similar fashion. In the example shown in Figure 5(a), all characters blur and fade in and out. The first two characters move in the vertical direction, the third character scales in size, the fourth character appears at a fixed location and the fifth character rotates.

This CAPTCHA can be vertically segmented using the PDM to identify the areas containing the text characters. An example of this is depicted in Figure 5(b). From the PDM, it can be seen that the characteristics of each of the individual characters can be identified. Some characters can be directly extracted from the PDM, for example the fourth character in Figure 5(b). Other characters can be extracted using the CL method. Figure 5(c) shows the first and second characters extracted using the CL method with a fixed line located at a position $\frac{1}{10}$th from the top of the CAPTCHA frame. In Figure 5(d), the third character is extracted using a line determined from the highest position at which the PDM for that character gives the highest value (as can be seen in Figure 5(b)). The fifth character which rotates, blurs and fades in and out is 'caught' using the CL method when the character's pixels reaches its maximum height. Figure 5(e) shows the resulting image containing all the extracted characters.

(a) Example frames from the animated CAPTCHA used on Atlantis-caps.com



(b) PDM with overlapping regions



(c) Extracted image

Fig. 6. Example of character extraction for the animated CAPTCHA used by Atlantis-caps.com using the PDM method

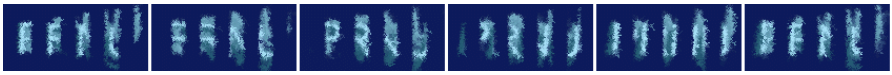
Animated CAPTCHA on the Atlantis website. The animated CAPTCHA used on the Atlantis-caps.com website[3] uses a text-on-text approach. What the user sees is a number of persistent characters over other continuously changing background characters. Frames from an example of this animated CAPTCHA are shown in Figure 6(a).

To extract characters into a single image, the PDM method can directly be used. However, in some cases the characters in the extracted image may not fully be segmented, resulting in lower accuracy when it comes to character recognition if passed directly into an OCR program. Nevertheless, these overlapping regions can easily be identified from the PDM because these regions have much higher PDM values, as can be seen from the example shown in Figure 6(b). Therefore, pre-processing the image to segment the characters can be done prior to character recognition. An example of an extracted image with separate characters is depicted in Figure 6(c).

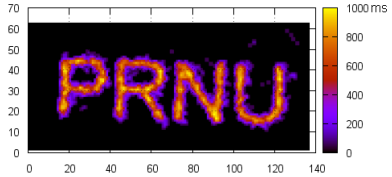
Animated CAPTCHA on the Sandbox website. Similar to the case of the Dracon CAPTCHA, text characters in the animated CAPTCHA available from the Sandbox website have a higher luminance value compared to the noise. As such, the PDM method can be used to attack this CAPTCHA scheme. The character extraction process is portrayed in Figure 7. Figure 7(a) shows several frames from a sample of this CAPTCHA scheme. The PDM for this CAPTCHA is shown in Figure 7(b) and the extracted image is provided in Figure 7(c).

Animated CAPTCHA on the AmourAngels website. As can be seen from the sample frames for this animated CAPTCHA, shown in Figure 8(a), the characters for this CAPTCHA scheme move vertically from frame to frame. All characters move a vertically at a constant speed (i.e. 2 pixels per frame) and some diagonal lines appear at random, presumably to deter segmentation.

[3] This is probably the JkCaptcha, which can be found at:
`http://www.kessels.biz/captcha/`



(a) Example animated CAPTCHA challenge available on the Sandbox website

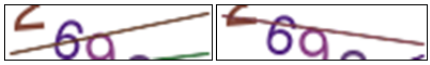


(b) PDM for challenge

(c) Extracted image

Fig. 7. Example of PDM character extraction for the animated CAPTCHA available on the Sandbox website

(a) Sample frames from an animated CAPTCHA used on the AmourAngels website



(b) Example showing 2 adjacent frames

(c) Noise removed

(d) Extracted image

Fig. 8. Example of character extraction using the CL method for the animated CAPTCHA used on the AmourAngels website

These diagonal lines can easily be removed before the character extraction process. Since all characters move from frame to frame at the same speed, whilst noise appears at random, the noise can be removed by simply comparing two adjacent animation frames. An example of two adjacent animation frames is shown in Figure 8(b). In adjacent frames, pixels representing characters simply move vertically by 2 pixel positions. All other pixels are treated as noise and can be removed, as shown in Figure 8(c). Subsequently, characters can be extracted using the CL method. Figure 8(d) shows the resulting image.

5 Results and Discussion

To test the accuracy of our methods, experiments were performed to attack the animated CAPTCHAs listed in Table 1. A total of 2,600 animated CAPTCHA samples were collected from the respective websites (i.e. 200 samples for each of the respective CAPTCHA schemes). A summary of our results is provided in Table 2. The table shows the breakdown of results for each of the different

Table 2. Experimental results

	CAPTCHA/ Website Name	Extracted Image	Extraction Method	Accuracy	Average Time
1	SiteBlackBox		PDM	97.5%	0.9s
2	Animierte CAPTCHA		PDM	94%	0.9s
3	Sandbox		PDM	87.5%	0.6s
4	CharitelBilling		PDM	79%	0.6s
5	iCaptcha		PDM	23.5%	1.8s
6	Atlantis		PDM	19%	2.6s
7	AmourAngels		CL	47%	0.7s
8	SnapPages		CL	99.5%	2.2
9	Bayu		PDM	100%	0.4s
10	Bulletdrive		PDM	100%	0.5s
11	CAPTCHAIM		CL	21%	1.9s
12	Dracon CAPTCHA		PDM	83.5%	2.1s
13	KillBot Professional		PDM and CL	18%	5.8s

schemes. For each scheme, the table shows the specific method that was used to perform character extraction. An example of an extracted image is also provided, along with the accuracy results and the average length of time it took for our automated attack to solve the respective CAPTCHA scheme. All experiments were conducted on an Intel Core 2 Duo 3.33Hz PC.

Based on the observations make by Bursztein et al. [5], a CAPTCHA scheme is deemed broken if it can be automatically solved more than 1% of the time, and they note that it is enough to know if a CAPTCHA scheme can be broken within the first 100 CAPTCHA samples. As such, our results show that the animated CAPTCHA schemes used in our experiments are effectively broken. In addition, the average length of time required to solve the animated CAPTCHAs is well within the length of time that a human user would take to solve a CAPTCHA.

The success of our approach can mainly be attributed to the fact that these animated CAPTCHA schemes are designed based on the assumption that the addition of the time dimension alone sufficiently increases the security of the resulting CAPTCHA. While it is certainly true that most animated CAPTCHAs cannot be solved using a single animation frame, this does not preclude automated attacks from gathering information from multiple animation frames. In fact, in some sense increasing the number animation frames also increases the amount of information that can be exploited to attack animated CAPTCHA schemes that are not well designed.

Since a CAPTCHA scheme must be designed to be human usable, there are a number of common features that many animated CAPTCHAs use to attract a user's focus of attention. This is mainly done to aid human perception in identifying the important information required to successfully solve the CAPTCHA challenge, amidst noise and the other elements that are included to deter automated attacks. While the implementation of these features can improve the usability of the resulting CAPTCHA scheme, several of them can also be exploited to break the CAPTCHA. In particular, some of the features exploited by our approach of using the PDM and CL methods to extract characters include:

- **Characters and positions:** In a number of animated CAPTCHAs, the text characters either remain at fixed locations over multiple animation frames, or only change (e.g. move and/or scale) within their respective columns. While this facilitates usability, as it is easy for a human to determine the correct order of the character string, it also makes the resulting CAPTCHA easier to break. Furthermore, many of the CAPTCHA scheme have a fixed number of characters, which makes it easier to determine the correct number of characters to extract.
- **Time delays:** Another feature commonly adopted is one where the important information appears and remains displayed for a longer duration as compared to the noise elements. This is done so that a human has time to actually read the challenge characters before they disappear, fade out or blur. It is particularly irritating for a user to 'miss' characters and be forced to wait for the animation cycle to repeat itself, which consequently also increases the length of time that the user must spend to solve the CAPTCHA challenge. However, this disproportion in the length of time given to information relevant to the challenge and the noise elements, allows automated attacks to efficiently filter out the noise.
- **Movement direction:** In some CAPTCHA schemes, the main text characters move in a certain direction (or remain stationary) while noise and other impediments, which may be in the form of other text characters, move in different directions. In most cases, to maintain the correct character order, the main characters normally only move vertically. This difference in movement allows *both* humans *and computers* to distinguish between the main characters and the noise elements.
- **Color or brightness:** Several animated CAPTCHA schemes use distinct colors, or luminance values, for the main text characters. While this succeeds in drawing the user's attention to the main characters contained within the CAPTCHA challenge, it also allows computers to easily filter out the other non-important elements. In such cases, it does not even matter whether the noise elements, the foreground or the background are changing from frame to frame, as they are simply removed when the unimportant colors or brightness levels are filtered out.

Not only have our results shown that naively adding a time dimension does not improve the security of a CAPTCHA scheme, it can be seen from the example extracted images provided in Table 2, that the resulting images are even

easier to break than some of the existing traditional single image CAPTCHAs. The reasons for this are two fold. First, the absence of any attempt to include segmentation-resistant mechanisms in the animated CAPTCHAs makes it is easy to extract individual characters using simple techniques like the PDM or CL methods. Standard segmentation-resistance techniques like overlapping or crowding characters together would certainly make the character extraction process more difficult. Second, unlike existing single image CAPTCHAs that must rely on the use of character warping and distortion to deter automated attacks, these animated CAPTCHAs have not really made use of such techniques. As such, in most cases the extracted characters are clearly legible to any good OCR program, without having to improve the quality of the image. Automated attacks on single image CAPTCHAs may require extensive pre-processing to improve character quality before the character recognition process.

Limitations. Other than horizontal or vertical movement, rotation is another type of movement that can be applied to characters in animated CAPTCHAs. While it is straightforward enough to automatically extract characters that are rotating separately, there may be some ambiguity in the resulting extracted image. In particular, depending on the extracted character's orientation, ambiguity arises between certain characters, e.g. 'N' and 'Z', 'b' and 'q', 'd' and 'p', 'n' and 'u', etc. The subtleties between such characters maybe recognizable by a human, but it is more difficult for automated attacks because the extracted image may contain noise or other impediments.

It should also be noted that in its current form neither the PDM method nor the CL method will work in the case of animated CAPTCHA schemes that are designed to be segmentation-resistant like NuCaptcha [14]. This state-of-the-art animated CAPTCHA has a much better design, because characters in NuCaptcha are constantly moving and do not remain within fixed columns. In addition, as the characters are overlapping and crowded together, this would prevent the PDM or CL methods from separating the characters. However, there are more sophisticated attacks that have been used to successfully break NuCaptcha [4].

6 Conclusion

Unlike traditional visual CAPTCHAs where information required to solve the challenge must be presented within a single image, animated CAPTCHAs provide a mechanism for important information to be spread over multiple animation frames. In this paper we demonstrate that naively adding the time dimension alone does not make an animated CAPTCHA scheme more secure. Using automated character extraction methods, we present our results on attacking 13 existing animated CAPTCHAs that were not designed to be segmentation-resistant. In addition, we describe several design flaws that can be exploited in animated CAPTCHAs and discuss various design issues that have to be considered in the design of animated CAPTCHA schemes.

References

1. ABBYY. ABBYY FineReader, http://finereader.abbyy.com/
2. Ahmad, A.S.E., Yan, J., Marshall, L.: The Robustness of a New CAPTCHA. In: Costa, M., Kirda, E. (eds.) EUROSEC, pp. 36–41. ACM (2010)
3. Baecher, P., Büscher, N., Fischlin, M., Milde, B.: Breaking reCAPTCHA: A Holistic Approach via Shape Recognition. In: Camenisch, J., Fischer-Hübner, S., Murayama, Y., Portmann, A., Rieder, C. (eds.) SEC 2011. IFIP AICT, vol. 354, pp. 56–67. Springer, Heidelberg (2011)
4. Bursztein, E.: Elie Bursztein's Blog, http://elie.im/blog/security/how-we-broke-the-nucaptcha-video-scheme-and-what-we-propose-to-fix-it/
5. Bursztein, E., Martin, M., Mitchell, J.C.: Text-based CAPTCHA Strengths and Weaknesses. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM Conference on Computer and Communications Security, pp. 125–138. ACM (2011)
6. Chellapilla, K., Larson, K., Simard, P.Y., Czerwinski, M.: Building Segmentation Based Human-Friendly Human Interaction Proofs (HIPs). In: Baird, H.S., Lopresti, D.P. (eds.) HIP 2005. LNCS, vol. 3517, pp. 1–26. Springer, Heidelberg (2005)
7. Chellapilla, K., Larson, K., Simard, P.Y., Czerwinski, M.: Computers beat Humans at Single Character Recognition in Reading based Human Interaction Proofs (HIPs). In: CEAS (2005)
8. Chellapilla, K., Simard, P.Y.: Using Machine Learning to Break Visual Human Interaction Proofs (HIPs). In: NIPS (2004)
9. Cui, J.-S., Mei, J.-T., Zhang, W.-Z., Wang, X., Zhang, D.: A CAPTCHA Implementation Based on Moving Objects Recognition Problem. In: ICEE, pp. 1277–1280. IEEE (2010)
10. Li, S., Shah, S.A.H., Khan, M.A.U., Khayam, S.A., Sadeghi, A.-R., Schmitz, R.: Breaking e-Banking CAPTCHAs. In: Proceedings of the 26th Annual Computer Security Applications Conference, ACSAC 2010, pp. 171–180. ACM, New York (2010)
11. Mori, G., Malik, J.: Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA. In: CVPR (1), pp. 134–144 (2003)
12. Moy, G., Jones, N., Harkless, C., Potter, R.: Distortion Estimation Techniques in Solving Visual CAPTCHAs. In: CVPR (2), pp. 23–28 (2004)
13. Nguyen, V.D., Chow, Y.-W., Susilo, W.: Breaking an Animated CAPTCHA Scheme. In: Bao, F., Samarati, P., Zhou, J. (eds.) ACNS 2012. LNCS, vol. 7341, pp. 12–29. Springer, Heidelberg (2012)
14. NuCaptcha Inc. NuCaptcha, http://www.nucaptcha.com/
15. von Ahn, L., Blum, M., Hopper, N.J., Langford, J.: CAPTCHA: Using Hard AI Problems for Security. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 294–311. Springer, Heidelberg (2003)
16. Yan, J., Ahmad, A.S.E.: Breaking Visual CAPTCHAs with Naive Pattern Recognition Algorithms. In: ACSAC, pp. 279–291. IEEE Computer Society (2007)
17. Yan, J., Ahmad, A.S.E.: A Low-Cost Attack on a Microsoft CAPTCHA. In: Ning, P., Syverson, P.F., Jha, S. (eds.) ACM Conference on Computer and Communications Security, pp. 543–554. ACM (2008)
18. Yan, J., Ahmad, A.S.E.: CAPTCHA Security: A Case Study. IEEE Security & Privacy 7(4), 22–28 (2009)

Analysis of Rogue Anti-Virus Campaigns Using Hidden Structures in k-Partite Graphs*

Orestis Tsigkas and Dimitrios Tzovaras

Information Technologies Institute
Centre for Research and Technology Hellas
Thessaloniki, Greece
{torestis,tzovaras}@iti.gr
http://www.iti.gr

Abstract. Driven by the potential economic profits, cyber-criminals are on the rise and use the Web to exploit unsuspecting users. Indeed, a real underground black market with thousands of collaborating organizations and individuals has developed, which brings together malicious users who trade exploits, malware, virtual assets, stolen credentials, and more. Among the various malicious activities of cyber-criminals, rogue security software campaigns have evolved into one of the most lucrative criminal operations on the Internet. In this paper, we present a novel method to analyze rogue security software campaigns, by studying a number of different features that are related to their operation. Contrary to existing data mining techniques for multivariate data, which are mostly based on the definition of appropriate proximity measures on a per-feature basis and data fusion techniques to combine per-feature mining results, we take advantage of the structural properties of the k-partite graph formed by considering the natural interconnections between objects of different types. We show that the proposed method is straightforward, fast and scalable. The results of the analysis of rogue security software campaigns are further assessed by a visual analysis tool and their accuracy is documented.

Keywords: unsupervised learning, security, k-partite graphs.

1 Introduction

Over the last decade, there has been a significant shift in the nature of cybercrime, from server-side to client-side attacks and from mainly destructive (e.g. fast spreading worms) to omnivorously profit-oriented activities like identity theft, fraud, spam, phishing, online gambling, extortion [1]. It is now evident that cybercriminals become increasingly collaborative and organized, changing

* This work has been partially supported by the European Commission through project FP7-ICT-257495-VIS-SENSE funded by the 7th framework program. The opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the European Commission.

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 114–125, 2012.

the ways that cybercrimes are committed. Individuals with different skill-sets join in ephemeral relationships to commit a common act and to reproduce their skills and knowledge. All the facts and figures presented in public threat reports are certainly valuable and help to shed some light on those cyber-criminal phenomena, but a lot of unknowns remain.

Among the various malicious activities of cyber-criminals, the spreading of fake antivirus (AV) programs stands out. Fake AV software has been utilized to defraud millions of computer users into paying as for services that the never actually receive. Rogue security software is actually the most common form of scam software, also called *scareware*, which makes use of social engineering to exploit a computer user's fear of revealing sensitive information, losing important data, and/or causing irreversible hardware damage. Therefore, a fake AV program impersonates an antivirus scanner and displays misleading or fraudulent alerts in an attempt to dupe a victim into purchasing a license for a commercial version that is capable of removing non-existent security threats. However, users not only do they never receive what they have paid for, but, to make things worse, their machines get compromised by the installed software, offering new attack opportunities to cyber-attackers. As a result, fake AV software has evolved into one of the most lucrative criminal operations on the Internet.

Moreover, as cyber crime is becoming more organized, new crime mechanisms utilise all available means to automate their malicious activities. This leads to patterns or fingerprints in relevant datasets that are valuable if identified. Such identification within a large set of heterogeneous data is a very difficult and time-consuming task, particularly across layers (network transport, service, transaction). Furthermore, Internet criminals have become adept at modifying their strategies and tactics as new methods are developed to combat their activities. As such, the tools used to identify and characterise their activities must be able to cope with fast-changing requirements. In order to be successful, the techniques used to commit crimes need to be as automated as possible and, of course, stealthy. This automation, by definition, leaves fingerprints that, if found, offer valuable information for the implementation of new detection strategies or for forensic purposes. The problem is that these fingerprints are, a priori, unknown and hidden in a massive amount of data. However, current analysis techniques do not allow us to automatically discover new relevant knowledge about attack phenomena, certainly not from a strategic viewpoint.

Consequently, many open issues remain. Who is behind the deployment of rogue AV websites, how many organized communities are responsible for them, where do they originate, what are the emerging strategies used in cybercrime and how do they evolve over time? Are cyber-criminals able to coordinate their actions? All previously described issues are related to a common security problem often referred to as "attack attribution" [1]. In this paper, we present an unsupervised method for root cause analysis of rogue AV campaigns, by studying a number of different features that are related to their operation and by ascribing large-scale attack phenomena to the same group of individuals or communities.

Contrary to existing data mining techniques for multivariate data, which are mostly based on the definition of appropriate proximity measures on a per-feature basis and data fusion techniques to combine per-feature mining results, we take advantage of the structural properties of the k-partite graph formed by considering the natural interconnections between objects of different types.

The rest of the paper is structured as follows. In Section 2, we provide an overview of the background work in analysis of security software campaigns. Section 3 presents the developed method for unsupervised learning on k-partite graphs, while Section 4 provides an overview of the analysis results. Finally, Section 5 concludes the paper.

2 Background Work

The spreading of rogue security software has been observed as early as in 2005 [2]. A thorough description of various instances of rogue software is presented in security industry reports [3] [4]. These studies aim to shed light on the strategies of cyber-criminals, the prevalence of rogue AV software and its distribution mechanisms. In [5], [6] and [7], the authors provide a study of malicious websites and their underground economy. Last, in the their seminal work in [8] and [9], Cova et al. present a methodology for ascribing rogue security software websites to the same campaign. The proposed methodology requires the definition of proximity measures and clustering on a per-feature basis. Then, the per-feature clustering results are combined using a data fusion technique based on mutli-criteria decision analysis. While the presented technique yields meaningful results, its accuracy largely depends on the security analyst who has to parametrize it at various steps.

3 Clustering Analysis Using k-Partite Graphs

The proposed clustering algorithm aims at identifying the structural properties of graphs by applying dynamic methods based on the class of flow-based graph clustering algorithms represented by Markov Clustering (MCL) [10]. MCL offers several advantages in that it is an elegant approach based on the natural phenomenon of flow, or transition probability, in graphs [11]. Contrary to clustering techniques that are based on the selection of appropriate distance or dissimilarity metrics and on fusion of per-feature results, flow-based graph clustering algorithms take advantage of the similarities of data instances as these are reflected on the structural properties of the graph (e.g. common connections). Therefore, we extend flow-based graph clustering algorithms to search for natural groups of rogue websites (common campaign) in k-partite graphs, and we present a number of enhancements to improve their performance, as well as to enhance the meaningfulness of results. Their operation relies on an iterative process which applies three operators - *expansion*, *inflation* and *pruning* - on an initial transition matrix $\mathbf{P}$, in alternation, until convergence.

The merit of the proposed method compared to existing distance-based clustering methods is threefold. First, defining appropriate distance measures is a not straightforward procedure and different distance measures result in different clustering results. On the contrary, our proposed method searches for similarities rather than dissimilarities between data objects, by considering their interconnections with respect to different features. Second, in many cases defining an appropriate distance metric may not be feasible. For example, one cannot define an appropriate distance metric for two blocks of IP addresses that belong to the same ISP but are located far apart in the IP space. The proposed method does not take into account the actual IP address, but the interconnections of rogue websites with different IP addresses and, therefore, it can capture the case of different IP blocks belonging to the same ISP. Last, our method can work with categorical, numerical, ordinal and binary data without requiring complex data transformations which usually depend of the security analyst's knowledge and expertise.

The security problem of rogue AV websites analysis involves data objects of multiple types that are related to each other, which can be naturally formulated as a k-partite graph. For example, rogue websites are related to malware types, geolocation of the websites, IP address of the website and the nameserver, etc. However, the research on mining the hidden structures from a k-partite graph is still limited and preliminary. Therefore, our research work aims at proposing a principal framework for unsupervised learning on k-partite graphs of various structures. Under this model, we derive a novel algorithm to identify the hidden structures of the graph by identifying strongly connected nodes, using neighbourhood information. The strength of our approach resides in its ability to incorporate multiple features, searching for clusters in the multidimensional space.

3.1 Problem Definition

A k-partite graph is a graph where nodes can be divided in k disjoint groups $(V_0, \ldots, V_{k-1})$, such that no edge connects the vertices in the same group. More formally, a k-partite graph G is defined as $G = \langle V_0 \cup \ldots \cup V_{k-1}, E \rangle$, where $V_l = \{n_i | 1 \leq i \leq N_l\}$, $\forall l \in [0, k-1]$, and $E \subset \bigcup_{l=1}^{k-1} \{V_0 \times V_l\}$ as shown in Fig. 1.

We assume an edge-weighted directed k-partite graph. Moreover, nodes in V_0 (white circles in Fi. 1) correspond to rogue security software websites, while nodes in $V_l, l \in [1, k-1]$ (coloured circles) correspond to feature values of a specific feature. Nodes in $V_{l \neq 0}$ can have connections only to nodes in V_0.

Given a query node n_i in V_l, $l \in [0, k-1]$, our clustering algorithm computes an *attractor* node. All nodes that are attached to the same *attractor* node belong to a single cluster. Based on the graph structure, the *attractor* node can be either a rogue website or a feature value of any given feature. Nodes that belong to the same group V_l have the same type; it is the connections between the k types of

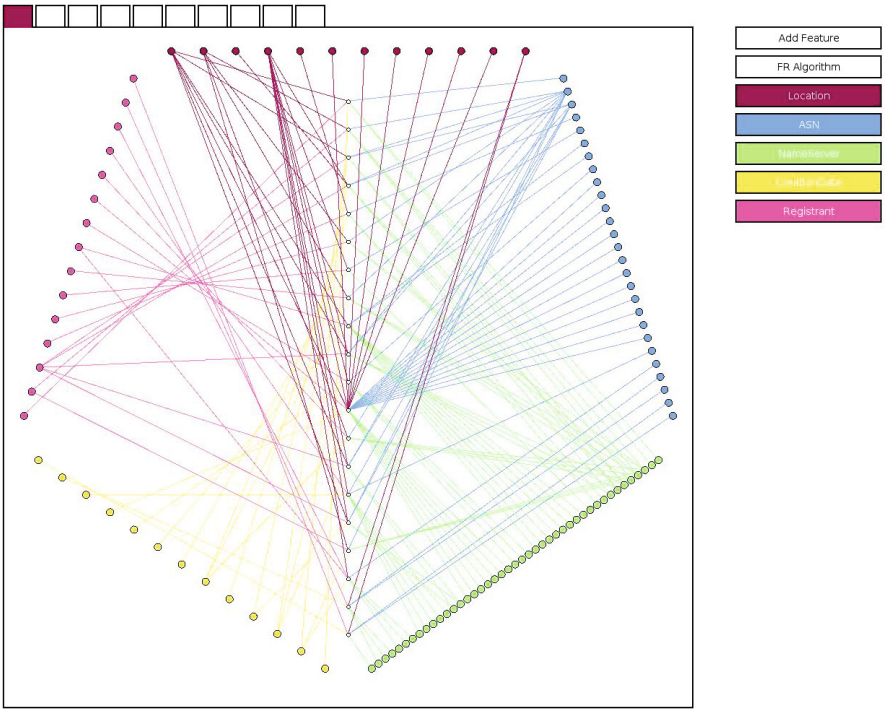


Fig. 1. A 6-partite graph. White circle nodes in the middle represent rogue security software websites. The nodes corresponding to the feature values of 5 different features are placed on the sides of a pentagon using different colouring schemes. Feature values can be connected only to rogue websites and not with each other.

objects that hold the key to mining the hidden structures in the k-partite graph. Given the natural inter-group connections (between V_l and V_m), our objective is to discover the intra-group relationships, such as the clusters within the group. An effective mining algorithm should thus be able to utilize these links across the $(k-1)$ natural groups that are formed by considering the connections between rogue websites and each of the $(k-1)$ features.

3.2 Building the *k*-Partite Graph

The subgraph G_l formed by considering nodes only in V_0 and V_l, $l \in [1, k-1]$, can be conceptually stored in a N_0-by-N_l matrix $\mathbf{M_l}$, where $M_l(i,j)$ is the weight of the edge $\langle i,j \rangle$. The nodes in $V_0(V_l)$ are called row (column) nodes. Note that a column node links to a row node if the corresponding matrix element is not zero. Moreover, row node n_i connects to another row node n_j if there is a column node c linking to both n_i and n_j. We call that path a connection between n_i and n_j through c. Nodes n_i and n_j can have multiple connections via different column nodes.

For each subgraph G_l, $l \in [1, k-1]$, we can construct the adjacency matrix $\mathbf{A_l}$ of G_l using $\mathbf{M_l}$:

$$\mathbf{A_l} = \begin{pmatrix} \mathbf{0} & \mathbf{M_l} \\ \mathbf{M_l^T} & \mathbf{0} \end{pmatrix}$$

In particular, $A_l(i,j)$ denotes the element at i-th row and j-th column in $\mathbf{A_l}$. Suppose we want to traverse the subgraph starting from the row node n_i. Then, we have to transform matrix $\mathbf{A_l}$ into a transition matrix $\mathbf{P_l}$, such that the sum of the probabilities of taking an edge $\langle i,j \rangle$, starting from the row node n_i, does not exceed 1. Therefore, the most common approach is that, for each row node n_i, the normalization of the weight of any edge $\langle i,j \rangle$ is proportional to the edge weight over all the outgoing edges from n_i. More formally:

$$P_l(i,j) = \frac{A_l(i,j)}{\sum_{m=1}^{N_l} A_l(i,m)}$$

and

$$\mathbf{P_l} = \begin{pmatrix} \mathbf{0} & \mathbf{M'_l} \\ \mathbf{M'^T_l} & \mathbf{0} \end{pmatrix}$$

Then, by considering the transition matrices $\mathbf{P_l}$ corresponding to each subgraph G_l, we can construct the transition matrix $\mathbf{P}$ of G as follows :

$$\mathbf{P} = \begin{pmatrix} \mathbf{0} & \mathbf{M'_1} & \mathbf{M'_2} & \dots & \mathbf{M'_{k-1}} \\ \mathbf{M'^T_1} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{M'^T_2} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{M'^T_{k-1}} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \end{pmatrix}$$

3.3 Clustering Formation

To find the hidden clusters in a graph we make use of Markov Clustering (MCL) algorithm which is based on (stochastic) flow simulation. This algorithm shares the ideas behind random walks. However, unlike random walks which compute a relevance score from a given node in a group to any other node in the group, MCL aims at calculating an "*attractor*" node, by which all nodes belonging to the same cluster will be attracted.

The MCL algorithm is an iterative process of applying three operators - *expansion*, *inflation* and *pruning* - on an initial transition matrix P, in alternation, until convergence. Each of these steps is defined below:

The *expansion* step requires that matrix $\mathbf{C_{N \times N}}$, which will finally hold the *attractor* nodes for each node n_i, is multiplied with the transition matrix $\mathbf{P}$:

$$\mathbf{C} = \mathbf{P} \cdot \mathbf{C} \tag{1}$$

The ith row of matrix $\mathbf{C}$ can be interpreted as the final distribution of a random walk of length 1 starting from node n_i, with the transition probabilities of the random walk given by $\mathbf{P}$.

The *inflation* step requires raising each entry in the matrix $\mathbf{P}$ to the power r and then normalizing the rows to sum to 1.

$$C(i,j) = \frac{C(i,j)^r}{\sum_{m=1}^{N} C(i,m)^r} \tag{2}$$

The inflation step has the effect of strengthening intra-cluster flow and weakening inter-cluster flow, by reducing the probability of visiting nodes that do not belong to the same cluster. This is due to the fact that there are more paths between two nodes that are in the same cluster than between those in different clusters and, therefore, there is a higher probability of visiting the inter-cluster nodes.

Last, the *prune* step removes the entries below a threshold q:

$$C(i,j) = \begin{cases} 0 & \text{, if } C(i,j) \leq q \cdot \max\limits_{j=1}^{n}\{C(i,j)\} \\ C(i,j) & \text{, otherwise} \end{cases} \tag{3}$$

Then, the retained entries are rescaled to have the row sum to 1. This step is primarily meant to reduce the number of non-zero entries in the matrix and hence save memory.

4 Experimental Results

The set of studied rogue AV domains is built by aggregating information from a number of different sources [8]. The considered dataset consists of 5,852 DNS entries, collected in July and August 2009, pointing to 3,581 distinct IP addresses hosting rogue AV servers. It is worth noting that at least 45% of these domains were registered through just 29 out of several hundred existing domain registrars.

To study the dynamics of rogue domains and their relation with the associated web servers, we make use of the data collected by HARMUR (HARMUR, a Historical ARchive of Malicious URLs), which enables us to study the relation between client side threats and the underlying server infrastructure, and their evolution over time [12]. The HARMUR dataset was developed by Symantec in the context of the WOMBAT EU-FP7 project [13] and extended in the framework of the VIS-SENSE EU-FP7 project [14] where Symantec is also involved in as a key partner. Instead of developing new detection technologies (e.g., based on honeyclients, or special web crawlers), HARMUR integrates multiple information sources and takes advantage of various data feeds that are dedicated to detecting Web threats. By doing so, HARMUR aims at enabling the creation of a "big picture" of the client-side threat landscape and its evolution.

4.1 Feature Selection

HARMUR collects a number of features associated with each rogue AV domain described in the following list:

- *Geolocation* (F_{Geo}). The country in which the web server of the domain is located.
- *ASN* (F_{ASN}). The number of the Autonomous System associated with the web server IP address.
- *Registrant email address* (F_{Regn}). The email address provided upon registration of the domain.
- *Registrar* (F_{Regr}). The organization which registered the domain.
- *Creation date* (F_{CD}). The date that the domain was registered.
- *Web Server IP address* (F_{IP}). The IP address associated with the domain.
- *Class C* (F_{IP_C}) *and Class B* (F_{IP_B}) *subnets of Web Server IP addresses.* To allow the identification of servers belonging to the same infrastructure, the /24 and /16 network prefix of each IP address is extracted.
- *Web Server version* (F_{Ver}). The version of the web server of the domain.
- *Nameserver IP address* (F_{NS}). The IP address of the authoritative nameserver(s).
- *Registered domain name* (F_{Dom}). The domain name can reveal common naming schemes.

Among the different information tracked through HARMUR, we select a number of features that we believe to be likely to reveal the organized operation of one specific individual or group. Therefore, we define the following feature set: $\mathcal{F} = \{F_{Regn}, F_{NS}, F_{IP}, F_{IP_C}, F_{IP_B}\}$, which will be used by the proposed method to link rogue domains to the same campaign.

Moreover, the set $\mathcal{F}' = \{F_{Geo}, F_{ASN}, F_{Regr}, F_{CD}, F_{Ver}, F_{Dom}\}$ of the remaining features is used to validate the accuracy of our results. Indeed, rogue domains that are grouped in a single cluster should exhibit high homogeneity in terms of their location, the associated AS number, the registrar of the domain and the version of the webserver they are running on. Moreover, rogue domains that are linked to the same campaign are probably registered on the same dates and the domain names should follow similar patterns. This is due to the fact that, cyber-criminals registering a high number of rogue domains try to automate their methods in order to save time and increase their revenue.

4.2 Cluster Analysis

The 12-partite graph that is constructed by considering each of the 11 features is shown in Fig. 2. The graph is positioned using a force-directed algorithm. Force-directed algorithms aim at positioning the vertices of a graph in such a way that preserves the structure of the high-dimensional data as possible in the 2-dimensional space. Therefore, two nearby vertices on the 2-dimensional space have highly similar feature vectors, whereas two distant points should have nothing in common. This allows us to visualize the high-dimensional data set, but also to assess the consistency of the obtained clustering results.

However, the visual clusters formed by the force-directed algorithm should not be considered as indicative of the actual clusters in the dataset for a number

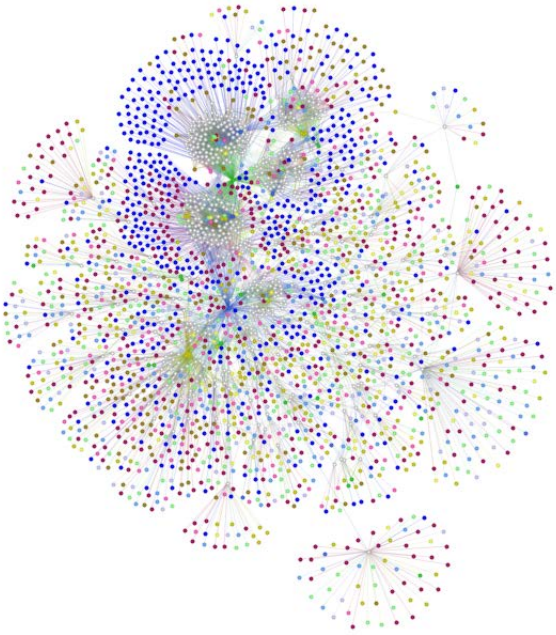


Fig. 2. The k-partite graph comprising of nodes corresponding to both websites and feature values positioned with a force-directed layout algorithm

of reasons. First, the force-directed algorithm takes into account the connections to all features and not only to the features in set $\mathcal{F}$. Second, the force-directed algorithm does not take into account the weight assigned to each feature. Last, force-directed algorithms are known to converge to local minima, which results in sub-optimal positioning of the vertices in a graph.

The weights assigned to each feature in set $\mathcal{F}$ is given by vector $\mathbf{w}$:

$$\mathbf{w} = [0.35, 0.2, 0.2, 0.15, 0.10]$$

In our discriminant analysis, we assign a higher weight to features F_{Regn}, F_{NS} and F_{IP}, since these specific features will yield a high probability that correlated rogue sites are likely due to the same campaign. On the other hand, by assigning lower weight to features F_{IP_C}, F_{IP_B} we give them a little less confidence, since these features are redundant with feature F_{IP}. The inflation parameter r and the cutting threshold q were set equal to 1.15 and 0.01 respectively.

Fig. 3 shows the results of the clustering analysis, where a different colour is used to represent a single cluster. The comparison of clusters corresponding to the results of the force-directed algorithm (position in the 2D space) with clusters corresponding to the results of our cluster analysis (colouring scheme) validate the accuracy of our method. The colour mapping allows us to have a clear overview of the coherency and high homogeneity of each cluster. Moreover,

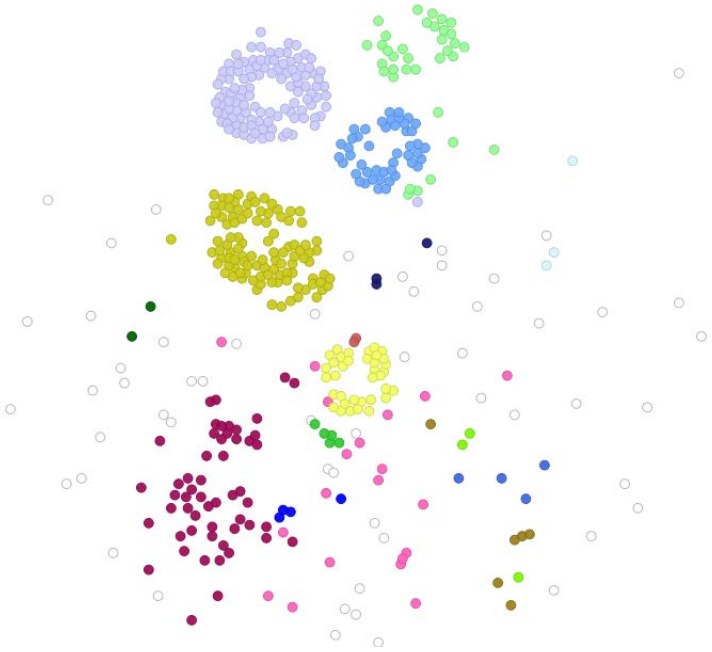


Fig. 3. Results of the clustering analysis. Only nodes corresponding to websites are depicted.

to gain insight into the root causes of each rogue AV campaign, we have to look at the contribution of each separate feature in the formation of clusters.

Indeed, from our clustering analysis, it is evident that, as far as the features in set $\mathcal{F}$ are concerned, one or a few clusters of the separate features contribute to the formation of a single cluster in the big graph. This is not always the case with features in set $\mathcal{F}'$, where a single cluster of the separate features is related to multiple clusters in the big graph. For example, rogue websites located in the USA belong to many different clusters, meaning that many different rogue campaigns are hosted in the USA. By paying special attention to the contribution of each separate feature in the formation of clusters, our clustering analysis allows us to make an interesting observation. For a specific campaign, although the rogue websites address Chinese people, as it is made obvious by the ".cn" extension of their domains, the websites themselves are hosted either in the USA or in Germany.

5 Conclusion

In this paper, we presented an unsupervised method for learning on k-partite graphs for the analysis of rogue AV campaigns. The proposed method takes

advantage of the structural properties of the k-partite graph formed by considering the natural interconnections between objects of different types. We showed that the proposed method is straightforward, fast and scalable. The results of the analysis of rogue security software campaigns were further assessed by a visual analysis tool where their validity was documented.

Acknowledgments. The authors wish to thank Symantec's senior researchers Dr. Marc Dacier, Dr. Olivier Thonnard and Dr. Corrado Leita for providing us with the HARMUR dataset and for their helpful comments that shed light into various aspects of the dynamics of rogue anti-virus campaigns.

References

1. Thonnard, O.: A multi-criteria clustering approach to support attack attribution in cyberspace. PhD thesis, École Doctorale d'Informatique, Télécommunications et Électronique de Paris (March 2010)
2. Wang, Y.M., Beck, D., Jiang, X., Roussev, R.: Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In: NDSS (2006)
3. Fossi, M., Turner, D., Johnson, E., Mack, T., Adams, T., Blackbird, J., Low, M.K., McKinney, D., Dacier, M., Keromytis, A., Leita, C., Cova, M., Overton, J., Thonnard, O.: Symantec report on rogue security software. Technical report, Symantec (October 2009)
4. Rajab, M.A., Ballard, L., Mavrommatis, P., Provos, N., Zhao, X.: The Nocebo Effect on the Web: An Analysis of Fake Anti-Virus Distribution. In: Workshop on Large-Scale Exploits and Emergent Threats (April 2010)
5. Zhuge, J., Holz, T., Song, C., Guo, J., Han, X., Zou, W.: Studying Malicious Websites and the Underground Economy on the Chinese Web. In: 2008 Workshop on the Economics of Information Security, WEIS 2008 (2008)
6. Franklin, J., Paxson, V.: An inquiry into the nature and causes of the wealth of internet miscreants. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS 2007, pp. 375–388. ACM, New York (2007)
7. Stone-Gross, B., Abman, R., Kemmerer, R., Kruegel, C., Steigerwald, D., Vigna, G.: The Underground Economy of Fake Antivirus Software. In: Proceedings of the Workshop on Economics of Information Security, WEIS (2011)
8. Cova, M., Leita, C., Thonnard, O., Keromytis, A.D., Dacier, M.: An Analysis of Rogue AV Campaigns. In: Jha, S., Sommer, R., Kreibich, C. (eds.) RAID 2010. LNCS, vol. 6307, pp. 442–463. Springer, Heidelberg (2010)
9. Cova, M., Leita, C., Thonnard, O., Keromytis, A., Dacier, M.: Gone Rogue: An Analysis of Rogue Security Software Campaigns. In: Proceedings of the 2009 European Conference on Computer Network Defense, EC2ND 2009, pp. 1–3. IEEE Computer Society (2009)
10. Dongen, S.V.: Graph Clustering by Flow Simulation. PhD thesis, University of Utrecht (2000)
11. Satuluri, V., Parthasarathy, S.: Scalable graph clustering using stochastic flows: applications to community discovery. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2009, pp. 737–746. ACM, New York (2009)

12. Leita, C., Cova, M.: HARMUR: storing and analyzing historic data on malicious domains. In: Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS 2011, pp. 46–53. ACM, New York (2011)
13. The WOMBAT Project, http://www.wombat-project.eu
14. The VIS-SENSE Project, http://www.vis-sense.eu/

Mobile Evil Twin Malnets – The Worst of Both Worlds

Christian Szongott, Benjamin Henne, and Matthew Smith

Distributed Computing & Security Group
Gottfried Wilhelm Leibniz University of Hannover
{szongott,henne,smith}@dcsec.uni-hannover.de
http://www.dcsec.uni-hannover.de

Abstract. The mobile computing world is undergoing major changes both in the capability as well as in the proliferation of mobile devices. While, up to now, mobile malware has played a relatively small role compared to the behemoth of desktop malware, the changing environment is steadily increasing the attractiveness of mobile devices as exploitable resources. The increased usage and connectivity of mobile devices opens up a much larger set of attack vectors to compromise them. In this paper, we adapt the evil twin rogue access point attack to the mobile domain and show how it can be used to create a mobile malnet, which is capable of spreading epidemically. We implemented the key components of the concept for the iPhone to study its properties in a laboratory environment. To demonstrate the dangers which come along with this kind of attack we simulate a metropolitan area and show how fast a malware can spread in a mobile environment.

1 Introduction

Researchers have predicted the epidemic spread of mobile malware many times in the last decade. However, a true outbreak of mobile malware has not occurred in the wild yet, leading to a certain level of complacency towards the threat of mobile malware.

That mobile devices have not been targeted in the past is partly due to the fact that the proliferation and capabilities of mobile devices were so limited that they were consequently also only of limited use to attackers. Compared to the effectiveness of Wi-Fi desktop worms with epidemic qualities such as the wildfire worm presented by Akritidis et al. [1] the capabilities of worms for mobile phones have been relatively small.

However, several key factors in mobile computing have changed in recent years, significantly altering the playing field and giving rise to a new threat of mobile malware [2]. The proliferation and capabilities of mobile, networked devices has increased markedly and the mobile phone has become the central digital hub of our lives, storing personal information, multimedia content as well as offering access to social networks, online banking, work networks and a host of

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 126–141, 2012.

other functionality. This increases both the value as a target as well as the attack surface. Recent work has also shown the potential dangers of mobile malnets [3]. Malnets are botnets created from routers, cellphones, and other non-traditional computational Wi-Fi devices. The main focus of malnet research has been in the area of router attacks, such as presented by McDaniel [4].

In this paper we show how recent advances in mobile devices can be used to combine the most dangerous aspects of these two worlds - mobile malnets and Wi-Fi worms. We will show how the inclusion of mobile hotspot capabilities in virtually all new mobile devices opens the door to leverage the evil twin attack[1] to create a mobile malnet capable of spreading from device to device. Unlike in previous work we do not require vulnerabilities in the Wi-Fi access points, nor it is a problem that most private and corporate Wi-Fi networks now use WPA to encrypt their traffic. To test the feasibility of our approach we implemented the key components needed to create the malnet for iOS. In lab experiments we analyze the basic parameters and requirements of this malware mechanism and measure infection and propagation times in a controlled environment.

The rest of the paper is organized as follows: In section 2 related work in the field of mobile malware and evil twins is presented. Section 3 describes how mobile evil twin malnets work and which weaknesses of todays smartphones are utilized to render this type of attack possible. Additionally our prototypical implementation is described in-depth. In Section 4 we present the simulations we used to study the spreading characteristics, their assumptions and results. Section 5 concludes the paper and gives an overview of possible future work in this research area.

2 Related Work

There is a large body of related work discussing the spread of mobile device malware. Most of this work [5,6,7,8,9,10,11] relies on Bluetooth or user errors as an infection vector. One prominent example for Bluetooth-based worm research is described in a paper from Wang et al. [12] where they analyzed the spreading patterns of mobile viruses. They considered Bluetooth and multimedia messaging as distribution channels for this kind of worms. They predict serious threats to mobile phones once an operating system reaches high enough market shares. How vulnerable even *feature phones*[2] are, is shown in the work by Collin et al. [11]. They studied the vulnerabilities of these mobile phones and possible attacks against the mobile network using SMS and Bluetooth.

In 2007 Akritidis et al. [1] presented a simulated study of Wi-Fi-based malware in metropolitan area networks. The work explores the idea to utilize the proximity of Wi-Fi routers to spread malware. Akritidis et al. present two modes

[1] Evil twin is a term for a malicious Wi-Fi access point that appears to be a legitimate one by spoofing its SSID.

[2] Phones that are not considered to be smartphones, but have additional functionality beyond standard mobile services.

of infection labeled push and pull. Push infection resembles traditional worm infection methods most closely. With push infection a vulnerability in a device connected to a vulnerable hotspot is exploited to infect that device, which then goes on to infect further devices. The second method is the pull method. In this case the infected node waits for a device connected to a vulnerable access point to make some form of network request and then injects the worm code into this connection. The advantage of this method is that instead of relying on a service vulnerability, the attacker exploits vulnerabilities, such as browser vulnerabilities which are much more frequent and patched at a slower rate. In the studies case both the push and the pull method rely on visible, open and unencrypted access points to launch the attacks against other connected devices.

Another source of related work and inspiration for our work are malnets. Malnets are botnets created from non-traditional Wi-Fi devices such as mobile phones, but in particular wireless routers. In their work McDaniel [4], Hu et al. [13] and Traynor et al. [14] show that traditional network routers can be used to build up large malnets in areas with a high density of routers. The attack uses weak or default passwords to compromise wireless routers, which are in range, which then in turn attempt to infect other vulnerable routers in range. They developed epidemiological models and showed that in densely populated areas with enough vulnerable routers an epidemic spread of the malnet is completed within hours. While not strictly being a malnet the research presented by Akritidis et al. above has similar properties. But since these works only consider static environments where APs themselves have to be infected we describe in this work a new kind of infection and explore an alternative route to creating malnets, which does not require vulnerable routers.

The attack vector we utilize is based on the evil twin attack presented by Bauer et al. [15]. They showed that it is possible to trick a wireless client into associating with a rouge (evil twin) access point. The paper describes how an attacker can execute MITM attacks against selected victims by tricking them into connecting with the evil twin hotspot instead of the legitimate one. While this was viable and serious even then, it did not receive a great deal of attention. We assume this lack of interest was due to the fact that at the time of the attack it was only discussed in the context of infrastructure-based Wi-Fis, where targeted attacks against specific locations were the goal. The proliferation of WEP, WPA/WPA2 made this attack significantly harder in most scenarios. Also the effort of setting up the evil twin and the targeted nature of the attack reduced the mass-effect of the approach. The changes in the mobile computing landscape and our research into extending this attack by leveraging the capabilities of current smart mobile devices to create a potentially massive mobile malnet increases the threat significantly.

3 Mobile Evil Twin Malnets

In this section we show that the currently dormant threats shown above can be combined and extended to once again be a serious threat in the new mobile

computing landscape. There are two key factors which enable the approach presented in this paper; the proliferation of Universal Authentication Mechanism (UAM) Wi-Fi hotspots [16] accessed through captive portals and the capability of modern smart mobile devices to act as a mobile hotspot. Before we present the concept of the mobile malnet, we discuss the usage and security environment which has now made this type of malware possible.

3.1 Wi-Fi Hotspot Usage and Security

Wi-Fi access has become a commodity service in most urban areas, such as malls, coffee shops, hotels, airports and other public locations. A relatively small number of Wireless Internet Service Providers (WISPs) offer Wi-Fi access for either ad-hoc usage or bundled with other services such as mobile phone plans.

While most Wi-Fi routers now usually come pre-configured with WPA/WPA2 or similar channel encryption mechanisms, public Wi-Fi is often unencrypted. The lack of channel encryption for public Wi-Fi is mostly due to the fact that WISPs have not found a way to incorporate the necessary configuration steps into their business and usage model. While the lack of encryption brings with it a number of security problems such as impersonation, credential theft and other violations of a Wi-Fi user's privacy, the lack of strong mutual authentication during the connection to a public Wi-Fi hotspot and the unsecured use of SSIDs (Service Set Identifiers) for both ESS (Extended Service Set) and IBSS (Independent Basic Service Set) networks is the main problem.

In most public Wi-Fis authentication of users is handled by the UAM. With UAM, any device is allowed to associate with the Wi-Fi access point (AP) and is issued an IP address and other network information such as the standard gateway automatically via DHCP. After association, the user opens a web browser and enters any URL. A transparent HTTP proxy (also called captive portal) on the AP (or the underlying infrastructure) captures the request and redirects it to a login page. In the case of free Wi-Fi, the user is usually just required to accept the terms of use; in the case of subscription or pay-as-you-go Wi-Fi the user needs to provide valid credentials or purchase access on the fly. The credentials entered in the login form are forwarded to a back-end processing facility (AAA server, credit card processor, etc.) and after successful validation the user's session is started. Now the user's primary HTTP request is sent and the response is delivered to the user. Critically, the content of this portal is controlled entirely by the WISP. Later in this paper we will show that this issue creates several dangerous problems.

Today there is no practical way to authenticate APs wherefore it is neither mandatory nor automated and common. If at all, it needs to be performed by the user checking a SSL/TLS certificate. Unlike the use of certificates in traditional client/server Internet applications (which also can be fraught with difficulty), this measure has very little security effect in practice, when applied to hotspot security. SSL certificates are issued for a FQDN (fully qualified domain name). However, since there is no verifiable or even any form of specified link between the FQDN and the SSID, the certificate-based approach is not feasible. Thus,

the vast majority of users will not be able to associate a hotspot SSID with the "correct" host name for the hotspot's authentication service.

The final problem in current Wi-Fi security is the fact that for increased usability many mobile devices are configured to automatically connect to known Wi-Fi hotspots, where "known hotspot" is defined only by the SSIDs being equal. Since SSIDs are not authenticated this opens up the possibility of the evil twin attack [17].

3.2 Mobile Evil Twin Attack

If a device has been connected to a hotspot with a given SSID at any time in the past, it is susceptible to an "evil twin" attack. An attacker positioned outside the Wi-Fi range of the real hotspot or with a stronger signal can create an evil twin hotspot by spoofing its SSID. In the previous section, we showed that there is currently no practical way for users of a UAM-enabled Wi-Fi hotspot to reliably determine its authenticity. So if an evil twin broadcasts a "known" SSID and a device connects to it, the evil twin is then capable of executing man in the middle attacks (MITMAs) against that device. In this paper we introduce an adaptation of the evil twin attack to create a far more potent attack type, the Mobile Evil Twin attack (MET). Unlike the evil twin attack, the MET does not aim at subverting a specific AP to carry out targeted attacks, but to misuse the UAM weaknesses as an infection channel and replicate itself.

In order to spread, any malware needs to find vulnerable hosts and an infection vector. The environment described above offers both. In a lot of countries, mobile hotspots are ubiquitous in many establishments like Starbucks, Barnes&Noble and McDonald's. These hotspots usually have a single SSID per brand to allow customers to easily connect to hotspots across any of their locations. In some cases even the WISP's default SSID is used (i.e. by BT in the US and the UK, which use the SSID `BTOpenzone`). While the default SSID for public Wi-Fi makes sense from a usability perspective, it opens the door to the MET attack since it creates a large user base which has connected to a well known SSID. In the following, we will show the design of the MET attack and the prototypical implementation of the key components. The implementation extends the jailbreakme.com jailbreak to deploy all components for a self-replicating mobile malnet. All components were separately tested in the lab using iOS 4.3.3 running on an iPhone 4, an iPad 1 and an iPad 2. The principle can be ported to any OS with a root vulnerability. However, the usability feature of an automatic browser-based captive portal popup makes iOS the more attractive target. The whole attack process consists of four steps that are graphed in Figure 1 and described below.

Step 1: Masquerading and Injection. The first step of the MET attack is the exploitation of the auto-reconnection feature to known hotspots. The MET attack starts with the initial infection host broadcasting an SSID corresponding to a well used SSID such as those used by public hotspots like *tmobile*. If a mobile

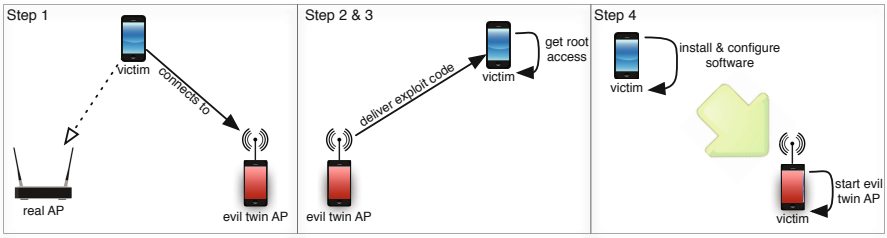


Fig. 1. MET attack overview

device has previously been connected to a legitimate AP of that SSID label, it is likely that it will automatically connect to the MET hotspot, unless there is a competing AP with higher signal strength or a SSID with higher priority.

Once the mobile device is connected to the MET hotspot all Internet activity of the device goes via the hotspot. With this setup, we can run a rogue Wi-Fi hotspot which can not only sniff, but also modify all unencrypted communication sent through it and inject malicious content.

The proof of concept malware presented in this paper uses `pf`, the iOS internal firewall and packet forwarding engine, to redirect all incoming traffic on port 80 to a locally deployed web server where the malicious payload is hosted. A more detailed description will follow later in this paper.

Step 2: Application Exploit. Once the mobile device is connected to the MET hotspot the second step of infection can occur. Since we use iOS for the prototypical implementation, which creates a MobileSafari popup window during the login process on the captive portal we can utilize a browser-based attack.

For all of the relevant smartphone platforms, i.e. iOS, Android, Palm/HP WebOS, BlackBerry and Symbian, the preinstalled web browsers are based on the WebKit [18] rendering engine. The WebKit engine is under constant scrutiny by security researchers and has had a number of security issues in the past (e.g.. [19,20,21]). Many of these security issues could be exploited by an attacker to execute arbitrary code on the phone with the same permissions as the web browser. Based on the long years of experience with desktop browsers and the recent history of mobile browsers, it can be assumed that mobile browsers will continue to be affected by security issues that allow arbitrary code execution for a fair while. For the proof of concept implementation we used the jailbreakme.com jailbreak which makes use of a PDF rendering vulnerability of the CoreGraphics framework in iOS 4.3.3 [22,23].

Step 3: Kernel Exploit. Executing code with the browser's permissions is rarely sufficient to install malware on a device. Compartmentalization and "sandboxing" of processes is a typical security approach for modern operating systems, including those on smartphones. Usually this means that the web browser – which runs under

an unprivileged user id – does not have privileges to install software on the phone. Therefore, another successful exploit on the operating system level is necessary to obtain full administrative privileges. Although harder to find than bugs in WebKit, these exploits exist and have existed for most mobile operating systems and it is likely that they will be available for future OS versions.

Step 4: Crossing-Over into the Evil Twin Malnet. Once a kernel-level exploit was executed on a mobile device the attacker gains complete control over that device, including the ability to control network and radio functions. This control is necessary for the propagation concept which we will outline in the following section. It makes use of the "personal hotspot" feature (Apple iOS $\geq$4.3). This hotspot functionality can be used by the device owner to share their 3G Internet connection with others by creating a mobile Wi-Fi access point. It conveniently bundles a small DHCP server, routing capabilities and Wi-Fi channel management and the setup is trivially easy. During the MET attack the mobile hotspot is activated using the target SSID making the device a new MET carrier.

By deploying *lighttpd* as a light weight web server on the infected mobile phone and rerouting all HTTP traffic of connected devices to it, we can deliver malicious code within any HTTP response and thus infect more mobile devices. Now, the infection cycle starts back at step 1.

3.3 Automating Infection

In the steps above an infection only occurs when a user manually triggers an event that receives content from the Internet into which the attack code can be injected. However, iOS has a convenient usability feature which enables almost instantaneous infection. When the iOS device is active, it automatically connects to the MET captive portal and as soon as any application requires Internet access an automated pop-up meant for credential input is shown.

This login page contains content which is fully controlled by whoever runs the Wi-Fi hotspot. It is, moreover, rendered with the same browser engine that is used for all other browsing activities. Therefore, virtually all exploits and security issues that exist for the WebKit browser family can also be exploited on that login page. Since the login page is displayed automatically, the MET infection routine can be executed as soon as the iOS device requests data from the MET hotspot.

Under the hood the following happens. After successfully associating with an AP and receiving an IP address, the iPhone's network stack sends one HTTP request to a specified URL. In the majority of cases the request goes to `http://www.apple.com/library/test/success.html`.

The response is then evaluated by iOS and if it consists of a valid HTML document containing only HTML metadata and exactly the word "Success" in the title tag, the network stack uses this as an indicator for the Internet

access being functional and finishes the connection setup. If any other response is received, the `apple.com` website is requested. In a second step the corresponding response is shown to the user in an automatic full-screen pop-up window which we use to deliver the exploit. The reason for this is the Universal Authentication Mechanism (see Section 3.1) where the user is prompted for his credentials by the captive portal login page.

3.4 Infection Implementation

In the following we describe the infection process with all involved systems and configurations. In the proof of concept presented in this paper we used a jailbroken iPhone 4 running iOS 4.3.3 as the initial infecting device. All further devices did not need to be prepared or jailbroken since all jailbreaks were executed automatically by the MET. The components of the prototype were tested on iPhone 4, iPad 1 and iPad 2.

To run an evil twin access point hotspot software needs to be configured that opens up Wi-Fi networks spoofing legitimate hotspot SSIDs. As the iOS internal personal hotspot cannot be activated and configured from the command line[3], we deployed MyWi 4[4], an app for jailbroken devices that has the same features, but can be configured through a plist file. It can be started through an undocumented program parameter from a shell script. This adds 1.8 MB to the MET prototype which could be avoided, but since optimizing the malware is not in the focus of this paper we chose the convenience MyWi offers. In addition to MyWi we installed the web server lighttpd due to its easy configuration and small footprint. The server is set up to listen on port 80 and is ready to deliver the exploit and the payload.

When a victim connects to this MET hotspot he gets an IP address from the local DHCP server that comes along with MyWi. We set up the firewall rules shown in Listing 1.1 to redirect all port 80 HTTP traffic from the hotspot network to the local lighttpd server.

```
1  nat on pdp_ip0 inet
2    from 192.168.40.0/24 to any
3    -> (pdp_ip0:0) static-port
4  no nat on ap0 inet
5    from 192.168.40.1 to 192.168.40.0/24
6
7  rdr on ap0 proto tcp
8    from 192.168.40.0/24 to any port 80
9    -> 127.0.0.1 port 80
10
11 pass on pdp_ip0
12   from any to any flags S/SA keep state
13 pass on ap0 all flags any
14   xkeep state (source-track global) rtable 4
```

Listing 1.1. PF firewall rules including the redirection for incoming traffic (line 7-9)

[3] The hotspot functionality relies on undocumented iOS Kernel APIs.
[4] Product website: `http://intelliborn.com/mywi.html`

When the victim's device checks the Internet connection, the web server responds to the first request of `http://www.apple.com/library/test/success.html` with a non-success webpage. Thus, the pop-up opens and displays the content of `http://www.apple.com`, which is also delivered by the local web server.

In this case it contains a hidden HTML iframe element showing a PDF file. Even within the popup and the hidden iframe, the iOS browser MobileSafari renders the PDF file. The malicious PDF file then exploits a vulnerability of the CoreGraphics library which is called by the Webkit engine. We use the exploit code from `http://www.jailbreakme.com` which first leads to a root shell and then uses it to download and install the actual jailbreak.

At this point there is an implementation issue with the jailbreakme.com PDF exploit. After the initial exploit is executed it tries to download the files that are required for the rest of the jailbreak and Cydia installation process. But since we are still in captive portal mode and the network state is set to "no internet connection" the download fails. Serving the success page for the portal and recapturing the device could solve this. However, the download would have to be delayed briefly until a functioning internet connection is recognized by iOS. This is not a conceptual problem and only stems from the fact that we are recycling an existing exploit which can not be modified at that point. If the malware was to be created for real a new exploit would be needed in any case and the delay would not be a problem. This issue does not affect the first infection case described above when a web browser is used. During the jailbreak process file downloads are initiated through plain HTTP requests without any authentication of the download server. Thus, we can redirect all requests to `http://www.jailbreakme.com` to the local web server as well and replace the jailbreakme.com payload with our own.

The jailbreak itself consists of three main parts: an initial filesystem image contains the directory structure including the `/bin` directory with utilities such as dpkg; a Debian package containing further programs; and a dynamic library that triggers and controls the installation of all these components. During the jailbreak process the integrity of the initial filesystem is checked. However, there is no integrity check made for the Debian package. Thus, we can modify the package and deliver it to the victim's device when the jailbreak process attempts to download the original one.

The modified Debian package contains not only the jailbreak files, all MET-related packages and configurations, a metadata file and a description of the package, but also command and control shell scripts that are executed before and after the installation or removal of the package. In the post-installation script of the modified package the installation of further packages is triggered. However, it is not possible to use the dpkg package manager directly for this purpose since the packaging system is locked during a running installation process. To circumvent this problem we register a daemon script to the iOS launch daemon `launchd` which usually monitors running iOS services and restarts them in the case of an abnormal termination (the script can be found in Listing 1.2 of appendix A).

Thus, after the post-installation script of the Debian package terminates successfully and removes the lock, iOS activates the daemon script and installs the rest of the malware. The daemon script checks if it was executed successfully in the past. If so, the daemon script deregisters itself from the iOS launchd service daemon and terminates. This is done to make sure that the installation of the evil twin software is not triggered twice. Otherwise it launches the installation of Debian packages that have been transferred to the victim's device within the Debian package. The hotspot software MyWi, the web server lighttpd and all of their package dependencies are included. The daemon script can be found in Listing 1.3 of appendix A.

To operate the malicious hotspot the two main components are started. The hotspot is started by a simple shell script call. For the web server a start script is registered as a daemon at the `launchd` service, like in the installation process. Now we have a fully operational evil twin hotspot that is ready to infect other devices.

There are currently three usability issues in the MET prototype. Firstly, as mentioned above, the popup-based installation would require a short delay to restore Internet access after the local redirection. Secondly, the installation of MyWi requires a restart of the device which would either be noticeable by the user or require the malware to wait for the next ordinary reboot. Thirdly, we do not hide the fact that we jailbreak the device. While the jailbreak process is executed in the background, we leave the AppStore equivalent Cydia in place. All three issues are implementation issues which do not affect the MET concept. Delaying the additional downloads by a couple of seconds solves the first issue. The second could be solved by subverting the iOS hotspot capability and the third would require the removal of the visible jailbreak signs. However, since it is not the goal of this work to create a stealthy real world malware and since these issues do not affect the concept of the MET attack there is currently no plan to address these issues.

4 MET Simulation

We used simulations to analyze the potential spreading characteristics of the mobile malware prototype presented in this paper. We developed a simulation framework [24] to study various security and privacy related issues in mobile environments. The framework allows us to carry out agent-based simulations on real-world road networks that are based on geospatial data from OpenStreetMap[5]. For the simulations in this work we used the framework and chose Downtown Chicago as the simulation area. Based on transport statistics we estimated a total number of 400,000 people using smartphones in downtown Chicago. Since we are simulating an exploit for the iOS platform we take the iOS market share of 20% [25] and the vulnerable iOS versions ($\sim$ 5% of iPhones are still running iOS version 4) into account. Thus, we have an infection base of about 4,000 devices. Considering the

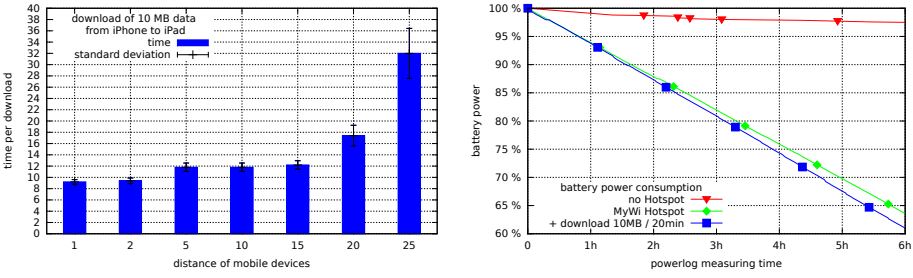
[5] `http://www.openstreetmap.org`

rising iOS market share and possible future multiplatform exploits we ran simulations also with a population size of 10,000 people. We implemented several types of agents which have different behavior patterns and walking speeds as well as diverse mobile device usage patterns. An agent chooses between different actions with a specified probability depending on his type.

Crucial factors for the simulation are the transmission speed and the battery lifetime of the devices. Therefore we investigated parameters like the infection duration and battery consumption of the prototype in order to set up the simulations as realistic as possible. We also conducted several parameter studies, varying population size as well as other parameters and found out characteristics for this type of attack in urban areas. For a more detailed description of the simulations we refer to [26].

4.1 Infection Duration and Battery Consumption

We conducted several lab experiments to determine the parameters of the MET prototype with the above limitations. The PDF file containing the initial exploit has a size of 17kB and its transfer plus the execution of the initial exploit takes under 1 second. This was measured using the browser based infection vector. Following this, the rest of the malware is downloaded by the initial exploit. Since this value is of relevance for the spreading characteristics of the malware, we measured the time needed for this download. Therefore we transferred a comparable 10MB MET package between the iPhone running the MET malware and a victim iPad at different distances.



(a) MET transfer duration measured over the distance between the mobile devices

(b) Battery consumption of the MET

Fig. 2. MET-related measurements

Figure 2a shows that the download time at the range of 1-2 meters was about 9 seconds. At distances of up to 15 meters download times rise to approximately 12 seconds, increasing to more than 32 seconds at a 25-meter distance. The measurements were conducted outside in a populated area with four other

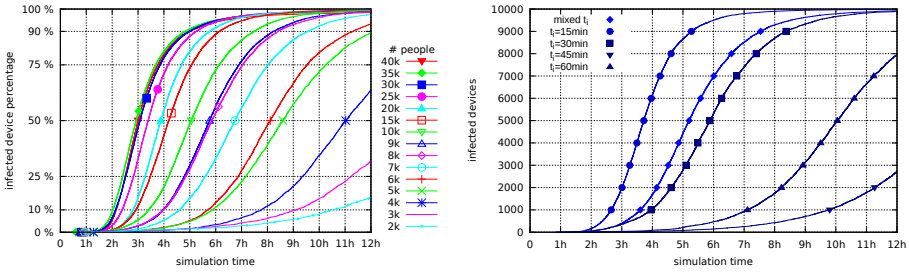
Wi-Fi hotspots active in the immediate vicinity. Each measurement was repeated five times and the standard deviation is shown in the figure.

Figure 2b shows the mean battery consumption of the MET measured over a six hour period. The red line marked with a triangle shows the battery consumption of the uninfected phone being idle. The green line marked with a diamond represents an infected phone with an active MET hotspot but otherwise idle. The blue line marked with a square shows a MET device which infects another device every 20 minutes. As can be seen the MET, if operated continuously, might be noticeable for the user due to the shortened battery lifetime. But with this shortened battery lifetime the infection of numerous other devices is still possible. Even if a user notices a suspicious behavior of his device, he will be not able to remove the MET without a complete phone reset. However the current version of the MET has not been optimized for stealth or longevity, since it is out of the focus of this paper.

The measured values are the basis for the simulations of mobile phones with limited battery life. In addition to the regular consumption each time a device infects another one, a small amount of battery lifetime is subtracted representing the running hotspot and the transfer of the mobile malware. If a device becomes infected its battery lifetime is reduced according to the measured values. Devices that run out of battery during the simulation neither can become infected nor can infect other nearby devices.

4.2 Simulation Results

In the following we present the results of the simulations to give a rough idea of how this kind of mobile malware could spread in an urban area.



(a) Infection simulation with different population sizes

(b) Infection simulation with different internet usage intervals

Fig. 3. Parametric study of the MET attack

Since the number of infectable devices has a significant effect on the epidemic spread of the malware, we conducted a parameter study over the number of devices in a closed world scenario, where no agent enters or leaves the simulated area. Figure 3a shows the normalized results of this study. As can be seen from

5,000 devices onwards an almost total infection occurs within 12 hours. From 10,000 devices onwards almost total infection is achieved within 8 hours.

To show the effect of usage patterns on the spread of the mobile malware, Figure 3b plots the amount of infections with different device activation intervals. As can be seen in the Figure the spreading of the malware has an epidemiological character for Internet usage intervals less than 30 minutes. Device usage intervals greater than 30 minutes lead to a significantly slower but nevertheless substantial spreading of the malware.

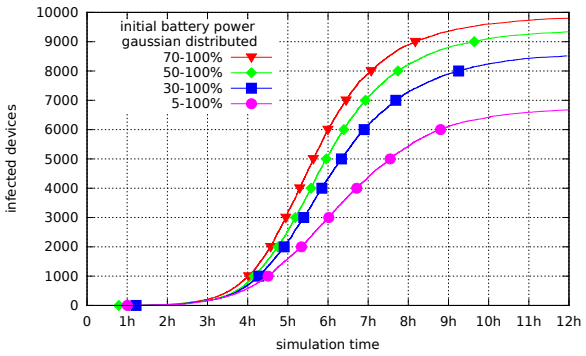


Fig. 4. Infection of 10,000 people depending on initial battery levels

Figure 4 shows a parameter study with different distributions of initial charge. Each simulation starts with a Gaussian distribution of initial charges in the range shown in the key of the figure. As can be seen the distribution of initial battery levels has a greater effect than the battery power drain. Surprisingly, even when a large number of devices run out of battery during the simulation there are still enough active infected devices to create almost full infection.

4.3 Mobile Evil Twin Malnet Summary

To summarize, the Mobile Evil Twin attack introduced above represents a new approach for creating mobile malnets. Unlike current Internet- (e.g.. spyeye [27]) or App Store-based (e.g.. Droid Dream [28]) attacks the user is not required to surf to a specific site or install a specific App to be infected. Unlike previous malnet approaches we do not require vulnerabilities in hotspots nor does the wide scale deployment of WPA/WPA2 pose a problem as long as there are some popular captive portal-based hotspots in operation somewhere. The concept for the automatic popup based infection was shown in theory and the web browser-based infection vector was fully implemented and shows the viability of the attack. The type of mobile malware presented in this work is made possible by both personal hotspot capabilities of smart devices as well as the fact that the usability and security measures in place for using and protecting Wi-Fi devices

were designed without this scale of capabilities in mind. Thus they offer very little protection against attacks in this realm. The conducted lab tests show the viability of the approach on a device-to-device basis.

5 Conclusion and Outlook

In this paper, we showed how new capabilities of mobile devices can be used to create a mobile evil twin malnet which is capable of spreading epidemically in current metropolitan areas. We present a proof of concept implementation for iOS 4.3.3 and measured the key attributes of the MET. In lab experiments we measured the infection time and battery consumption of the prototype to use these values in simulations. We simulated a metropolitan area to analyze the spreading characteristics of this new kind of mobile malware and showed that a large user base gets infected within a small amount of time.

There are many areas of future work. The mobile malware is just a proof of concept and could easily be made more effective or stealthy depending on the desired use. The malware can be extended to exploit more than one OS and cycle between different SSIDs to significantly increase the potential infection base and thus also significantly speed up the infection rate. Or the rate could be artificially slowed to make the spread more stealthy and energy saving. More importantly, an effective countermeasure needs to be researched to combat this type of malware. Current countermeasures do not work well against MET, thus opening up room for new ideas and research in this area. One promising idea is to use context information, such as the location or other environmental parameters to augment the decision when to automatically connect to known SSIDs or raise an alarm. IPSs and IDSs could also be ported to mobile platforms to collaboratively detect emerging malnets.

References

1. Akritidis, P., Chin, W.Y., Lam, V.T., Sidiroglou, S., Anagnostakis, K.G.: Proximity Breeds Danger: Emerging Threats in Metro-area Wireless Networks. In: Proceedings of 16th USENIX Security Symposium, pp. 22:1–22:16 (2007)
2. Becher, M., Freiling, F.C., Hoffmann, J., Holz, T., Uellenbeck, S., Wolf, C.: Mobile security catching up? revealing the nuts and bolts of the security of mobile devices. In: 2011 IEEE Symposium on Security and Privacy (May 2011)
3. Husted, N., Myers, S.: Mobile location tracking in metro areas: malnets and others. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 85–96. ACM (2010)
4. McDaniel, P.: Malnets: large-scale malicious networks via compromised wireless access points. Security and Communication Networks (2009)
5. Khouzani, M.H.R., Sarkar, S., Altman, E.: Maximum damage malware attack in mobile wireless networks. In: Proceedings of the 29th Conference on Information Communications, INFOCOM 2010, pp. 749–757. IEEE Press (2010)
6. Yan, G., Flores, H.D., Cuellar, L., Hengartner, N., Eidenbenz, S., Vu, V.: Bluetooth worm propagation. In: Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security - ASIACCS 2007, p. 32 (2007)

7. Singh, K., Sangal, S., Jain, N., Traynor, P., Lee, W.: Evaluating Bluetooth as a Medium for Botnet Command and Control. In: Kreibich, C., Jahnke, M. (eds.) DIMVA 2010. LNCS, vol. 6201, pp. 61–80. Springer, Heidelberg (2010)
8. Fleizach, C., Liljenstam, M., Johansson, P., Voelker, G.M., Mehes, A.: Can you infect me now? In: Proceedings of the 2007 ACM Workshop on Recurring Malcode, WORM 2007, p. 61 (2007)
9. Dagon, D., Martin, T., Starner, T.: Mobile Phones as Computing Devices: The Viruses are Coming! IEEE Pervasive Computing 3(4), 11–15 (2004)
10. Carettoni, L., Merloni, C., Zanero, S.: Studying Bluetooth Malware Propagation: The BlueBag Project. IEEE Security and Privacy Magazine 5(2), 17–25 (2007)
11. Mulliner, C., Golde, N., Seifert, J.P.: SMS of Death: From Analyzing to Attacking Mobile Phones on a Large Scale. In: Proceedings of the 20th USENIX Security Symposium (2011)
12. Wang, P., González, M.C., Hidalgo, C.A., Barabási, A.L.: Understanding the spreading patterns of mobile phone viruses. Science 324(5930), 1071–1076 (2009)
13. Hu, H., Myers, S., Colizza, V., Vespignani, A.: WiFi networks and malware epidemiology. Proceedings of the National Academy of Sciences of the United States of America 106(5), 1318–1323 (2009)
14. Traynor, P., Butler, K., Enck, W., McDaniel, P., Borders, K.: Malnets: large-scale malicious networks via compromised wireless access points. Security and Communication Networks 3(2-3), 102–113 (2010)
15. Bauer, K., Gonzales, H., McCoy, D.: Mitigating Evil Twin Attacks in 802.11. In: 2008 IEEE International Performance, Computing and Communications Conference, pp. 513–516 (December 2008)
16. Anton, B., Bullock, B., Short, J.: Best Current Practices for Wireless Internet Service Provider (WISP) Roaming. Wi-Fi Alliance (February 2003)
17. Cranfield University: 'evil twin' hotspots are a new menace for internet users, warns cranfield university (January 2005), http://www.sciencedaily.com/releases/2005/01/050120102036.html
18. Webkit Development Team: The webkit open source project (November 2011), http://www.webkit.org/
19. US-CERT: CVE-2011-0157: WebKit (March 2011), http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-0157
20. US-CERT: CVE-2011-1290: WebKit (March 2011), http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-1290
21. US-CERT: CVE-2011-1344: WebKit (March 2011), http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-1344
22. US-CERT: CVE-2010-3855: CoreGraphics (November 2010), http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-3855
23. US-CERT: CVE-2011-0226: CoreGraphics (July 2011), http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-0226
24. Henne, B., Szongott, C., Smith, M.: Towards a mobile security & privacy simulator. In: 2011 IEEE Conference on Open Systems, ICOS 2011, Langkawi, Malaysia (September 2011)
25. Savitz, E.: Windows Phone To Top iOS Market Share By 2016, IDC Says (June 2012), http://www.forbes.com/sites/ericsavitz/2012/06/06/windows-phone-to-top-ios-market-share-by-2016-idc-says/
26. Szongott, C., Henne, B., Smith, M.: Evaluating the threat of epidemic mobile malware. In: Proceedings of the 8th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (2012)

27. RSA FraudAction Research Labs: New SpyEye Gains Zeus Features: Detailed Analysis of SpyEye Trojan v1.3 (February 2011), modified for brevity: http://bit.ly/eAtn0B
28. Weintraub, S.: Google to throw the kill switch on malicious apps (March 2011), edited for brevity: http://bit.ly/eKPLQj

A Scripts and Configurations

```
#!/bin/sh
launchctl submit -l evilTwinInstall -- /tmp/evilTwinInstall.sh
launchctl start evilTwinInstall
```

Listing 1.2. Post installation script of the Debian package registering the daemon script to the launch daemon

```
#!/bin/sh
name=evilTwinInstall
sleep 10
if [ -f /tmp/evilTwinInstall.success ]
then
  launchctl remove evilTwinInstall
else
  dpkg -i /tmp/lighttpd_1.4.18-6_iphoneos-arm.deb
  [...]
  dpkg -i --force-all /tmp/com.mywi4.ondemand_4.50.6_iphoneos-arm.deb
  dpkg -i /tmp/com.mywi4_5.03.2_iphoneos-arm.deb

  mv /tmp/www /var/
  mv /tmp/com.mywi.plist /private/var/mobile/Library/Preferences/com.mywi.
      plist
  mv /tmp/lighttpd.conf /etc/lighttpd.conf
  mv /tmp/mypfrules.conf /etc/mypfrules.conf

  launchctl submit -l webserver -- /usr/sbin/lighttpd -f /etc/lighttpd.
      conf
  /Applications/MyWi.app/MyWiApp_ startmywi
  pfctl -F all
  pfctl -f /etc/mypfrules.conf
  touch /tmp/evilTwinInstall.success
fi
```

Listing 1.3. Daemon script triggering the installation of all required packages

Firm Grip Handshakes: A Tool for Bidirectional Vouching

Omer Berkman[1,⋆], Benny Pinkas[2,⋆⋆], and Moti Yung[3]

[1] Academic College of Tel Aviv Yaffo
[2] Bar Ilan University
[3] Google Inc. and Columbia University

Abstract. Clients trust servers over the Internet due to their trust in digital signatures of certification authorities (CAs) which comprise the Internet's trust infrastructure. Based on the recent DigiNotar attack and other attacks on CAs, we formulate here a very strong attack denoted "Certificate in The Middle" (CiTM) and propose a mitigation for this attack. The solution is embedded in a handshake protocol and makes it more robust: It adds to the usual aspect of "CA vouching" a client side vouching for the server "continuity of service," thus, allowing clients and server to detect past and future breaches of the trust infrastructure. We had simplicity, flexibility, and scalability in mind, solving the problem within the context of the protocol (with the underlying goal of embedding the solution in the TLS layer) with minor field changes, minimal cryptographic additions, no interaction with other protocol layers, and no added trusted parties.

1 Introduction

August 2011 presented a wakeup call for Internet security, when it was revealed that an unidentified attacker hacked into the computers of DigiNotar, a Dutch CA, and issued a fraudulent certificate for, among others, google.com. This certificate was subsequently used in a Man-in-the-Middle attack deployed against users in Iran. The certificate fooled browsers into assuming that they were encrypting data with a public key assigned to Google, where in fact they were using a public key chosen by the attacker. As a result, users' data, including their Gmail credentials such as their passwords and cookies, was revealed to the attacker.

The attack was possible since "trust" is assumed to operate "top down". Namely, the normal way for browsers to operate, based on the SSL/TLS protocol, is to trust any assertion made by a certificate authority (CA) that is trusted by the browser. In most cases this means that any CA whose key is preinstalled with the browser is trusted. The attack was identified since the Google certificate was "pinned" in the Chrome browser. Namely, its value was hardcoded in the

⋆ Most of this work was done while visiting Google.

⋆⋆ Most of this work was done while visiting Google. Partly supported by the SFEROT project funded by the European Research Council.

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 142–157, 2012.

browser, and the browser was not willing to accept certificates for Google (the browser's manufacturer) except from a very small number of CAs [10] (incidentally, this technology was deployed a mere two months before the attack). As a result, a suspicious user noticed the error and posted the rogue certificate on the web, and subsequently the certificate was identified as a fake one.

In the days that followed, it was revealed that the attackers issued hundreds of certificates for different high-importance web services, such as Google, Yahoo, Skype, Facebook, Microsoft Windows Update services and the anonymous communication system Tor, as well as some intelligence agencies. It was also revealed that DigiNotar knew of the breach more than a month before the rogue certificate became public, but has not notified anyone about it. Consequently, the root certificates of DigiNotar were revoked from all major browsers and the company went out of business [19,1].

The goal of this work is to mitigate attacks like the one described here, by fundamentally changing the notion of trust over the web. This is done by adding a "bottom-up" component allowing clients to vouch for the server's trustworthiness: following the first server-client handshake, the client has a cross-session "firm grip" on the server. The goal is to add this property with relatively small efforts and modifications, and yet to achieve a much more robust authentication for the typical client-server web interactions (i.e., via a modified SSL/TLS protocol).

1.1 The Current Trust Infrastructure and Our Goals

The web's trust infrastructure based on its early days' need to bootstrap trust in an essentially unlimited number of sites from an initial trust in a limited number of CAs. The trust model is built top-down (reflecting on the CA infrastructure and X.509 standards). The core idea is that each browser ships with the public keys of a limited number of root CAs, and subsequently trusts a public key presented by a web site if it is accompanied by a certificate chain leading to the public key from one of the root CAs. Over the years, the number of CAs trusted by browsers became very large: more that 650 organizations, located in 52 countries, were identified as valid CAs for Mozilla or Microsoft browsers [7,5]. It is likely that some of these CAs suffer from security vulnerabilities or are not managed properly. Indeed, earlier in 2011 an attacker obtained bogus certificates from Comodo, a major CA, but no actual attack using these certificate was identified in the wild [20].

The threat that the current infrastructure poses to users is very serious. It is sufficient for an attacker to forge certificates from a single "trusted CA" in order to apply Meet-in-the-Middle (MiTM) attacks to all web sites in all jurisdictions (with the help of phishing, malware or DNS contamination). This situation, allowing a very local exposure to potentially affect the web globally, is a colossal failure of the trust infrastructure from a risk managemnt perspective. [1]

[1] Indeed, it is very hard to come up with a set of policies defining a "normal" usage of CAs and certificates, and any such policy will result in many false warnings. We do not suggest to use such a policy, but rather use this situation to exemplify that the current trust model is very flawed and must be fixed.

Toward our goals, let us define the attack we aim to prevent, which we denote as the "certificate in the middle" (CiTM) attack. It is modeled after the DigiNotar incident, and is stronger than a man in the middle (MiTM) attack since it assumes that the attacker can both mount an active attack (e.g., by controlling the Domain Name System) *and* forge arbitrary valid certificate chains.

Definition 1 (CiTM attack). *A "certificate in the middle" (CiTM) attack is an attack by an adversary with the following capabilities:*

- *It can eavesdrop on any communication channel (in particular, to communication between the client and server).*
- *It can change messages that are sent on any communication channel.*
- *It can generate arbitrary valid certificate chains. In particular, the adversary can generate a certificate which states that a certain public key (for which the adversary knows the corresponding private key) is the public key of a different party.*

Being polynomial-time bounded and deprived of access to secure private memories, the adversary cannot*, however, learn the private keys of specific parties. In particular it does not know the private keys of the server which it tries to attack.*

The current web trust infrastructure is completely vulnerable to CiTM attacks since its top-down approach assumes that all CAs are trusted, whereas the CiTM attacker is able to issue certificates in the name of trusted CAs.

It is easy to verify that due to the strong capabilities of the adversary, one cannot protect unfortunate users who have all their communication channels permanently controlled by a CiTM attacker. These users will always receive the same certificates forged by the attacker, whereas the server will always observe messages that will comply with whatever policy the server might have.

Have we formulated above an attack we cannot prevent? Are we at a loss here since we formulated such a strong adversary? We claim that, nevertheless, there is hope for protecting users very efficiently. This is so since not all users will be permanently under the control of the attacker. Some clients will use uncompromised connections before they are subject to a CiTM attack, whereas others might first be subject to the attack and then be able to connect to the server through a legitimate channel (say, when they travel outside of the affected country or if the attacker's infrastructure fails for a short period of time). Although not all users might fall into these categories, it is important to note that, as with the DigiNotar breach, even a single educated user who notifies the world about the attack is likely to lead to a complete revocation of the relevant CA. Furthermore, deploying a CiTM attack requires compromising a CA, and therefore even the most determined attackers cannot deploy the attack too many times. We thus argue that even alerting a limited number of users to the existence of the attack can severely diminish its utility to attackers.

1.2 Contributions

We propose a concept and a framework for handshake protocols, that extends an initial handshake and minimizes the "window of opportunity for an attack" that can be employed by the adversary. This type of limitation is a principle in designing robust and secure systems. Specifically, our framework uses a form of "chaining" between all protocol handshakes that are performed between a specific client-server pair (hence we call it a "firm grip handshake"). As a result, if a client establishes an uncompromised handshake before it is subject to a CiTM attack then the client identifies the attack at the moment that it is attempted due to lack of chaining. Alternatively, even if all client-server handshakes were subject to an attack, it is sufficient to have a single uncompromised handshake in the future in order to inform the client about the attack by breaking the chaining to the compromised certificate.

Regarding server side security, we cannot guarantee that the server identifies an attack: due to the stateless nature of web servers, the attacker can modify all communication from the client to look as if each interaction is the first interaction between the client and server, and therefore no chaining with a previous handshake exists. However, by forcing attackers to resort to such behavior we enable servers to identify attacks by examining connection statistics and attempting to use, say, off-line anomaly detection techniques on the server's logs; (for example, if an exceedingly large proportion of connections from a certain location seem to be coming from new clients, e.g., new web browsers, then further examination of this phenomenon might be needed).

To highlight the new concepts and avoid implementation variations, we describe a high-level conceptual version of the protocol, rather than the specifics of embedding the protocol in existing TLS implementations (although we do discuss issues relevant to such an implementation). Our protocol is based on two main ideas:

- Extending the initial client-server handshake by having the server choose a key that will be used, together with the initial certificate chain, in order to MAC all future client-server handshakes. (The MAC value can be considered part of the reconstructable state and can be further hashed and MACed in existing fields, say in TLS).
- In order to improve scalability and retain the stateless-ness of web servers, the protocol stores the MAC key in the client side in encrypted and authenticated "sandwich cookies", or oreos, rather than storing it at the server.

Desired Properties. We suggest the following protocol properties:

- Security requirements
  - *Client identifying future attacks.* If the initial client-server handshake is not affected by a CiTM attacker, then any future CiTM attack is identified by the client.

  - *Client identifying past attacks.* If the initial client-server handshake is compromised by a successful CiTM attack, then once the client performs a handshake with the server over an uncompromised channel, i.e., a channel whose contents cannot be changed by the attacker, the client identifies that an attack has previously occurred.
  - *Server identifying irregular behavior.* Any active attack will result in either the server identifying the attack, or it identifying a usage pattern which is different than normal.

– *Simplicity and flexibility.* Changes required for preventing CiTM attacks must be easily integrateable with existing protocols. The changes to the existing implementations of the protocol must be minimal.
  Changes should be applied to only one layer of the communication stack (e.g., TLS) rather than several layers (e.g. TLS and the application which uses it). Ideally, they can be implemented without changing existing standards (e.g., consist of a few added fields in existing messages and data structures). No infrastructure changes should be required (e.g., no additional trusted parties are added).
– *Scalability.* The server need not keep a long-lived state for each client.
– *Efficiency.* The protocol requires only a few added crypto operations, preferably symmetric key operations.

The protocol does require the server to store a single short long-lived state. In Section 3.1 we describe how to support recovery from the server erasing this state, or from having this state being compromised.

1.3 Related Work

There have been several recent proposals for moving away from the current complete trust in certificate authorities, and they all deserve credit for worthy efforts to solve this problem. One such approach which is already deployed is the pinning of certificates in Chrome for the google.com domain [10], which proved to be successful in the DigiNotar attack. This approach, however, cannot scale since it requires to encode in the browser an entry for each supported domain.

In the origin-bound certificates [4] solution, the client browser generates a self-signed client certificate in its initial handshake with the server, and passes it to the server in all future handshakes. The server can then embed that certificate in the authentication processes, for example by storing it in a cookie in the application layer, and use it for authentication. This project has done remarkable work in implementing this solution as a TLS extension in the Chrome browser and in Google's web serving infrastructure. There are, however, some major differences between our solution and this project: Our protocol can be implemented in the TLS layer alone, and does not require integration with cookies served by the application layer. In addition, it identifies CiTM attacks even if they occur before the first uncompromised interaction between the client and the legitimate server, whereas such attacks against the origin-bound certificate solution remain undetected. A solution implemented in [2] is similar to origin-bound certificates but employs client's passwords available from any browser.

It allows web applications to use secrets that they share with clients, in order to attest for the authenticity of their certificates.

The TACK Internet draft [14] describes a TLS extension that enables a server to assert the authenticity of its public key by signing it with a server "TACK key". TLS connections to a pinned hostname require the server to present a TACK containing a pinned TACK key and a corresponding signature of the TLS server's public key. Unlike our solution, TACK enables only the client, and not the server, to identify CiTM attacks. Another difference stems from the way TACK implements its goal of limiting the damage from transient attacks on servers. Unlike our approach (see Section 3.1), TACK tries to minimize the "window of attack opportunity" by using time. Specifically, it limits the duration in which a TACK key is pinned in the client to be the minimum between 30 days and the length of time in which this TACK key has been observed by the client. As a result, a user who, for example, connects to his bank quite rarely, say once every 31 days, is not offered any protection by TACK.

Another set of solutions is based on using third-party services for extending the current trust infrastructure. For example, it is suggested in [13] to form a public auditable repository of every publicly visible certificate. Each certificate issued must be accompanied by an audit proof, and servers must send these proofs along with the certificates to browsers, which will then check them. Unlike our suggestion, this solution adds a third party to the infrastructure. In addition, domain owners must regularly monitor the public logs to ensure that no rogue certificates were issued for their domain. This is a commendable project that will provide complete transparency of certificate usage. Yet, this project is somewhat orthogonal to our local protocol extension approach, and requires considerable global resources in order to be implemented and maintained. A project with similar goals is EFF's Sovereign Keys [6]. Another suggestion along these lines is to use DNSSEC to bound certificates to domain names [16]. Another third-party approach is to use a notary service, i.e. to ask a third-party observer whether it observes the same certificate as the client. This approach was taken in [18,15,11].

2 The Protocol

We give a high-level conceptual version of the protocol, which can be applied to any handshake protocol (where embedding it in TLS and other certificate-based protocols is a major goal). All that we assume is that the initial message in the protocol is sent from the client to the server, and that it is possible to add fields to protocol messages. This latter assumption can be justified by instantiating these additional fields in several ways, as we discuss in Section 2.1.

The protocol is based on two main ideas. The first idea is that in the initial handshake between the client and server, the server chooses a fresh key, denoted as the "cream" for reasons that will shortly become clear, and sends it to the client. (This key can be sent encrypted or in the clear; see discussion in Section 2.1.) Later, each handshake is accompanied with MACs, keyed with this key, of the messages seen by the parties in the handshake, as well as with hashes

of the initial certificate chain that each of the parties had received. Therefore, an attacker who does not know the cream key will not be able to mount an active attack on future handshakes, even if it can issue a certificate of the server. Also, if the client receives an initial certificate chain different than the one sent by the server, then their hash values will be different and this fact will be identified in the first uncompromised communication.

The second idea is that the server need not store a state containing the cream key. Rather, it generates a sandwich cookie, or oreo, which contains an encrypted and authenticated envelope over the cream key and over a hash of the initial certificate chain. The oreo is stored at the client side and is sent by the client to the server in future handshakes. The server can then decrypt the oreo, verify its authenticity and use the resulting key for generating MACs and verifying MACs received from the client. We note that given the stateless nature of web servers the idea of keeping state at users has been suggested before (perhaps for the first time in [12]), and is typically used for providing the server with encrypted keys/states in cookies, for the server to restore keys and common state. We describe the protocol below. Its main steps are also depicted in Figure 1.

The Basic Protocol

Long Lived States: The server stores a long-lived server key, $\boldsymbol{sk}$. This is a symmetric key and is the only state that must be stored by the server. The client stores a "cream file", which is defined in the protocol below.

The Protocol:

When initiating a connection to the server the client sends an additional bit, the $\boldsymbol{sbit}$, which states whether the client already stores a state for this server.
If $sbit = 0$ (no state) then

1. The server picks a random key, denoted as the "cream key", or $\boldsymbol{ck}$. This key is a symmetric key chosen using fresh randomness.
2. The server computes a hash of the certificate chain that it sends to the client. Denote this value as the server hash, $\boldsymbol{sh} = H(\text{certificate chain sent})$, where $H()$ is a collision intractable hash function, for example a function from the SHA family.
3. The server uses the long-lived server key sk to generate an encrypted and authenticated version of the cream key ck and of the server hash sh. The result is denoted as the "sandwich cookie", or **oreo** (which contains a cream filling in it).
4. During the handshake, or immediately afterwards, the server sends the key ck and the oreo to the client. (See discussion in Section 2.1 on how these values can be sent, and whether they should be encrypted or not.)
5. The client receives the cream key ck. It also computes a hash of the certificate chain that it received. Namely, computes a client hash $\boldsymbol{ch} = H(\text{certificate-chain received})$ using the same function $H()$ as the server. The values ck and ch will never be sent in the clear by the client.

The client stores a state for the server in an entry which includes the cream key *ck*, the client hash *ch* and the oreo, and is stored in a special cream file. If all went well, the oreo contains a server hash *sh* that is identical to the client hash *ch*. (See discussion in Section 2.1 on storing this cream file.)

If $sbit = 1$ (namely, the client connects to a server for which it already has an entry in its cream file) then

1. The client sends the oreo to the server. This is done during the handshake or immediately afterwards. (See discussion in Section 2.1 on how this data can be sent.)
2. At the end of the handshake the client sends to the server a MAC, keyed with the cream key *ck*, of the concatenation of the string "client view" to all messages seen by the client in the handshake (messages both sent and received by the client). It also sends *ch*, the hash of the certificate chain received by the client in the initial handshake. (See discussion in Section 2.1 on why this value is not included in the MAC.) We assume that the "view" (of client/ server) is different in each interaction, and thus serves as a mechanism that prevents "replay attacks."
3. The server decrypts the oreo and learns the cream key *ck* and the server hash *sh* of the certificate chain sent by the server in the initial handshake. If the check of the authenticity of the oreo fails then the server aborts and notifies its operator.
   Otherwise, the server uses *ck* to check the MAC received from the client (namely, compute a MAC keyed by *ck* of the string "client view" concatenated to the messages seen by the server, and compare it to the MAC received from the client). It also compares the received *ch* value to its own *sh* value. If any of these checks fails then the server aborts and notifies its operator.
4. The server sends to the client a MAC, keyed with the cream key *ck*, of the string "server view" containing the concatenation of all messages seen by the server in the handshake. It also sends to the client the server hash *sh*. The client checks the MAC using the cream key *ck* and compares the received *sh* value to the *ch* value (recall that *ck* and *ch* are stored in the client's cream file). If any of these checks fails then the client aborts and notifies the user. (In a way similar to the message that is presented when certificate pinning is used and the wrong certificate is received.)

Note that if the client has an entry for the server in its oreo file, then the client always expects to receive a MAC at the end of the protocol, assuming (for now) that the MAC key is always available at the server.

2.1 Comments

Sending ck from server to client. In the initial handshake the server chooses a cream key *ck* and sends it to the client. It is possible to send this value encrypted

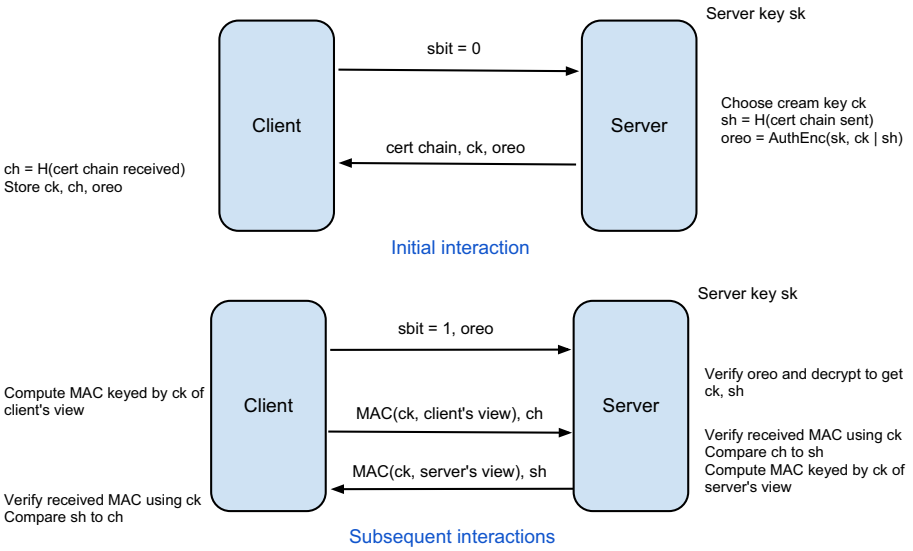


Fig. 1. The basic protocol

or unencrypted. The advantage in sending *ck* encrypted is that this prevents an adversary from learning the value even if the adversary is able to eavesdrop on the initial handshake. The disadvantage in encrypting *ck* is that encryption can only be performed after the two parties agree on a common key. Therefore an encrypted *ck* can only be communicated in the last message sent by the server in the handshake protocol (the server-finished message in TLS), or immediately after the handshake. This might complicate the implementation, as well as its integration with the existing protocol and the applications using it.

Note that the only advantage in encrypting *ck* is against an adversary that at the time of the initial interaction has capabilities which allow it to eavesdrop on the first handshake, but do not enable it to forge a server certificate or mount an active attack. Yet at a later time the adversary has the full capabilities required for mouting a full CiTM attack. The decision about whether to use encryption depends on whether one expects to encounter adversaries with this set of capabilities.

Sending separately the hash of the certificate chain. The last steps of the protocol have each party send to the other party a MAC of its respective view, as well as a hash of the initial certificate chain. A more natural way of implementing this step would have been to send just a single value equal to the MAC of both the view and the initial certificate chain. Namely, have, say, the client, send $\mathrm{MAC}_{ck}(\text{client}'\text{s view} \mid ch)$. This, however, prevents us from proving security

based on standard assumptions[2] and therefore we resulted to sending the hash of the certificate separately.

Storing the cream file. For each server the client must store an entry which contains the cream key ck, the client hash ch, and the oreo that the client received from the server. The client might store additional information that could be useful for forensic examination in case that a CiTM attack is identified, such as the time of the initial handshake and the certificate chain received in it.

This data can be stored in a special "cream file". This file is similar to a cookie file except for one important difference: the contents of the cream file are never sent outside of the client, except for the oreo data which is sent to the relevant server. Similar to a cookie file, the cream file contains sensitive private information that reveals, for example, which sites were visited by the user. Therefore it must be possible to apply to the cream file the same privacy controls as for cookie files, for example the option of deleting entries for specific sites (at the cost of losing CiTM protection for connections to these sites).

Changing to a new CA. The protocol does not prevent the server from having a new CA sign its certificates (say, because that CA gives the owner of the server a better financial offer). Future interactions with the client will be made using certificates signed by the new CA, but the oreo will not change, and handshakes will be MACed using the original cream key ck, and will include the hash of the certificate chain used in the initial client-server interaction.

Using existing TLS modes. Following the initial handshake which establishes the cream key ck, which then servers as a shared key between client and server, the protocol could be modified in many ways. Two, perhaps interesting, variants are to use ck as the pre-shared-key in [8] (using, say, the RSA-PSK key exchange) or as the password in [17].

Correctness. Let us state an easy and yet important claim about the well-functioning of the protocol if no active attack is used.

Claim. If no change is made to the messages exchanged between the client and server, then the new protocol provides the same functionality as the original handshake protocol.

Proof. If no changes are made to the messages then, in particular, the certificate chain sent by the server is identical to the certificate chain received by the client, and therefore the server hash sh is equal to the client hash ch. In addition, both parties have identical values for the key ck, and also the MAC values received

[2] The problem is with the case of an adversary that controls the initial communication with the server. That adversary can send the client an arbitrary MAC key ck and has to make sure that MACs computed with this key, of the certificate chain that it sent, are equal to the MACs that the server expects to receive. This seems as a very hard task, which is indeed impossible if we assume the MAC to be computed by a random oracle. However, the security definition of MACs in the standard model assumes that the adversary has no information about the key that is used, and this is not the case with this attack. Therefore we cannot prove the security of this protocol variant in the standard model.

are equal to those that are sent. Therefore all checks made by the parties are successful and the original protocol is allowed to run in its entirety.

2.2 Embedding the Protocol Data in Existing Protocols

The new protocol requires sending additional fields between the client and server, namely *sbit*, oreo, MAC and *ch* from the client to the server, and *ck*, oreo, MAC and *sh* from the server to the client. The most straightforward way would have been to change existing protocols, such as TLS, in order to support these new fields. However, changing TLS would be a lengthy process and it is preferable to be able to communicate the new fields without changing the original protocol. This goal is aided by the fact the new fields are not very long: if we assume a symmetric key to be 10-16 bytes long, the output of $H()$ to be 20 bytes long, and a MAC value to be 8-12 bytes long (shorter than a key, since attacking it requires an online attack), then the length of the oreo would be 18-28 bytes, and the length of the hash values ch, sh would be 20 bytes.

We describe here how the additional fields can be embedded over TLS messages. In particular, we describe how superflous certificate and implicit sending of values, can be used to communication the required information with minimal or no changes to the messages that are sent in the protocol.

Client to server communication. With regard to TLS, the oreo and *ch* value sent by the client can be embedded in either the client-hello or client-finished messages. The MAC sent by the client must be sent in the client-finished message, or after the handshake is over. The client-hello message contains an "extra data" field which can be used for sending arbitrarily long data [3]. In addition, the client-random field of the client-hello message contains a 28 byte long random string, part of which can be used to send data. The client-finished message contains a client-certificate field, which can be used in order to send a certificate that contains the MAC and the oreo (and which does not need not be signed by a CA since it will be otherwise ignored by the server).

Superflous certificates. In the first handshake the server needs to send *ck* and the oreo to the client in the server-hello or server-finished message. A natural solution is to embed these values in a new field of a certificate sent by the server. An unfortunate disadvantage of this approach is that CAs are known to charge significant amounts of money for adding new fields to certificates. A workaround is to use "superflous certificates", based on an undocumented feature of TLS.[3] This method works in the following way. The server server-hello message sent by the server contains a set of certificates. Ideally these certificates should form a certificate chain to a CA trusted by the client, but it is often the case that the

[3] See references to this method in
http://www.ietf.org/mail-archive/web/tls/current/msg08820.html,
http://tools.ietf.org/agenda/81/slides/tls-2.pdf, and
http://code.google.com/p/certificate-transparency/source/browse/src/client/ct.cc?r=103ff6cd41788fb51d37c3362632f639759ef4a7

server sends a set of certificates, only some of which form this chain. (This happens, for example, if instead of removing old certificates the server just adds new certificates to the set that it is sending to clients.) As a result, all major browsers accept server-hello messages in which only a subset of the certificates sent by the server form a chain, and the other certificates are superflous. Therefore, if in our protocol the server wishes to send additional information to the client, it can encode this information in a new certificate and add it to the certificate set sent to the client. It is not required to have this certificate signed by any CA since the client will identify a different valid certificate chain leading to a trusted CA. Note that the entire server-hello message is authenticated in the MAC sent in the server-finished message, keyed by the key agreed upon in the handshake protocol (here we refer the MAC sent as part of the TLS protocol, not the MAC in our protocol). Therefore an attacker cannot change this set of certificates or add new ones. In effect, the result is that by using superflous certificates, servers can add arbitrary authenticated information to TLS handshakes.

Using implicit MACs and hash values. The last steps of the protocol have each of the parties send to the other party a MAC of its view and a hash of the initial certificate. The other party checks these messages by computing its own version of these values and comparing it to the the values that it receives. Therefore, there is no need to send these values but only to make sure that both parties agree on them. Note also that TLS already requires each party to send, as its last message, a hash of all the messages it sent and received in the handshake protocol. These messages are denoted in TLS as the client-finished/server-finished messages, respectively.

We can therefore make the following change to the protocol: the client (and similarly the server) does not send the MAC and hash as required by our protocol but instead computes the last hash in the client-finished message as if it has sent these values. The server, which knows which values it expects to receive as the MAC and hash, uses these values to verify the client-finished message. The effective result is that each party can verify that the other party could have sent the correct MAC and hash values, but this is accomplished without sending any value expect for normal TLS fields.

3 Security and Extensions

We show that the client can identify CiTM attacks as long as it runs a single uncompromised handshake with the server. We also claim that although the server might not be explicitly warned about these attacks, it is likely to have sufficient information to implicitly suspect the presence of the attacks.

Intuitively, if the initial handshake is uncompromised then the client and server share the cream key ck which is unknown to any attacker, and which is used for MACing future handshakes. Therefore no active attack can be applied to a future handshake. If, on the hand, an attacker mounts a successful CiTM attack on the initial handshake, and since we assume that the attacker does not know the secret keys of the server, then the attacker must have used a certificate

chain different than any certificate chain used by the server. As a result, whenever the client performs a handshake with the original server they do not agree on the hash values, and the client informs its user about this discrepancy.

A note about the cream key ck. Our analysis here assumes that the cream key *ck* is sent encrypted in the initial client-server handshake. As is discussed in Section 2.1, this value can be sent either encrypted or unencrypted. The same analysis holds even if *ck* is sent unencrypted, as long as we assume that the attacker does not eavesdrop on that initial handshake.

The claims about security are based on assuming that some connections are made over "uncompromised channels", which we now define.

Definition 2 (Uncompromised channel). *A channel is* uncompromised *if an adversary can eavesdrop on communication carried out over the channel but cannot change it. (Namely, the adversary is only a passive eavesdropper.)*

We first state and prove two claims about the client identifying CiTM attacks if it has an uncompromised connection with the server.

Claim. Assuming that the cryptographic primitives used in the protocol are secure, then if the initial client-server handshake is uncompromised, the client will identify any future CiTM attack.

Proof. (Sketch) Since the initial handshake was uncompromised, the client received in it certificate chain and an oreo identical to the ones sent to it by the server, and therefore the client hash *ch* is equal to the server hash *sh*, and the oreo contains valid encrypted and authenticated values of *ck* and of *sh*.

Consider what happens before the first active attack attempt by the adversary. The adversary might have eavesdropped on the initial handshake and learned the certificate chain as well as an encryption of *ck* and the encrypted and authenticated oreo. (These encryptions are done using keys which are indistinguishable from random by the attacker.) The adversary might have also eavesdropped on subsequent handshakes which contained copies of the same oreo, and MACs keyed by *ck*. The adversary might have eavesdropped on communications of other clients, but these used key values which were independent of those used by the attacked client, except for the oreos which are all encrypted by the same server key *sk*. Standard cryptographic arguments can therefore show that if the adversary can learn anything about the cream key *ck* then either the encryption functions or the MAC are insecure. Since we assume these to be secure we conclude that the adversary cannot distinguish *ck* from a random string.

Now, consider the first handshake in which the adversary aims to change any of the messages. The adversary must send to the client a MAC of the new transcript of messages sent and received by the client, keyed by *ck*. Since part of these messages contain randomness chosen by the client, it holds with overwhelming probability that the transcript of messages in the current handshake is not identical to any of the transcripts of previous handshakes. Therefore the adversary must forge a MAC with a key it has no information about. A secure MAC algorithm ensures the adversary's failure, except with negligible probability.

As for CiTM attacks on the initial handshake, note that we need only consider attacks in which the attacker sends a certificate chain different than the certificate chain sent by the server to the client, since using a certificate chain of the original server requires the attacker to complete the initial handshake with a MAC based on a key encrypted with the public key of the server, which the attacker cannot decrypt. We are now ready to state our second claim.

Claim. If the initial handshake is compromised by a CiTM attack, and the attacker sends in it a certificate chain that is different than the one used by the server, then the client will identify the attack when it first connects to the server through an uncompromised channel.

Proof. (Sketch) The client receives an oreo from the attacker in the initial handshake. Then, in the client's first handshake with the real server it sends it this same oreo. The server checks the authenticity of the oreo, and therefore aborts the protocol unless the oreo was generated by the server itself. (It is possible that the oreo is not rejected, if the attacker sends to a client a valid oreo that a different client received from the same server.) The oreo contains a server hash *sh* of a certificate chain sent by the server. That certificate chain is different than the one received by the client, and as a result *sh* is different than the client hash *ch*, since both are computed by applying a collision intractable hash function to the certificate chain seen by the server and client, respectively. The last step of the handshake requires the server to send *sh* to the client, which then compares it to *ch* and aborts, since the two are different.

Attack identification by the server. In many cases it is sufficient to alert the client alone to the fact that a CiTM is taking place, since, as in the DigiNotar case, clients can use alternative communication channels to notify the rest of the world about the attack. Still, it is preferable to let the server learn in realtime that an attack is taking place.

The server checks the MAC sent to it by the client at the end of the handshake, and this check rejects any changes that an attacker injected into the messages. There is, however, an easy way for the attacker to prevent the server from identifying the attack, by setting the *sbit* to 0 and pretending that this handshake is the first handshake made by the client. In this case the server assumes that there is no oreo stored at the client, and it does not expect to receive a MAC at the end of the handshake (and therefore the attacker can drop this MAC from the messages transferred to the server).

This attack works in principle, but it should be possible for the server to use other signals in order to identify that an attack is taking place. Normally, we can assume that clients will set the *sbit* to 0 whenever they first connect to a server (say, when a user switches to a new web browser and connects to a site it frequented in the past). However, this event should not happen too often. The server could therefore gather relevant data and analyze it, perhaps using machine learning techniques, in order to identify suspicious signals. These signals could include, for example, a large percentage of users from the same physical location or country who all seem to be using a new browser. Or a specific user who,

in every new connection, seems to be connecting from a new browser. Overall, we expect that large scale or persistant CiTM attacks will be identified by the server as well as by the client.

3.1 Recovery of Server Keys

The basic protocol assumes that the server key sk is always accessible by the server and is never learned by an attacker. The protocol states that once the client stores an entry for a server in its cream file, it always expects to receive MACs from that server. Therefore, if the server loses its server key sk and is unable to decrypt oreos, it will not be able to communicate with the client. Furthermore, if sk becomes known to an adversary then that adversary will be able to compute valid MACs on future handshakes and apply CiTM attacks.

An extension to the protocol must therefore enable the server to recover from losses or compromises of the server key sk. This functionality is supported using a "recovery key" prk. This key is part of a public key-pair of secret/public keys (srk, prk) which are used for signining and for signature verification, respectively. An important property is that the private key srk can be easily kept offline until the time that it is needed (i.e., until the unlikely case that the server key sk is lost or is compromised). The key srk can be stored disconnected from the network, or even be stored in a non-electronic format, in order to minimize the chances of it being compromised.

The protocol is changed so that in the initial handshake the server sends the public key prk to the client, which stores it in its cream file. Afterwards, the protocol continues as usual except for the following change: the client sends an $sbit$ equal to 1, but it agrees to receive server answers corresponding to $sbit = 0$ if those answers and the entire handshake are signed with the secret key srk corresponding to the key prk in the server's entry in the cream file. Namely, the client accepts a change of the handshake by the server to an initial client-server handshake, as long as this change is signed by srk. Note that this feature can be used for periodic update of keys, as well as for recovery from key compromise.

Acknowledgments. We would like to thank Úlfar Erlingsson, Adam Langley and Cem Paya for valuable discussions.

References

1. Arthur, C.: Rogue web certificate could have been used to attack iran dissidents (August 2011), http://www.guardian.co.uk/technology/2011/aug/30/faked-web-certificate-iran-dissidents
2. Dacosta, I., Ahamad, M., Traynor, P.: Trust no one else: Detecting MITM attacks against SSL/TLS without third-parties. In: Foresti, et al. (eds.) [9], pp. 199–216
3. Dierks, T., Allen, C.: The TLS protocol version 1.0. RFC-2246 (1999)
4. Dietz, M., Czeskis, A., Balfanz, D., Wallach, D.S.: Origin-bound certificates: a fresh approach to strong client authentication for the web. In: USENIX Security, Berkeley, CA, USA (2012)

5. Eckersley, P., Burns, J.: An observatory for the SSLiverse (2010), https://www.eff.org/files/DefconSSLiverse.pdf
6. EFF: The Sovereign Keys project, https://www.eff.org/sovereign-keys
7. EFF: The EFF SSL observastory (2010), https://www.eff.org/observatory
8. Eronen, P., Tschofenig, H.: Pre-shared key ciphersuites for transport layer security (TLS). RFC-4279 (2005)
9. Foresti, S., Yung, M., Martinelli, F. (eds.): ESORICS 2012. LNCS, vol. 7459. Springer, Heidelberg (2012)
10. Google: New chromium security features (June 2011), http://blog.chromium.org/2011/06/new-chromium-security-features-june.html
11. Holz, R., Riedmaier, T., Kammenhuber, N., Carle, G.: X.509 forensics: Detecting and localising the SSL/TLS men-in-the-middle. In: Foresti, et al. (eds.) [9], pp. 217–234
12. Janson, P., Tsudik, G., Yung, M.: Scalability and flexibility in authentication services: The kryptoknight approach. In: Annual Joint Conference of the IEEE Computer and Communications Societies (1997)
13. Laurie, B., Langley, A.: Certificate authority transparency and auditability (2011), http://www.links.org/files/CertificateAuthorityTransparencyandAuditability.pdf
14. Marlinspike, M., Perrin, T.: Trust assertions for certificate keys. draft-perrin-tls-tack-00.txt (2012)
15. Marlinspike, M.: Convergence, http://convergence.io
16. Osterweil, E., Kaliski, B., Larson, M., McPherson, D.: Reducing the X.509 attack surface with DNSSEC's DANE. In: SATIN: Securing and Trusting Internet Names (March 2012)
17. Taylor, D., Wu, T., Mavrogiannopoulos, N., Perrin, T.: Using the secure remote password (SRP) protocol for TLS authentication. RFC-5054 (2007)
18. Wendlandt, D., Andersen, D.G., Perrig, A.: *Perspectives:* improving SSH-style host authentication with multi-path probing. In: Isaacs, R., Zhou, Y. (eds.) USENIX Annual Technical Conference, pp. 321–334. USENIX Association (2008)
19. Wikipedia: DigiNotar — Wikipedia, the free encyclopedia (2012), http://en.wikipedia.org/wiki/DigiNotar
20. Zetter, K.: Hack obtains 9 bogus certificates for prominent websites; traced to Iran (2011), http://www.wired.com/threatlevel/2011/03/comodo-compromise/

Group Key Establishment: Adding Perfect Forward Secrecy at the Cost of One Round

Kashi Neupane[1], Rainer Steinwandt[2,*], and Adriana Suárez Corona[2,**]

[1] Atlanta Metropolitan State College, Atlanta, GA 30310
kneupane@atlm.edu
[2] Florida Atlantic University, Boca Raton, FL 33431
{rsteinwa,asuarezc}@fau.edu

Abstract. A compiler is presented which, in the random oracle model, allows to add perfect forward secrecy to any secure authenticated group key establishment protocol P which has at least one round. The compiler does not modify the session identifier and does not impose changes on the underlying public key infrastructure. Building on a secure unauthenticated 1-round 2-party key establishment Q with perfect forward secrecy as auxiliary input, P is transformed into an authenticated group key establishment protocol with perfect forward secrecy and with one more round than P.

Keywords: protocol compiler, group key establishment, forward secrecy.

1 Introduction

The establishment of a common session key among a set of users over an insecure network is a fundamental cryptographic task. The most basic security requirement usually considered in group key establishment is semantic security of the session key in the presence of a suitably formalized adversary. To address the problem of compromised long-term secrets, this basic security requirement is often extended so that perfect forward secrecy is provided—even if all long-term secrets are leaked, the session key remains computationally indistinguishable from a uniformly at random chosen element of the session key space.

A substantial body of work on modularizing the design of group key establishment protocols is available, including techniques to augment a passively secure protocol so that it offers security against an active adversary [KY03] and malicious insiders [Boh06], or to scale a 2-party solution to an n-party solution [ABVS07, NPW11]. Interestingly, there does not seem to be much work available on how to achieve perfect forward secrecy in a systematic way. While it is a common design practice to separate secret key material used for authentication

* RS was supported by the Spanish *Ministerio de Economía y Competitividad* through the project grant MTM-2012-15167.
** ASC was supported by project MTM2010 - 18370 - C04- 01 and FPU grant AP2007-03141, cofinanced by the European Social Fund.

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 158–168, 2012.

purposes (such as a signing key) from the actual derivation of a session key, we are not aware of a discussion of provable generic techniques which allow to 'lift' a given group key establishment protocol—e. g., a key transport—to one which provides perfect forward secrecy. A one-round construction for group key establishment is available [GBNM10], but no such solution with perfect forward secrecy appears to be available. Combining one-round constructions with an efficient compiler to add forward-secrecy in a generic way appears to be an attractive modular design approach to identify new two-round protocols for group key establishment.

Below we suggest, in the random oracle model, a compiler which does not make use of digital signatures and does not require parties to make additional data publicly available. From a practical point of view this means that the compiler does not impose the introduction of a modified or new public key infrastructure. The construction builds on the availability of an unauthenticated 1-round key establishment for two parties which provides perfect forward secrecy—such as a 2-party Diffie-Hellman with a session key of the form $H(abP)$. The session identifier of the underlying group key establishment is invariant under the compiler.

2 Technical Preliminaries

To formalize the task of group key establishment we use a model of Bohli et al. [BVS07] which builds on Bresson et al. [BCP01] and Katz and Yung [KY03].

2.1 Security Model and Security Goals

Protocol participants. Let k be the security parameter and $\mathcal{U}$ the set of protocol participants which we assume to be polynomial in k. Each *user* $U \in \mathcal{U}$ is a probabilistic polynomial time algorithm and we allow each $U \in \mathcal{U}$ to execute concurrently a polynomial number of protocol instances Π_U^s ($s \in \mathbb{N}$). User identities are assumed to be bitstrings of identical length k and to keep notation simple, throughout we will not distinguish between the bitstring identifying a user U and the algorithm U itself. To a protocol instance Π_U^s, the following seven variables are associated:

acc_U^s: is set to TRUE if the session key stored in sk_U^s has been accepted;
pid_U^s: stores the identities of those users in $\mathcal{U}$ with which a key is to be established, including U (the latter ensures that being partnered is a reflexive relation);
sid_U^s: stores a session identifier, i. e., a non-secret identifier for the session key stored in sk_U^s;
sk_U^s: is initialized with a distinguished NULL value and after a successful protocol run holds the session key;
state_U^s: stores state information needed for executing the protocol (e. g., a secret scalar of an ephemeral Diffie-Hellman key);
term_U^s: is set to TRUE if this protocol execution has terminated;
used_U^s: indicates if this instance is used, i. e., currently involved in a protocol execution.

Initialization. Before actual protocol executions take place, we allow an optional trusted initialization phase *without adversarial interference*. In this phase, for each $U \in \mathcal{U}$ a (public key, secret key)-pair (pk_U, ak_U) can be generated, ak_U is given to U only, and pk_U is handed to all users in $\mathcal{U}$ and to the adversary. The initialization phase can also be used to disseminate further public parameters, if necessary.

Adversarial capabilities and communication network. The adversary $\mathcal{A}$ is represented as a probabilistic polynomial time algorithm with full control over the communication network. This results in a fully asynchronous, non-private network allowing arbitrary point-to-point connections among users. More specifically, $\mathcal{A}$'s capabilities are expressed through the following *oracles*:

$\mathsf{Send}(U, s, M)$: This oracle serves two purposes.

- With the Send oracle, $\mathcal{A}$ can initialize a protocol execution; sending the special message $M = \{U_{i_1}, \ldots, U_{i_r}\} \subseteq \mathcal{U}$ to an unused instance Π_U^s with $U \in M$ initializes a protocol run among $U_{i_1}, \ldots, U_{i_r}$. After such a query, Π_U^s sets $\mathsf{pid}_U^s := \{U_{i_1}, \ldots, U_{i_r}\}$, $\mathsf{used}_U^s :=$ TRUE, and processes the first step of the protocol.
- The message M is sent to instance Π_U^s and the protocol message output by Π_U^s after receiving M is returned.

$\mathsf{Reveal}(U, s)$: returns the stored session key sk_U^s if $\mathsf{acc}_U^s =$ TRUE and a NULL value otherwise.

$\mathsf{Corrupt}(U)$: for a user $U \in \mathcal{U}$ this query returns U's long-term secret key ak_U.

It is worth noting that, unlike Reveal, Corrupt refers to a *user* rather than an individual *protocol instance*. An adversary with access to all of the above oracles is considered *active*. To capture a *passive* adversary, access to Send is replaced with access to an Execute oracle, returning a complete protocol transcript among the specified unused instances.—An active adversary can simulate such an Execute oracle by means of Send in the obvious manner.

For technical reasons, there is one more oracle, Test, and $\mathcal{A}$ must submit exactly one query of the form $\mathsf{Test}(U, s)$ with an instance Π_U^s that has accepted a session key, i. e., with $\mathsf{acc}_U^s =$ TRUE. In response to such a query, a bit $b \leftarrow \{0, 1\}$ is chosen uniformly at random and for $b = 1$ the established session key stored in sk_U^s is returned. For $b = 0$ the output is a uniformly at random chosen element from the space of session keys. The idea is that for a secure group key establishment protocol, no efficient adversary can distinguish between $b = 0$ and $b = 1$. To turn this idea into a definition we first exclude trivialities and restrict our discussion to *correct* group key establishment protocols:

Definition 1 (Correctness). *A group key establishment is* correct *if on honest delivery of all messages and all users being honest, a single protocol execution among users $U_0, \ldots, U_{n-1}$ involves n instances $\Pi_0^{s_0}, \ldots, \Pi_{n-1}^{s_{n-1}}$ such that with overwhelming probability all of the following hold:*

- *all users accept, i. e.,* $\mathsf{acc}_0^{s_0} = \cdots = \mathsf{acc}_{n-1}^{s_{n-1}} = \text{TRUE}$;
- *all users obtain the same session identifier, i. e.,* $\mathsf{sid}_0^{s_0} = \cdots = \mathsf{sid}_{n-1}^{s_{n-1}}$;
- *all users accept the same session key, i. e.,* $\mathsf{sk}_0^{s_0} = \cdots = \mathsf{sk}_{n-1}^{s_{n-1}} \neq \text{NULL}$ *associated with the same session identifier* $\mathsf{sid}_0^{s_0}$;
- *all communication partners are specified as desired communication partner, i. e.,* $\mathsf{pid}_0^{s_0} = \cdots = \mathsf{pid}_{n-1}^{s_{n-1}} = \{U_0, \ldots, U_{n-1}\}$.

Correctness refers to a scenario where no attack takes place; to formulate security guarantees we have to specify under which circumstances a correct guess for the random bit used by the Test oracle constitutes a viable attack. For this we use the following notions of partnering and freshness.

Definition 2 (Partnering). *Two instances* $\Pi_{U_i}^{s_i}$ *and* $\Pi_{U_j}^{s_j}$ *are* partnered *if* $\mathsf{sid}_{U_i}^{s_i} = \mathsf{sid}_{U_j}^{s_j}$, $\mathsf{pid}_{U_i}^{s_i} = \mathsf{pid}_{U_j}^{s_j}$ *and* $\mathsf{acc}_{U_i}^{s_i} = \mathsf{acc}_{U_j}^{s_j} = \text{TRUE}$.

Based on this notion of partnering, we can specify what we mean by a *fresh* instance—an instance where the adversary does not know the session key for trivial reasons. We consider two types of freshness. In the first one the adversary cannot corrupt any user associated with the Test-instance, and therefore forward secrecy is not implied:

Definition 3 (Freshness without forward secrecy). *An instance* $\Pi_i^{s_i}$ *is called* fresh *if none of the following two conditions hold:*

- *For some* $U_j \in \mathsf{pid}_i^{s_i}$ *a* $\mathsf{Corrupt}(U_j)$*-query has been executed.*
- *A query* $\mathsf{Reveal}(U_j, s_j)$ *with* $\Pi_i^{s_i}$ *and* $\Pi_j^{s_j}$ *being partnered occurred.*

The second formulation allows an adversary $\mathcal{A}$ to reveal *all* secret keys without violating freshness, provided $\mathcal{A}$ does not send any "relevant" messages after having received the secret keys. As a consequence, security in the sense of Definition 6 below implies forward secrecy:

Definition 4 (Freshness with forward secrecy). *An instance* $\Pi_i^{s_i}$ *is called* fs-fresh *if none of the following two conditions hold:*

- *For some* $U_j \in \mathsf{pid}_i^{s_i}$ *a* $\mathsf{Corrupt}(U_j)$ *query was executed before a query of the form* $\mathsf{Send}(U_k, s_k, *)$ *has taken place where* $U_k \in \mathsf{pid}_i^{s_i}$.
- *A query* $\mathsf{Reveal}(U_j, s_j)$ *with* $\Pi_i^{s_i}$ *and* $\Pi_j^{s_j}$ *being partnered occurred.*

We write $\mathsf{Succ}_{\mathcal{A}}^{\mathsf{ke}}$ for the event that $\mathcal{A}$ queries Test with a fresh instance according to Definition 3 and outputs a correct guess for the Test oracle's bit b. Similarly, we write $\mathsf{Succ}_{\mathcal{A}}^{\mathsf{fs\text{-}ke}}$ for $\mathcal{A}$ correctly identifying b when querying Test with an fs-fresh instance in the sense of Definition 4.

Definition 5 (Semantic security without forward secrecy). *A key establishment protocol is said to be* (semantically) secure, *if the advantage* $\mathrm{Adv}_{\mathcal{A}}^{\mathsf{ke}} = \mathrm{Adv}_{\mathcal{A}}^{\mathsf{ke}}(k) := \left|2 \cdot \Pr[\mathsf{Succ}_{\mathcal{A}}^{\mathsf{ke}}] - 1\right|$ *is negligible for all probabilistic polynomial time algorithms* $\mathcal{A}$.

Analogously, with $\mathrm{Adv}^{\mathsf{fs\text{-}ke}}_{\mathcal{A}} = \mathrm{Adv}^{\mathsf{fs\text{-}ke}}_{\mathcal{A}}(k) := \left|2 \cdot \Pr[\mathsf{Succ}^{\mathsf{fs\text{-}ke}}_{\mathcal{A}}] - 1\right|$ taking the role of $\mathrm{Adv}^{\mathsf{ke}}_{\mathcal{A}}$, we obtain the stronger

Definition 6 (Semantic security with forward secrecy). *A key establishment protocol is said to be* fs-(semantically) secure*, if* $\mathrm{Adv}^{\mathsf{fs\text{-}ke}}_{\mathcal{A}} = \mathrm{Adv}^{\mathsf{fs\text{-}ke}}_{\mathcal{A}}(k)$ *is negligible for all probabilistic polynomial time algorithms* $\mathcal{A}$*.*

To make explicit that adversaries are considered to be active, i. e., have access to Send, it is common to refer to a group key establishment protocol as *authenticated*. Complementing this terminology, we speak of *unauthenticated* group key establishment to indicate that adversaries are passive.

Remark 1. In Definitions 5 and 6 the advantage can equivalently be written as $\mathrm{Adv}^{(\mathsf{fs\text{-}})\mathsf{ke}}_{\mathcal{A}} = |\Pr[1 \leftarrow \mathcal{A} \mid b = 1] - \Pr[1 \leftarrow \mathcal{A} \mid b = 0]|$.

Remark 2. In case of ambiguity we will also include the name of the protocol in the index, writing $\mathrm{Adv}^{(\mathsf{fs\text{-}})\mathsf{ke}}_{\mathcal{A},\mathsf{R}}$ for the advantage of adversary $\mathcal{A}$ when attacking protocol R.

3 A Compiler to Achieve Forward Secrecy

Throughout, we refer to the set of protocol participants aiming at a common session key as $U_0, \ldots, U_{n-1}$ and think of them as being arranged in a circle with indices being taken mod n. By $H : \{0,1\}^* \longrightarrow \{0,1\}^k$ we denote a random oracle, and we assume the session key space to be $\{0,1\}^k$.

3.1 Construction

As a protocol P which is not forward-secure may leak the complete session key when long-term secret keys leak, the compiler below uses the session key established by P as ephemeral authentication key only. The session identifier of P is preserved, however. To derive the new session key, our construction assumes the availability of a secure unauthenticated 1-round 2-party key establishment Q with perfect forward secrecy.[1] A natural choice is a 2-party Diffie-Hellman key establishment over a suitable cyclic group $\langle P \rangle$ with an established key of the form $H(abP)$, but more conceptual proposals such as [KLC+00] indicate that our compiler might also be of interest in a post-quantum setting. The proposed compiler assumes that the given authenticated group key establishment P involves at least one round of communication and modifies P in two ways:

Modification of Round 1. In addition to the first round of protocol P, U_i broadcasts two messages m_i^{L} and m_i^{R} to execute two instances of Q, establishing 2-party keys sk_i^{L} and sk_i^{R} with $U_{i-1 \pmod n}$ and $U_{i+1 \pmod n}$.

[1] The formal definition is obtained by restricting Definition 6 in the obvious way to a 2-party setting.

Addition of a final round. After U_i has accepted the session key $sk_{\mathsf{P},i}$ with session identifier $sid_{\mathsf{P},i}$ in protocol P, participant U_i performs the following additional steps:

- Compute the values

$$T_i := sk_i^{\mathrm{L}} \oplus sk_i^{\mathrm{R}},$$
$$\theta_i := H(sid_{\mathsf{P},i}||\mathsf{pid}_{U_i}||(m_0^{\mathrm{L}}, m_0^{\mathrm{R}})|| \ldots ||(m_{n-1}^{\mathrm{L}}, m_{n-1}^{\mathrm{R}})||T_i||U_i||sk_{\mathsf{P},i}).$$

- Broadcast (T_i, θ_i, U_i).
- Recover sk_0^{R} from

$$sk_0^{\mathrm{R}} = sk_i^{\mathrm{L}} \oplus T_0 \oplus \bigoplus_{j=i}^{n-1} T_j, \tag{1}$$

  verify $\theta_0, \ldots, \theta_{n-1}$, and check if $T_0 \oplus \cdots \oplus T_{n-1} = 0$. If any check fails, terminate without accepting a session key.
- Accept the session key sk_0^{R} with session identifier sid_{P}.

Remark 3. Security in the sense of Definition 5 does not guarantee that accepting parties hold the same session key or session identifier. The index i in $sk_{\mathsf{P},i}$ and $sid_{\mathsf{P},i}$ tries to make this explicit.

3.2 Design Rationale

As mentioned in the previous section, we assume no guarantees about the forward secrecy of the key established in protocol P, and hence the computation of the actual session key established by the compiler relies on the 2-party protocol Q only. Our use of 2-party keys in a circle of participants follows the well-known construction of Burmester and Desmedt [BD94], using the key established in P to authenticate legitimate protocol participants. As the security definitions allow the session key to be determined by a proper subset of the protocol participants, doing without an elaborate key derivation and using one of the 2-party keys as established session key seems a viable and inexpensive option.

The tag θ_i in the additional round provides an explicit key confirmation for the group key established in P and uses this key to authenticate messages added by the compiler. When aiming at a standard model construction it is tempting to derive the tags θ_i by means of a message authentication code, using $sk_{\mathsf{P},i}$ as secret key. The technical issue one encounters here, however, is that the relation among the keys accepted by $U_1, \ldots, U_{n-1}$ in P is not clear—neither do they have to be equal nor independent. So when trying to argue that the tags cannot be forged, one had to deal with a situation where an adversary has access to tags under potentially related keys, a scenario which is not directly addressed in the usual definition of existential unforgeability. When using a random oracle, as is done in our compiler, possible relations among the $sk_{\mathsf{P},i}$ are of no concern.

Finally, in the described form, the compiler uses protocol Q as a black box, but it is worth noting that for a specific Q it can be possible to avoid the execution of two independent instances of Q in Round 1. Specifically, for a Diffie-Hellman key exchange, U_i can use a single message $(u_i P, U_i)$ to establish both keys, therewith reducing the communication cost.

3.3 Security Analysis

The following theorem shows that the above compiler indeed augments P in the desired way.

Theorem 1. *Let* P *be authenticated and secure according to Definition 5 and* Q *an unauthenticated 1-round 2-party key establishment which is secure in the sense of Definition 6. Then the group key establishment obtained from the compiler in Section 3.1 is authenticated and secure according to Definition 6.*

Proof. Let q_{send} be a polynomial upper bound for the total number of $\mathcal{A}$'s queries to the Send oracle. Moreover, let q_{ro} be a polynomial upper bound on the total number of $\mathcal{A}$'s queries to the random oracle H, including both direct calls of $\mathcal{A}$ to H and indirect calls through queries to Send.[2] We prove the security of the protocol by "game hopping", letting the adversary $\mathcal{A}$ interact with a simulator. The advantage of $\mathcal{A}$ in Game i will be denoted by $\text{Adv}_{\mathcal{A}}^{\text{Game } i}$:

Game 0: This game is identical to the original attack game, with all oracles of the adversary being simulated faithfully. Consequently,

$$\text{Adv}_{\mathcal{A}}^{\mathsf{fs\text{-}ke}} = \text{Adv}_{\mathcal{A}}^{\text{Game } 0}.$$

Game 1: In this game we modify the adversary in such a way that at the beginning she guesses uniformly at random which instance $\Pi_{i_0}^{s_{i_0}}$ will be queried to the Test oracle as well as one more instance $\Pi_{i_0-1}^{s_{i_0-1}}$ with which $\Pi_{i_0}^{s_{i_0}}$ will, after the compiler's modification of Round 1, establish a 2-party key $sk_{i_0}^{\text{R}}$. Whenever at least one of these guesses turns out to be wrong, we abort the simulation and consider the adversary to be at loss. Otherwise the game is identical with Game 0. Consequently,

$$\text{Adv}_{\mathcal{A}}^{\text{Game } 0} \leq 2 \cdot q_{\text{send}}^2 \cdot \text{Adv}_{\mathcal{A}}^{\text{Game } 1},$$

and as q_{send} is polynomial in k, it suffices to recognize $\text{Adv}_{\mathcal{A}}^{\text{Game } 1}$ as negligible.

Game 2: Let $\mathsf{Collision}$ be the event that two values $x \neq x'$ are queried to H such that $H(x) = H(x')$. Whenever this event occurs, we abort and count this as success for the adversary. There are at most $\binom{q_{\text{ro}}}{2} \leq q_{\text{ro}}^2/2$ candidate pairs (x, x') and the range of H is of size 2^k. Hence

$$|\text{Adv}_{\mathcal{A}}^{\text{Game } 2} - \text{Adv}_{\mathcal{A}}^{\text{Game } 1}| \leq 2 \cdot \Pr[\mathsf{Collision}] \leq \frac{q_{\text{ro}}^2}{2^k}.$$

Game 3: Let Forge be the event that at least one of the messages m_i^{L}, m_i^{R}, T_i of at least one party U_i has been modified, and $\Pi_{i_0}^{s_0}$ terminates successfully, i. e., accepts a session key despite the message modification(s). Whenever this event occurs, we abort and count this as success for the adversary. As $\mathsf{Collision}$ did not occur, for Forge to take place, $\mathcal{A}$ has either guessed the tag

[2] In particular, q_{ro} depends on the number of random oracle calls in P and Q.

θ_i correctly (which happens with probability $\leq 2^{-k}$) or submitted a random oracle query of the form $H(\cdots||sk_{\mathsf{P},0})$. As transmitting θ_i to $\Pi_{i_0}^{s_0}$ requires a Send query, the freshness definition ensures that $\mathcal{A}$ submitted the query involving $sk_{\mathsf{P},0}$ to H before possibly corrupting U_{i_0} or any other users holding an instance partnered with $\Pi_{i_0}^{s_0}$. This enables us to turn $\mathcal{A}$ into an adversary $\mathcal{B}$ against protocol P:
The adversary $\mathcal{B}$ runs a simulation of $\mathcal{A}$ and answers oracle queries of $\mathcal{A}$ by means of its own oracles as detailed below.

- Random oracle H: $\mathcal{B}$ uses its own random oracle in the obvious way. Moreover, when receiving a direct random oracle query of $\mathcal{A}$, then $\mathcal{B}$ adds the last k bits of the query to an (initially empty) list L_{ro}.
- Send: Performing the necessary computations for executing protocol Q itself, $\mathcal{B}$ can simulate $\mathcal{A}$'s queries to Send perfectly with its own Send oracle—with the exception of the computation of the θ_i-values for those instances which in protocol P are partnered with $\Pi_{i_0}^{s_0}$. (For all other instances, $\mathcal{B}$ can simply reveal the group key established in protocol P.) To simulate θ_i-values of instances partnered with $\Pi_{i_0}^{s_0}$ in P, $\mathcal{B}$ replaces such values θ_i with independently chosen random values; note that $\theta_i = H(\ldots||U_i||\ldots)$, i. e., even with identical group keys in protocol P the random oracle queries for θ_i differ for each U_i. If $\mathcal{A}$ never submits a query of the form $H(\ldots||sk_{\mathsf{P},i})$ with $sk_{\mathsf{P},i}$ being held by an instance partnered with $\Pi_{i_0}^{s_0}$ in P, this simulation is perfect. If such a query is ever submitted by $\mathcal{A}$, we make no claims about $\mathcal{A}$'s further behavior.
- Reveal: If $\mathcal{A}$ should ever submit a query to this oracle that violates the freshness of $\Pi_{i_0}^{s_0}$ (which in principle could happen in the case where $\mathcal{B}$'s simulation of Send is incorrect), the simulation of $\mathcal{A}$ is aborted. Otherwise $\mathcal{B}$ can compute the required session key from its simulated executions of Q.
- Corrupt: If $\mathcal{A}$ should ever submit a query to Corrupt that violates the freshness of $\Pi_{i_0}^{s_0}$ (which in principle could happen in the case where $\mathcal{B}$'s simulation of Send is incorrect), the simulation of $\mathcal{A}$ is aborted. In all other cases $\mathcal{B}$ uses its own Corrupt oracle (and the data of the simulated Q executions, if Q should involve long-term secrets) in the obvious way.

At the latest when $\mathcal{A}$ queries Test, the adversary $\mathcal{B}$ ends the simulation of $\mathcal{A}$ and chooses a candidate session key among the entries of L_{ro} at random—if the event Forge occurred and $\mathcal{A}$ did not guess θ_i, then L_{ro} is non-empty. Then $\mathcal{B}$ queries Test with an instance randomly chosen among all instances partnered with $\Pi_{i_0}^{s_{i_0}}$ in P. This instance is fresh for protocol P, as the session identifier of the compiled protocol and of P are identical and the compiler does not modify partner identifiers. If the challenge obtained from Test is equal to the candidate session key selected in L_{ro}, $\mathcal{B}$ outputs 1, otherwise $\mathcal{B}$ outputs 0.

Using q_{ro} as upper bound for the number of entries in L_{ro} and q_{send} as upper bound for the number of instances possibly partnered with $\Pi_{i_0}^{s_0}$, we see that $\mathcal{B}$'s success probability is at least

$$\Pr[\mathsf{Succ}^{\mathsf{ke}}_{\mathcal{B}}] \geq \frac{1}{2} \cdot \left(\frac{1}{q_{\mathrm{ro}} \cdot q_{\mathrm{send}}} \cdot (\Pr[\mathsf{Forge}] - \frac{1}{2^k}) - \frac{1}{2^k} \right) + \frac{1}{2} \cdot \left(1 - \frac{1}{2^k} \right),$$

and hence

$$|\mathrm{Adv}^{\mathrm{Game\ 3}}_{\mathcal{A}} - \mathrm{Adv}^{\mathrm{Game\ 2}}_{\mathcal{A}}| \leq 2 \cdot q_{\mathrm{ro}} \cdot q_{\mathrm{send}} \cdot \left(\mathrm{Adv}^{\mathsf{ke}}_{\mathcal{B}} + \frac{1}{2^{k-1}} \right) + \frac{1}{2^{k-1}}.$$

Game 4: This game differs from Game 3 in the simulator's response in the final round. If the simulator has to output the message of instance $\Pi^{s_{i_0}}_{i_0}$ then the simulator replaces $sk^{\mathrm{R}}_{i_0}$ with a random value from $\{0,1\}^k$ in all computations of $\Pi^{s_{i_0}}_{i_0}$. To keep consistency, the same value has to be used for $sk^{\mathrm{L}}_{i_0+1}$ in the neighbored instance $\Pi^{s_{i_0+1}}_{i_0+1}$.

An adversary $\mathcal{A}$ that distinguishes Game 3 and Game 4 can be used as black box to construct an adversary against the 2-party protocol Q:

$\mathcal{C}$ runs a simulation of $\mathcal{A}$ and faithfully simulates all instances for $\mathcal{A}$ with the exception of $\Pi^{s_{i_0}}_{i_0}$ and $\Pi^{s_{i_0+1}}_{i_0+1}$. To answer $\mathcal{A}$'s oracle queries, $\mathcal{C}$ can proceed as follows.

- Random oracle H: To answer $\mathcal{A}$'s random oracle queries, $\mathcal{C}$ queries its own random oracle.
- $\mathsf{Corrupt}$: For instances other than $\Pi^{s_{i_0}}_{i_0}$ and $\Pi^{s_{i_0-1}}_{i_0-1}$, $\mathcal{C}$ has all information available as part of its simulation. For the latter two instances, $\mathcal{C}$ has the long-term secrets for protocol P available, and uses its own $\mathsf{Corrupt}$ oracle to obtain the long-term secret for protocol Q.
- Reveal: For instances other than $\Pi^{s_{i_0}}_{i_0}$ and $\Pi^{s_{i_0-1}}_{i_0-1}$, $\mathcal{C}$ simply computes the session key that needs to be revealed. Moreover, as $\Pi^{s_{i_0}}_{i_0}$ is $\mathcal{A}$'s 'Test instance' and partnered with $\Pi^{s_{i_0-1}}_{i_0-1}$, both of these instances must remain fresh and cannot be revealed. Therefore $\mathcal{C}$ can faithfully simulate all Reveal queries.
- Send: For instances other than $\Pi^{s_{i_0}}_{i_0}$ and $\Pi^{s_{i_0-1}}_{i_0-1}$, $\mathcal{C}$ simply performs the necessary computations. The same holds for these two instances in regard to computations for the protocol P. To compute in Round 1 the protocol messages of protocol Q, the adversary $\mathcal{C}$ uses its own $\mathsf{Execute}$ oracle and takes from the respective reply the corresponding messages of protocol Q for the answer for $\mathcal{A}$.

  To simulate the messages in the final round for instances $\Pi^{s_{i_0}}_{i_0}$ and $\Pi^{s_{i_0-1}}_{i_0-1}$, $\mathcal{C}$ queries its own Test oracle on instance $\Pi^{s_{i_0}}_{i_0}$ and uses this value $\widetilde{sk^{\mathrm{R}}_{i_0}}$ to compute T_{i_0} and T_{i_0+1}. To compute the tag θ_{i_0} and the tags for the instances partnered, $\mathcal{C}$ uses the messages for protocol Q obtained through the $\mathsf{Execute}$ query. The other elements involved in the random oracle query can be faithfully simulated by $\mathcal{C}$.
- Test: When $\mathcal{A}$ queries $\mathsf{Test}(U_{i_0}, s_{i_0})$, the queried instance must be fresh for $\mathcal{A}$ and hence is ensured to be fresh for $\mathcal{C}$, too. In this case, $\mathcal{C}$ chooses a random $b^{\mathsf{Test}} \in \{0,1\}$. If $b = 1$, $\mathcal{C}$ hands the answer $\widetilde{sk^{\mathrm{R}}_{i_0}}$ obtained as answer to its own Test query to $\mathcal{A}$, otherwise it hands a uniformly at random chosen k-bit string to $\mathcal{A}$.

Whenever $\mathcal{A}$ correctly identifies b^{Test}, $\mathcal{C}$ outputs 1, i. e., claims that the real session key was given, whenever $\mathcal{A}$ guesses incorrectly, $\mathcal{C}$ outputs 0. Denoting by b^{Q} the internal random bit of the Test oracle faced by $\mathcal{C}$, we obtain (with a slight abuse of notation) the following bound.

$$\left|\mathrm{Adv}_{\mathcal{A}}^{\text{Game 3}} - \mathrm{Adv}_{\mathcal{A}}^{\text{Game 4}}\right|$$

$$\begin{aligned}
&\leq \big|\Pr[1 \leftarrow \mathcal{A}^{b^{\mathsf{Test}}=1} \mid b^{\mathsf{Q}} = 1] - \Pr[1 \leftarrow \mathcal{A}^{b^{\mathsf{Test}}=0} \mid b^{\mathsf{Q}} = 1] - \\
&\quad \Pr[1 \leftarrow \mathcal{A}^{b^{\mathsf{Test}}=1} \mid b^{\mathsf{Q}} = 0] + \Pr[1 \leftarrow \mathcal{A}^{b^{\mathsf{Test}}=0} \mid b^{\mathsf{Q}} = 0]\big| \\
&= \big|\Pr[1 \leftarrow \mathcal{C} \mid b^{\mathsf{Q}} = 1] - \Pr[0 \leftarrow \mathcal{C} \mid b^{\mathsf{Q}} = 1] - \\
&\quad \Pr[1 \leftarrow \mathcal{C} \mid b^{\mathsf{Q}} = 0] + \Pr[0 \leftarrow \mathcal{C} \mid b^{\mathsf{Q}} = 0]\big| \\
&\leq \mathrm{Adv}_{\mathcal{C}}^{\mathsf{Q}} + \mathrm{Adv}_{\mathcal{C}}^{\mathsf{Q}} \\
&= 2 \cdot \mathrm{Adv}_{\mathcal{C}}^{\mathsf{Q}}.
\end{aligned}$$

In particular, under the assumption that protocol Q is fs-secure, the right-hand side of this inequality is negligible in k.

None of the partners of the adversary's Test-instance are allowed to be corrupted or to be revealed, because of the definition of freshness. Thereby, those instances were affected in Game 4 and use a random value as session key. Therefore, the adversary has only a probability of $\frac{1}{2}$ for guessing the internal random bit of Test, yielding

$$\mathrm{Adv}_{\mathcal{A}}^{\text{Game 4}} = 0.$$

Putting the probabilities together we recognize the adversary's advantage in the real model as negligible:

$$\mathrm{Adv}_{\mathcal{A}}^{\mathsf{fs\text{-}ke}} \leq 2q_{\text{send}}^2 \cdot \left(\frac{q_{\text{ro}}^2}{2^k} + 2q_{\text{ro}}q_{\text{send}} \cdot \left(\mathrm{Adv}_{\mathcal{B},\mathsf{P}}^{\mathsf{ke}} + \frac{1}{2^{k-1}} \right) + \frac{1}{2^{k-1}} + 2 \cdot \mathrm{Adv}_{\mathcal{C},\mathsf{Q}}^{\mathsf{fs\text{-}ke}} \right).$$

□

4 Conclusion

The presented compiler offers a generic technique to augment a secure group key establishment protocol which does not offer perfect forward secrecy—e. g., a key transport—to one that provides forward secrecy. For this, no modifications of the session identifier or of an underlying public key infrastructure are needed. In connection with one-round constructions for group key establishment, this seems an interesting option to design two-round protocols for group key establishment with perfect forward secrecy in a modular way.

References

[ABVS07] Abdalla, M., Bohli, J.-M., González Vasco, M.I., Steinwandt, R.: (Password) Authenticated Key Establishment: From 2-Party to Group. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 499–514. Springer, Heidelberg (2007)

[BCP01] Bresson, E., Chevassut, O., Pointcheval, D.: Provably Authenticated Group Diffie-Hellman Key Exchange - The Dynamic Case. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 290–309. Springer, Heidelberg (2001)

[BD94] Burmester, M., Desmedt, Y.: A Secure and Efficient Conference Key Distribution System (Extended Abstract). In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 275–286. Springer, Heidelberg (1995)

[Boh06] Bohli, J.-M.: A Framework for Robust Group Key Agreement. In: Gavrilova, M., Gervasi, O., Kumar, V., Tan, C.J.K., Taniar, D., Laganá, A., Mun, Y., Choo, H. (eds.) ICCSA 2006. LNCS, vol. 3982, pp. 355–364. Springer, Heidelberg (2006)

[BVS07] Bohli, J.-M., González Vasco, M.I., Steinwandt, R.: Secure group key establishment revisited. International Journal of Information Security 6(4), 243–254 (2007)

[GBNM10] Gorantla, M.C., Boyd, C., González Nieto, J.M., Manulis, M.: Generic One Round Group Key Exchange in the Standard Model. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 1–15. Springer, Heidelberg (2010)

[KLC+00] Ko, K.H., Lee, S.-J., Cheon, J.H., Han, J.W., Kang, J.-S., Park, C.: New Public-Key Cryptosystem Using Braid Groups. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 166–183. Springer, Heidelberg (2000)

[KY03] Katz, J., Yung, M.: Scalable Protocols for Authenticated Group Key Exchange. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 110–125. Springer, Heidelberg (2003)

[NPW11] Nam, J., Paik, J., Won, D.: A security weakness in Abdalla et al.'s generic construction of a group key exchange protocol. Information Sciences 181(1), 234–238 (2011)

Applicability of OR-Proof Techniques to Hierarchical Identity-Based Identification

Atsushi Fujioka[1], Taiichi Saito[2], and Keita Xagawa[1]

[1] NTT Secure Platform Laboratories,
3-9-11 Midori-cho, Musashino-shi, Tokyo 180-8585, Japan
{fujioka.atsushi,xagawa.keita}@lab.ntt.co.jp
[2] Tokyo Denki University,
5 Senju Asahi-cho, Adachi-ku, Tokyo 120-8551, Japan
taiichi@c.dendai.ac.jp

Abstract. We discuss the applicability of the well known OR-proof technique to hierarchical identity-based identification (HIBI) protocols for enhancing their security. We first describe formal security definitions for HIBI protocol not only in the adaptive hierarchical-identity setting but also in both "static" and "weak selective" hierarchical-identity settings. Next, we investigate whether the security enhancement transformations for identity-based identifications presented at ACNS 2012, which is based on the OR-proof technique, can be applied to HIBI protocols. We formally prove that several of these transformations are applicable to HIBI with slight modification. Curiously, the rest do not seem applicable, which stems from hierarchy and delegation. We also present a variant transformation and show that it can enhance the security of HIBI protocols in all three hierarchical-identity settings.

Keywords: hierarchical identity-based identification, OR-proof, impersonation under concurrent attacks.

1 Introduction

Identification is a protocol between a *prover* and a *verifier* through which the prover tries to convince the verifier of his/her identity. The security of identification protocols is defined by an experiment consisting of *learning* and *challenge phases*. In the learning phase, an adversary acts as many verifiers to gather much information and in the challenge phase, acts as a prover to impersonate an entity. A strong security model of identification protocols is formulated as a *model against impersonation under concurrent attacks* [2], in which an adversary is allowed to *concurrently* access entities who prove their identities, even in the challenge phase. On the other hand, in the security *model against impersonation under passive attacks* [2], an adversary is only allowed to eavesdrop on identification communications in the learning phase.

After the proposal of identity-based cryptography [13], identification in the identity-based setting, called *identity-based identification* (IBI), has also been

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 169–184, 2012.

investigated. In an IBI protocol, the existence of a *private key generator* (PKG) is assumed as well as in other identity-based cryptographic schemes. The PKG generates a secret key corresponding to the inputted identity of an entity, and gives the secret key to the entity. For IBI protocols, we can consider other types of attack models with respect to an adversary's selection of identities. Under *adaptive identity attacks* (i.e., in the adapt-id-imp-atk security model) [1,12], an adversary is allowed to adaptively ask identities to oracles. Under *static identity attacks* (i.e., in the stat-id-imp-atk security model) [12], an adversary declares, only at the beginning of the learning phase, the identities of all entities to be in queries or challenged. Under *weak selective identity attacks* (i.e., in the wsid-imp-atk security model) [14], an adversary requests secret keys of identities only at the beginning of the learning phase. Here, atk denotes a type of attack such that $\mathsf{atk} \in \{\mathsf{pa}, \mathsf{ca}\}$, and pa and ca mean *passive attack* and *concurrent attack*, respectively.

It is natural to extend IBI to identification in the hierarchical identity setting, called *hierarchical identity-based identification* (HIBI). In an HIBI protocol, the single PKG functionality of generating secret keys is divided into partial ones and the divided functionalities are delegated to multiple PKGs. If a PKG is assigned a hierarchical identity, $\mathsf{ID}^{(k-1)} = (I_1, \ldots, I_{k-1})$, and given a secret key, $sk_{\mathsf{ID}^{(k-1)}}$, corresponding to the hierarchical identity, then it can generate a secret key, $sk_{\mathsf{ID}^{(k)}}$, corresponding to a hierarchical identity, $\mathsf{ID}^{(k)} = (I_1, \ldots, I_k)$. We may omit the word "hierarchical" to indicate a hierarchical identity if its meaning is clear and denote a (hierarchical) identity by ID if we do not need to specify its hierarchy depth. Since IBI has three phases, HIBI also has three: Setup, Extract, and Identification.

Security for HIBI. The first security formulation for HIBI was given by Chin, Heng, and Goi [3] with their first proposal of an HIBI protocol. However, they formulated the passive and concurrent security only under adaptive hierarchical-identity attacks. Therefore, we can consider three types of attack models regarding an adversary's selection of hierarchical identities for HIBI protocols, as well as for IBI protocols. One is called *security against impersonation under adaptive hierarchical-identity attacks* (adapt-hid-imp-atk security), which is an extension of adapt-id-imp-atk security. The second one is called *security against impersonation under static hierarchical-identity attacks* (stat-hid-imp-atk security), which is an extension of stat-id-imp-atk security and in which an adversary requests secret keys of hierarchical identities only at the beginning of the learning phase. The third one is called *security against impersonation under weak selective hierarchical-identity attacks* (wshid-imp-atk security), which is an extension of wsid-imp-atk security and in which an adversary declares, only at the beginning of the learning phase, the hierarchical identities of all entities to be in queries or challenged.

Security Enhancement Transformations of IBI. It is well known that the OR-proof technique [5,4] enhances the security of not only standard identification

but also IBI protocols from passive security to the concurrent security [11,6]. Thus, we expect that the OR-proof technique can be also applied to HIBI protocols to enhance their security. Actually, HIBI protocols proposed in [7] utilize the OR-proof technique, and they are concurrently secure.

In [6], Fujioka, Saito, and Xagawa investigated OR-proof techniques for IBI protocols, which are formulated as three transformations, *DIsk*, *MI*, and *DPsk transformations* with double-key variants *DIdk* and *DPdk transformations*.[1] The authors proved that all the transformations can enhance an adapt-id-imp-pa (resp. wsid-imp-pa) secure IBI protocol to an adapt-id-imp-ca (resp. wsid-imp-ca) secure one, and that the DPsk, and DPdk transformations can enhance a stat-id-imp-pa secure IBI protocol to a stat-id-imp-ca secure one [6].

Our Contributions. We formally define *security against impersonation under static hierarchical-identity attacks* (stat-hid-imp-atk security) and *security against impersonation under weak selective hierarchical-identity attacks* (wshid-imp-atk security), along with the existing adapt-hid-imp-atk security, where atk denotes a type of attack such that $\mathsf{atk} \in \{\mathsf{pa}, \mathsf{ca}\}$.

Next we introduce two properties of HIBI protocols, which are extensions of the Σ^+*-type* and Σ^**-type* properties defined for IBI protocols [6]. The requirement of Σ^+-type is weaker than that of Σ^*-type in HIBI.

We examine whether the transformations discussed in [6] can be applied to HIBI protocols to enhance their security, and show the following points:

- To apply the DIdk transformation to a Σ^+-type HIBI protocol, we need a slight modification in choosing (imaginary) identities.
- The DIdk, MI, and DPdk transformations can convert an adapt-hid-imp-pa (resp. wshid-imp-pa) secure Σ^+-type HIBI protocol to an adapt-hid-imp-ca (resp. wshid-imp-ca) secure one.
- The DPdk transformation can convert a stat-hid-imp-pa secure Σ^*-type HIBI protocol to a stat-hid-imp-ca secure one.
- It seems difficult to enhance the passive security of an HIBI protocol in the static hierarchical-identity attack model by the DIdk and MI transformations.
- It seems difficult to enhance the passive security of an HIBI protocol in all the three hierarchical-identity attack models by the DIsk and DPsk transformations.

We also present a modified version of the DIdk transformation (named *mDIdk transformation*) and show that the mDIdk transformation can also convert a stat-hid-imp-pa secure HIBI protocol to a stat-hid-imp-ca secure one. While it is an open problem [6] whether there exists an OR-proof security enhancement

[1] DIsk, MI and DPsk stand for *Dual-Identity single-key*, *Master-Identity*, and *Double-Parameter single-key*, respectively. Also DIdk and DPsk stand for *Dual-Identity double-key* and *Double-Parameter double-key*, respectively [6]. The "double-parameter" means two master public keys, and "single/double-key" indicates the numbers of user's secret keys.

transformation based on a single master public key that converts a stat-id-imp-pa secure IBI protocol to a stat-id-imp-ca one, there exists an OR-proof security enhancement transformation based on a single master public key for stat-hid-imp-pa secure HIBI protocols.

For the summary and comparison, see Table 1.

Table 1. Applicability of Transformations

	HIBI [this paper]			IBI [6]		
	adapt-hid	stat-hid	wshid	adapt-id	stat-id	wsid
DIsk				✓		✓
DIdk	✓		✓	✓		✓
MI	✓		✓	✓		✓
DPsk				✓	✓	✓
DPdk	✓	✓	✓	✓	✓	✓
mDIdk	✓	✓	✓			

2 Definitions

We formally define hierarchical identity-based identification (HIBI) protocols and their security by following the formal definition of IBI protocols [1].

Hierarchical Identity-Based Identification. Two types of formal definitions for key generation in hierarchical identity-based cryptography have been proposed. One consists of three algorithms, *Root Setup, Lower-level Setup*, and *Extraction*, as in the Gentry-Silverberg hierarchical identity-based encryption (HIBE) scheme [9], and the other consists of two algorithms, *root-key-generation algorithm* and *node-key-generation algorithm*, as in the Horwitz-Lynn HIBE scheme [10]. The two types of definitions are essentially the same. We adopt the formal definition of HIBI protocols proposed by Chin et al. [3]. Note that their key generation is the former type, but we here describe it in the latter type.

Let $\mathsf{HIBI} = (\mathsf{SetUp}, \mathsf{KG}, \mathsf{P}, \mathsf{V})$ be an HIBI protocol, and κ denote the security parameter. In HIBI, SetUp is the root-key-generation algorithm that on input 1^κ outputs mpk and msk. To simplify notation, we set $sk_{\mathsf{ID}^{(0)}} = msk$. KG is the node-key-generation algorithm that on input $(mpk, sk_{\mathsf{ID}^{(k-1)}}, \mathsf{ID}^{(k)})$ outputs $sk_{\mathsf{ID}^{(k)}}$, P is the prover algorithm that takes mpk, ID, and sk_{ID} as inputs and interacts with V, and V is the verifier algorithm that takes mpk and ID as inputs, interacts with P, and finally outputs $dec \in \{accept, reject\}$, where $\mathsf{ID}^{(k-1)} = (I_1, \ldots, I_{k-1})$, $\mathsf{ID}^{(k)} = (I_1, \ldots, I_k)$. Thus, SetUp is used in Setup, KG is used in Extract, and P and V are used in Identification. Throughout this paper, we denote $\mathsf{pref}(\mathsf{ID})$ as the set of all prefixes of ID, i.e., $\mathsf{pref}(\mathsf{ID}) = \{(I_1), (I_1, I_2), \ldots, (I_1, \ldots, I_{k-1}), (I_1, \ldots, I_k)\}$ when $\mathsf{ID} = (I_1, \ldots, I_k)$. Note that $\mathsf{pref}(\mathsf{ID})$ includes ID.

We describe the formal definitions of the security of HIBI based on the following experiment $\mathbf{Exp}^{\mathsf{adapt\text{-}hid\text{-}imp\text{-}atk}}_{\mathsf{HIBI},\mathcal{I}}(\kappa)$ between a challenger and an impersonator $\mathcal{I} = (\mathsf{CV}, \mathsf{CP})$, where atk denotes a type of attack such that $\mathsf{atk} \in \{\mathsf{pa}, \mathsf{ca}\}$.

Experiment $\mathbf{Exp}^{\mathsf{adapt\text{-}hid\text{-}imp\text{-}atk}}_{\mathsf{HIBI},\mathcal{I}}(\kappa)$**:**

Setup Phase: The challenger obtains $(mpk,\ msk) \leftarrow \mathsf{SetUp}(1^\kappa)$ and initializes $HU,\ CU,\ TU,\ PS \leftarrow \emptyset$, where HU, CU, and TU denote the sets of honest users, corrupted users, and target users, respectively, and PS denotes the set of provers' sessions. The impersonator CV is given the security parameter 1^κ and the master public key mpk.

Learning Phase: The CV can ask queries to the Init, Corr, and Conv oracles when $\mathsf{atk} = \mathsf{pa}$ and also to Prov when $\mathsf{atk} = \mathsf{ca}$. Note that $\mathsf{ID} \notin HU \setminus TU$ means that ID is a target identity, a prefix of target identity, corrupted identity, or non-initiated identity.

- The oracle Init receives input $\mathsf{ID}^{(k)}$. If $\mathsf{ID}^{(k)} \in HU \cup CU \cup TU$, then it returns $\perp$. Otherwise, it computes k secret keys $sk_{\mathsf{ID}^{(1)}}, \ldots, sk_{\mathsf{ID}^{(k)}}$ by running $sk_{\mathsf{ID}^{(i)}} \leftarrow \mathsf{KG}(mpk,\ sk_{\mathsf{ID}^{(i-1)}},\ \mathsf{ID}^{(i)})$ if all prefixes of $\mathsf{ID}^{(k)}$ are not in HU, adds $\mathsf{pref}(\mathsf{ID}^{(k)})$ to HU, and provides the CV with $\mathsf{ID}^{(k)}$. If some prefixes of $\mathsf{ID}^{(k)}$ are in HU and if $\mathsf{ID}^{(j)}$ is the longest one in the prefixes, it computes $k-j$ secret keys $sk_{\mathsf{ID}^{(j+1)}}, \ldots, sk_{\mathsf{ID}^{(k)}}$.
- The oracle Corr receives input ID. If $\mathsf{ID} \notin HU \setminus TU$, then it returns $\perp$. Otherwise, it deletes all ID′s in HU such that $\mathsf{ID} \in \mathsf{pref}(\mathsf{ID}')$, adds them to CU, and returns sk_{ID} to the CV.
- The oracle Conv receives input ID. If $\mathsf{ID} \notin HU$, then it returns $\perp$. Otherwise it returns a transcript of a transaction between the prover with identity ID and a verifier.
- (only when $\mathsf{atk} = \mathsf{ca}$) The oracle Prov receives inputs ID, s, and M_{in}. If $\mathsf{ID} \notin HU \setminus TU$, then it returns $\perp$. If $(\mathsf{ID},\ s) \notin PS$, then it adds $(\mathsf{ID},\ s)$ to PS, selects a random coin ρ, and sets a state of the prover $st_{\mathsf{P}}[(\mathsf{ID}, s)] \leftarrow (mpk,\ sk_{\mathsf{ID}},\ \rho)$. Next, it obtains $(M_{out},\ st_{\mathsf{P}}[(\mathsf{ID}, s)]) \leftarrow \mathsf{P}(M_{in},\ st_{\mathsf{P}}[(\mathsf{ID}, s)])$. Finally, it returns M_{out}. Note that we require that $\mathsf{ID} \notin HU \setminus TU$ since we do not consider man-in-the-middle attacks [8] in this paper.

Challenge Phase: The CV outputs a target identity ID^* and state information st_{CP}. If ID^* is not in HU, then the challenger outputs *reject* and halts. Otherwise, the challenger sets $TU \leftarrow \mathsf{pref}(\mathsf{ID}^*)$ and gives st_{CP} to CP. CP can ask queries to Init, Corr, and Conv, (and Prov when $\mathsf{atk} = \mathsf{ca}$) as in the learning phase. Finally, the challenger obtains $(tr,\ dec) \leftarrow \mathbf{Run}[\mathsf{CP}(st_{\mathsf{CP}})^{\textsc{Init},\textsc{Corr},\textsc{Conv}(,\textsc{Prov})} \leftrightarrow \mathsf{V}(mpk,\ \mathsf{ID}^*)]$ and outputs dec.

In these experiments, the impersonator is allowed to obtain a secret key of an adaptively chosen (hierarchical) identity and a transcript of a transaction between the prover of an adaptively chosen (hierarchical) identity and a verifier. In the case of $\mathsf{atk} = \mathsf{ca}$, the Prov oracle allows multiple sessions at the same time.

Definition 2.1. *Let* $\mathsf{HIBI} = (\mathsf{SetUp}, \mathsf{KG}, \mathsf{P}, \mathsf{V})$ *be an HIBI protocol and* $\mathcal{I} = (\mathsf{CV}, \mathsf{CP})$ *an impersonator. Let* κ *be a security parameter. The advantage of* $\mathcal{I}$ *in attacking* HIBI *is defined by*

$$\mathbf{Adv}^{\mathsf{adapt\text{-}hid\text{-}imp\text{-}atk}}_{\mathsf{HIBI},\mathcal{I}}(\kappa) := \Pr\left[\ \mathbf{Exp}^{\mathsf{adapt\text{-}hid\text{-}imp\text{-}atk}}_{\mathsf{HIBI},\mathcal{I}}(\kappa) = \mathit{accept}\ \right].$$

We say that HIBI *is* secure against impersonation under adaptive hierarchical-identity and concurrent attacks (adapt-hid-imp-ca secure) *if* $\mathbf{Adv}^{\mathsf{adapt\text{-}hid\text{-}imp\text{-}ca}}_{\mathsf{HIBI},\mathcal{I}}(\kappa)$ *is negligible for every polynomial-time* $\mathcal{I}$ *and is* secure against impersonation under adaptive hierarchical-identity and passive attacks (adapt-hid-imp-pa secure) *if* $\mathbf{Adv}^{\mathsf{adapt\text{-}hid\text{-}imp\text{-}pa}}_{\mathsf{HIBI},\mathcal{I}}(\kappa)$ *is negligible for every polynomial-time* $\mathcal{I}$.

Static and Weak Selective Hierarchical-Identity Attack Models. Following Rückert [12] and Yang et al. [14], we describe two other security definitions, which are weaker than the adapt-hid-imp-atk security, of HIBI based on the following experiments, $\mathbf{Exp}^{\mathsf{stat\text{-}hid\text{-}imp\text{-}atk}}_{\mathsf{HIBI},\mathcal{I}}(\kappa)$ and $\mathbf{Exp}^{\mathsf{wshid\text{-}imp\text{-}atk}}_{\mathsf{HIBI},\mathcal{I}}(\kappa)$ ($\mathsf{atk} \in \{\mathsf{pa}, \mathsf{ca}\}$), between a challenger and impersonator $\mathcal{I} = (\mathsf{CV}, \mathsf{CP})$.

Experiment $\mathbf{Exp}^{\mathsf{stat\text{-}hid\text{-}imp\text{-}atk}}_{\mathsf{HIBI},\mathcal{I}}(\kappa)$**:**

- **Setup Phase:** At the beginning of this phase, the CV on input 1^κ issues a single corrupt query $(\mathsf{ID}_1, \ldots, \mathsf{ID}_t)$ to the challenger before receiving the master public key. The challenger is given the security parameter 1^κ, obtains $(mpk, msk) \leftarrow \mathsf{SetUp}(1^\kappa)$, and computes $sk_{\mathsf{ID}_i} \leftarrow \mathsf{KG}(mpk, msk, \mathsf{ID}_i)$ $(1 \le i \le t)$. It sets $CU \leftarrow \{\mathsf{ID}_1, \mathsf{ID}_2, \ldots, \mathsf{ID}_t\}$ and then returns $(sk_{\mathsf{ID}_1}, \ldots, sk_{\mathsf{ID}_t})$ to the CV. The challenger initializes HU, TU, and $PS \leftarrow \emptyset$. The CV is given the master public key mpk.
- **Learning and Challenge Phases:** The learning and challenge phases are defined the same as those in experiment $\mathbf{Exp}^{\mathsf{adapt\text{-}hid\text{-}imp\text{-}atk}}_{\mathsf{HIBI},\mathcal{I}}(\kappa)$, except that impersonator $\mathcal{I}$ is not allowed additional queries to Corr during these phases.

Experiment $\mathbf{Exp}^{\mathsf{wshid\text{-}imp\text{-}atk}}_{\mathsf{HIBI},\mathcal{I}}(\kappa)$**:**

- **Setup Phase:** At the beginning of this phase, the CV on input 1^κ issues a single initialization query $(\mathsf{ID}_1, \ldots, \mathsf{ID}_t)$ to the challenger before receiving the master public key. The challenger is given the security parameter 1^κ and obtains $(mpk, msk) \leftarrow \mathsf{SetUp}(1^\kappa)$. It sets $HU \leftarrow \bigcup_{i=1}^{t} \mathsf{pref}(\mathsf{ID}_1)$ and provides the CV with $(\mathsf{ID}_1, \ldots, \mathsf{ID}_t)$. The challenger initializes CU, TU, and $PS \leftarrow \emptyset$. The CV is given the master public key mpk.
- **Learning and Challenge Phases:** The learning and challenge phases are defined the same as those in experiment $\mathbf{Exp}^{\mathsf{adapt\text{-}hid\text{-}imp\text{-}atk}}_{\mathsf{HIBI},\mathcal{I}}(\kappa)$, except that $\mathcal{I}$ is not allowed additional queries to the Init oracle during these phases.

In the stat-hid-imp-atk experiment, the impersonator has to choose all (hierarchical) identities that it wants to corrupt at the beginning of the experiment. After that, it is allowed to access oracles except for Corr. In the wshid-imp-atk experiment, the impersonator has to select all (hierarchical) identities that it

wants to initialize at the beginning of the experiment. Then, it is allowed to send queries of only the (hierarchical) identities chosen at the beginning,

Let $\mathsf{HIBI} = (\mathsf{SetUp}, \mathsf{KG}, \mathsf{P}, \mathsf{V})$ be an HIBI protocol and $\mathcal{I} = (\mathsf{CV}, \mathsf{CP})$ an impersonator. Let κ be a security parameter. The advantages of $\mathcal{I}$ in attacking HIBI are defined as

$$\mathbf{Adv}^{\mathsf{stat\text{-}hid\text{-}imp\text{-}atk}}_{\mathsf{HIBI},\mathcal{I}}(\kappa) := \Pr\left[\ \mathbf{Exp}^{\mathsf{stat\text{-}hid\text{-}imp\text{-}atk}}_{\mathsf{HIBI},\mathcal{I}}(\kappa) = \mathit{accept}\ \right] \text{ and}$$

$$\mathbf{Adv}^{\mathsf{wshid\text{-}imp\text{-}atk}}_{\mathsf{HIBI},\mathcal{I}}(\kappa) := \Pr\left[\ \mathbf{Exp}^{\mathsf{wshid\text{-}imp\text{-}atk}}_{\mathsf{HIBI},\mathcal{I}}(\kappa) = \mathit{accept}\ \right].$$

We say that HIBI is *secure against impersonation under static (resp. weak selective) hierarchical-identity and concurrent attacks (*$\mathsf{stat\text{-}hid\text{-}imp\text{-}ca}$ *(resp.* $\mathsf{wshid\text{-}imp\text{-}ca}$*) secure)* if $\mathbf{Adv}^{\mathsf{stat\text{-}hid\text{-}imp\text{-}ca}}_{\mathsf{HIBI},\mathcal{I}}(\kappa)$ (resp. $\mathbf{Adv}^{\mathsf{wshid\text{-}imp\text{-}ca}}_{\mathsf{HIBI},\mathcal{I}}(\kappa)$) is negligible for every polynomial-time $\mathcal{I}$, and is *secure against impersonation under static (resp. weak selective) hierarchical-identity and passive attacks (*$\mathsf{stat\text{-}hid\text{-}imp\text{-}pa}$ *(resp.* $\mathsf{wshid\text{-}imp\text{-}ca}$*) secure)* if $\mathbf{Adv}^{\mathsf{stat\text{-}hid\text{-}imp\text{-}pa}}_{\mathsf{HIBI},\mathcal{I}}(\kappa)$ (resp. $\mathbf{Adv}^{\mathsf{wshid\text{-}imp\text{-}pa}}_{\mathsf{HIBI},\mathcal{I}}(\kappa)$) is negligible for every polynomial-time $\mathcal{I}$.

Σ^+- and Σ^*-Type HIBI Protocols. We define two analogues of Σ-type IBI-protocols [6] in the context of HIBI protocols. Let $\mathsf{HIBI} = (\mathsf{SetUp}, \mathsf{KG}, \mathsf{P}, \mathsf{V})$ be an HIBI protocol. Suppose that P and V interact by using four probabilistic polynomial-time algorithms $(\Sigma_{\mathsf{hibi\text{-}com}}, \Sigma_{\mathsf{hibi\text{-}ch}}, \Sigma_{\mathsf{hibi\text{-}res}}, \Sigma_{\mathsf{hibi\text{-}vrfy}})$ as follows:

$\mathsf{P} \to \mathsf{V}$: P computes $(a, st) \leftarrow \Sigma_{\mathsf{hibi\text{-}com}}(mpk, \mathsf{ID}, sk_{\mathsf{ID}})$ and sends a to V.
$\mathsf{V} \to \mathsf{P}$: V computes $c \leftarrow \Sigma_{\mathsf{hibi\text{-}ch}}(mpk, \mathsf{ID})$ and sends c to P.
$\mathsf{P} \to \mathsf{V}$: P computes $z \leftarrow \Sigma_{\mathsf{hibi\text{-}res}}(mpk, \mathsf{ID}, sk_{\mathsf{ID}}, a, c, st)$ and sends z to V.
V: V computes $dec \leftarrow \Sigma_{\mathsf{hibi\text{-}vrfy}}(mpk, \mathsf{ID}, a, c, z)$ and outputs $dec \in \{\mathit{accept}, \mathit{reject}\}$.

We call this type of three-move HIBI protocol *canonical* [2]. We also call an HIBI protocol HIBI *Σ^+-type* if it is canonical and satisfies the following three properties: *special zero-knowledge*, *special soundness*, and *special challenge*:

Special Zero-Knowledge: We can obtain an accepting transcript from a challenge c, mpk, and ID. That is, there is a probabilistic polynomial-time algorithm $\Sigma_{\mathsf{hibi\text{-}sim}}$ that takes on input mpk, ID, and c such that $c \leftarrow \Sigma_{\mathsf{hibi\text{-}ch}}(mpk, \mathsf{ID})$ and outputs (a, z) such that $\mathit{accept} = \Sigma_{\mathsf{hibi\text{-}vrfy}}(mpk, \mathsf{ID}, a, c, z)$. The distribution of transcripts generated by $\Sigma_{\mathsf{hibi\text{-}ch}}$ and $\Sigma_{\mathsf{hibi\text{-}sim}}$ is indistinguishable from that of real transcripts.

Special Soundness: We can compute the user secret key sk_{ID} for an identity ID from mpk, ID, and two accepting transcripts (a, c, z) and $(a, \tilde{c}, \tilde{z})$ such that $c \neq \tilde{c}$. That is, there is a probabilistic polynomial-time algorithm $\Sigma_{\mathsf{hibi\text{-}ext}}$ that takes as input mpk, ID, and two transcripts (a, c, z) and $(a, \tilde{c}, \tilde{z})$ satisfying $\mathit{accept} = \Sigma_{\mathsf{hibi\text{-}vrfy}}(mpk, \mathsf{ID}, a, c, z) = \Sigma_{\mathsf{hibi\text{-}vrfy}}(mpk, \mathsf{ID}, a, \tilde{c}, \tilde{z})$ and $c \neq \tilde{c}$, and outputs sk_{ID}.

Special Challenge: $\Sigma_{\mathsf{hibi\text{-}ch}}$ depends only on mpk, not on (mpk, ID), and the output c is uniformly distributed over a commutative group $\mathbb{G}$. In addition, the

group operation + is computable in polynomial time, and $\mathbb{G}$ is determined only by mpk, not by (mpk, ID). That is, there is a probabilistic polynomial time algorithm $\Sigma^{+}_{\mathsf{hibi\text{-}ch}}$ such that it takes as input mpk (without ID) and outputs c, and c is uniformly distributed over $\mathbb{G}$.

We call an HIBI protocol HIBI Σ^{*}-*type* if the special challenge property is replaced with the following property:

Strongly Special Challenge: $\Sigma_{\mathsf{hibi\text{-}ch}}$ depends only on 1^{κ}, not on mpk, and the output c is uniformly distributed over $\mathbb{G}$. In addition, + is computable in polynomial time, and $\mathbb{G}$ is determined only by 1^{κ}, not by mpk. That is, there is a probabilistic polynomial time algorithm $\Sigma^{*}_{\mathsf{hibi\text{-}ch}}$ such that it takes as input 1^{κ} (not mpk) and outputs c, and c is uniformly distributed over $\mathbb{G}$.

3 Security Enhancement Transformations

In this section, we describe three security enhancement transformations, DIdk, MI, and DPdk, and show that they can enhance the passive security of an HIBI protocol to the concurrent security in both adaptive and weak selective hierarchical-identity settings. Next, we present a modified version of the DIdk transformation (mDIdk) and show that the DPdk and mDIdk transformation can enhance a $\mathsf{stat\text{-}hid\text{-}imp\text{-}pa}$ secure HIBI protocol to a $\mathsf{stat\text{-}hid\text{-}imp\text{-}ca}$ secure one. In addition, we discuss the difficulty in enhancing security in the static hierarchical-identity models by the DIdk, DIsk, MI, DPsk, or a single key variant of the mDIdk (mDIsk) transformation.

Let $\mathsf{HIBI}' = (\mathsf{SetUp}', \mathsf{KG}', \mathsf{P}', \mathsf{V}')$ be a Σ^{+}-type (Σ^{*}-type) HIBI protocol in which $(\mathsf{P}', \mathsf{V}')$ use four probabilistic polynomial time algorithms $\Sigma_{\mathsf{hibi\text{-}com}}$, $\Sigma^{+}_{\mathsf{hibi\text{-}ch}}$ ($\Sigma^{*}_{\mathsf{hibi\text{-}ch}}$), $\Sigma_{\mathsf{hibi\text{-}res}}$, and $\Sigma_{\mathsf{hibi\text{-}vrfy}}$, and have the special zero-knowledge property with a probabilistic polynomial time algorithm $\Sigma_{\mathsf{hibi\text{-}sim}}$ and the special soundness with a probabilistic polynomial time algorithm $\Sigma_{\mathsf{hibi\text{-}ext}}$.

3.1 Dual-Identity Double-Key Transformation

We show a security enhancement transformation based on the OR-proof technique, applicable to the Σ^{+}-type HIBI protocol. We call this transformation *DIdk transformation*.

Let $i.\mathsf{ID}^{(k)}$ denote $(i||I_1, I_2, \ldots, I_k)$ when $\mathsf{ID}^{(k)} = (I_1, \ldots, I_k)$. In an HIBI protocol generated by the DIdk transformation, an entity of an identity ID is given two secret keys corresponding to imaginary (hierarchical) identities $0.\mathsf{ID}$ and $1.\mathsf{ID}$ for the underlying HIBI protocol, and shows his/her identity by proving that the imaginary (hierarchical) identity is either $0.\mathsf{ID}$ or $1.\mathsf{ID}$.

We describe an HIBI protocol $\mathsf{HIBI} = (\mathsf{SetUp}, \mathsf{KG}, \mathsf{P}, \mathsf{V})$ produced by applying the DIdk transformation to HIBI' in Fig. 1. Note that in the EXTRACT algorithm in this figure, we let $sk_{\mathsf{ID}^{(0)}} = (msk', msk')$.

Due to the special challenge property, c is an element in $\mathbb{G}$ determined by mpk', so are c_0 and c_1 since $c = c_0 + c_1$ and the operation + is defined in $\mathbb{G}$. Then, c_0 and c_1 are possible challenges under mpk'.

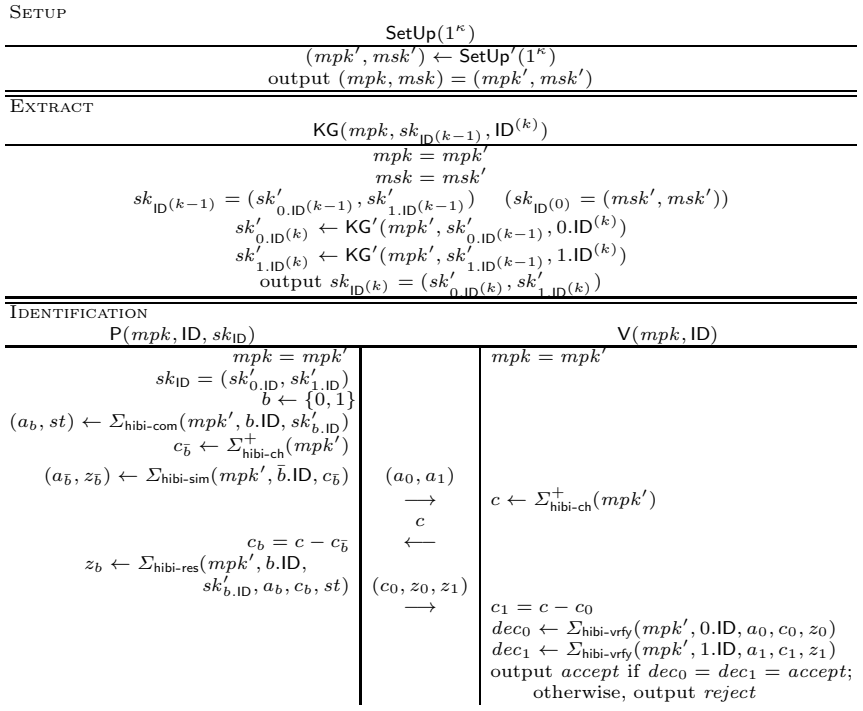


Fig. 1. DIdk Transformation

It is easy to have a variant of the DIdk transformation (DIsk transformation) such that each entity is given only either secret key of identities 0.ID or 1.ID, and the entity shows that it has either the secret key of 0.ID or 1.ID. However, it seems difficult to enhance the passive security of an HIBI protocol in any hierarchical-identity attack model with this variant.

3.2 Master-Identity Transformation

We show another security enhancement transformation based on the OR-proof technique, applicable to the Σ^+-type HIBI protocol. We call this transformation *MI transformation*.

In an HIBI protocol generated by the MI transformation, an entity with an identity ID is simply given a secret key corresponding to the identity ID for the underlying HIBI protocol, and the entity of the identity ID proves that his/her identity is either ID or an (imaginary) master identity.

We describe an HIBI protocol $\mathsf{HIBI} = (\mathsf{SetUp}, \mathsf{KG}, \mathsf{P}, \mathsf{V})$ produced by applying the MI transformation to HIBI' in Fig. 2. In the EXTRACT algorithm in this figure, we let $sk_{\mathsf{ID}^{(0)}} = msk'$.

Due to the special challenge property, c is an element in $\mathbb{G}$ determined by mpk', so are c_0 and c_1 since $c = c_0 + c_1$ and the operation $+$ is defined in $\mathbb{G}$. Then, c_0 and c_1 are possible challenges under mpk'.

SETUP

$\mathsf{SetUp}(1^\kappa)$

$(mpk', msk') \leftarrow \mathsf{SetUp}'(1^\kappa)$
choose a master identity ID_{master} from the set of identities of depth 1
output $(mpk, msk) = ((mpk', \mathsf{ID}_{master}), msk')$

EXTRACT

$\mathsf{KG}(mpk, sk_{\mathsf{ID}^{(k-1)}}, \mathsf{ID}^{(k)})$

$mpk = (mpk', \mathsf{ID}_{master})$
output $\perp$ if $\mathsf{ID}_{master} \in \mathsf{pref}(\mathsf{ID}^{(k)})$
$sk_{\mathsf{ID}^{(k-1)}} = sk'_{\mathsf{ID}^{(k-1)}} \quad (sk_{\mathsf{ID}^{(0)}} = msk')$
$sk'_{\mathsf{ID}^{(k)}} \leftarrow \mathsf{KG}'(mpk', sk'_{\mathsf{ID}^{(k-1)}}, \mathsf{ID}^{(k)})$
output $sk_{\mathsf{ID}^{(k)}} = sk'_{\mathsf{ID}^{(k)}}$

IDENTIFICATION

$\mathsf{P}(mpk, \mathsf{ID}, sk_{\mathsf{ID}})$		$\mathsf{V}(mpk, \mathsf{ID})$
$mpk = (mpk', \mathsf{ID}_{master})$		$mpk = (mpk', \mathsf{ID}_{master})$
$sk_{\mathsf{ID}} = sk'_{\mathsf{ID}}$		
$(a_0, st) \leftarrow \Sigma_{\mathsf{hibi\text{-}com}}(mpk', \mathsf{ID}, sk'_{\mathsf{ID}})$		
$c_1 \leftarrow \Sigma^+_{\mathsf{hibi\text{-}ch}}(mpk')$		
$(a_1, z_1) \leftarrow \Sigma_{\mathsf{hibi\text{-}sim}}(mpk', \mathsf{ID}_{master}, c_1)$	(a_0, a_1) $\longrightarrow$	$c \leftarrow \Sigma^+_{\mathsf{hibi\text{-}ch}}(mpk')$
	c $\longleftarrow$	
$c_0 = c - c_1$		
$z_0 \leftarrow \Sigma_{\mathsf{hibi\text{-}res}}(mpk', \mathsf{ID}, sk'_{\mathsf{ID}}, a_0, c_0, st)$	(c_0, z_0, z_1) $\longrightarrow$	$c_1 = c - c_0$ $dec_0 \leftarrow \Sigma_{\mathsf{hibi\text{-}vrfy}}(mpk', \mathsf{ID}, a_0, c_0, z_0)$ $dec_1 \leftarrow \Sigma_{\mathsf{hibi\text{-}vrfy}}(mpk', \mathsf{ID}_{master}, a_1, c_1, z_1)$ output *accept* if $dec_0 = dec_1 = accept$s; otherwise, output *reject*

Fig. 2. MI Transformation

Here, ID_{master} is randomly chosen from the set of identities but does not coincide with any identities of real entities. It is clear that an impersonator should not be allowed to obtain the secret key of ID_{master}. In the construction of KG, $\mathsf{KG}(mpk, sk_{\mathsf{ID}^{(k-1)}}, \mathsf{ID}^{(k)})$ outputs $\perp$ if $\mathsf{ID}_{master} \in \mathsf{pref}(\mathsf{ID}^{(k)})$. This means that the space of all possible identities of entities does not include ID_{master}. Note that since ID_{master} is randomly chosen from the set of identities, it is a hierarchical identity of depth 1.

3.3 Double-Parameter Double-Key Transformation

We show the other security enhancement transformation based on the OR-proof technique, applicable only to the Σ^*-type HIBI protocol, not to the Σ^+-type. We call this transformation *DPdk transformation*.

In an HIBI protocol generated by the DPdk transformation, an entity of an identity ID is given secret keys corresponding to the identity based on both master public keys in the underlying HIBI protocol and proves that his/her (hierarchical) identity is ID under either master public key.

We describe an HIBI protocol $\mathsf{HIBI} = (\mathsf{SetUp}, \mathsf{KG}, \mathsf{P}, \mathsf{V})$ produced by applying the DIdk transformation to HIBI' in Fig. 3. In the EXTRACT algorithm in this figure, we let $sk_{\mathsf{ID}^{(0)}} = (msk'_0, msk'_1)$.

Due to the strongly special challenge property, c is an element in $\mathbb{G}$ determined only by 1^κ, so are c_0 and c_1 since $c = c_0 + c_1$ and the operation $+$ is defined in $\mathbb{G}$. Therefore, c_0 and c_1 are possible challenges under mpk'_0 and mpk'_1, respectively.

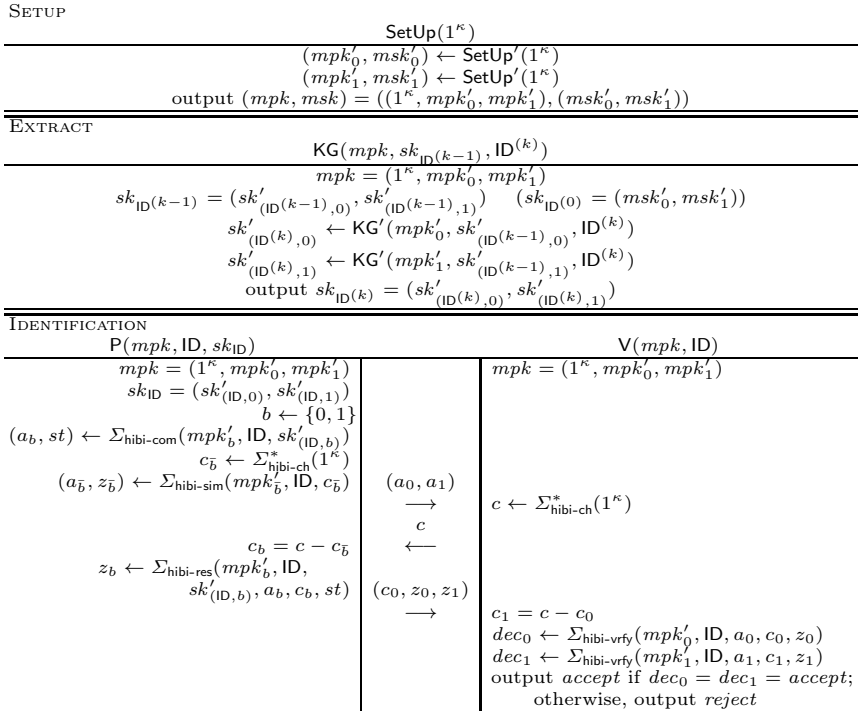


Fig. 3. DPdk Transformation

Although the DPdk transformation requires two master public keys and is less efficient than the DIdk and MI transformations, the transformation can enhance the security of a Σ^*-type HIBI protocol even in the static hierarchical-identity attack model.

It is easy to have a variant of the DPdk transformation (DPsk transformation) such that each entity is given either a secret key based on mpk'_0 or mpk'_1, and the entity shows that it has either the secret key in mpk'_0 or mpk'_1. However, it seems difficult to enhance the passive security of HIBI protocols in any hierarchical-identity attack mode with this variant.

3.4 Modified Dual-Identity Double-Key Transformation

We modify the DIdk transformation and call the modified transformation *mDIdk transformation*, which is also applicable to the Σ^+-type HIBI protocol.

We have seen in the previous subsection that, in the DIdk transformation, a user of identity $\mathsf{ID}^{(k)}$ $(= (I_1, I_2, \ldots, I_k))$ is given two secret keys corresponding to imaginary identities $0.\mathsf{ID}^{(k)} = (0||I_1, I_2, \ldots, I_k)$ and $1.\mathsf{ID}^{(k)} = (1||I_1, I_2, \ldots, I_k)$ in the underlying HIBI protocol. We modify the DIdk transformation in a way that a user of identity $\mathsf{ID}^{(k)}$ is given two secret keys corresponding to imaginary

identities $(0, I_1, I_2, \ldots, I_k)$ and $(1, I_1, I_2, \ldots, I_k)$ in the underlying HIBI protocol. We then see that in this mDIdk transformation, if the underlying HIBI protocol is ℓ-level (i.e., the maximum length of hierarchical identities is ℓ), the resulting HIBI protocol is $(\ell - 1)$-level. We let $i \circ \mathsf{ID}^{(k)}$ denote $(i, I_1, I_2, \ldots, I_k)$ when $\mathsf{ID}^{(k)} = (I_1, \ldots, I_k)$.

We describe an HIBI protocol, $\mathsf{HIBI} = (\mathsf{SetUp}, \mathsf{KG}, \mathsf{P}, \mathsf{V})$ produced by applying the mDIdk transformation to HIBI' in Fig. 4. In the EXTRACT algorithm in this figure, we let $sk_{\mathsf{ID}^{(0)}} = (sk'_{(0)}, sk'_{(1)})$.

SETUP

$\mathsf{SetUp}(1^\kappa)$

$(mpk', msk') \leftarrow \mathsf{SetUp}'(1^\kappa)$
$sk'_{(0)} \leftarrow \mathsf{KG}'(mpk', msk', (0))$
$sk'_{(1)} \leftarrow \mathsf{KG}'(mpk', msk', (1))$
output $(mpk, msk) = (mpk', (sk'_{(0)}, sk'_{(1)}))$

EXTRACT

$\mathsf{KG}(mpk, sk_{\mathsf{ID}(k-1)}, \mathsf{ID}^{(k)})$

$mpk = mpk'$
$sk_{\mathsf{ID}(k-1)} = (sk'_{0\circ\mathsf{ID}(k-1)}, sk'_{1\circ\mathsf{ID}(k-1)})$ $\quad (sk_{\mathsf{ID}^{(0)}} = (sk'_{(0)}, sk'_{(1)}))$
$sk'_{0\circ\mathsf{ID}^{(k)}} \leftarrow \mathsf{KG}'(mpk', sk'_{0\circ\mathsf{ID}^{(k-1)}}, 0 \circ \mathsf{ID}^{(k)})$
$sk'_{1\circ\mathsf{ID}^{(k)}} \leftarrow \mathsf{KG}'(mpk', sk'_{1\circ\mathsf{ID}^{(k-1)}}, 1 \circ \mathsf{ID}^{(k)})$
output $sk_{\mathsf{ID}^{(k)}} = (sk'_{0\circ\mathsf{ID}^{(k)}}, sk'_{1\circ\mathsf{ID}^{(k)}})$

IDENTIFICATION

$\mathsf{P}(mpk, \mathsf{ID}, sk_{\mathsf{ID}})$		$\mathsf{V}(mpk, \mathsf{ID})$
$mpk = mpk'$		$mpk = mpk'$
$sk_{\mathsf{ID}} = (sk'_{0\circ\mathsf{ID}}, sk'_{1\circ\mathsf{ID}})$		
$b \leftarrow \{0, 1\}$		
$(a_b, st) \leftarrow \Sigma_{\mathsf{hibi\text{-}com}}(mpk', b \circ \mathsf{ID}, sk'_{b\circ\mathsf{ID}})$		
$c_{\bar{b}} \leftarrow \Sigma^{+}_{\mathsf{hibi\text{-}ch}}(mpk')$		
$(a_{\bar{b}}, z_{\bar{b}}) \leftarrow \Sigma_{\mathsf{hibi\text{-}sim}}(mpk', \bar{b} \circ \mathsf{ID}, c_{\bar{b}})$	(a_0, a_1)	
	$\longrightarrow$	$c \leftarrow \Sigma^{+}_{\mathsf{hibi\text{-}ch}}(mpk')$
	c	
$c_b = c - c_{\bar{b}}$	$\longleftarrow$	
$z_b \leftarrow \Sigma_{\mathsf{hibi\text{-}res}}(mpk', b \circ \mathsf{ID},$		
$sk'_{b\circ\mathsf{ID}}, a_b, c_b, st)$	(c_0, z_0, z_1)	
	$\longrightarrow$	$c_1 = c - c_0$
		$dec_0 \leftarrow \Sigma_{\mathsf{hibi\text{-}vrfy}}(mpk', 0 \circ \mathsf{ID}, a_0, c_0, z_0)$
		$dec_1 \leftarrow \Sigma_{\mathsf{hibi\text{-}vrfy}}(mpk', 1 \circ \mathsf{ID}, a_1, c_1, z_1)$
		output *accept* if $dec_0 = dec_1 = accept$; otherwise, output *reject*

Fig. 4. mDIdk Transformation

Due to the special challenge property, c is an element in $\mathbb{G}$ determined by mpk', so are c_0 and c_1 since $c = c_0 + c_1$ and the operation $+$ is defined in $\mathbb{G}$. Therefore, c_0 and c_1 are possible challenges under mpk'.

It is easy to have a variant of the mDIdk transformation (mDIsk transformation) such that each entity is given only either secret key of identities $0 \circ \mathsf{ID}$ or $1 \circ \mathsf{ID}$, and the entity shows that it has either the secret key of $0 \circ \mathsf{ID}$ or $1 \circ \mathsf{ID}$. However, it seems difficult to enhance the passive security of HIBI protocols in any hierarchical-identity attack model with this variant.

3.5 Security of DIdk, MI, DPdk, and mDIdk Transformations

We prove that the DIdk, MI, DPdk, and mDIdk transformations can convert an adapt-hid-imp-pa secure HIBI protocol to an adapt-hid-imp-ca secure one. We construct an adapt-hid-imp-pa impersonator, $\mathcal{I}'$, from an adapt-hid-imp-ca impersonator, $\mathcal{I}$.

Theorem 3.1. *The DIdk transformation converts an* adapt-hid-imp-pa *secure Σ^+-type HIBI protocol into an* adapt-hid-imp-ca *secure one.*

In the reduction from an adapt-hid-imp-ca experiment to an adapt-hid-imp-pa experiment, $\mathcal{I}'$ needs to simulate the PROV oracle. In the DIdk transformation, $\mathcal{I}'$ can have either secret key of an (imaginary) identity 0.ID or 1.ID; thus, $\mathcal{I}'$ can simulate all oracles.

Theorem 3.2. *The MI transformation converts an* adapt-hid-imp-pa *secure Σ^+-type HIBI protocol into an* adapt-hid-imp-ca *secure one.*

We assume two types of impersonators and show that there exist reductions from each impersonator to $\mathcal{I}'$. We let $\mathcal{I}_{master}$ be an impersonator from which $\mathcal{I}'$ derives the secret key corresponding to ID_{master}, and $\mathcal{I}_{user}$ be the other impersonator from which $\mathcal{I}'$ derives a secret key of a user. In the reduction from $\mathcal{I}_{master}$, $\mathcal{I}'$ can perfectly simulate the PROV oracle by obtaining secret keys of users from the external CORR oracle. Thus, $\mathcal{I}'$ can extract a secret key of ID_{master} from $\mathcal{I}_{master}$ (by using the Reset Lemma), and can impersonate ID_{master}. Note that since ID_{master} is randomly chosen from the set of identities, it is a hierarchical identity of depth 1. In the reduction from $\mathcal{I}_{user}$, $\mathcal{I}'$ can perfectly simulate the PROV oracle with a secret key of ID_{master} obtained from the external CORR oracle, and extract a secret key of the target identity ID^* from $\mathcal{I}_{user}$ (by using the Reset Lemma). Thus, it can impersonate ID^*.

Theorem 3.3. *The DPdk transformation converts an* adapt-hid-imp-pa *secure Σ^*-type HIBI protocol into an* adapt-hid-imp-ca *secure one.*

After $\mathcal{I}'$ receives mpk' from the challenger, $\mathcal{I}'$ internally generates another key pair (mpk'_*, msk'_*), and can perfectly simulate all oracles since it obtains the secret keys of all users with this msk'_*. Thus, $\mathcal{I}'$ can extract from $\mathcal{I}$ a secret key of the target identity ID^* either for mpk' or mpk'_* (by using the Reset Lemma). If $\mathcal{I}'$ obtains secret key for ID^* in mpk', it can impersonate ID^* in mpk'.

Theorem 3.4. *The mDIdk transformation converts an* adapt-hid-imp-pa *secure Σ^+-type HIBI protocol into an* adapt-hid-imp-ca *secure one.*

This theorem is proved in almost the same way as **Theorem 3.1**.

We see in the following theorems that the security enhancement transformations can be also applied to wshid-imp-pa secure HIBI protocols. We construct an wshid-imp-pa impersonator, $\mathcal{I}'$, from an wshid-imp-ca impersonator, $\mathcal{I}$.

Theorem 3.5. *The DIdk transformation converts a* wshid-imp-pa *secure Σ^+-type HIBI protocol into a* wshid-imp-ca *secure one.*

In the Setup phase, $\mathcal{I}$ issues $(\mathsf{ID}_1, \ldots, \mathsf{ID}_t)$ to $\mathcal{I}'$ acting as the challenger in the wshid-imp-ca experiment. Then, $\mathcal{I}'$ issues $(0.\mathsf{ID}_1,\ 1.\mathsf{ID}_1, \ldots,\ 0.\mathsf{ID}_t,\ 1.\mathsf{ID}_t)$ to the external challenger. After receiving mpk', $\mathcal{I}'$ sends $(b^*||I_1^{(i)})$ $(1 \leq i \leq t)$ to the external CORR where b^* is a random bit and $\mathsf{ID}_i = (I_1^{(i)}, \ldots, I_{k_i}^{(i)})$, and receives the keys $sk'_{(b^*||I_1^{(i)})}$. $\mathcal{I}'$ can simulate the oracles in the same way as in the proof of **Theorem 3.1**.

Theorem 3.6. *The MI transformation converts a* wshid-imp-pa *secure Σ^+-type HIBI protocol into a* wshid-imp-ca *secure one.*

Theorem 3.7. *The DPdk transformation converts a* wshid-imp-pa *secure Σ^*-type HIBI protocol into a* wshid-imp-ca *secure one.*

Theorem 3.8. *The mDIdk transformation converts a* wshid-imp-pa *secure Σ^+-type HIBI protocol into a* wshid-imp-ca *secure one.*

On the other hand, the DPdk and mDIdk transformations can also convert a passively secure HIBI protocol to a concurrently secure one in the static hierarchical-identity attack model. We construct an stat-hid-imp-pa impersonator, $\mathcal{I}'$, from an stat-hid-imp-ca impersonator, $\mathcal{I}$.

Theorem 3.9. *The DPdk transformation converts a* stat-hid-imp-pa *secure Σ^*-type HIBI protocol into a* stat-hid-imp-ca *secure one.*

In the reduction from a stat-hid-imp-ca experiment to a stat-hid-imp-pa experiment, $\mathcal{I}'$ needs to simulate the PROV oracle. In the DPdk transformation, $\mathcal{I}'$ can have either the master secret key of the master public key mpk'_0 or mpk'_1, can generate a secret key of any identity; thus, $\mathcal{I}'$ can simulate all oracles.

Theorem 3.10. *The mDIdk transformation converts a* stat-hid-imp-pa *secure Σ^+-type HIBI protocol into a* stat-hid-imp-ca *secure one.*

This theorem is proven by a reduction similar to that of **Theorem 3.9**. At the beginning of the Setup phase, $\mathcal{I}'$ randomly chooses $b^* \in \{0, 1\}$, sends a single corrupt query including the 1-level hierarchical identity (b^*) to the external challenger and obtains a secret key $sk'_{(b^*)}$. After this, $\mathcal{I}'$ can generate secret keys of the underlying HIBI protocol for any hierarchical identities of the form $(b^*, I_1, I_2, \ldots, I_k)$ (i.e., any hierarchical identities in which each identity at first level is b^*) and can simulate the PROV oracle.

3.6 Discussion

The DPdk transformation can convert a stat-hid-imp-pa secure Σ^*-type HIBI protocol to a stat-hid-imp-ca secure one, and the mDIdk transformation can convert a stat-hid-imp-pa secure Σ^+-type HIBI protocol to a stat-hid-imp-ca

secure one. On the other hand, the DIdk and MI transformations seem not to be able to do so. See **Table 1** for a summary.

In the DPdk transformation, two master public keys in the underlying HIBI protocol, mpk'_0 and mpk'_1, compose a master public key in the resulting HIBI protocol, and the secret key of each entity in the resulting HIBI protocol is computed with either the master secret keys, msk'_0 or msk'_1, in the underlying HIBI protocol. Even in the stat-hid-imp-atk security model, since the simulator has a master secret key msk'_{b^*} for the master public key mpk'_{b^*}, it can generate a secret key for any entity and then simulate the PROV oracle.

The mDIdk transformation can be applied only to HIBI protocols with a hierarchical depth larger than one. That is, it is not applicable to IBI protocols. Though it is similar to the DIdk transformation, its security proof is similar to that of the DPdk transformation. In the mDIdk transformation, if we obtain $sk'_{(b^*)}$ ($b^* = 0$ or 1), we can compute secret keys for any descendant of the 1-level identity (b^*) and simulate the PROV oracle, even in the stat-hid-imp-atk security model.

On the other hand, we face a problem in simulating the PROV oracle in the DIdk and MI transformations. In the DIdk transformation, the master public key in the resulting HIBI protocol is set with the master public key in the underlying HIBI protocol, mpk', and a secret key of an entity corresponding to identity ID in the resulting HIBI protocol is a secret key corresponding to either identity, $0.\mathsf{ID}$ or $1.\mathsf{ID}$, in the underlying HIBI protocol. The secret key is computed with the master secret key, msk', corresponding to mpk'. Since the simulator does not have msk' in the proof for the stat-hid-imp-atk security and can no longer corrupt users in the learning phase, it cannot obtain secret keys for identities queried to the PROV oracle and fails to simulate it.

In the MI transformation, the master public and secret keys and the secret keys of entities in the resulting HIBI protocol are the same as those in the underlying HIBI protocol. In the proof for the stat-hid-imp-atk security, the simulator might obtain the secret key for the master identity, ID_{master}, at the Setup phase and simulate the CONV oracle. In the challenge phase, however, if the secret key extracted from transcripts coincides with the key for ID_{master}, the simulator would not obtain the non-trivial secret key and the impersonation would fail. We do not know how to address the problem.

We finally observe that the single-key variants may fail to enhance security because corruption may leak information of a secret key. In an HIBI protocol by the DIsk transformation, an entity of ID has only either $sk'_{0.\mathsf{ID}}$ or $sk'_{1.\mathsf{ID}}$, which determines b of its descendants. Precisely speaking, if an impersonator obtains a secret key $sk'_{b.\mathsf{ID}'}$ such that $\mathsf{ID} \in \mathsf{pref}(\mathsf{ID}')$ by issuing a CORR query ID', it may know b for $sk'_{b.\mathsf{ID}'}$ and then for $sk'_{b.\mathsf{ID}}$. A similar discussion with DIsk can be applied to the mDIsk transformation, Also in the DPsk transformation, an entity of ID has only either $sk'_{(\mathsf{ID},0)}$ or $sk'_{(\mathsf{ID},1)}$, and an impersonator may guess b for $sk'_{(\mathsf{ID},b)}$. Thus, in all three cases, from the impersonator that knows b, a simulator in proof may extract only the trivial secret key corresponding to b, and constructing their security proofs seems to be difficult.

References

1. Bellare, M., Namprempre, C., Neven, G.: Security proofs for identity-based identification and signature schemes. Journal of Cryptology 22(1), 1–61 (2009)
2. Bellare, M., Palacio, A.: GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 162–177. Springer, Heidelberg (2002)
3. Chin, J.-J., Heng, S.-H., Goi, B.-M.: Hierarchical Identity-Based Identification Schemes. In: Ślęzak, D., Kim, T.-H., Fang, W.-C., Arnett, K.P. (eds.) SecTech 2009. CCIS, vol. 58, pp. 93–99. Springer, Heidelberg (2009)
4. Cramer, R., Damgård, I., Schoenmakers, B.: Proof of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994)
5. Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: STOC 1990, pp. 416–426. ACM (1990)
6. Fujioka, A., Saito, T., Xagawa, K.: Security Enhancements by OR-Proof in Identity-Based Identification. In: Bao, F., Samarati, P., Zhou, J. (eds.) ACNS 2012. LNCS, vol. 7341, pp. 135–152. Springer, Heidelberg (2012)
7. Fujioka, A., Saito, T., Xagawa, K.: Secure Hierarchical Identity-Based Identification without Random Oracles. In: Gollmann, D., Freiling, F.C. (eds.) ISC 2012. LNCS, vol. 7483, pp. 258–273. Springer, Heidelberg (2012)
8. Gennaro, R.: Multi-trapdoor Commitments and Their Applications to Proofs of Knowledge Secure Under Concurrent Man-in-the-Middle Attacks. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 220–236. Springer, Heidelberg (2004)
9. Gentry, C., Silverberg, A.: Hierarchical ID-Based Cryptography. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 548–566. Springer, Heidelberg (2002)
10. Horwitz, J., Lynn, B.: Toward Hierarchical Identity-Based Encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 466–481. Springer, Heidelberg (2002)
11. Kurosawa, K., Heng, S.-H.: Identity-Based Identification Without Random Oracles. In: Gervasi, O., Gavrilova, M.L., Kumar, V., Laganà, A., Lee, H.P., Mun, Y., Taniar, D., Tan, C.J.K. (eds.) ICCSA 2005, Part II. LNCS, vol. 3481, pp. 603–613. Springer, Heidelberg (2005)
12. Rückert, M.: Adaptively Secure Identity-Based Identification from Lattices without Random Oracles. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 345–362. Springer, Heidelberg (2010)
13. Shamir, A.: Identity-Based Cryptosystems and Signature Schemes. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985)
14. Yang, G., Chen, J., Wong, D.S., Deng, X., Wang, D.: A new framework for the design and analysis of identity-based identification schemes. Theoretical Computer Science 407(1-3), 370–388 (2008); A preliminary version appeared ACNS 2007 (2007)

LiBrA-CAN: A Lightweight Broadcast Authentication Protocol for Controller Area Networks

Bogdan Groza[1], Stefan Murvay[1], Anthony van Herrewege[2], and Ingrid Verbauwhede[2]

[1] Faculty of Automatics and Computers, Politehnica University of Timisoara
{bogdan.groza,pal-stefan.murvay}@aut.upt.ro
[2] ESAT/COSIC - IBBT, KU Leuven, Belgium
{anthony.vanherrewege,ingrid.verbauwhede}@esat.kuleuven.be

Abstract. Security in vehicular networks established itself as a highly active research area in the last few years. However, there are only a few results so far on assuring security for communication buses inside vehicles. Here we advocate the use of a protocol based entirely on simple symmetric primitives that takes advantage of two interesting procedures which we call key splitting and MAC mixing. Rather than achieving authentication independently for each node, we split authentication keys between groups of multiple nodes. This leads to a more efficient progressive authentication that is effective especially in the case when compromised nodes form only a minority and we believe such an assumption to be realistic in automotive networks. To gain more security we also account an interesting construction in which message authentication codes are amalgamated using systems of linear equations. We study several protocol variants which are extremely flexible allowing different trade-offs on bus load, computational cost and security level. Experimental results are presented on state-of-the-art Infineon TriCore controllers which are contrasted with low end controllers with Freescale S12X cores, all these devices are wide spread in the automotive industry. Finally, we discuss a completely backward compatible solution based on CAN+, a recent improvement of CAN.

1 Motivation and Related Work

Vehicular network security established itself as an intense research topic in the last few years. Remarkable research papers from Koscher et al. [7] and later Checkoway et al. [4] showed vehicles to be easy targets for malicious adversaries.

While most of previous research was focused on vehicle to vehicle and vehicle to infrastructure communication there seem to be only a few results for assuring security on communication buses inside vehicles. There are several reasons behind this. First, the relevance of security inside vehicles was decisively shown only in the last two years [7], [4]. Second, the design principles used by manufacturers are somewhat out of reach for the academic community, being hard in this way to understand many assertions behind protocol design. Third, which is relevant for our research here, intra-vehicle communication is subject to constraints and specifications that are quite different from other well studied protocols. Most of the approaches advocate the use of secure gateways between different ECUs (Electronic Control Unit) or subnetworks [1], [13] and rely on basic building blocks from cryptography (encryptions, signatures, etc.). However, none

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 185–200, 2012.

of these approaches is meant specifically for assuring broadcast authentication on CAN which is still the most common communication bus in automotives.

In this respect two main results in assuring CAN security can be found so far, one of them is based on the well known TESLA protocol [6] and the other proposes a new paradigm which closely follows CAN specifications [12]. Van Herrewege et al. [12] design their protocol from scratch and clearly note that the constraints of CAN "eliminate all the authentication protocols published so far". We do agree with this conclusion in the sense that we believe that standard authentication approaches, may cover only some of the application areas for CAN and new approaches (even non-standard) are needed.

Previous proposals. TESLA like protocols proved to be highly effective in sensor networks [10], [9] and so far are the most efficient alternative for assuring broadcast authentication with efficient Message Authentication Codes (MAC). However, when it comes to CAN bus, this protocol family has one drawback that is critical for automotives: delays, which by the nature of TESLA are unavoidable. The main purpose of the work in [6] is to determine a lower bound on these delays. Delays in the order of milliseconds or below, as shown to be achievable in [6], are satisfactory for many scenarios, but such delays do not appear to be small enough for intra-vehicle communication. There is no obvious way to improve on these delays further. Of course one alternative is in using a bus with a higher throughput, more computational power and better electronic components (e.g., oscillators) but this will greatly increase the cost of components, nullifying in this way the cost effectiveness of CAN. CANAuth [12] is a protocol that has the merit to follow in great detail the specifications of CAN, its security is specifically designed to meet the requirements of the CAN bus. In particular, CANAuth is not intended to achieve source authentication as the authentication is binded to the message IDs and messages may originate from different sources which will be impossible to trace. This fits the specification of CAN which has a message oriented communication. However, a first issue is that the number of CAN IDs is quite high, in the order of hundreds (11 bits) or even millions in the case of extended frames (29 bits) and storing a key for each possible ID does not seem to be so practical. For this purpose, in [12] a clever solution is imagined: the keys are linked with acceptance codes and masks, which fortunately are not numerous. But still, this leads to some security concerns as we discuss next. Traditionally, keys are associated to entities to ensure that they are not impersonated by adversaries, but the effect of associating keys to messages is less obvious. For example, any external tool (assume On-Board Diagnostics (OBD) tools which are wide spread) that is produced by external third parties will have to embed the keys associated for each ID that it sends over or even just listens on CAN. It is thus unclear which keys can be shared with different manufacturers and how or what are the security outcomes for this. Obviously, if a third party device, even an innocuous one designed just as passive receiver, is easier to compromise then all the IDs which it was allowed to send or just receive are equally compromised.

Our proposal. We take advantage of a progressive authentication mechanism, by which only a few bits of the MAC are revealed in each packet to each verifier, and each part of the MAC can be verified by more than one receiver. To achieve this flexible authentication mechanism we base our proposal on two paradigms: key splitting and MAC

mixing, the later being an optional procedure to increase security by allowing any node to detect a potential forgery.

Key splitting allows a higher entropy for each mixed MAC that is sent at the cost of loosing some security for groups that contain malicious nodes. In scenarios with high number of nodes, an adversarial majority will be required to break the protocol, while if there are fewer adversarial nodes, the security level is drastically increased. Consequently, this appears to give a flexible and efficient trade-off. This procedure is not new, similar techniques were proposed in the past in the context of broadcast encryption. We could trace this back up to the work of Fiat and Naor [5] but there is a high amount of papers on this subject. However, the constraints of our application in CAN networks are entirely different from related work where this procedure was suggested or used in scenarios such as sensor networks [2], pay-tv [8], etc. The main idea behind such schemes is that groups of k corrupted receivers cannot learn the secret (in settings with $n > k$ users).

In addition to this we exhibit a distinct contribution in the construction of Linearly Mixed MACs which allow us to amalgamate more authentication codes in one via a system of linear equations. This construction has the advantage that if one of the MACs is wrong then this will affect all other MACs and thus the mixed MAC will fail to verify on any of the multiple keys. This increases the chance of a forgery being detected and ultimately it increases the reliability in front of benign nodes that are in possession of a wrong key. To best of our knowledge this procedure is new. The closest work that we could find are the multi-verifier signatures proposed by Roeder et al. [11]. In their work, linear systems of equations are used as well upon message authentication codes but the security properties and goals of their construction are different.

These procedures allow us to design a protocol that is more flexible and efficient. For our setting we assume a reduced number of participants. While indeed ECUs inside cars come from different manufacturers which may or may not be trustworthy, we believe that suspicious ECUs should be limited in number, since the potential insertion of a trapdoor in some component will discredit the public image of the manufacturer too much and it appears to be little or no benefit for this. More, ECUs coming from the same manufacturer should be trustworthy with each other and can use the same shared key (randomly generated at runtime for each (sub)network that they are part of). In this way the number of actual keys needed to assure broadcast security should be more limited than it appears to be on a first sight. In our design we try to take advantage of this assumption, and our approach is more efficient in the case when compromised nodes form only a minority.

2 The Protocol

We begin with a brief overview of the CAN protocol followed by a description of the frame structure employed in our protocol. Then we outline the main authentication scheme which builds upon keys shared between groups of receivers, a procedure which we call key splitting. Further, we discuss some variations of the main scheme that can be used for different trade-offs. Subsequently we introduce a construction which we call Linearly Mixed MAC (LM-MAC) which gives additional security benefits.

2.1 Overview of the CAN Protocol

Controller Area Network (CAN) is a broadcast serial bus. The typical topology consists of a differential bus which connects multiple nodes by two wires (called CAN-H and CAN-L). This is also suggested in Figure 3 which is related to the main version of our protocol. To avoid collisions an arbitration based on message identifiers (29 bits in extended frames and 11 bits in standard frames) is used. Each CAN frame begins with a start bit and is followed by the arbitration field, a control field (6 bits), data bits (0-64), CRC sequence (15 bits), a 2 bit acknowledgment and 7 bits that mark the end of the frame. Additional stuffing bits (distinct in value to the previous bit) are added after each 6 consecutive bits of identical value. This structure is suggested in Figure 1.

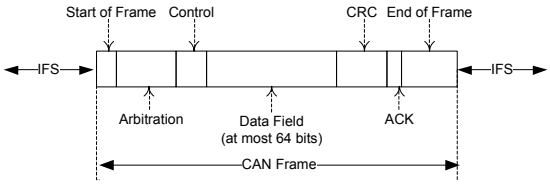


Fig. 1. Structure of a CAN frame

2.2 Frame Structure

As a general procedure, we separate between frames that carry messages and frames that carry authentication tags. This seems to be a correct option due to a widely employed CAN mechanism which is ID filtering that is used to restrict certain frames to arrive to a particular node. While we do want to keep this feature, we want the node to be able to carry additional authentication tasks, e.g., in the case of the two-stage authentication discussed further, a reason for which we intend for the authentication frames to be able to reach the node and thus they may need to have a different ID than the message frame. The last bit of the identifier field specifies whether a frame carries an authentication tag or message. This procedure is employed in our experimental setup while in section 4 we discuss a backward compatible solution which can embed all the authentication information inside the message.

Larger data blocks or authentication tags can be split across multiple frames with the same ID field and counter. Other adjustments can be done at the implementation level. For example, since the ID field is quite short, both the node and window identifiers (which denote the source and the number of the authentication frame) can be moved in the data field. We preferred to place these identifiers in the ID field since it is a frequent choice of developers to place a unique ID for each node in the CAN ID field. But indeed, such an option can affect real-time requirements and for this purpose placing these IDs in the data field is safer. The size of the counter c could be roughly around 20–40 bits but this greatly depends on the bus speed (which determines the number of packets released each second).

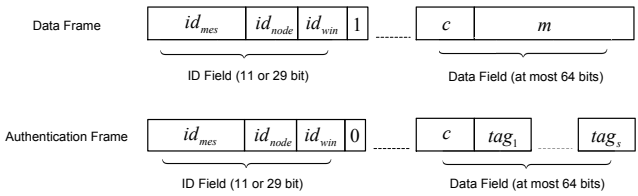


Fig. 2. Data frames and authentication frames

2.3 The Main Scheme: Centralized Authentication

A master oriented communication makes sense since it is practical to have one node with higher computational power that can take care of the most intensive part of the authentication. Figure 3 shows the master node and the slave nodes connected to the bus, it also outlines the keys that are shared between nodes. For the key sharing procedure, all slaves register to the master which distributes the keys.In practice associating nodes to a group and sharing the keys is done by standard techniques, e.g., key-exchange protocols, we do not insist on this since such issues are straight-forward to solve.

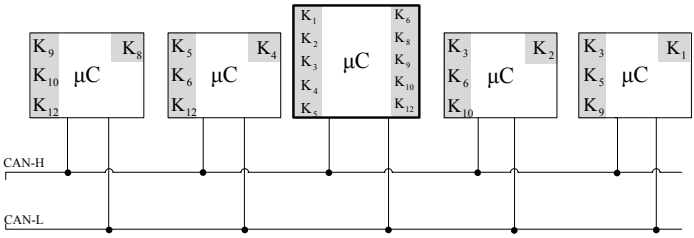


Fig. 3. Master and slave microcontrollers (μC) in a setting for centralized authentication

In the main scheme we make use of Mixed Message Authentication Codes (M-MAC) which amalgamate more MACs into one. Here we give an abstract definition for this construction while in a forthcoming section we provide a more elaborate instance with additional security properties. Indeed, the easiest way to build an M-MAC is simply by concatenating multiple tags, such a construction is fine for our protocol and can be safely embodied in the main scheme (still, we can achieve more security with the LM-MAC introduced in an upcoming section).

Construction 1. *(Mixed Message Authentication Code) A mixed message authentication code* M-MAC *is a tuple* $(\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ *of probabilistic polynomial-time algorithms such that:*

1. $\mathbb{K} \leftarrow \mathsf{Gen}(1^\ell, s)$ *is the key generation algorithm which takes as input the security parameter* ℓ *and set size* s *then outputs a key set* $\mathbb{K} = \{k_0, k_1, ..., k_s\}$ *of* s *keys,*
2. $\tau \leftarrow \mathsf{Tag}(\mathbb{K}, \mathbb{M})$ *is the* MAC *generation algorithm which takes as input the key set* $\mathbb{K}$ *and message tuple* $\mathbb{M} = (m_0, m_1, ..., m_s)$ *where each* $m_i \in \{0,1\}^*$ *then outputs a tag* τ *(whenever needed, to avoid ambiguities on the message and key, we use the notation* $\mathsf{M\text{-}MAC}_{\mathbb{K}}(\mathbb{M})$ *to depict this tag),*
3. $v \leftarrow \mathsf{Ver}(k, m, \tau)$ *is the verification algorithm which takes as input a key* $k \in \mathbb{K}$*, a message* $m \in \{0,1\}^*$ *and a tag* τ *and outputs a bit* v *which is* 1 *if and only if the tag is valid with respect to the key* k*, otherwise the bit* v *is* 0*. For correctness we require that if* $k \in \mathbb{K}$ *and* $m \in \mathbb{M}$ *then* $1 \leftarrow \mathsf{Ver}(k, m, \mathsf{Tag}(\mathbb{K}, \mathbb{M}))$.

The centralized scheme is summarized by the next construction. For simplicity of the exposition, since the main scheme is used to authenticate the same message to all nodes (rather than authenticate a tuple of messages as in the cummulative authentication scheme), we replace $\mathbb{M}$ with a simple array that points out the values that are authenticated, e.g., $id_{node}, id_{win}, c, m$, etc. Obviously in this case the M-MAC receives as input a message tuple of s identical messages.

Construction 2. *(Centralized Authentication) Given a mixed message authentication code algorithm* M-MAC *for some security parameter* ℓ*, size* s *and a group of* n *nodes, we define protocol* $\mathsf{CN\text{-}CAN\text{-}LiBrA}_{\mathcal{M},\mathcal{S}^*}(\mathsf{M\text{-}MAC}, \ell, s, n, b, w)$ *as the following set of actions for the master* $\mathcal{M}$*:*

1. $\mathsf{Setup}(\ell, n, s)$ *on which master* $\mathcal{M}$ *generates all subsets of* s *slaves out of* n *slaves, let* $t = \binom{n}{s}$ *be the number of subsets, and randomly picks* t *keys, each of* ℓ *bits, then places them in the keyset* $\mathbb{K}_{\mathcal{M}} = \{k_1, k_2, ..., k_t\}$*. Subsequently master* $\mathcal{M}$ *uses a secure channel to send each node the corresponding keys (alternatively these keys can be distributed in an off-line manner). Let* $\mathbb{K}^i_{\mathcal{S}} = \{k_1, k_2, ..., k_{t'}\}$ *with* $t' = \binom{n-1}{s-1}$ *denote the key set received by each slave* $\mathcal{S}$*.*
2. $\mathsf{RecMes}(id_{node}, id_{win}, c, m)$ *on which master* $\mathcal{M}$ *receives a data frame containing message* m *from slave* $\mathcal{S}$ *checks if the counter is up-to-date then stores the packet in a queue of messages to be authenticated.*
3. $\mathsf{RecTag}(id_{node}, id_{win}, c, \mathsf{M\text{-}MAC}_{\mathbb{K}^i_{\mathcal{S}}}(id_{node}, id_{win}, c, m))$ *on which master* $\mathcal{M}$ *receives an authentication frame containing tag* $\mathsf{M\text{-}MAC}_{\mathbb{K}^i_{\mathcal{S}}}(id_{node}, id_{win}, c, m)$ *from slave* $\mathcal{S}$*. Further, he retrieves the packet matching the identifiers and counter from the queue and if the message proves to be authentic, i.e.,* $1 \leftarrow \mathsf{Ver}((id_{node}, id_{win}, c, m), k, \mathsf{M\text{-}MAC}_{\mathbb{K}^i_{\mathcal{S}}}(id_{node}, id_{win}, c, m))$*,* $\forall k \in \mathbb{K}^i_{\mathcal{S}}$*, he proceeds to authenticating the tag to other nodes with* $\mathsf{SendTag}(id_{node}, id_{win}, m, \mathbb{K}^i)$*. If the message is not available in the queue then the tag is discarded (subsequently an error message can be sent).*
4. $\mathsf{SendTag}(id_{node}, id_{win}, m, \mathbb{K}^i)$ *on which master* $\mathcal{M}$ *after receiving a message and its valid tag, groups all the remaining keys* $\mathbb{K}_{\mathcal{M}} \setminus \mathbb{K}^i_{\mathcal{S}}$ *in sets of size* v *then for each such set* $\widetilde{\mathbb{K}}^j_{\mathcal{S}}$ *computes* $\mathsf{M\text{-}MAC}_{\widetilde{\mathbb{K}}^j_{\mathcal{S}}}(id_{node}, j, m)$ *and broadcasts it in authentication frames with node identifier* id_{node} *and window identifier set to* j *(obviously there are* $|\mathbb{K}_{\mathcal{M}} \setminus \mathbb{K}^i_{\mathcal{S}}|/v$ *windows).*

and for each of the slaves $\mathcal{S}^*$:

1. $\mathsf{Setup}(\ell, n, s)$ *on which slave* $\mathcal{S}_i$ *obtains its key set* $\mathbb{K}_{\mathcal{S}}^i = \{k_1, k_2, ..., k_{t'}\}$ *with* $t' = \binom{n-1}{s-1}$ *from master* $\mathcal{M}$ *(either offline or via a secure channel).*
2. $\mathsf{RecMes}(id_{node}, id_{win}, c, m)$ *on which slave* $\mathcal{S}$ *receives a data frame containing message* m *from another slave* $\mathcal{S}_j$ *and proceeds similarly to master* $\mathcal{M}$ *by storing it in a queue of messages to be authenticated.*
3. $\mathsf{RecTag}(id_{node}, id_{win}, c, \mathsf{M\text{-}MAC}_{\mathbb{K}_{\mathcal{S}}^j}(id_{node}, id_{win}, c, m))$ *on which* $\mathcal{S}_i$ *receives an authentication frame containing tag* $\mathsf{M\text{-}MAC}_{\mathbb{K}_{\mathcal{S}}^i}(id_{node}, id_{win}, c, m))$ *from the master* $\mathcal{M}$ *or another slave* $\mathcal{S}_j$ *and verifies for all keys* $k \in \mathbb{K}^i \cap \mathbb{K}^j$ *if the tag is correct. If for all keys in its keyset a correct tag was received*
then message m *is deemed authentic.*
4. $\mathsf{SendMes}(m, \mathbb{K}_i)$ *on which slave* $\mathcal{S}_i$ *whenever wants to broadcast a message* m *increments its local counter, computes the tag* $\mathsf{M\text{-}MAC}_{\mathbb{K}_{\mathcal{S}}^i}(id_{node}, 0, c, m)$ *with its keyset* $\mathbb{K}^i$ *and sends the data frame containing* m *and an authentication frame containing the tag on the bus (note that in the case of slaves* id_{win} *is set to 0).*

Example 1. The key allocation done by the Setup procedure allows the keys to be split between groups of n slaves. Here we clarify our intentions with the *key splitting* procedure by giving an example. Table 1 shows the groups that can be formed in the case of 4 nodes. If we consider groups formed by exactly 2 nodes we have $\binom{4}{2} = 6$ groups and each two nodes share exactly $\binom{2}{0} = 1$ group. Table 1 outlines the groups shared by $\mathcal{S}_1$, i.e., G_9, G_{10}, G_{12}, and those shared by $\mathcal{S}_2$, i.e., G_5, G_6, G_{12}. Note that they intersect in one group G_{12}. In Table 2 the case of $n = 4$ and $n = 8$ nodes are explored, with complete groups of all sizes k and any number of corrupted nodes l. The total number of groups and the subgroup shared by each node as well as the percentage of secure bits, i.e., bits that cannot be forged by an adversary, from each M-MAC are outlined. Indeed, the percent of authenticated bits from each tag is higher and decreases significantly with the number of corrupted nodes.

Table 1. Possible groups with 4 nodes, groups of size 2 outlined in gray

	G_0	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}	G_{13}	G_{14}	G_{15}
$\mathcal{S}_1$	0	0	0	0	0	0	0	0	1	(1)	(1)	1	(1)	1	1	1
$\mathcal{S}_2$	0	0	0	0	1	[1]	[1]	1	0	0	0	0	[1]	1	1	1
$\mathcal{S}_3$	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
$\mathcal{S}_4$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

2.4 Variations of the Main Scheme: Two-Stage and Cumulative Authentication

For practical reasons we discuss two variations of the main scheme. In the experimental results section, the first variation is shown to have certain advantages in front of the main scheme for scenarios when nodes have equal computational power.

Table 2. Authentication rate in the case of $n = 4, 8$ participants, groups of size k and l corrupted nodes

				Authentication bits from one M-MAC (%)							
n	k	groups	sub-groups	$l=0$	$l=1$	$l=2$	$l=3$	$l=4$	$l=5$	$l=6$	$l=7$
4	1	4	1	25	25	25	25	-	-	-	-
4	2	6	3	50	33	33	16	-	-	-	-
4	3	3		75	25	0	0	-	-	-	-
8	1	8	1	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5
8	2	28	7	25	21	17	14	10	7	3.5	0
8	3	56	21	37.5	26	17	10	5	1.7	0	0
8	4	70	35	50	28	14	5	1.4	0	0	0
8	5	56	35	62	26	8.9	1.7	0	0	0	0
8	6	28	21	75	21	3.5	0	0	0	0	0
8	7	8	7	87	12.5	0	0	0	0	0	0

In the case of *two-stage authentication* we assume a scenario in which only slave nodes are present, i.e., nodes with equal computational power. In this case each node can start broadcasting by sending a tag which includes only a part of the keys for the subgroups that he is part of and a second slave (pointed out by some flag, or predefined in protocol actions) continues with the authentication. The procedure is repeated until the desired number of authentication frames is reached. Various ways for tag allocation can be imagined. Consider the case of 8 nodes in subgroups of size 3 and 4 authentication frames (codenamed TS-8S3F4). If M-MACs are used then these can be set up to work in $GF(2^{16})$ or $GF(2^{32})$. Subsequently each node sends an M-MAC with keys for 4 of the nodes (or 2 in case $GF(2^{32})$) and the nodes reply in a round-robin fashion (note that a frame carries at most 64 bits). To save some computational power and have even more flexibility in tag allocation it is also possible to skip the use of the M-MAC. In Table 3 we give an example for this case. Each row corresponds to one of the 8 slaves and each column to one of the 56 groups that are formed with 3 slaves, $\times$ is used as placeholder to denote that a node is part of a group. Here f_j^i denotes the j-th part of frame i and the authentication is started by slave $\mathcal{S}_1$ with frame f_*^1 followed by $\mathcal{S}_2$ with f_*^2 then again $\mathcal{S}_1$ with f_*^3 but this time followed by $\mathcal{S}_3$ with f_*^4 (here $*$ is a placeholder for any of the frame components). We can set the size of each tag in f_*^2 and f_*^2 to 16 bits and for f_*^1 and f_*^3 use around 5-7 bits for each tag. This will result in a security level of around 64 bits for each node.

Since in some scenarios small delays may be acceptable, we can take benefit of them and increase the efficiency of the main scheme. In the *cumulative authentication* scheme a timer can be used and all messages are accumulated by the master over a predefined period δ then authenticated at once (this procedure can be employed in the slave-only settings as well). While this introduces an additional delay δ, similar to the case of the TESLA protocol, this delay can be chosen as small as needed to cover application requirements. Different to the case of the delay from TESLA like protocols, this delay is not strongly constrained by external parameters (such as oscillator precision, synchronization error, bus speed, etc.).

Table 3. Example of tag scheduling with two-stage authentication TS-8S2F4 (8 nodes with groups of size 3)

	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}	G_{13}	G_{14}	G_{15}	G_{16}	G_{17}	G_{18}	G_{19}	G_{20}	G_{21}	G_{22}	G_{23}	G_{24}	G_{25}	G_{26}	G_{27}	G_{28}
S_1	f_1^1	f_2^1	f_3^1	f_4^1	f_5^1	f_6^1	f_7^1	f_8^1	f_9^1	f_{10}^1	f_1^3	f_2^3	f_3^3	f_4^3	f_5^3	f_6^3	f_7^3	f_8^3	f_9^3	f_{10}^3	f_{11}^3							
S_2	×	×	×	×	×	×																f_1^2	f_2^2	f_3^2	f_4^2	×	×	×
S_3	×						×	×	×	×	×											×	×	×	×	×		
S_4		×					×					×	×	×	×							×					×	×
S_5			×					×				×				×	×	×					×				×	
S_6				×					×				×			×			×	×				×				×
S_7					×					×				×			×		×		×					×		
S_8						×					×				×			×		×	×						×	

	G_{29}	G_{30}	G_{31}	G_{32}	G_{33}	G_{34}	G_{35}	G_{36}	G_{37}	G_{38}	G_{39}	G_{40}	G_{41}	G_{42}	G_{43}	G_{44}	G_{45}	G_{46}	G_{47}	G_{48}	G_{49}	G_{50}	G_{51}	G_{52}	G_{53}	G_{54}	G_{55}	G_{56}
S_1																												
S_2	×	×	×	×	×	×	×	×																				
S_3									f_1^4	f_2^4	f_3^4	f_4^4	×	×	×	×	×	×										
S_4	×	×							×	×	×	×							×	×	×	×	×	×				
S_5			×	×	×				×				×	×	×				×	×	×				×	×	×	
S_6			×			×	×			×			×			×	×		×			×	×		×	×		×
S_7	×			×		×		×			×			×		×		×		×		×		×		×	×	
S_8		×			×		×	×				×			×		×	×		×	×		×	×		×	×	×

2.5 Increasing Security with LM-MACs (Linearly Mixed MACs)

As outlined in our abstract description, M-MACs use an array of keys to build a tag which is verifiable by any of the keys. The first security property which we require for an M-MAC is *unforgeability* and is a standard property for any MAC code, thus it merely derives from the main building block. We do develop on this by requiring a new property which we call *strong non-malleability* and which we show to be achievable by our more advanced LM-MAC construction.

Construction 3. *(Linearly Mixed MAC) We define the* LM-MAC *as the tuple of probabilistic polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ *that work as follow:*

1. $\mathbb{K} \leftarrow \mathsf{Gen}(1^\ell, s)$ *is the key generation algorithm which flips coins and returns a key set* $\mathbb{K} = \{k_0, k_1, ..., k_s\}$ *where each key has* ℓ *bits (*ℓ *is the security parameter of the scheme),*
2. $\tau \leftarrow \mathsf{Tag}(\mathbb{K}, \mathbb{M})$ *is the mac generation algorithm which returns a tag* $\tau = \{x_1, x_2, ..., x_s\}$ *where each* x_i *is the solution of the following linear system in* $GF(2^b)$*:*

$$\begin{cases} \mathsf{KD}_1(k_1, m_1) \cdot x_1 + \mathsf{KD}_2(k_1, m_1) \cdot x_2 + ... + \mathsf{KD}_s(k_1, m_1) \cdot x_s \equiv \mathsf{MAC}_{k_1}(m_1) \\ \mathsf{KD}_1(k_2, m_2) \cdot x_1 + \mathsf{KD}_2(k_2, m_2) \cdot x_2 + ... + \mathsf{KD}_s(k_2, m_2) \cdot x_s \equiv \mathsf{MAC}_{k_2}(m_2) \\ ... \\ \mathsf{KD}_1(k_s, m_s) \cdot x_1 + \mathsf{KD}_2(k_s, m_s) \cdot x_2 + ... + \mathsf{KD}_s(k_s, m_s) \cdot x_s \equiv \mathsf{MAC}_{k_s}(m_s) \end{cases}$$

Here b *is polynomial in the security parameter* ℓ *and* KD *stands for a key derivation process. If such a solution does not exist, then the* M-MAC *algorithm fails and returns* $\perp$*.*
3. $v \leftarrow \mathsf{Ver}(k, m, \tau)$ *is the verification algorithm which returns* 1 *if and only if having* $\tau' = \mathsf{MAC}_k(m)$ *it holds* $\tau' \equiv \mathsf{KD}_1(k, m) \cdot x_1 + \mathsf{KD}_2(k, m) \cdot x_2 + ... + \mathsf{KD}_s(k, m) \cdot x_s$*. Otherwise it returns* 0*.*

Let us emphasize that the probability that the M-MAC fails to return a solution is negligible in the security parameter (if proper b and s are chosen). As shown in [3] the probability that an n by n matrix with random elements from $GF(q)$ is non-singular converges to $\prod_{i=1}^{\infty}(1-1/q^i)$ as $n \to \infty$. For example in case when $s = 4$ we have a chance of around 10^{-5} for $b = 16$ and 10^{-10} for $b = 32$ for the M-MAC to fail.

Example 2. We want to clarify here our intentions on M-MACs with respect to the protocol design. Consider a case when master M broadcasts messages m_1 and m_2 to slaves S_1, S_2 along with the authentication tag. To increase efficiency of our protocol we want to authenticate both messages with the same mixed MAC and more, since only a portion of each tag is disclosed (reducing the bus overhead but also the security level), we want one of the slaves to be able to carry out the authentication further with a new part of a valid tag (note that this is what happens in the case of the two-stage authentication). Consider that the following packets arrive on the bus: message m_1, message m_2 and the mixed tag obtained by simply concatenating the two tags $\mathsf{MAC}_{k_1}(m_1)||\mathsf{MAC}_{k_2}(m_2)$. However, due to the message filtering of the CAN bus it may be that the two messages do not reach both slaves. Assume message m_1 reaches S_1 and m_2 reaches S_2. Now neither S_1 or S_2 can carry the authentication further, even in the case when they both have k_1 and k_2 they are not in possession of the message that reached the other slave and thus they can not validate the other part of the tag. More relevant, note that the nodes are unable to detect if the other part of the tag is compromised. Now consider the case of the LM-MAC. In this case the tag is obtained by mixing the two tags via the linear equation system, e.g., the two components of the tag x_1, x_2 verify a relation of the form $\alpha_1 x_1 + \alpha_2 x_2 = \mathsf{MAC}_{k_1}(m_1)$ and $\beta_1 x_1 + \beta_2 x_2 = \mathsf{MAC}_{k_2}(m_2)$ (here α's and β's are derived from the secret keys k_1, k_2). If an adversary compromises any part of the tag, i.e., either x_1 or x_2, then both equations will fail to verify and any of the receivers can detect this (indeed, we assume that the adversary is not in possession of the secret keys k_1 and k_2 since in such case he can compute correct LM-MACs anyway). Consequently, with the LM-MACs any of them can check the tag for correctness and this validation will also hold for the other receiver, this is inherited from the strong non-malleability property for M-MACs.

We now sketch a more formal account of the properties that we require for our building blocks. These are mediated by two attack games against unforgeability, i.e., $\mathsf{Game}^{\mathsf{UF}}_{\mathsf{M\text{-}MAC}}$, and strong non-malleability, i.e., $\mathsf{Game}^{\mathsf{SNM}}_{\mathsf{M\text{-}MAC}}$. Both games are defined for a generic M-MAC construction and in particular the LM-MAC can be proved to resist such attacks. The attack game on strong non-malleability $\mathsf{Game}^{\mathsf{SNM}}_{\mathsf{M\text{-}MAC}}$ against an M-MAC requires an adversary to be able to construct an M-MAC in such way that verification fails with at least one of the keys but succeeds with another. An M-MAC that is resilient to such an attack is called *strongly non-malleable*.

Definition 1. *(Unforgeability Attack Game) We define the* M-MAC *unforgeability game* $\mathsf{Game}^{\mathsf{UF}}_{\mathsf{M\text{-}MAC}}$ *as the following five stage game between challenger* $\mathcal{C}$ *and adversary* Adv*:*

1. Challenger $\mathcal{C}$ *runs the key generation algorithm* $\mathsf{Gen}(1^\ell, s)$ *to get a key set* $\mathbb{K} = \{k_0, k_1, ..., k_s\}$.
2. Adversary Adv *is allowed to requests* $\mathcal{C}$ *any subset of the keyset* $\mathbb{K}' = \{k_{j_0}, k_{j_1}, ..., k_{j_t}\}$, $t < s$ *where* $\forall j_i \in [1..s]$. *That is, the adversary is always missing at least 1 of the keys.*

3. Adversary Adv is allowed to make queries to the MAC generation oracle $\mathcal{O}^{\mathsf{Tag}}(\mathbb{K}, \mathbb{M})$ for any message tuple $\mathbb{M}$ to obtain the corresponding tag $\tau \leftarrow \mathsf{Tag}(\mathbb{K}, \mathbb{M})$ and to the verification oracle $\mathcal{O}^{\mathsf{Ver}}(i, \tau, m)$ with any key index i, tag τ and message m and the oracle will return 1 if and only if τ is a correct tag under key k_i for message m.
4. Eventually, the adversary outputs the tuple $(m^{\Diamond}, \tau^{\Diamond}, i)$ for some index i such that he is not in possession of k_i.
5. The game output is 1 if the following two conditions hold: Ver *outputs 1 on (τ, m, k_i) and the adversary never queried m to the* Tag *oracle. Otherwise the game output is 0.*

Definition 2. *(Unforgeability) We say that a mixed message authentication code* M-MAC *is unforgeable if:* $\Pr\left[\mathsf{Game}^{\mathsf{UF}}_{\mathsf{M\text{-}MAC}}(1^{\ell}, s) = 1\right] < \mathrm{negl}(\ell)$.

Definition 3. *(Strong Non-malleability Attack Game) We define the* M-MAC *strong non-malleability game $\mathsf{Game}^{\mathsf{SNM}}_{\mathsf{M\text{-}MAC}}$ as the following five stage game between challenger $\mathcal{C}$ and adversary Adv:*

1. Challenger $\mathcal{C}$ runs the key generation algorithm $\mathsf{Gen}(1^{\ell}, s)$ to get a key set $\mathbb{K} = \{k_0, k_1, ..., k_s\}$.
2. Adversary Adv is allowed to requests $\mathcal{C}$ any subset of the keyset $\mathbb{K}' = \{k_{j_0}, k_{j_1}, ..., k_{j_t}\}$, $t < s - 1$ where $\forall j_i \in [1..s]$. That is, the adversary is always missing at least 2 of the keys.
3. Adversary Adv is allowed to make queries to the MAC generation oracle $\mathcal{O}^{\mathsf{Tag}}(\mathbb{K}, \mathbb{M})$ for any message tuple $\mathbb{M}$ to obtain the corresponding tag $\tau \leftarrow \mathsf{Tag}(\mathbb{K}, \mathbb{M})$ and to the verification oracle $\mathcal{O}^{\mathsf{Ver}}(i, \tau, m)$ with any key index i, tag τ and message m and the oracle will return 1 if and only if τ is correct tag under key k_i for message m.
4. Eventually, the adversary outputs the pair $(m^{\Diamond}, \tau^{\Diamond})$.
5. The game output is 1 if there are at least two keys $k, k' \in \mathbb{K}$ such that the following two conditions hold: Ver *outputs 1 on (τ, m, k) but outputs 0 on (τ, m, k') and the keys k, k' are not part of the adversary keyset $\mathbb{K}'$. Otherwise the game output is 0.*

Definition 4. *(Strong Non-malleability) We say that a mixed message authentication code* M-MAC *is strongly non-malleable if:* $\Pr\left[\mathsf{Game}^{\mathsf{SNM}}_{\mathsf{M\text{-}MAC}}(1^{\ell}, s) = 1\right] < \mathrm{negl}(\ell)$.

Theorem 1. *The* LM-MAC *construction is unforgeable if the underlying MAC is unforgeable and is strongly non-malleable in the random oracle model.*

Due to space limitations, a proof of this theorem in the random oracle model is deferred for the extended version of this work.

3 Experimental Results

To evaluate the performance of the proposed protocol suite, we used several setups with different hardware components to determine the minimum authentication delay. Automotive grade embedded devices from Freescale and Infineon as well as a notebook equipped with an adapter for CAN communication from Vector were employed to build the nodes of our experimental CAN network. The embedded platforms that we used are representatives for industry's low-end and high-end edges.

3.1 Test Beds

Using the aforementioned components we built several test beds. First, the case of a system using the centralized authentication approach with one master node and 4 slave nodes was considered:

- *Testbed 1:* $S12+4\times S12$. Both master and slave nodes are built on identical S12 development boards with CAN communication speed set to 125kbps.
- *Testbed 2:* $TC1782+4\times TC1797$. Master and slave nodes are built on similar TriCore development boards having the same computational and communication capabilities. CAN communication speed is set to 1Mbps.
- *Testbed 3:* $Intel\ T7700+4\times S12$. The master node is implemented on a PC (Intel Core2Duo CPU T7700@2.4GHz) while slave nodes are built on the S12 boards. The master-slave CAN communication is done through the CANcardXL using a low speed CANcab for 125kbps.
- *Testbed 4:* $Intel\ T7700+4\times TC1797$. The master node is implemented on the same PC as in the previous case while slave nodes are built on the TriCore platform. This time a high speed CANcab is used with the CANcardXL to enable a 1Mbps communication speed.

A different testbed was set up to compare the different variants of the key splitting protocol on a system with 8 slaves based on S12X nodes. Two variants were considered as we further discus: centralized authentication (in this case one extra node was added to act as the master) and two-stage authentication.

3.2 Protocol Performance

Centralized authentication was implemented on the four testbeds prepared for this purpose. Our implementation considers 6 groups of two nodes each formed by combining the four available nodes. Messages and authentication tags are always sent as separate frames and the message size is always 8 bits. The MAC size for each group is set to 21 bits so that 3 authentication tags fit a single 64 bit CAN frame. The MAC is computed using the MD5 hash function over an input formed by concatenating the group key to the message. The resulting hash is then truncated to the desired size. Table 4 holds the timings and bus loads for each test bed. Here δ is the authentication delay, i.e., the time needed by a node to authenticate the message once it receives it. For the bus load we considered the fraction of traffic caused by the authentication tags over the entire bandwidth.

As expected, scenarios in which high end devices played the role of master nodes (PC, TriCore) showed better performance than in the case of low end master nodes. The case of a PC master with TriCore slaves does not perform better, despite the considerable difference in computational power between master nodes (TriCore vs. Intel Core2Duo) due to limitations of CAN adapters. Because of their internal hardware/software design, these adapters introduce some limitations, e.g., the average response time specified by Vector for the CANcardXL is $100\mu s$.

To evaluate the protocol behavior when using different trade-offs we implemented different variants of the key splitting authentication protocol on a system with 8 slaves

built on S12X nodes. By grouping the eight nodes two by two we obtain a total of 28 groups. The size of the authentication tags and the truncated MAC size differ in each variant. We set up the implementations as follows:

- *Centralized:* The message sending node computes and sends one MAC for each group that he is part of. The master computes and sends one MAC for each of the other 21 groups (if groups of size 2 are used). If the master is to perform the authentication in only 2 frames then each MAC can be truncated to 5 bits and this will lead to a total of 35 security bits for each node. But if we increase the number of authentication frames from the master to 3, then each MAC can be truncated to 9 bits giving a total of 63 authentication bits for each node which is a reasonable level for real-time security.
- *Two-stage:* The master node is missing in this implementation, therefore we use two helper nodes for computing and sending the complete authentication tag. In the two-stage variant, the sender node will first put one authentication tag on the bus which contains the full 36 authentication bits for one of the helper nodes, 20 bits for the second one and 8 extra bits for another node. This first tag is followed by a second tag generated by the first helper node which contains the remaining 16 authentication bits for the second helper node and 48 bits equally distributed for three of the remaining nodes. To complete the 36 authentication bits for each of the remaining nodes, the sender node and the second helper node will each put an authentication tag on the bus. As discussed previously, the security level can be raised to around 64 bits by using groups of size 3 and the described tag allocation procedure.

Table 5 holds the results achieved with these two implementations. The worst performer in terms of authentication delay is the implementation of the centralized authentication variant as it involves computing MACs for each of the 28 groups in a sequential manner. In the other implementation, a smaller number of MACs are computed some of which are done by different nodes in parallel. A smaller authentication delay is obtained when using the two-stage implementation at the cost of an increased CPU load on the sender side. However, this cost is somewhat compensated by the higher level of security offered by the fact that the sender node offers more authentication bits.

3.3 Computational Performance with Linearly Mixed Tags

The results from Tables 4 and 5 use the simple concatenation of individual MACs computed with MD5 as the underlying hash function. We now take a brief account of the impact of mixing tags using linear systems of equations, complete experimental results on this will be available in the extended version of our work, here we make an accurate estimation of the computational costs. To begin with, in Table 6 we give an overview on the computational timings for various hash functions and input sizes on both of the employed platforms. For the Linearly Mixed MACs, in addition to the computation of the MACs, two supplemental computational tasks are required: solving the linear system of equations on the sender side (a task which should be usually done by the master which has higher computational power) and reconstructing the MAC on the receiver side. Our experimental results obtained on the communication master equipped with the Intel 2.4GHz core with the well known NTL library (http://www.shoup.net/ntl/) showed that the computational cost of solving the system for 2 nodes in $GF(2^8)$ up to $GF(2^{32})$ are

Table 4. Centralized authentication with 4 nodes

Master	Slave	δ	Bitrate	Bus load
S12X	4xS12X	2.54 ms	125 kbps	53.84%
PC	4xS12X	1.848 ms	125 kbps	72.22%
TriCore	4xTriCore	267 μs	1 Mbps	54.31%
PC	4xTriCore	378 μs	1 Mbps	42.54%

Table 5. Centralized & Cascade with 8 nodes

Variant	Master	Slave	δ	Bus load
Centralized	S12X	8xS12X	22.624 ms	11.27%
Two-stage	-	8xS12X	6.806 ms	46.21%

around 3–6 times more intensive than an MD5 computation and this increases to 10–20 times the MD5 computation in the case of 4 nodes. Since this task should be done by the master node it shouldn't raise computational issues. The reconstruction of the MAC was around 10 times cheaper compared to the linear mixing procedure and compared to MD5 it was in the range of 0.5–5 times more intensive, the later in the case of 8 nodes and $GF(2^{32})$. All these are reasonable amounts of computations and we believe that they can be significantly improved with platform dependent tweaks.

Table 6. Computational performance of employed embedded platforms

Hash function	Input size (bytes)					
	S12			TriCore		
	0	16	64	0	16	64
MD5	371μs	374μs	1414μs	10.16μs	11.00μs	18.34μs
SHA1	1.144ms	1.148ms	4.510ms	14.64μs	15.10μs	27.60μs
SHA256	2.755ms	2.755ms	5.440ms	41.70μs	42.35μs	80.80μs

4 Backward Compatibility with CAN+

There are two main drawbacks to the LiBrA-CAN protocol. First of all, depending on the setup, it can require quite a lot of the CAN bus' bandwidth, and second, all nodes in the system need to be aware of the LiBrA-CAN protocol for it to work. In this section, we show a method of eliminating both of these drawbacks using an unofficial extension of the CAN protocol, called CAN+ [14].

The CAN+ protocol allows transmission of extra data along with a CAN packet on an out-of-band channel. It does this by transmitting data at an increased rate in between CAN sample points. At least 225 extra bits can be transmitted with the CAN+ protocol alongside a CAN message.

Using the CAN+ protocol for LiBrA-CAN data transmission helps in two ways. First of all, the required bandwidth drops. For LiBrA-CAN schemes whereby a single node never needs to transmit more than $\lceil \frac{225}{64} \rceil = 3$ authentication tags, all LiBrA-CAN data can be transmitted as CAN+ data. This reduces the LiBrA-CAN overhead for those schemes to 0%. Nodes that need to transmit just a tag, can do so by transmitting a 0-byte CAN message and embedding the tag as CAN+ data, thereby reducing the time they use

the bus from 108 bit lengths (for an 8-byte message) to 44 bit lengths in non-extended CAN mode, which is a 60% decrease.

Second, if LiBrA-CAN authentication data is only transmitted as CAN+ data, then nodes that do not support CAN+ will not even see the LiBrA-CAN data. Thus, a system can be setup whereby important nodes are outfitted with a CAN+ transceiver, while non-important nodes aren't. This makes the LiBrA-CAN protocol completely backwards compatible with existing CAN networks: nodes supporting CAN+ could be dropped into the network at will and start authentication messages with LiBrA-CAN, while existing CAN nodes will be completely oblivious as to what is going on and continue functioning as before. An added bonus is that this also drastically reduces roll-out cost.

5 Discussion and Conclusions

The proposed protocol is efficient when the number of nodes is low. We expect this to be the case in many automotive scenarios where, although the number of ECUs may be high, the numbers of manufacturers from which they come may not be high and distributing trust between several groups is an acceptable solution. If the number of nodes is too high we see only two resolutions: public key cryptography (with the drawback of high computational requirements, at least 2 orders of magnitude) or TESLA like protocols (with the drawback of authentication delays as shown in [6]). CANAuth [12] is also a solution for high number of nodes if one considers that source authentication is not relevant and associating keys to message groups is sound from a security perspective. While a decision on what protocol should be used for in-vehicle authentication can be taken only by manufacturers and by means of consortium, we believe that this proposal should be considered as an interesting alternative. The use of MAC mixing, key splitting and the features of the CAN arbitration seems to give an efficient management for source authentication.

Acknowledgement. This work was supported by in part by research grants PNII-IDEI 940/2008 and POSDRU 107/1.5/S/77265, inside POSDRU Romania 2007-2013. It was also supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007), by the Flemish IBBT projects, and by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. In addition, it was supported by the Flemish Government, FWO G.0550.12N and by the European Commission through the ICT programme under contract FP7-ICT-2011-284833 PUFFIN. We also thank the reviewers for helpful comments on our work.

References

1. Bar-El, H.: Intra-vehicle information security framework. In: Proceedings of 9th Embedded Security in Cars Conference, ESCAR (2009)
2. Chan, H., Perrig, A., Song, D.X.: Random key predistribution schemes for sensor networks. In: 2003 Proceedings of the Symposium on Security and Privacy, pp. 197–213. IEEE (2003)
3. Charlap, L.S., Rees, H.D., Robbins, D.P.: The asymptotic probability that a random biased matrix is invertible. Discrete Mathematics 82(2), 153–163 (1990)

4. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T.: Comprehensive experimental analyses of automotive attack surfaces. In: USENIX Security 2011 (2011)
5. Fiat, A., Naor, M.: Broadcast Encryption. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 480–491. Springer, Heidelberg (1994)
6. Groza, B., Murvay, P.-S.: Higher layer authentication for broadcast in Controller Area Networks. In: International Conference on Security and Cryptography, SECRYPT (2011)
7. Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S.: Experimental security analysis of a modern automobile. In: 2010 IEEE Symposium on Security and Privacy, SP, pp. 447–462 (May 2010)
8. Naor, M., Pinkas, B.: Threshold Traitor Tracing. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 502–517. Springer, Heidelberg (1998)
9. Perrig, A., Canetti, R., Song, D.X., Tygar, J.D.: SPINS: Security protocols for sensor networks. In: Seventh Annual ACM International Conference on Mobile Computing and Networks, MobiCom 2001, pp. 189–199 (2001)
10. Perrig, A., Canetti, R., Tygar, J.D., Song, D.X.: Efficient authentication and signing of multicast streams over lossy channels. In: IEEE Symposium on Security and Privacy, pp. 56–73 (2000)
11. Roeder, T., Pass, R., Schneider, F.: Multi-verifier signatures. Journal of Cryptology 25(2), 310–348 (2012)
12. Van Herrewege, A., Singelee, D., Verbauwhede, I.: CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus. In: 9th Embedded Security in Cars Conference (2011)
13. Wolf, M., Weimerskirch, A., Paar, C.: Secure in-vehicle communication. In: Embedded Security in Cars, pp. 95–109 (2006)
14. Ziermann, T., Wildermann, S., Teich, J.: CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16x higher data rates. In: DATE, pp. 1088–1093. IEEE (2009)

Efficient Verification of Input Consistency in Server-Assisted Secure Function Evaluation

Vladimir Kolesnikov[1,⋆], Ranjit Kumaresan[2,⋆⋆], and Abdullatif Shikfa[3]

[1] Bell Labs Research, Alcatel-Lucent, Murray Hill, NJ 07974, USA
kolesnikov@research.bell-labs.com
[2] University of Maryland, College Park, MD 20740, USA
ranjit@cs.umd.edu
[3] Bell Labs Research, Alcatel-Lucent, 91620 Nozay, France
abdullatif.shikfa@alcatel-lucent.com

Abstract. We consider generic secure computation in the setting where a semi-honest server assists malicious clients in performing multiple secure two-party evaluations (SFE).

We present practical schemes secure in the above model. The main technical difficulty that we address is efficiently ensuring input consistency of the malicious players across multiple executions. That is, we show how any player can prove he is using the same input he had used in another execution. We discuss applications of our solution, such as online profile matching.

1 Introduction

Secure multiparty computation allows players to compute the value of a multivariate function on their inputs while keeping the inputs private. Generically, the problem of secure computation has been solved [41,12,7], for both semi-honest and malicious players. Since then, extensive body of work concentrated on optimizing these approaches so that they can be used in practice with acceptable overhead.

In this work, we consider a setting where the malicious players wish to securely compute on their inputs and are assisted by a semi-honest server to do so. This setting, although not fully general, is gaining prominence, due to the increasing success of collaborative platforms and online social networks. These

⋆ Supported in part by the Intelligence Advanced Research Project Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) contract Number D11PC20194. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

⋆⋆ Work partly done while the author was visiting Bell Labs. Work partly supported by NSF grant #1111599.

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 201–217, 2012.

platforms offer (and depend upon) a wide variety of applications that would benefit from privacy protection for their users. One such functionality is online profile matching (or match ratio computation). On dating sites, users search for other users whose characteristics/profile match their *desiderata*, while online job portals connect companies and job seekers who best match job openings. Typical applications considered for general SFE, such as auctions, payments, etc., are of interest also in our setting, with the added advantage of possible efficiency gain due to the opportunity to engage the help of the server.

As mentioned, these and other scenarios naturally introduce a third-party, the server, who is in the position to assist the users with their computation, and who can also protect users' identities. Asking the server to perform the computation himself implies revealing private data, which is often sensitive and needs protection. At the same time, the server is an established business and often can be trusted not to deviate from the prescribed protocol. It is therefore reasonable to assume that the server is semi-honest.

In our approach, we will use Yao's Garbled Circuit (GC) [41] as the basic tool. See Section 3.1 for the description of the technique. Having access to the semi-honest server resolves the problem of malicious circuit generation. We also rely on Oblivious Transfer (OT) secure against malicious receiver and semi-honest sender. Combination of the two gives us a natural solution for our setting (see Section 2 for more details). This solution is complete for the standard standalone executions, usually considered in SFE research.

1.1 Input Consistency Verification

In this work we consider multiple SFE executions. One issue that arises here, and which is not addressed by the standard SFE model is that of input consistency between executions.

The Need for Input Consistency. We first argue the importance of input consistency verification. We consider several motivating examples first.

Consider profile matching and match ratio computation, also considered, e.g., in [38]. These are the underlying functionalities in online dating, resume/job matching, profiling for advertisement and other services, etc. In many of these applications, it is critical to the business model that users cannot manipulate their inputs to extract maximum benefit, but, rather, that user's inputs are consistent among executions.

Consider, for example, the online dating application, where Alice and Bob evaluate their compatibility by creating (and sometimes modifying) their profiles and matching them to their preferences. This process may be interactive, and the functions of interest may be adaptively selected. It may be important that the corresponding inputs provided in these functions, are chosen consistently. For example, if Bob's private profile indicates that he is working on Ph.D. in cryptography, and he later finds out that Alice likes kittens, he should not be able to later claim a veterinary or firefighting degree.

Similarly, if one system user, a corporation, is running a promotion campaign targeting a certain demographic, other users should not be able to improperly adjust their profiles to take advantage of the promotion.

High-Level Approach. One natural approach to this issue is to have the server certify the players' profiles by issuing a certificate. This works well in limited circumstances, but not always. There are several problems with this approach. Firstly, certification is often understood as involving verification and approval. While certifying profile characteristics, such as *Age* is reasonable, certifying *favorite color* may look unusual. Further, some characteristics, such as *favorite color* are dynamic, and may change during the lifetime of the system. Finally, sometimes a need arises in considering a personal feature which was not expected to be in the profile, as its usefulness may have been discovered during the interactive profile matching.

We consider a more light-weight approach, where no inputs are certified by anybody. However, once a certain input had been used by Alice in communication with Bob, Bob can always ask her to supply the same input in future communication, and vice versa. In our previous example, once Bob supplied the profile that indicated he studies cryptography, Alice will be able to ensure that in all future SFE where the field of studies is involved, Bob will input *cryptography*.

Similarly, if Alice communicated with corporation `CoffeeCorp`, and Alice's profile indicated she has graduated, she will not be able to participate in `CoffeeCorp`'s promotion for free coffee for college students.

We note that the fact that some inputs may change over the course of time will not break the fundamentals of our approach. The user with changed profile attribute might simply inform the other user, if needed, that a particular attribute was updated. The computation will go through, and other user will simply be additionally informed of the changed input.

1.2 Our Setting

We summarize our setting, which we motivated in the previous section.

Two parties, Alice and Bob want to evaluate a function, without disclosing their inputs to each other. Either of the parties can be maliciously corrupted. They are assisted by a semi-honest server $\mathcal{S}$ who does not collude with any of the other players. $\mathcal{S}$ has no input, and he obtains no output in the computation. We allow both Alice and Bob to verify, with the help of the server, that, for two SFE evaluations, a particular input wire is set to the same plaintext value.

We do not discuss which input wires are allowed to be checked, and how Alice and Bob agree on which wires they are checking. We assume that this agreement is reached over an insecure channel, and disagreement in this matter will simply result in non-participation in the SFE and/or input verification protocols.

In particular, importantly, we will not allow a player to verify consistency of two inputs of the other player without the other player's consent.

1.3 Related Work

We consider SFE with malicious players, who use the help of a semi-honest server. Most relevant to us is a comparatively small body of work that provides improvements in settings similar to ours. We mention, but do not discuss in detail here the works that specifically concentrate on the malicious setting, such as [27,21,35,29,37,34,5,8]. This is because malicious-secure protocols are much more costly than the protocols we are considering.

The issue of input consistency comes up in secure two-party computation. Cut-and-choose, a very popular technique, requires evaluation of a security-parameter number of circuits, and a consistency check among all the inputs of all the evaluated circuits [31,27,37,29]. This requires a quadratic-complexity solution[1] [31,27] and solves a harder problem than ours, since we have the help of the semi-honest server.

This server-assisted computation model has been considered as early as [11], where the authors consider players A and B who wish to let a third party C learn the output of their computation. The helping oblivious server has often been appealed to in circumstances where such a player is natural, and where regular two- or multi-party SFE would have been too costly. One example is that of auctions [33], where secure computation is achieved with the help of two non-colluding servers. Here one (semi-honest) server creates garbled circuit implementing the auction, and the other (malicious) server evaluates it based on the inputs of the clients, which were submitted through a proxy Oblivious Transfer protocol. Several protocols have also been developed for the special case of secure auctions using only one server [17,6,9,10]. More recently, [15] argues that the server model is well-suited for the web (where clients connect and interact only once with servers, and simultaneous availability of all clients is not possible) and present several protocols for a number of functions of interest. Other recent works [23,30,3] also consider secure computation in the server-assisted model but allow the server to be malicious. As mentioned earlier, protocols secure against malicious parties are much more expensive than the protocols we are considering. [22] similarly uses the helping server to overcome the non-simultaneous nature of survey submissions in survey processing. This work is incomparable to ours because, firstly, most of it concentrates on specific functions of interest (e.g., auctions). More importantly, our distinguishing feature is the input consistency verification across several executions, which is not considered in these works.

Input consistency checking is recognized in security literature as an important ingredient in system building. Zero-knowledge proofs on commitments and related techniques are often used in consistency checking. For example, [16] consider privacy-preserving data mining and identify the need for input consistency checking in computing specific functions of interest. They further propose to solve it by involving expensive public key techniques. This body of work is incomparable to ours, because, firstly, they consider specific problems, their solutions are often informal and presented without proofs (e.g. that of [16]), and further

[1] An expander-graph-based linear-complexity solution [40] is also available, but it is more costly for practical parameters.

rely on expensive public-key tools. Some of their ideas (e.g., [16] using same randomness in encrypting same inputs), however, are related to our approach.

Finally, input consistency can be achieved via Certificate Authority (CA) issuing credentials for players' inputs. However, as we discussed in Introduction, this approach is not sufficiently general, and is more costly than our proposed approach.

In contrast with all of the above approaches, we show how to ensure input consistency across several executions while only relying on a small number of symmetric-key primitives, and minimal additional storage by the players only (one bit per input to be cross-referenced), and no additional storage by the server other than one master secret of security-parameter length.

1.4 Our Contributions and Outline of the Work

We propose a quite general solution to the reactive SFE among two malicious players and the helping non-colluding semi-honest server. Our main technical contribution is a technique to ensure input consistency among several executions. Our solution is very efficient and is comparable to that of Yao's Garbled Circuit (GC) in the semi-honest model.

We start our presentation with a high-level description of our technical idea in Section 2. We then provide the overview of the preliminaries in Section 3, before presenting the detailed protocol and proof in Section 4. We discuss several natural extensions in Section 5.

2 Overview of Our Approach

We base our solution on Yao's GC [41] (and its state-of-the-art optimizations such as garbled row reduction [36], free-XOR [25]). GC is secure against malicious circuit evaluator and semi-honest circuit constructor, therefore we will have the semi-honest server $\mathcal{S}$ generate the garbled circuit for the chosen function (as communicated to $\mathcal{S}$ by both clients). As for inputs, we will use OT extension [20] secure against malicious receivers and semi-honest server [18]. Each player runs above OT with the server to obtain wire secrets corresponding to their input. Then they send these wire secrets to the other player (and receive the other player's input secrets). The computed GC is then sent by $\mathcal{S}$ to both players for evaluation (it is important to send the GC after the inputs have been delivered so that, e.g., players cannot abort based on the output of SFE). At this point, each player can complete GC evaluation and compute their output.

The above is a complete solution with the exception of the input consistency verification.

Our main contribution is precisely the method for input consistency verification. Our main idea is as follows. The input wire secrets in the constructed (by $\mathcal{S}$) garbled circuit will encode their corresponding plaintext values according to a secret stored by $\mathcal{S}$. This can be done, for example, by $\mathcal{S}$ choosing and storing a random bit b_i and setting the last bit of the 0-secret of wire i to b_i and the last

bit of 1-secret to be $\neg b_i$. Now, when, say, Bob, receives Alice's wire secret from Alice, he will store the last bit of the wire of interest. Note that effectively the plaintext value of this wire is shared between $\mathcal{S}$ and Bob. Now, when Bob wishes to confirm that plaintext values of two of Alice's wires across two executions are the same, he simply needs to compare the XOR of the two values he stored with the XOR of the corresponding values stored by $\mathcal{S}$. If the XOR values are the same, then Alice supplied the same input. Indeed, in both good-behavior cases (Alice supplying either $0,0$ or $1,1$ in the two executions), Bob's stored bits will XOR to the XOR of the two stored bits of $\mathcal{S}$. We stress that this check can be done "in plaintext", i.e., simply by $\mathcal{S}$ sending the corresponding XOR value to Bob. This approach is symmetrically applied to both players.

Finally, we note that the server generates the encoding bits b_i using a PRFG, so he does not need to store any of the plaintext encoding bits, as he can always regenerate them from his master secret, client ids and SFE id. We note that including client ids into the circuit generation seed derivation is important. If not included, two malicious players P_1, P_2 might open an honest Alice's input of execution C_i by pretending that C_i was their prior execution.

We now make several observations regarding our presentation and the result.

Observation 1. *We stress that since we encode the wire-secret to wire-key correspondence in a single bit of the wire key, it is important that this correspondence can not be violated by a malicious player. For example, a semantically secure encryption may ignore the last bit of the key. If such an encryption were used in GC, a malicious player could flip the last bit and later falsely convince the verifier of input consistency. To prevent this, we ensure that players cannot malleate the received wire keys by having the server $\mathcal{S}$ send hashes of all wire keys to both players.*

Observation 2. *In our formal presentation, we will consider functions F that output the same value to both Alice and Bob. Our theorems and protocols can be naturally extended to cover the general case of the multi-output functionalities.*

Observation 3. *While we concentrate our discussion on* input *consistency, we can verify consistency of any wires of the evaluated circuits using a natural generalization of our approach.*

Observation 4. *The server will aid in consistency verification only if it is approved by both players by correspondingly conveying their consent to $\mathcal{S}$. We do not discuss how players know which wires are to be checked, we assume this is given to players as an additional and insecure input.*

Observation 5. *The server will aid in SFE only if the evaluated circuit is approved by both players by correspondingly sending the (identical to each other) circuit description $\mathcal{S}$. We do not discuss how players know which function they wish to compute; we assume this is given to players as an additional and insecure input.*

Observation 6. *Our protocols achieve* fairness *with respect to both clients. Recall, fairness requires that both participants of SFE learn the output simultaneously. In other words, players are not allowed to abort early after learning the output. Full, and even partial fairness is quite expensive to achieve (see e.g., [13,14]) in the standard two-party setting, and full fairness comes "for free" in our setting.*

Observation 7. *Each player only needs to remember the bit corresponding to a single instance of a specific semantic input (say the first one) to achieve comparison. For example, Alice will need to remember only one instance of bits corresponding to Bob's age. Indeed, all future uses of a particular semantic input can be compared to its first use.*

3 Preliminaries and Notation

3.1 Garbled Circuits (GC)

Yao's Garbled Circuit approach [41], excellently presented in [28], is the most efficient method for one-round secure evaluation of a boolean circuit C. We summarize its ideas in the following. The circuit *constructor* $\mathcal{S}$ creates a *garbled circuit* $\widetilde{C}$: for each wire w_i of the circuit, he randomly chooses two garblings $\widetilde{w}_i^0, \widetilde{w}_i^1$, where $\widetilde{w}_i^j$ is the *garbled value* of w_i's value j. (Note: $\widetilde{w}_i^j$ does not reveal j.) Further, for each gate G_i, $\mathcal{S}$ creates a *garbled table* $\widetilde{T}_i$ with the following property: given a set of garbled values of G_i's inputs, $\widetilde{T}_i$ allows to recover the garbled value of the corresponding G_i's output, but nothing else. $\mathcal{S}$ sends these garbled tables, called *garbled circuit* $\widetilde{C}$ to the *evaluator* $\mathcal{C}$. Additionally, $\mathcal{C}$ obliviously obtains the *garbled inputs* $\widetilde{w}_i$ corresponding to inputs of both parties: the garbled inputs $\widetilde{x}$ corresponding to the inputs x of $\mathcal{S}$ are sent directly and $\widetilde{y}$ are obtained with a parallel 1-out-of-2 oblivious transfer (OT) protocol [32,2,28]. Now, $\mathcal{C}$ can evaluate the garbled circuit $\widetilde{C}$ on the garbled inputs to obtain the *garbled outputs* by evaluating $\widetilde{C}$ gate by gate, using the garbled tables $\widetilde{T}_i$. Finally, $\mathcal{C}$ determines the plain values corresponding to the obtained garbled output values using an output translation table received from $\mathcal{S}$. Correctness of GC follows from the way garbled tables $\widetilde{T}_i$ are constructed.

We note that GC evaluator cannot deviate from the prescribed protocol, and GC is therefore secure against malicious GC evaluator, given an appropriate malicious-secure OT protocol.

3.2 Notation

Let κ be the computational security parameter. The server $\mathcal{S}$ assists parties P_1 and P_2 to secure evaluate arbitrary functions over their inputs multiple times. In each iteration, $\mathcal{S}$ will be provided circuit C_i that party P_1 with client id id_1, and input x_i, and party P_2 with client id id_2, and input y_i, wish to evaluate. We let $x_{i,j}$ (resp. $y_{i,j}$) denote the j-th bit of x_i (resp. y_i). We assume that x_i

(resp. y_i) is of length m (resp. n), and that I_1 (resp. I_2) represents the set of P_1's (resp. P_2's) input wires in C_i.

Server $\mathcal{S}$ maintains a master secret – a state, denoted by σ, across executions. Given a circuit C_i, and state σ, the server uses algorithm $\mathsf{GarbGen}(i, C_i, \mathsf{id}_1, \mathsf{id}_2, \sigma)$ to generate a garbled version of C_i which we denote by $\widetilde{C}_i$.

In circuit C_i, we let $u_{i,j}$ (resp. $v_{i,j}$) denote the j-th input wire belonging to party P_1 (resp. P_2). For a wire $u_{i,j}$ (resp. $v_{i,j}$), we refer to the garbled values corresponding to 0 and 1 by $\widetilde{u}^0_{i,j}, \widetilde{u}^1_{i,j}$ (resp. $\widetilde{v}^0_{i,j}, \widetilde{v}^1_{i,j}$) respectively. While evaluating the garbled circuit, the evaluator will possess only one of two garbled values for each wire in the circuit. We let $\widetilde{w}'_{i,j}$ denote the garbled value on wire $w_{i,j}$ that is possessed by the evaluator.

Our protocols are designed in the random oracle model. In our protocols H, E, and H' represent hash functions that are modeled as non-programmable random oracles.

4 Input Consistency Verification in Server-Assisted SFE

We start this section with the definition of security. Then, in Section 4.2, we present a simple protocol for server-assisted SFE secure against malicious players. This is the natural protocol based on GC. Finally, in Section 4.3, we show how to allow to perform input consistency checks.

4.1 Definitions

We provide a formal definition of the SFE functionality we will realize. We want P_1 and P_2 to securely compute with the help of the server, and, in addition, to be able to check each other's inputs for consistency. Our functionality provides two interfaces, evaluate and check, which, respectively, evaluate the given circuit, and checks two inputs for consistency. It is easy to see that such a functionality provides the utility we desire, i.e., allows P_1 to ensure consistency of P_2's inputs, and vice versa.

Definition 1. *The* consistency checking functionality $\mathcal{F}_{\mathsf{cc}}$ *is a reactive functionality that interacts with a server* $\mathcal{S}$ *and parties* P_1 *and* P_2 *in the following way.*

- *If it receives an* evaluate *request from both parties, then* $\mathcal{F}_{\mathsf{cc}}$ *obtains* (i, C_i, x_i) *from* P_1*, and* (i, C_i, y_i) *from* P_2*. If the values* i, C_i*, provided by* P_1 *and* P_2 *differ, then* $\mathcal{F}_{\mathsf{cc}}$ *returns* $\perp$ *to* $\mathcal{S}$*.* $\mathcal{F}_{\mathsf{cc}}$ *records* (i, x_i, y_i)*. Then* $\mathcal{F}_{\mathsf{cc}}$ *returns* $(i, z_i = C_i(x_i, y_i))$ *to both* P_1 *and* P_2*. Finally,* $\mathcal{F}_{\mathsf{cc}}$ *returns* (i, C_i) *to* $\mathcal{S}$*.*
- *If it receives a* check *request from both parties, then* $\mathcal{F}_{\mathsf{cc}}$ *obtains the same string* (i_1, j_1, i_2, j_2) *from both parties.* $\mathcal{F}_{\mathsf{cc}}$ *returns* (i_1, j_1, i_2, j_2) *to* $\mathcal{S}$*. Let* $w_{i,j}$ *denote the plaintext value carried on* j*-th wire in circuit* C_i *when evaluated using inputs* x_i *and* y_i*.* $\mathcal{F}_{\mathsf{cc}}$ *checks if* $w_{i_1,j_1} = w_{i_2,j_2}$*, and returns* pass *to both parties if the check passed, else it returns* fail *to both parties.*

We say that a stateful protocol π is a server-assisted secure computation protocol that allows consistency checking across multiple sequential executions if it securely realizes $\mathcal{F}_{\mathsf{cc}}$ in the presence of an adversary $\mathcal{A}$.

Remark: Note that the above definition reveals each circuit C_i as well as each check request (i_1, j_1, i_2, j_2) to the semi-honest server. This is allowed in our model, and is consistent with the standard definitions of SFE security. However, even this information can be hidden, and we discuss natural ways to do so in Section 5.

4.2 Server-Assisted Secure Computation

In this section, we describe a simple protocol for secure computation that supports multiple executions in the server-assisted setting. Our protocol is a natural adjustment of the secure computation protocols based on garbling schemes [41,4]. The main idea of the protocol of this section is that the semi-honest server will generate the GC and distribute it to both players, after running the OT protocol. We note that we do not yet address input consistency in this section, and the following protocol is simply a building block. We will use the protocol of this section as a subprotocol in the scheme presented in Section 4.3.

In this protocol, $\widetilde{C}_i$ used by $\mathcal{S}$ is constructed as described in Section 3.1. Let input keys for P_1 be $\{\widetilde{u}^0_{i,j}, \widetilde{u}^1_{i,j}\}_{j\in I_1}$, and those corresponding to P_2 be $\{\widetilde{v}^0_{i,j}, \widetilde{v}^1_{i,j}\}_{j\in I_2}$. We describe our protocol below.

Protocol 1

1. $\mathcal{S}$ and P_1 participate in m OT instances in the following way. In the j-th instance:
   - $\mathcal{S}$ acts as sender with input $(\widetilde{u}^0_{i,j}, \widetilde{u}^1_{i,j})$.
   - P_1 acts as receiver with input $x_{i,j}$.
   - P_1 obtains $\widetilde{u}'_{i,j}$ as output.
2. $\mathcal{S}$ and P_2 participate in n OT instances in the following way. In the j-th instance:
   - $\mathcal{S}$ acts as sender with input $(\widetilde{v}^0_{i,j}, \widetilde{v}^1_{i,j})$.
   - P_2 acts as receiver with input $y_{i,j}$.
   - P_2 obtains $\widetilde{v}'_{i,j}$ as output.
3. For each $j \in I_2$, $\mathcal{S}$ sends $H(\widetilde{v}^0_{i,j}), H(\widetilde{v}^1_{i,j})$ in random order to P_1. Let P_1 receive these as $\{g_{i,j}, g'_{i,j}\}_{j\in I_2}$.
4. For each $j \in I_1$, $\mathcal{S}$ sends $H(\widetilde{u}^0_{i,j}), H(\widetilde{u}^1_{i,j})$ in random order to P_2. Let P_2 receive these as $\{h_{i,j}, h'_{i,j}\}_{j\in I_1}$.
5. P_1 sends $\{\widetilde{u}'_{i,j}\}_{j\in I_1}$ to P_2.
6. P_2 sends $\{\widetilde{v}'_{i,j}\}_{j\in I_2}$ to P_1.
7. P_1 aborts the protocol if for some $j \in I_2$, $H(\widetilde{v}'_{i,j}) \notin \{g_{i,j}, g'_{i,j}\}$ holds.
8. P_2 aborts the protocol if for some $j \in I_1$, $H(\widetilde{u}'_{i,j}) \notin \{h_{i,j}, h'_{i,j}\}$ holds.
9. $\mathcal{S}$ sends the garbled circuit $\widetilde{C}_i$ to both P_1 and P_2.
10. Using keys $\{\widetilde{v}'_{i,j}\}_{j\in I_2}$ and $\{\widetilde{u}'_{i,j}\}_{j\in I_1}$, P_1 and P_2 evaluate $\widetilde{C}_i$ to obtain output z_i.

Intuition for Security. (A formal proof is included in the proof of Theorem 1.) We informally argue that Protocol 1 allows for multiple secure evaluations in the presence of an adversary that either passively corrupts $\mathcal{S}$, or actively corrupts one of P_1, P_2.

Security against semi-honest $\mathcal{S}$. Note that the server does not learn what keys are sent by P_1 to P_2 and vice versa. Furthermore, by receiver-security of OT, the keys selected by P_1 and P_2 (in their respective OT executions) is computationally hidden from the server. This is enough to guarantee security against a semi-honest $\mathcal{S}$.

Security against malicious P_1. The simulator extracts P_1's inputs by using the simulator of the secure OT protocol. It then checks whether the keys $\{\widetilde{u}'_{i,j}\}_{j\in I_1}$ correspond to the extracted input. If not, the simulator aborts the simulation at this stage, and outputs whatever P_1 outputs. If the checks pass, then the simulator sends the extracted input to the trusted party. Upon receiving the output back from the trusted party, the simulator "fakes" the garbled circuit, and provides the appropriate output translation tables. This completes an informal description of the simulation. Indistinguishability of simulation follows from the fact that H is a random oracle, and the encryption scheme (used to create garbled tables) is semantically secure, and the security of OT protocol.

Security against malicious P_2 follows by an argument similar to the above.

We will use Protocol 1 as a subprotocol to construct our main protocol that also enables consistency verification in addition to secure computation.

4.3 Verifying Consistency across Multiple Executions

Our main technical contribution is a design of a new garbling scheme that will allow efficient consistency verification in our setting. Recall that Yao's garbled circuit is constructed by choosing for each wire $w_{i,j}$, garblings $\widetilde{w}^0_{i,j}, \widetilde{w}^1_{i,j}$ at random from $\{0,1\}^\kappa$, and creating the garbled tables $\widetilde{T}_{i,j}$ using any semantically-secure encryption scheme.

GC Encryption. In our GC garbling schemes, we employ the following encryption. For simplicity of presentation, we work in the random oracle model.

Let $E : \{0,1\}^* \to \{0,1\}^\kappa$ be a random oracle. For encrypting the value x in the truth table of the ℓ-th gate in the i-th execution, we use the following encryption scheme that takes two keys k_a, k_b as follows:

$$\mathsf{Enc}_{k_a,k_b}(x,i,\ell) = E(k_a \| k_b \| i \| \ell) \oplus x$$

Before presenting our main protocol, we describe our main amendment to the traditional Yao GC-based garbling scheme (and their benefits) that we take advantage of.

Correlating the Keys across Multiple Executions. We achieve verifiable consistency across multiple executions by correlating the keys at the input level.

More precisely, the server $\mathcal{S}$ chooses at random his master secret $\sigma \in \{0,1\}^\kappa$, permanently stores it, and uses it in the following way to covertly "mark" each input wire with its plaintext label. Let P_1's (resp. P_2) id be id_1 (resp. id_2), and $H' : \{0,1\}^* \to \{0,1\}$ be a one-bit RO. For $w_{i,j}$ that is the input wire of either P_1 or P_2: (1) the first $\kappa - 1$ bits of $\widetilde{w}^0_{i,j}, \widetilde{w}^1_{i,j}$ are picked at random, and (2) the last bit of $\widetilde{w}^0_{i,j}$ is set to $H'(i\|j\|\mathsf{id}_1\|\mathsf{id}_2\|\sigma)$, while the last bit of $\widetilde{w}^1_{i,j}$ is set to $\widetilde{w}^0_{i,j}$'s complement $1 \oplus H'(i\|j\|\mathsf{id}_1\|\mathsf{id}_2\|\sigma)$. As we will formally show below, correlating the keys in the manner described above will allow for efficient consistency verification. We stress that the remaining keys (i.e., those that do not correspond to input or output wires of C_i) are still picked at random from $\{0,1\}^\kappa$.

We are now ready to formalize the above discussion and to describe $\mathsf{GarbGen}$ which is the algorithm $\mathcal{S}$ uses to create the garbled circuit for the i-th execution. The algorithm $\mathsf{GarbGen}$ takes the execution index i, the circuit C_i, and the server's state (master secret) σ to produce garbled circuit $\widetilde{C}_i$.

In $\mathsf{GarbGen}$, we generate the wires garblings at random. We note that in practice, we would probably use a PRFG such as AES.

Algorithm $\mathsf{GarbGen}(i, C_i, \mathsf{id}_1, \mathsf{id}_2, \sigma)$**.**
In C_i, let $u_{i,j}$ and $v_{i,j}$ represent the input wires corresponding to P_1 whose client id is id_1 and P_2 whose client id is id_2 respectively.

- For every $w_{i,j}$ that is an input wire of either P_1 or P_2, do the following:
  1. Set $\widehat{w}_{i,j} := H'(i\|j\|\mathsf{id}_1\|\mathsf{id}_2\|\sigma)$. (Recall, H''s output is one-bit.)
  2. Choose $r_0, r_1 \leftarrow \{0,1\}^{\kappa-1}$ at random.
  3. Set $\widetilde{w}^0_{ij} := r_0\|\widehat{w}_{i,j}$.
  4. Set $\widetilde{w}^1_{ij} := r_1\|(1 \oplus \widehat{w}_{i,j})$.
- For every internal wire $w_{i,j}$ of C_i, choose $\widetilde{w}^0_{i,j}, \widetilde{w}^1_{i,j} \leftarrow \{0,1\}^\kappa$.
- For each gate G in C_i do the following: Let the gate index of G be ℓ. Suppose w_1 and w_2 represent input wires, and w_3 represent the output wires of gate G. For $j \in \{1,2,3\}$, let $\widetilde{w}^0_j, \widetilde{w}^1_j$ represent the garblings corresponding to 0 and 1 respectively. Given this, the garbled table $\widetilde{T}$, corresponding to gate G with gate function g, in $\widetilde{C}_i$ consists of a random permutation of the set $\{E(\widetilde{w}^{b_1}_1\|\widetilde{w}^{b_2}_2\|i\|\ell) \oplus \widetilde{w}^{g(b_1,b_2)}_3\}_{b_1,b_2 \in \{0,1\}}$. (Recall E is a random oracle.)

We are now ready to describe our final protocol that enables efficient verification of consistency across multiple executions. We refer the reader to technical overview of the protocol and its intuition presented in Section 2.

Protocol 2

Setup: $\mathcal{S}$ chooses random seed $\sigma \leftarrow \{0,1\}^\kappa$.

Evaluation: In the i-th execution:

- P_1 and P_2 provide C_i to $\mathcal{S}$. If submissions of P_1 and P_2 differ, the server aborts.
- $\mathcal{S}$ creates $\widetilde{C}_i \leftarrow \mathsf{GarbGen}(i, C_i, \mathsf{id}_1, \mathsf{id}_2, \sigma)$.

- $\mathcal{S}$ uses $\widetilde{C}_i$ as the garbled circuit, and participates in Protocol 1 with P_1 and P_2. At the end of the protocol, P_1 and P_2 obtain their respective outputs of the execution.
- For $w_{i,j}$ that represents the input wire of either P_1 or P_2, both P_1 and P_2 do the following.
  - Set $\widehat{w}'_{i,j}$ to the last bit of $\widetilde{w}'_{i,j}$.
  - Add $\widehat{w}'_{i,j}$ to the local state.

Consistency Verification:

- Both P_1 and P_2 specify (i_1, j_1, i_2, j_2) to $\mathcal{S}$. If submissions of P_1 and P_2 differ, the server aborts.
- $\mathcal{S}$ retrieves $\widehat{w}_{i_1,j_1} \leftarrow H'(i_1 \| j_1 \| \mathsf{id}_1 \| \mathsf{id}_2 \| \sigma)$, and $\widehat{w}_{i_2,j_2} \leftarrow H'(i_2 \| j_2 \| \mathsf{id}_1 \| \mathsf{id}_2 \| \sigma)$. $\mathcal{S}$ sends the bit $(\widehat{w}_{i_1,j_1} \oplus \widehat{w}_{i_2,j_2})$ to both P_1 and P_2. Denote the bit received by P_1 and P_2 as c.
- P_1 and P_2 retrieve $\widehat{w}'_{i_1,j_1}$ and $\widehat{w}'_{i_2,j_2}$ from their local state, and check if $c \stackrel{?}{=} \widehat{w}'_{i_1,j_1} \oplus \widehat{w}'_{i_2,j_2}$. If the check fails then the execution is aborted.

This completes the description of our protocol. We stress that unlike P_1 and P_2, the server $\mathcal{S}$ does not store any local state other than the master secret σ. We now provide the proof of security.

Theorem 8. *Let $\mathcal{A}$ be an adversary that either passively corrupts $\mathcal{S}$ or actively corrupts one of P_1, P_2. Then, Protocol 2 securely realizes $\mathcal{F}_{\mathsf{cc}}$ in the presence of $\mathcal{A}$.*

Proof. (sketch) **Security against Semi-honest $\mathcal{S}$.** We start with showing that the protocol is secure against the semi-honest server $\mathcal{S}$. The information received by $\mathcal{S}$ are transcripts of the underlying OTs. This does not leak information, because OTs are secure against semi-honest $\mathcal{S}$. Given this, the simulator $\mathsf{Sim}_{\mathcal{S}}$ follows naturally.

Secure against Malicious P_1^*. We present the simulator Sim_1 of a malicious P_1^*, and argue that it produces a good simulation.

Sim_1 chooses random seed $\sigma \leftarrow \{0,1\}^\kappa$. Sim_1 does the following in each iteration i.

Sim_1 first obtains $\widetilde{C}_i$ by running $\mathsf{GarbGen}(i, C_i, \sigma)$. Then, Sim_1 starts P_1^* and interacts with it, sending it messages it expects to receive, and playing the role of the trusted party for the OT oracle calls that P_1^* makes, in which P_1^* plays the role of the receiver.

Sim_1 plays OT trusted party m times, where P_1^* is the receiver; as such, Sim_1 receives all m OT selection bits (which are supposed to correspond to P_1^*'s input) from P_1^* and each time, for concreteness say j, uses $\{\widetilde{u}^0_{i,j}, \widetilde{u}^1_{i,j}\}$ (which are P_1's input keys specified by $\mathsf{GarbGen}$) to hand to P_1^* as his OT output. Let us denote this output by $\widetilde{u}_{i,j}$. If any of the underlying OTs abort, then Sim_1 sends abort to the trusted party and halts, outputting whatever P_1^* outputs.

Acting as $\mathcal{S}$, the simulator Sim_1 chooses random κ-bit string r_j, and sends $r_j, H(\widetilde{v}^0_{i,j})$ in random order. Then, acting as P_2, Sim_1 receives $\{\widetilde{u}'_{i,j}\}_{j \in I_1}$ from

P_1^*, and sends $\{\widetilde{v}^0_{i,j}\}_{j\in I_2}$ to P_1^*. Next, Sim_1 checks if for every $j \in I_1$, it holds that $\widetilde{u}'_{i,j} = \widetilde{u}_{i,j}$. If any of the checks fail, then the simulator sends abort to the trusted party, and terminates outputting whatever P_1^* outputs. Otherwise, Sim_1 forms x_i^* in the following way. For each $j \in I_1$, if $\widetilde{u}_{i,j} = \widetilde{u}^0_{i,j}$, then $x^*_{i,j}$ is set to 0; else it is set to 1. Then, Sim_1 feeds x_i^* as its input to the trusted party. Sim_1 gets back the output z_i from the trusted party.

Sim_1 creates a "fake" garbled circuit $\widetilde{C}'_i$ which is exactly the same as an honestly generated garbled circuit $\widetilde{C}_i$ (created using $\mathsf{GarbGen}$) except with the following modification. Let G denote an output gate with input wires w_1, w_2, and output wire w_3. Let G's gate index be ℓ. Recall Sim_1 obtained output z_i from the trusted party, so it knows the actual value b that is carried on the output wire w_3. Sim_1 creates garbled gate $\widetilde{T}$ as a random permutation of the set $\{E(\widetilde{w}_1^{b_1}\|\widetilde{w}_2^{b_2}\|i\|\ell)\oplus\widetilde{w}_3^b\}_{b_1,b_2\in\{0,1\}}$. That is, the "fake" garbled circuit, upon evaluation, will always yield the correct output z_i. Sim_1 sends the fake garbled circuit $\widetilde{C}'_i$ to P_1^*. This completes the description of the simulation of the evaluation phase.

We now describe the simulation of the consistency verification stage. Acting as $\mathcal{S}$, the simulator Sim_1 receives query (i_1, j_1, i_2, j_2) from P_1^*. Sim_1 checks if this is a valid query, and if not, it aborts the protocol, and terminates the simulation outputting whatever P_1^* outputs. Otherwise, Sim_1 returns $H(i_1\|j_1\|\mathsf{id}_1\|\mathsf{id}_2\|\sigma)\oplus H(i_2\|j_2\|\mathsf{id}_1\|\mathsf{id}_2\|\sigma)$ to both parties.

We now argue that Sim_1 produces a view indistinguishable from the real execution in both the evaluation and verification stages. We first note that Sim_1's interaction with P_1^* is indistinguishable from that of honest $\mathcal{S}$ and P_2. Indeed, OT secrets delivered to P_1^* are distributed identically to real execution. Further, since non-selected OT secrets remain hidden, P_1^* knows the value of exactly one key in $\{\widetilde{u}^0_{i,j}, \widetilde{u}^1_{i,j}\}$ for each of his input wires $j \in I_1$. Thus, if the execution is not aborted, then P_1^* must have sent the key that he obtained via OT with Sim_1 (acting as $\mathcal{S}$). Also, the fake garbled circuit sent to P_1^* from Sim_1 is indistinguishable from real, since the underlying encryption scheme is based on RO and produces uniform and independent ciphertexts.

Now we argue that the simulation of the consistency verification stage is indistinguishable from the real execution. Let (i_1, j_1, i_2, j_2) represent the query that was sent by P_1. First, we consider the case when wires j_1 and j_2 carry P_2's inputs. Since valid queries over (honest) P_2's inputs are actually consistent in the real execution, and since Sim_1 generates keys honestly according to $\mathsf{GarbGen}$, we conclude that Sim_1's answer is indistinguishable from $\mathcal{S}$'s answer in the real execution. Now suppose wires j_1 and j_2 carry P_1's inputs. Since an RO collision happens with negligible probability, we are guaranteed that the key $\widetilde{u}'_{i,j}$ that P_1 sent to P_2 is exactly the one that P_1 retrieved via OT. Indistinguishability of the simulation follows from the fact that Sim_1 answers the query honestly based on keys generated via $\mathsf{GarbGen}$. This completes the proof of correctness of the simulation when P_1^* is malicious.

The case when P_2 is malicious is symmetric and is skipped. ■

5 Extensions

As observed earlier, our definition of $\mathcal{F}_{\mathsf{cc}}$ functionality reveals information about the verification queries and the circuit C_i to the server $\mathcal{S}$. While, as we discussed, this is not a security violation, in some use cases, it is desired to hide this information as well. In this section, we discuss natural approaches to hiding this information.

Private Verification Queries. We provide a simple solution for preserving privacy of verification queries. Recall that in order to verify input consistency, parties need to retrieve the XOR of the least significant bits of the wire keys corresponding to wires specified in their queries. Note that these least significant bits can be obtained directly from the private state σ (i.e. the master secret) of $\mathcal{S}$. This motivates the following solution that preserves privacy of verification queries. Consider a circuit C' which takes as input queries q_1, q_2 from parties P_1 and P_2, and the private state σ of $\mathcal{S}$, and computes the desired output (i.e., XOR of the least significant bits of the keys specified by the queries). Clearly, if C' is evaluated securely, i.e., while keeping queries q_1, q_2 private from $\mathcal{S}$, and private state σ hidden from P_1 and P_2, then our problem is solved. We propose the following efficient solution to securely evaluate C' using garbled circuits.

We model H' as a PRF (as opposed to RO) in order to allow C' to internally generate the keys from σ. (We stress that modeling H' as a PRF does not violate the security of our construction in any way.) We also require C' to check if $q_1 = q_2$ holds, and produce output only when this check passes. This is necessary in order to guarantee that the malicious party does not obtain information other than what the output of the honest query reveals.

Now, without loss of generality, suppose party P_1 wishes to verify input consistency. Parties simply securely evaluate C' on corresponding inputs, and use the output of this computation as discussed in consistency verification subprotocol of Protocol 2 described in Section 4.3. Clearly, $\mathcal{S}$ will not know which wires are being verified. We note that two colluding players P_1 and P_2 will not obtain output related to a third player, since H' used for evaluation of the marker bits is evaluated on inputs which include both client ids.

We stress that the above solution is very efficient[2]; in particular its complexity is independent of the number of past executions between P_1 and P_2.

Function Privacy. We employ standard techniques such as universal circuits [39,1,26,24] to preserve function privacy. Our application is mildly complicated by the fact that we need to ensure that both parties provide the same function descriptors as input to the universal circuit. This is done to prevent a malicious party from evaluating an arbitrary function over the honest party's inputs. We resolve this issue using techniques similar to the ones we employed when privacy of queries needed to be preserved.

In more detail, let U' denote a circuit that takes as input two function descriptors f_1, f_2, and two inputs x and y. Circuit U' checks if $f_1 = f_2$, and if

[2] See [19] for the concrete cost of securely evaluating AES.

so, evaluates a universal circuit U on input f_1, x, y to produce output $f_1(x, y)$. Clearly, if U' is evaluated securely, i.e., while keeping function descriptors f_1, f_2 private from $\mathcal{S}$, then our problem is solved. Secure evaluation of U' is performed in the same way as described in the setting where privacy of queries needed to be preserved.

References

1. Abadi, M., Feigenbaum, J.: Secure circuit evaluation. Journal of Cryptology 2(1), 1–12 (1990)
2. Aiello, W., Ishai, Y., Reingold, O.: Priced Oblivious Transfer: How to Sell Digital Goods. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 119–135. Springer, Heidelberg (2001)
3. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (2012)
4. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. To Appear at ACM CCS 2012 (2012)
5. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic Encryption and Multiparty Computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 169–188. Springer, Heidelberg (2011)
6. Cachin, C.: Efficient private bidding and auctions with an oblivious third party. In: Proceedings of the 6th ACM Conference on Computer and Communications Security, pp. 120–127. ACM, New York (1999)
7. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: 20th Annual ACM Symposium on Theory of Computing, pp. 11–19. ACM Press (May 1988)
8. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty Computation from Somewhat Homomorphic Encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012)
9. Di Crescenzo, G.: Private Selective Payment Protocols. In: Frankel, Y. (ed.) FC 2000. LNCS, vol. 1962, pp. 72–89. Springer, Heidelberg (2001)
10. Di Crescenzo, G.: Privacy for the Stock Market. In: Syverson, P.F. (ed.) FC 2001. LNCS, vol. 2339, pp. 269–288. Springer, Heidelberg (2002)
11. Feige, U., Kilian, J., Naor, M.: A minimal model for secure computation (extended abstract). In: STOC 1994: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, pp. 554–563. ACM (1994)
12. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game, or a completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th Annual ACM Symposium on Theory of Computing, pp. 218–229. ACM Press (May 1987)
13. Gordon, S.D., Hazay, C., Katz, J., Lindell, Y.: Complete fairness in secure two-party computation. In: Ladner, R.E., Dwork, C. (eds.) 40th Annual ACM Symposium on Theory of Computing, pp. 413–422. ACM Press (May 2008)
14. Gordon, S.D., Katz, J.: Partial Fairness in Secure Two-Party Computation. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 157–176. Springer, Heidelberg (2010)

15. Halevi, S., Lindell, Y., Pinkas, B.: Secure Computation on the Web: Computing without Simultaneous Interaction. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 132–150. Springer, Heidelberg (2011)
16. Han, S., Ng, W.K.: Preemptive measures against malicious party in privacy-preserving data mining. In: SIAM International Conference on Data Mining, pp. 375–386 (2008)
17. Harkavy, M., Tygar, J.D., Kikuchi, H.: Electronic auctions with private bids. In: Proceedings of the 3rd Conference on USENIX Workshop on Electronic Commerce, vol. 3. USENIX Association, Berkeley (1998)
18. Harnik, D., Ishai, Y., Kushilevitz, E., Nielsen, J.B.: OT-Combiners via Secure Computation. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 393–411. Springer, Heidelberg (2008)
19. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: USENIX Security (2011)
20. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending Oblivious Transfers Efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
21. Jarecki, S., Shmatikov, V.: Efficient Two-Party Secure Computation on Committed Inputs. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 97–114. Springer, Heidelberg (2007)
22. Feigenbaum, J., Pinkas, B., Ryger, R., Saint-Jean, F.: Secure computation of surveys. In: EU Workshop on Secure Multiparty Protocols (2004)
23. Kamara, S., Mohassel, P., Raykova, M.: Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272 (2011)
24. Katz, J., Malka, L.: Constant-Round Private Function Evaluation with Linear Complexity. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 556–571. Springer, Heidelberg (2011)
25. Kolesnikov, V., Schneider, T.: Improved Garbled Circuit: Free XOR Gates and Applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
26. Kolesnikov, V., Schneider, T.: A Practical Universal Circuit Construction and Secure Evaluation of Private Functions. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 83–97. Springer, Heidelberg (2008)
27. Lindell, Y., Pinkas, B.: An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)
28. Lindell, Y., Pinkas, B.: A proof of security of Yao's protocol for two-party computation. Journal of Cryptology 22(2), 161–188 (2009)
29. Lindell, Y., Pinkas, B.: Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 329–346. Springer, Heidelberg (2011)
30. Lopez-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: 44th Annual ACM Symposium on Theory of Computing, pp. 1219–1234. ACM Press (2012)
31. Mohassel, P., Franklin, M.: Efficiency Tradeoffs for Malicious Two-Party Computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 458–473. Springer, Heidelberg (2006)
32. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: 12th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 448–457. ACM-SIAM (January 2001)

33. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Proceedings of the 1st ACM Conference on Electronic Commerce, pp. 129–139. ACM, New York (1999)
34. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A New Approach to Practical Active-Secure Two-Party Computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Heidelberg (2012)
35. Nielsen, J.B., Orlandi, C.: LEGO for Two-Party Secure Computation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 368–386. Springer, Heidelberg (2009)
36. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure Two-Party Computation Is Practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
37. Shelat, A., Shen, C.-H.: Two-Output Secure Computation with Malicious Adversaries. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 386–405. Springer, Heidelberg (2011)
38. Shikfa, A., Önen, M., Molva, R.: Broker-Based Private Matching. In: Fischer-Hübner, S., Hopper, N. (eds.) PETS 2011. LNCS, vol. 6794, pp. 264–284. Springer, Heidelberg (2011)
39. Valiant, L.: Universal circuits (preliminary report). In: STOC, pp. 196–203. ACM Press (1976)
40. Woodruff, D.P.: Revisiting the Efficiency of Malicious Two-Party Computation. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 79–96. Springer, Heidelberg (2007)
41. Yao, A.: How to generate and exchange secrets (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science, pp. 162–167. IEEE Computer Society Press (October 1986)

Fast and Private Computation of Cardinality of Set Intersection and Union

Emiliano De Cristofaro[1], Paolo Gasti[2,⋆], and Gene Tsudik[3]

[1] Palo Alto Research Center
edc@parc.com
[2] New York Institute of Technology
pgasti@nyit.edu
[3] University of California Irvine
gts@ics.uci.edu

Abstract. In many everyday scenarios, sensitive information must be shared between parties without complete mutual trust. Private set operations are particularly useful to enable sharing information with privacy, as they allow two or more parties to jointly compute operations on their sets (e.g., intersection, union, etc.), such that only the minimum required amount of information is disclosed. In the last few years, the research community has proposed a number of secure and efficient techniques for Private Set Intersection (PSI), however, somewhat less explored is the problem of computing the *magnitude*, rather than the contents, of the intersection – we denote this problem as Private Set Intersection Cardinality (PSI-CA). This paper explores a few PSI-CA variations and constructs several protocols that are more efficient than the state-of-the-art.

1 Introduction

Proliferation of, and growing reliance on, electronic information generate an increasing amount of sensitive data stored and processed in the cyberspace. Consequently, there is a compelling need for efficient cryptographic techniques that allow sharing information while protecting privacy. Among these, Private Set Intersection (PSI) [14,28,17,25,18,26,11,10,21], and Private Set Union (PSU) [28,18,20,15,35] have recently attracted a lot of attention from the research community. In particular, PSI allows one party (client) to compute the intersection of its set with that of another party (server), such that: (i) server does not learn client input, and (ii) client learns no information about server input, beyond the intersection. Efficient PSI protocols have been used as building blocks for many privacy-oriented applications, e.g., collaborative botnet detection [31], denial-of-service identification [2], on-line gaming [7], intelligence-community systems [23], location sharing [32], just to cite a few.

Nonetheless, in certain information-sharing settings, PSI and PSU functionalities offer very limited privacy to server. Consider the following scenario where,

⋆ Work done while at University of California, Irvine

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 218–231, 2012.

after running PSI, the set intersection learned by client corresponds to entire server input: server privacy is actually non-existent, while client's is fully preserved. This illustrates the need for server to enforce a policy, based on the cardinality of set intersection/union, that governs whether it is willing to take part in PSI or PSU protocols. (We explore this intuition in Section 6.)

This paper investigates Private Set Intersection Cardinality (PSI-CA) and Private Set Union Cardinality (PSU-CA). These functionalities are appealing in scenarios where client is only allowed to learn the *magnitude* – rather than the *content* – of set intersection/union. For instance, PSI-CA is useful in social networking, e.g., when two parties want to privately determine the number of common connections (or interests) in order to decide whether or not to become friends. Moreover, PSI-CA is useful to privately compare equal-size low-entropy vectors, e.g., to realize private computation of Hamming Distance between two strings on an arbitrarily large alphabet: two parties may use PSI-CA, by treating each symbol, together with its position in the string, as a unique set element, such that client privately learns the number of elements (symbols) in common (thereby also obtaining the Hamming Distance). Other relevant applications of PSI-CA include role-based association rule mining [27], affiliation-hiding authentication [3], as well as to estimating the similarity of sample sets [6]. Finally, efficient PSI-CA protocols are becoming instrumental to privacy-preserving genomic tests, as recently showed in [4].

1.1 Contributions

This paper focuses on PSI-CA – a cryptographic primitive, involving server (on input of a private set $\mathcal{S}$) and client (on input of a private set $\mathcal{C}$), that results in client outputting $|\mathcal{S} \cap \mathcal{C}|$. Computation of PSI-CA naturally implies that of PSU-CA, since $|\mathcal{S}|, |\mathcal{C}|$ are always mutually disclosed and $|\mathcal{S} \cup \mathcal{C}| = |\mathcal{S}| + |\mathcal{C}| - |\mathcal{S} \cap \mathcal{C}|$.

Although prior work has yielded some PSI-CA techniques (see Section 2), a number of open problems still remain to be addressed. This paper presents the following contributions:

1. We present a very efficient PSI-CA protocol that incurs computational and communication complexities linear in the set sizes. Our protocol is secure under the DDH assumption in the Random Oracle Model (ROM) against semi-honest adversaries. This protocol is a very close variant of the protocol of Agrawal, Evfimievski, and Srikant [1], and our merit is really a security analysis of this modification rather than the protocol itself.
2. We introduce the concept of Authorized PSI-CA (APSI-CA), whereby client input must be pre-authorized by an off-line mutually-trusted authority, and present an appropriate protocol extension with linear complexities (as opposed to quadratic in related prior techniques).
3. We show how to combine PSI-CA with PSI such that server can decide whether to allow client to obtain the set intersection according to its policy, based on the size of the intersection itself (privately obtained using PSI-CA). This first-of-a-kind approach is very efficient, as it requires only *one* additional message on top of PSI-CA protocol.

Paper Organization. Next section reviews related work. After preliminaries in Section 3, Section 4 presents our PSI-CA protocol. Then, Section 5 constructs a variant for APSI-CA, and finally, Section 6 sketches a three-round policy-based PSI variant. The paper concludes in Section 7.

2 Related Work

2.1 (Authorized) Private Set Intersection and Union

Agrawal, Evfimievski, and Srikant [1] introduce a Private Set Intersection (PSI) construction based on commutative encryption.[1] The protocol has linear complexity – that is, assuming that server and client sets contain w and v items, respectively, computation and communication complexity amounts to $O(w+v)$. [1] also presents a variant that only discloses the size of the intersection – we review it in Section 2.2 below.

The work in [14] propose a few protocols for Private Set Intersection (PSI) based on Oblivious Polynomial Evaluations (OPE-s) and additively homomorphic encryption (e.g., Paillier [33]). The intuition is to represent a set as a polynomial and its elements – as the polynomial's roots. Client encrypts the coefficients, that are then evaluated homomorphically by server. As a result, client learns the intersection and nothing else. Client's computation complexity amounts to $O(w+v)$, and server's to $O(wv)$ exponentiations. [14] proposes techniques to asymptotically reduce server workload to $O(w \log\log v)$, by using Horner's rule and balanced bucket allocation. [18] obtains similar complexities while also offering PSU techniques. Whereas, [28] extends OPE-s to more than two players, all learning the intersection/union, with quadratic computational and linear communication complexities. Additional PSU constructs appear in [20,15,35].

Other PSI constructs, such as [17,25], rely on Oblivious Pseudo-Random Functions (OPRF-s) and reduce computation overhead to a *linear* number of exponentiations. Recent results in the Random Oracle Model (ROM) have led to very efficient PSI protocols, also with linear complexities, while using much more efficient cryptographic tools. They replace OPRFs with unpredictable functions [26] and blind signatures [11], with security under *One-More-DH* and *One-More-RSA* assumptions [5], respectively. Finally, [10] achieves linear communication and computational complexities, using short exponents, with security in the malicious model, while [21] shows a construction in the semi-honest model based on garbled circuits [38] which, leveraging so-called Oblivious Transfer Extension [24], scales relatively gracefully for very large security parameters.

Authorization of client input in PSI has been first investigated in [8] and [9]. Authorized Private Set Intersection (APSI) is later formalized in [11] and [10] that construct efficient techniques with linear complexity in the presence of,

[1] It is quite interesting to observe that several PSI papers (e.g., [14,26]) erroneously cite the work by Evfimievski, Gerke, and Srikant [12] as the work introducing commutative-encryption based PSI, which is, in fact, [1]. Also, observe that protocols in [1] are essentially the same as those sketched earlier, in [22], although the latter provided no security analysis.

respectively, semi-honest and malicious adversaries. Finally, [36] proposes Policy-Enhanced PSI, allowing two parties to privately share information while enforcing complex policies. In this model, both parties' sets must be authorized, and both parties obtain the intersection.

2.2 Private Set Intersection Cardinality

Prior work yielded several PSI-CA protocols:

- Agrawal, Evfimievski, and Srikant [1] present an adaptation of their PSI protocol to PSI-CA, also secure under the DDH assumption in the presence of semi-honest adversaries. Their construction is actually similar to ours (presented in Figure 1), although we also present two extensions.
- The PSI protocol by Freedman, Nissim, and Pinkas [14] can be extended to PSI-CA with the same complexity, i.e., $O(w \log\log v)$ computation and $O(w + v)$ communication.
- Hohenberger and Weis [19] present a PSI-CA construction, also based on [14], and with similar (sub-quadratic) complexities.
- Kissner and Song [28] proposes a PSI-CA protocol for multiple ($n \geq 2$) parties, incurring $O(n^2 \cdot v)$ communication and $O(v^2)$ computational overhead.
- Vaidya and Clifton [37] construct a multi-party PSI-CA protocol, based on commutative one-way hash functions [30] and Pohlig-Hellman encryption [34]. It incurs n rounds, and involves $O(n^2 \cdot v)$ communication and $O(vn)$ computational overhead.
- Camenisch and Zaverucha [8] present an APSI variant (private intersection of certified sets) that computes the cardinality of (certified) set intersection and incurs *quadratic* communication and computation complexity.

3 Preliminaries

This section defines PSI-CA/PSU-CA functionalities, along with their privacy requirements, and introduces computational assumptions.

DDH Assumption. Let $\mathbb{G}$ be a cyclic group and g be its generator. We assume that bit-length of group size is l. The DDH problem is hard in $\mathbb{G}$ if, for every efficient algorithm $\mathcal{A}$, the following probability is a negligible function of κ:

$$\left|\Pr[x, y \leftarrow \{0,1\}^l : \mathcal{A}(g, g^x, g^y, g^{xy}) = 1] - \Pr[x, y, z \leftarrow \{0,1\}^l : \mathcal{A}(g, g^x, g^y, g^z) = 1]\right|$$

Definition 1 (Private Set Union Cardinality (PSU-CA)). *A protocol involving server, on input a set of w items $\mathcal{S} = \{s_1, \ldots, s_w\}$, and client, on input a set of v items $\mathcal{C} = \{c_1, \cdots, c_v\}$. It results in the latter outputting $|\mathcal{U}|$, where: $\mathcal{U} = \mathcal{S} \cup \mathcal{C}$.*

Definition 2 (Private Set Intersection Cardinality (PSI-CA)). *A protocol involving server, on input a set of w items $\mathcal{S} = \{s_1, \ldots, s_w\}$, and client, on input a set of v items $\mathcal{C} = \{c_1, \cdots, c_v\}$. It results in the latter outputting $|\mathcal{I}|$, where: $\mathcal{I} = \mathcal{S} \cap \mathcal{C}$.*

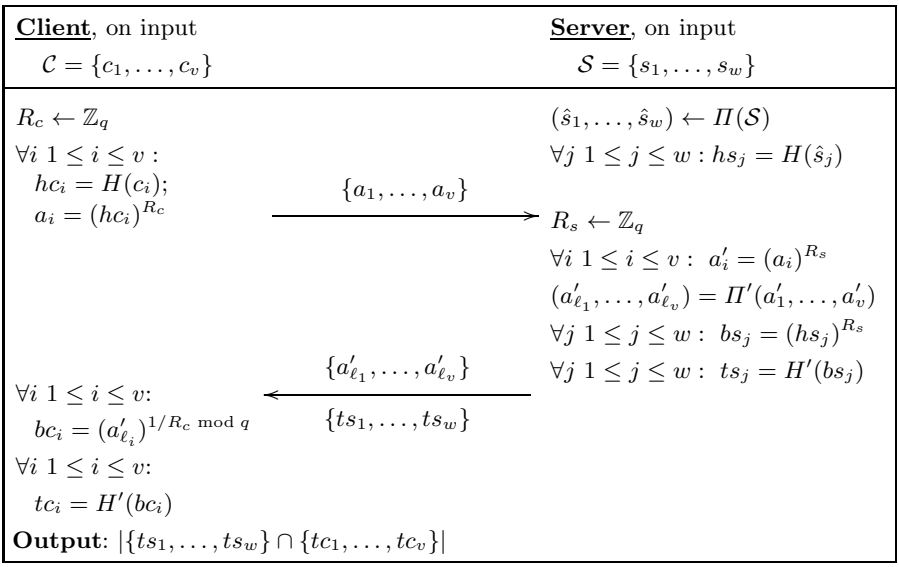


Fig. 1. Proposed PSI-CA Protocol. All computation is mod p. Π and Π' are random permutations

Informally, both PSI-CA and PSU-CA entail the following privacy requirements:

- *Server Privacy.* Client learns no information beyond: (1) cardinality of set intersection/union and (2) upper bound on the size of $\mathcal{S}$.
- *Client Privacy.* No information is leaked about client set $\mathcal{C}$, except an upper bound on its size.
- *Unlinkability.* Neither party can determine if any two instances of the protocol are related, i.e., executed on the same input by client or server, unless this can be inferred from the actual protocol output.

Remark: As mentioned earlier in the paper, for any $\mathcal{C}$ and $\mathcal{S}$, the size of $\mathcal{C} \cup \mathcal{S}$ can be computed as $|\mathcal{C}| + |\mathcal{S}| - |\mathcal{C} \cap \mathcal{S}|$. Thus, privately computing cardinality of the *intersection* of $\mathcal{C}$ and $\mathcal{S}$ allows one to privately compute the cardinality of their *union* as well. Consequently, the rest of the paper only focuses on PSI-CA.

4 New PSI-CA and PSU-CA

This section presents our PSI-CA construction, secure in the presence of semi-honest adversaries in the Random Oracle Model (ROM). We outline it in Figure 1. Protocol executes on common input of two primes p, q (where $q|p-1$), a generator g of a subgroup of size q, and two hash functions (modeled as random oracles), $H : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$ and $H' : \{0,1\}^* \rightarrow \{0,1\}^\kappa$, given the security parameter κ. (Notation $a \leftarrow A$ denotes that a is chosen uniformly at random from A).

Intuition. First, client masks its set items (c_i-s) with a random exponent (R_c) and sends resulting values (a_i-s) to server, which "blindly" exponentiates them with its own random value R_s. Server shuffles the resulting values (a'_i-s) and sends them to client. Then, server sends client the output of a one-way function, $H'(\cdot)$, computed over the exponentiations of server's items (s_j-s) to randomness R_s. Finally, client tries to match one-way function outputs received from server with one-way function outputs computed over the shuffled (a'_i-s) values, stripped of the initial randomness R_c. *Client* learns the set intersection cardinality (and nothing else) by counting the number of such matches. As showed below, unless they correspond to items in the intersection, one-way function outputs received from server cannot be used by client to learn related items in server's set (under the DDH assumption). Also, client does not learn *which* items are in the intersection as the matching occurs using *shuffled* a'_i values.

Complexity. Protocol complexity is *linear* in the sizes of the two sets. Let $|\mathcal{S}| = w$ and $|\mathcal{C}| = v$. Client performs $2(v+1)$ exponentiations with short, i.e., $|q|$-bit, exponents modulo $|p|$-bit and v modular multiplications. Server computes $(v+w)$ modular exponentiations with short exponents and w modular multiplications. In practice, one can select $|p| = 1024$ or $|p| = 2048$, and $|q| = 160$ or $|q| = 224$. Communication overhead amounts to $2(v+1)$ $|p|$-bit and w κ-bit values.

Semi-Honest Participants. We start with security in the semi-honest model. Note that the term *adversary* refers to insiders, i.e., protocol participants. Outside adversaries are not considered, since their actions can be mitigated via standard network security techniques.

Definition 3 (Correctness). *If both parties are honest, at the end of the protocol, executed on inputs* $((\mathcal{S}, v), (\mathcal{C}, w))$*, server outputs* $\perp$*, and client outputs* $(|\mathcal{S} \cap \mathcal{C}|)$*.*

The following client and server privacy definitions follow from those in related work [14,13,17]. In particular, as formalized in [16] (Sec. 7.2.2), in case of semi-honest parties, the traditional "real-versus-ideal" definition framework is *equivalent* to a much simpler framework that extends the formulation of honest-verifier zero-knowledge. Informally, a protocol privately computes certain functionality if whatever can be obtained from one party's view of a protocol execution can be obtained from input and output of that party. In other words, the view of a semi-honest party (including $\mathcal{C}$ or $\mathcal{S}$, all messages received during execution, and the outcome of that party's internal coin tosses), on each possible input $(\mathcal{C}, \mathcal{S})$, can be efficiently simulated considering only that party's own input and output.

Definition 4 (Client Privacy). *Let* $\mathsf{View}_S(\mathcal{C}, \mathcal{S})$ *be a random variable representing server's view during execution of PSI-CA with inputs* $\mathcal{C}, \mathcal{S}$*. There exists a* PPT *algorithm* S^* *such that:*

$$\{S^*(\mathcal{S}, |\mathcal{S} \cap \mathcal{C}|)\}_{(\mathcal{C},\mathcal{S})} \stackrel{c}{\equiv} \{\mathsf{View}_S(\mathcal{C}, \mathcal{S})\}_{(\mathcal{C},\mathcal{S})}$$

Definition 5 (Server Privacy). *Let* $\mathsf{View}_C(\mathcal{C},\mathcal{S})$ *be a random variable representing client's view during execution of PSI-CA with inputs* $\mathcal{C},\mathcal{S}$*. There exists a* PPT *algorithm* C^* *such that:*

$$\{C^*(\mathcal{C},|\mathcal{S}\cap\mathcal{C}|)\}_{(\mathcal{C},\mathcal{S})} \stackrel{c}{\equiv} \{\mathsf{View}_C(\mathcal{C},\mathcal{S})\}_{(\mathcal{C},\mathcal{S})}$$

In other words, on each possible pair of inputs $(\mathcal{C},\mathcal{S})$, client's view can be efficiently simulated by C^* on input: $\mathcal{C}$ and $|\mathcal{S}\cap\mathcal{C}|$ (as well as v,w). Thus, as in [16], we claim that the two distributions implicitly defined above are computationally indistinguishable. (Notation "$\stackrel{c}{\equiv}$" indicates computational indistinguishability.)

We claim that the protocol in Figure 1 is correct under Definition 3 and secure under Definitions 4 and 5 above. Proof of such claims is provided next.

4.1 Proofs

Correctness. For any c_i held by client and s_j held by server, if $c_i = s_j$, hence, $hc_i = hs_j$, we obtain:

$$tc_{\ell_i} = H'(bc_{\ell_i}) = H'({a_{\ell_i}}^{(1/R_c)}) = H'({hc_i}^{R_s}) = H'({hs_j}^{R_s}) = H'(bs_j) = ts_j$$

Hence, client learns set intersection cardinality by counting the number of matching pairs (tc_{ℓ_i}, ts_j). □

Client Privacy. We claim that the views of server – i.e., $\mathcal{S}$ and $a_i = H(c_i)^{R_c}$ for $i = 1,\ldots,v$ where H is modeled as a random oracle – is indistinguishable from $r_1,\ldots,r_v$ with $r_i \leftarrow \mathbb{Z}_p$. Therefore it is possible to construct a PPT algorithm S^* such that $\{S^*(\mathcal{S},|\mathcal{S}\cap\mathcal{C}|)\}_{(\mathcal{C},\mathcal{S})} \stackrel{c}{\equiv} \{\mathsf{View}_S(\mathcal{C},\mathcal{S})\}_{(\mathcal{C},\mathcal{S})}$.

When $v = 1$, for any $hc_1 = H(c_1)$ there exists R_{c_1} such that $a_1 = {hc_1}^{R_{c_1}}$. Therefore, a_1 is uniformly distributed – i.e., distributed identically to r_1.

For $v \geq 2$, elements $a_1,\ldots,a_v$ are indistinguishable from $r_1,\ldots,r_v$ assuming the hardness of DDH. In particular, the existence of an efficient distinguisher $\mathcal{D}$ that outputs 0 when presented with $r_1,\ldots,r_v$ and outputs 1 when it observes $a_1,\ldots,a_v$ allows us to construct a simulator $\overline{\mathsf{SIM}}_s$ that violates the DDH assumption, as follows.

Upon receiving a DDH challenge $(\bar{g},\bar{g}^x,\bar{g}^y,\bar{g}^z)$, $\overline{\mathsf{SIM}}_s$:

- Selects random set $\overline{\mathcal{C}}$ composed of v elements $\overline{\mathcal{C}} = \{c_1,\ldots,c_v\}$, $v-2$ random values $d_1,\ldots,d_{v-2}$ from $\mathbb{Z}_q$ and R_c at random from $\mathbb{Z}_q$.
- Sends $\{\overline{a}_1,\ldots,\overline{a}_v\} = \{\bar{g}^y,\bar{g}^z,(\bar{g}^y)^{d_1},\ldots,(\bar{g}^y)^{d_{v-2}}\}$ to $\mathcal{D}$.
- Answers queries for H as follows: $H(c_1) = \bar{g}$; $H(c_2) = \bar{g}^x$; $H(c_i) = \bar{g}^{d_{i-2}}$ for $3 \leq i \leq v$ and with a random value otherwise. Queries and responses to H are stored by $\overline{\mathsf{SIM}}_S$ for consistency.

Note that if $(\bar{g},\bar{g}^x,\bar{g}^y,\bar{g}^z)$ is a Diffie-Hellman tuple, i.e. $z = xy$, then $\overline{a}_1,\ldots,\overline{a}_v$ is distributed like $a_1,\ldots,a_v$; thus, $\mathcal{D}$ must output 1. If $(\bar{g},\bar{g}^x,\bar{g}^y,\bar{g}^z)$ is not a Diffie-Hellman tuple, then $\overline{a}_1,\ldots,\overline{a}_v$ is not properly distributed (since $a_2 \neq (H(c_2))^y$)

and therefore $\mathcal{D}$ must output 0. As a result, $\overline{\mathsf{SIM}}_s$ can use $\mathcal{D}$'s output to respond to the DDH challenge correctly iff $\mathcal{D}$'s output is correct. Therefore, $\mathcal{D}$ can only answer correctly with negligible advantage over random guessing. □

Server Privacy. We show that client's view can be efficiently simulated by a PPT algorithm SIM_C, i.e., $\{\mathsf{SIM}_C(\mathcal{C}, |\mathcal{S} \cap \mathcal{C}|)\}_{(\mathcal{C},\mathcal{S})} \stackrel{c}{\equiv} \{\mathsf{View}_C(\mathcal{C}, \mathcal{S})\}_{(\mathcal{C},\mathcal{S})}$. The simulator is constructed as follows:

1. SIM_C builds two tables $T_1 = (u, h)$ and $T_2 = (u', h')$ to answer the H and H' queries respectively. SIM_C responds to a query u (resp. u') with a value in $h \leftarrow \mathbb{Z}_p$ for H (resp. $h' \leftarrow \mathbb{Z}_p$ for H'), and stores (u, h) in T_1 ((u', h') in T_2 resp.). SIM_C uses T_1, T_2 to respond consistently to queries from client.
2. SIM_C constructs a set $TS = \{ts_1, \ldots, ts_w\}$, where $ts_i \leftarrow \{0,1\}^\kappa$, and a random subset $TS' = \{ts'_1, \ldots, ts'_{|\mathcal{S}\cap\mathcal{C}|}\} \subseteq TS$, such that $|TS'| = |\mathcal{S} \cap \mathcal{C}|$.
3. Then, SIM_C adds $|\mathcal{S} \cap \mathcal{C}|$ distinct pairs $(H(c_i)^{R_s}, ts'_i \in TS')$ to T_2 and continues to answer queries to H and H' consistently using T_1 and T_2 as defined in Step 1.
4. Upon receiving $\{a_1, \ldots, a_v\}$ from client, SIM_C picks $R_s \leftarrow \mathbb{Z}_q$ and computes $a'_i = a_i^{R_s}$. Finally SIM_C sends $\Pi'(a'_i, \ldots, a'_v)$ and $\{ts_1, \ldots, ts_w\}$ to client.

Any efficient semi-honest client C^* cannot distinguish between an interaction with an honest server with input $\mathcal{S} = \{s_1, \ldots, s_w\}$ and SIM_C.

By construction, C^*'s view differs from the interaction with an honest server only in the way elements $\{ts_1, \ldots, ts_w\}$ are constructed. Let distinguisher $\mathcal{D}$ be an algorithm that outputs 0 on input an element from distribution:

$$\begin{aligned} D_0 = \{&(H(s_1), \ldots, H(s_w)), \Pi(ts_1 = H'(H(s_1)^{R_s}), \ldots, ts_w = H'(H(s_w)^{R_s})), \\ &(a_1 = H(c_1)^{R_c}, \ldots, a_v = H(c_v)^{R_c}), \Pi'(a'_1 = H(c_1)^{R_c R_s}, \ldots, \\ &a'_v = H(c_v)^{R_c R_s})\}_{hs_i} \end{aligned}$$

and 1 on input an element from:

$$\begin{aligned} D_1 = \big\{&(hs_1, \ldots, hs_w), \Pi\big(ts_1 = H'(H(c_1)^{R_s}), \ldots, ts_{|\mathcal{S}\cap\mathcal{C}|} = H'(H(c_{|\mathcal{S}\cap\mathcal{C}|})^{R_s}), \\ &ts_{|\mathcal{S}\cap\mathcal{C}|+1} = H'(r_{|\mathcal{S}\cap\mathcal{C}|+1}), \ldots, ts_w = H'(r_w)\big), \\ &(a_1 = H(c_1)^{R_c}, \ldots, a_v = H(c_v)^{R_c}), \\ &\Pi'(a'_1 = H(c_1)^{R_c R_s}, \ldots, a'_v = H(c_v)^{R_c R_s})\big\}_{hs_i} \end{aligned}$$

with $r_{|\mathcal{S}\cap\mathcal{C}|+1}, \ldots, r_w$ random elements from $\mathbb{Z}_p$ and where $\mathcal{D}$ is allowed to select the elements in sets $\mathcal{C} = \{c_1, \ldots, c_v\}$ and $\mathcal{S} = \{s_1, \ldots, s_v\}$. The existence of $\mathcal{D}$ violates the hardness assumption of DDH: Let (g, g^x, g^y, g^z) be a DDH challenge for simulator $\overline{\mathsf{SIM}}$, which interacts with $\mathcal{D}$ as follows: $\overline{\mathsf{SIM}}$ responds to $H(x)$ queries from $\mathcal{D}$ with g^{rh_i} for a random $rh_i \in \mathbb{Z}_q$, and stores (x, g^{rh_i}) in table T_H for consistency and to queries $H'(x)$ with a random string, using $T_{H'}$ to store queries-response for consistency.

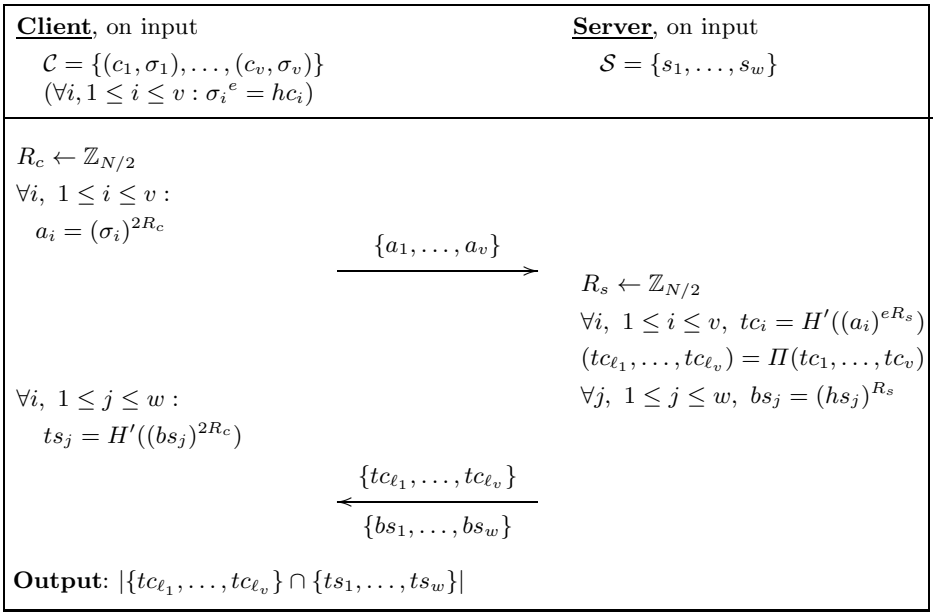


Fig. 2. Authorized PSI-CA. All computation is mod N

W.l.o.g., let $H(c_i) = g^{rh_i}$; $\overline{\mathsf{SIM}}$ computes $a'_i = (g^y)^{rh_i \cdot R_c}$ and constructs $ch = ((g, g^x, g^{r_3}, \ldots, g^{r_w}), \Pi(ts_1 = H'(g^y), ts_2 = H'(g^z), ts_3 = H'((g^y)^{r_3}), \ldots, ts_w = H'((g^y)^{r_w})), (a_1, \ldots, a_v), (a'_1, \ldots, a'_v))$ with $r_3, \ldots, r_w$ random elements in $\mathbb{Z}_p$. Note that ch belongs to distribution D_0 iff (g, g^x, g^y, g^z) is a proper Diffie-Hellman tuple, i.e., $z = xy$ and to D_1 otherwise. Moreover, while $\mathcal{D}$ can test for which elements $H'(a'_i) = ts_j$, pairs i, j are distributed as expected because of the permutation Π'. Therefore, $\mathcal{D}$ has only negligible advantage in distinguishing the two distributions. □

5 Fast Authorized PSI-CA

We now introduce the concept of Authorized PSI-CA (APSI-CA). It extends "plain" PSI-CA to enforce (pre-)*authorization* of client input. Similar to APSI [11] (reviewed in Section 2), APSI-CA involves an offline trusted third party – Certification Authority (CA) – that provides client with authorizations (in practice, signatures) to input into the set intersection cardinality computation.

Definition 6 (Authorized PSI-CA (APSI-CA)). *A protocol involving a server, on input of a set of w items: $\mathcal{S} = \{s_1, \cdots, s_w\}$, and a client, on input of a set of v items with associated authorizations (i.e., signatures), $\mathcal{C} = \{(c_1, \sigma_i) \cdots, (c_v, \sigma_v)\}$. It results in client outputting $|\mathcal{I}^*|$, where:*

$$\mathcal{I}^* = \{s_j \in \mathcal{S} \mid \exists (c_i, \sigma_i) \in \mathcal{C} \ s.t.\ c_i = s_j \wedge \mathsf{Verify}(\sigma_i, c_i) = 1\}.$$

APSI-CA entails the following informal privacy requirements:

- *Server Privacy (APSI-CA).* The client learns no information beyond what can be inferred from the protocol output, i.e., (1) cardinality of set intersection on authorized items and (2) upper bound on the size of $\mathcal{S}$ (the server could conceivably add "dummies" to its input; such dummies do not alter the output of the protocol, but conceal the exact number of elements in the server's set).
- *Client Privacy (APSI-CA).* No information is leaked about items or authorizations in client set (except an upper bound on their number).
- *Unlinkability.* Similar to PSI-CA, we require that neither server nor client can determine if any two instances of the protocol are related, i.e., executed on the same input by client or server.

We illustrate our APSI-CA protocol in Figure 2. Observe that the CA is responsible for generating all public parameters: on input the security parameter κ, it executes $(N, e, d, g) \leftarrow RSA.KGen(\kappa)$, where g is a generator of QR_N, and selects $H : \{0,1\}^* \rightarrow \mathbb{Z}_N{}^*$ (Full-Domain Hash) and $H' : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ (random oracles). The CA authorizes client input c_i by issuing $\sigma_i = H(c_i)^d \bmod N$ (i.e., an RSA signature). The protocol is executed between client and server, on common input (N, e, H, H'). We assume that server's input $(\mathcal{S})$ is randomly permuted before protocol execution to mask any *ordering* of the items contained in it. Finally, hc_i and hs_j denote, respectively, $H(c_i)$ and $H(s_j)$.

Similar to its PSI-CA counterpart, this APSI-CA has the following properties:

- ***Correctness.*** For any (σ_i, c_i) held by client and s_j held by server, if: (1) σ_i is a genuine CA signature on c_i, and (2) $c_i = s_j$, hence, $hc_i = hs_j$, we obtain: $tc_{\ell_i} = H'((\sigma_i)^{2eR_cR_s}) = H'((hc_i)^{2R_cR_s}) = ts_j$.
- ***Privacy.*** In this version of the paper, we only provide some intuition for our security arguments, and defer to future work formal proofs. Client privacy is based on its input being statistically indistinguishable from a random distribution in QR_N. Arguments regarding server privacy are similar to those for PSI-CA, thus, we do not repeat them here. We argue that if one could violate APSI-CA server privacy, then the one would also violate server privacy of the APSI construct in Figure 1 of [10], proven secure under the RSA and DDH assumptions. Finally, note that the protocol is unlinkable, given that random values, R_c, R_s, are selected fresh for each protocol execution.
- ***Efficiency.*** This APSI-CA protocol incurs linear computation (for both parties) and communication complexity. Specifically, client and server perform respectively $O(w)$ and $O(w + v)$ modular exponentiations. However, exponents are now taken in the RSA settings, while in PSI-CA can be taken from a smaller group, thus, be much shorter (e.g., 160-bit vs 1024-bit long). Communication complexity amounts to $O(w + v)$. Note that this is significantly lower than related work, i.e., [8], which incurs quadratic overhead (see Section 2.2).

6 Combining PSI-CA and PSI

As mentioned in Section 1, it is often desirable to privately assess the magnitude of the set intersection before engaging in an actual (private) set intersection computation. We are motivated by potential concerns with respect to server privacy, arising in PSI executions where the intersection obtained by client is close to the entire server set (i.e., $|\mathcal{S} \cap \mathcal{C}| \approx |\mathcal{S}|$).

We now show how to combine our proposed PSI-CA construct with with PSI functionality, in order to address such concerns. Specifically, rather than engaging in PSI, parties first run the PSI-CA protocol with their client/server roles reversed. This way, server learns (only) the intersection cardinality and the size of the parties' inputs, and uses this information to decide whether to proceed with PSI. In case it decides to proceed, client only needs to receive one more message from server to compute the intersection. In other words, server defines a *policy* – based on the size of (i) the two sets and (ii) the intersection – and only if the policy is satisfied, server engages in PSI protocol (thus, client privately obtains the set intersection).

The resulting protocol is presented in Figure 3. In the first two rounds, server and client run PSI-CA with their *roles reversed* (i.e., server learns the cardinality of the intersection), and, assuming server's policy is satisfied, the last round allows client to learn the set intersection. The same approach can be used for other private set operations, such as PSU [18]. Indeed, similar concerns about server privacy occur in a scenario where $|\mathcal{C} \cup \mathcal{S}| \approx |\mathcal{C}| + |\mathcal{S}|$, and can again be addressed by running PSI-CA with roles reversed. Observe that protocol in Figure 3 incurs complexities comparable to the underlying PSI-CA (illustrated in Figure 1): only one additional message must be sent to realize policy-based PSI.

The security of this protocol, in presence of semi-honest adversaries, trivially stems from that of the underlying PSI-CA. Nonetheless, we defer to future work extending our constructions to malicious security. In fact, there is no guarantee that malicious parties maintain the same input over multiple interactions or do not abort execution prematurely. This constitutes an interesting open challenge that we defer to future work.

Remark: Our technique in Figure 3 is not to be confused with the concept of Policy-Enhanced PSI, recently proposed by [36]. Using the latter, two parties privately obtain the intersection of their sets, while enforcing policies pertaining what/how to share, based on policies and authorizations related to single items. Whereas, policy enforced by server in our protocol is much simpler – it is based on the cardinality of set intersection: depending on this (and on its relationship to set size), server decides whether or not to disclose set intersection's content to client. A vaguely comparable approach is so-called Knowledge-oriented Multi-party Secure Computation [29], where each participating party is able to reason about the increase in knowledge that other parties could gain as a result of the secure computation, and may choose not to participate to restrict that gain.

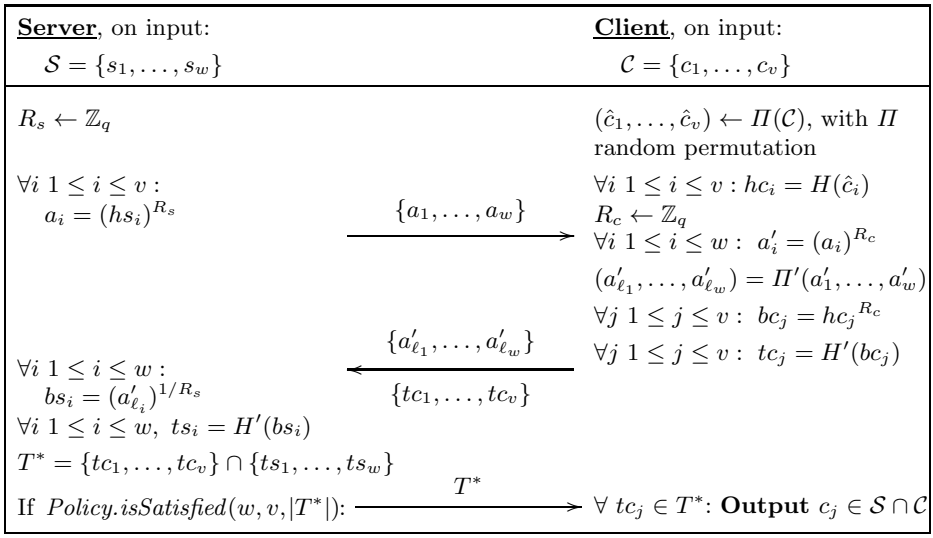


Fig. 3. Combining PSI-CA and PSI for a three-round policy-based Private Set Intersection protocol. (All computation is mod p).

7 Conclusion

This paper presented a protocol for PSI-CA, with *linear* computational and communication complexities. It can be used to compute PSU-CA, without introducing any additional overhead.

Next, we presented two novel extensions. We introduced Authorized PSI-CA, or APSI-CA, that is useful in settings where client input must be authorized by a certification authority. Then, we showed how PSI-CA can be used to realize a PSI protocol where server determines (in privacy-preserving manner) cardinality of set intersection before deciding whether or not to engage in a PSI interaction with client. Such an approach is very efficient, as it requires only *one* additional message on top of our PSI-CA protocol.

We will release an optimized implementation of all protocols presented in this paper along with the final version of the paper. As part of future work, we plan to investigate extensions to guarantee security in the presence of malicious adversaries and in the UC framework.

Acknowledgments. We wish to thank Stanislaw Jarecki and Jens Groth for their valuable feedback.

References

1. Agrawal, R., Evfimievski, A., Srikant, R.: Information sharing across private databases. In: SIGMOD (2003)
2. Applebaum, B., Ringberg, H., Freedman, M.J., Caesar, M., Rexford, J.: Collaborative, Privacy-Preserving Data Aggregation at Scale. In: Atallah, M.J., Hopper, N.J. (eds.) PETS 2010. LNCS, vol. 6205, pp. 56–74. Springer, Heidelberg (2010)

3. Ateniese, G., Blanton, M., Kirsch, J.: Secret handshakes with dynamic and fuzzy matching. In: NDSS (2007)
4. Baldi, P., Baronio, R., De Cristofaro, E., Gasti, P., Tsudik, G.: Countering GATTACA: Efficient and Secure Testing of Fully-Sequenced Human Genomes. In: CCS (2011)
5. Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M.: The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. Journal of Cryptology 16(3) (2003)
6. Blundo, C., De Cristofaro, E., Gasti, P.: EsPRESSo: Efficient Privacy-Preserving Evaluation of Sample Set Similarity. In: DPM (2012)
7. Bursztein, E., Lagarenne, J., Hamburg, M., Boneh, D.: OpenConflict: Preventing Real Time Map Hacks in Online Games. In: S&P (2011)
8. Camenisch, J., Zaverucha, G.M.: Private Intersection of Certified Sets. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 108–127. Springer, Heidelberg (2009)
9. De Cristofaro, E., Jarecki, S., Kim, J., Tsudik, G.: Privacy-Preserving Policy-Based Information Transfer. In: Goldberg, I., Atallah, M.J. (eds.) PETS 2009. LNCS, vol. 5672, pp. 164–184. Springer, Heidelberg (2009)
10. De Cristofaro, E., Kim, J., Tsudik, G.: Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 213–231. Springer, Heidelberg (2010)
11. De Cristofaro, E., Tsudik, G.: Practical Private Set Intersection Protocols with Linear Complexity. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 143–159. Springer, Heidelberg (2010)
12. Evfimievski, A., Gehrke, J., Srikant, R.: Limiting privacy breaches in privacy preserving data mining. In: PODS (2003)
13. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword Search and Oblivious Pseudorandom Functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Heidelberg (2005)
14. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient Private Matching and Set Intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004)
15. Frikken, K.: Privacy-Preserving Set Union. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 237–252. Springer, Heidelberg (2007)
16. Goldreich, O.: Foundations of Cryptography. Cambridge U. Press (2004)
17. Hazay, C., Lindell, Y.: Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 155–175. Springer, Heidelberg (2008)
18. Hazay, C., Nissim, K.: Efficient Set Operations in the Presence of Malicious Adversaries. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 312–331. Springer, Heidelberg (2010)
19. Hohenberger, S., Weis, S.A.: Honest-Verifier Private Disjointness Testing Without Random Oracles. In: Danezis, G., Golle, P. (eds.) PET 2006. LNCS, vol. 4258, pp. 277–294. Springer, Heidelberg (2006)
20. Hong, J., Kim, J.W., Kim, J., Park, K., Cheon, J.H.: Constant-Round Privacy Preserving Multiset Union. Cryptology ePrint Archive, Report 2011/138 (2011), http://eprint.iacr.org/2011/138
21. Huang, Y., Evans, D., Katz, J.: Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? In: NDSS (2012)
22. Huberman, B., Franklin, M., Hogg, T.: Enhancing privacy and trust in electronic communities. In: ACM Conference on Electronic Commerce (1999)

23. Intelligence Advanced Research Projects Activity (IARPA). Automatic Privacy Protection and Security and Privacy Assurance Research Programs, https://www.fbo.gov/utils/view?id=920029a5107a9974c2e379324a1dcc4e
24. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending Oblivious Transfers Efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
25. Jarecki, S., Liu, X.: Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 577–594. Springer, Heidelberg (2009)
26. Jarecki, S., Liu, X.: Fast Secure Computation of Set Intersection. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 418–435. Springer, Heidelberg (2010)
27. Kantarcioglu, M., Nix, R., Vaidya, J.: An Efficient Approximate Protocol for Privacy-Preserving Association Rule Mining. In: Theeramunkong, T., Kijsirikul, B., Cercone, N., Ho, T.-B. (eds.) PAKDD 2009. LNCS, vol. 5476, pp. 515–524. Springer, Heidelberg (2009)
28. Kissner, L., Song, D.: Privacy-Preserving Set Operations. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 241–257. Springer, Heidelberg (2005)
29. Mardziel, P., Hicks, M., Katz, J., Srivatsa, M.: Knowledge-oriented secure multiparty computation. In: PLAS (2012)
30. Menezes, A., Oorschot, P.V., Vanstone, S.: Handbook of Applied Cryptography. CRC (1997)
31. Nagaraja, S., Mittal, P., Hong, C., Caesar, M., Borisov, N.: BotGrep: Finding Bots with Structured Graph Analysis. In: Usenix Security (2010)
32. Narayanan, A., Thiagarajan, N., Lakhani, M., Hamburg, M., Boneh, D.: Location Privacy via Private Proximity Testing. In: NDSS (2011)
33. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
34. Pohlig, S., Hellman, M.: An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. IEEE Transactions on Information Theory 24(1) (1978)
35. Seo, J.H., Cheon, J.H., Katz, J.: Constant-Round Multi-party Private Set Union Using Reversed Laurent Series. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 398–412. Springer, Heidelberg (2012)
36. Stefanov, E., Shi, E., Song, D.: Policy-Enhanced Private Set Intersection: Sharing Information While Enforcing Privacy Policies. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 413–430. Springer, Heidelberg (2012)
37. Vaidya, J., Clifton, C.: Secure set intersection cardinality with application to association rule mining. Journal of Computer Security 13(4) (2005)
38. Yao, A.: Protocols for secure computations. In: FOCS (1982)

Fast and Secure Root Finding for Code-Based Cryptosystems

Falko Strenzke*

Cryptography and Computeralgebra, Department of Computer Science,
Technische Universität Darmstadt, Germany
fstrenzke@crypto-source.de

Abstract. In this work we analyze five previously published respectively trivial approaches and two new hybrid variants for the task of finding the roots of the error locator polynomial during the decryption operation of code-based encryption schemes. We compare the performance of these algorithms and show that optimizations concerning finite field element representations play a key role for the speed of software implementations. Furthermore, we point out a number of timing attack vulnerabilities that can arise in root-finding algorithms, some aimed at recovering the message, others at the secret support. We give experimental results of software implementations showing that manifestations of these vulnerabilities are present in straightforward implementations of most of the root-finding variants presented in this work. As a result, we find that one of the variants provides security with respect to all vulnerabilities as well as competitive computation time for code parameters that minimize the public key size.

Keywords: side channel attack, timing attack, implementation, code-based cryptography.

1 Introduction

Implementations of code-based cryptosystems like the McEliece [1] and Niederreiter [2] schemes have received growing interested from researchers in the past years and been analyzed with respect to efficiency on various platforms [3–7]. Furthermore, a growing number of works has investigated the side-channel security of code-based cryptosystems [8–12].

In this work, we will turn to an algorithmic task that arises in the decryption operation of both Niederreiter and McEliece cryptosystems. This is the root-finding algorithm. It deserves attention for two reasons: first of all, as addressed already in previous work, it is in general the most time-consuming part of the decoding algorithm [3, 5]. The second aspect is that of side-channel security of the root-finding and has so far, to the best of our knowledge, only been considered in

* To the most part, this work was done in the author's private capacity, a part of the work was done at Cryptography and Computeralgebra, Department of Computer Science, Technische Universität Darmstadt, Germany.

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 232–246, 2012.

[13]. We point out basically two types of timing side-channel vulnerabilities that can arise in the root-finding procedure. One is aimed at recovering the message to a given ciphertext, the other at finding the support that is part of code-based private keys.

Based on these considerations, we chose a number of different root-finding algorithms, which we describe in Sec. 4 after providing some elementary preliminaries about code-based cryptosystems in Sec. 2 and 3. In Sec. 5, we perform a timing side-channel analysis of these algorithms including theoretical discussions and experimental results. Afterwards, in Sec. 6, we give the results of a performance evaluation of the chosen algorithms on modern x86 architectures.

As the main result from our work, we find that the root-finding variant according to [14], which to the best of our knowledge has not been taken into consideration for code-based cryptosystems so far, achieves both competitive running time and security with respect to all potential timing side-channel vulnerabilities in the root-finding. But we wish to stress that it is not the aim of this work to give a definitive answer to the question which root-finding variant is the best to use in a code-based cryptosystem. This is mainly for the reason that the we are considering pure software implementations here, and the use of crypto coprocessors might change the picture. We will come back to this in the Conclusion in Sec. 7.

As the starting point for the implementation that the experimental results of this work are based on, we used the HyMES open source implementation [15] of the McEliece scheme presented in [3]. We added the countermeasure previously proposed in [11], and removed some fault attack vulnerabilities, the latter is addressed in App. B. Furthermore, we adopted the root-finding algorithm variant given by the Berlekamp Trace Algorithm [16], as it is found in HyMES. However, we performed some straightforward optimizations in the implementation of this algorithm which are mentioned in Sec. 4.2. We added the remaining root-finding variants that are presented in Sec. 4. In this context, we want to emphasize that we do not take any credit for the choice of the Berlekamp Trace Algorithm for the purposes of root-finding in code-based cryptosystems, the implementation of this algorithm used for the results in this work, and its description as given in this work, since all of this is adopted from [3] resp. [15].

2 Preliminaries

In the following, we describe those aspects of the encryption and decryption of code-based cryptosystems as McEliece [1] and Niederreiter [2] schemes that are relevant for the topics of this work. As these only depend on properties common to both types of cryptosystems, it is possible for us to basically omit any distinction between them.

Code-based cryptosystems build on error correcting codes. Specifically, the only codes known to be secure for this use are Goppa Codes [17]. The parameters of such a code are its length n, which is the length of the code words, $n \leqslant 2^m$; the dimension k (with $k < n$), which is the length of the message words; and the error correcting capability t (all lengths refer to the binary representation).

For such a code, if a message word is encoded into a code word, up to t bit flip errors in the code word may be corrected by a corresponding error correction algorithm, thus allowing to recover the message word. In the following, we will also make use of the expression of "adding errors" to the code word, by which we mean carrying out the "exclusive or" (XOR) operation with the code word $\boldsymbol{c}$ and the error vector $\boldsymbol{e}$, i.e. $\boldsymbol{c} \oplus \boldsymbol{e}$.

In both McEliece and Niederreiter schemes, the encryption involves the creation of an error vector $\boldsymbol{e}$, whose Hamming weight $\mathrm{wt}(\boldsymbol{e})$ is equal to the error correcting capability t of the employed code. The concrete realization of the encryption is different in both schemes, but in either case, it is vital for the privacy of the encrypted message that $\boldsymbol{e}$ remains secret.

In the McEliece and Niederreiter cryptosystems, syndrome decoding through Patterson's Algorithm [18] plays a key role. As the details of this algorithm are irrelevant for purposes of understanding the topics of this work, we give only a brief outline of this algorithm. In the McEliece cryptosystem, the syndrome is computed from the ciphertext by multiplying the ciphertext with the so called parity check matrix, in the Niederreiter scheme, the syndrome is the ciphertext itself. The Patterson decoding algorithm takes as input the so called syndrome vector $\boldsymbol{s}$ and outputs the error vector $\boldsymbol{e}$ that was added to the code word.

As already mentioned, this work deals with one part of the Patterson Algorithm, this is the finding of the roots, i.e. zeros of the so called error locator polynomial $\sigma(Y) \in \mathbb{F}_{2^m}[Y]$ which is computed in the course of the Patterson Algorithm. In case of $w = \mathrm{wt}(\boldsymbol{e}) \leqslant t$ it holds that

$$\sigma(Y) = \Pi_{i=1}^{w}(\alpha_{E_i} - Y), \tag{1}$$

where the ordered set $\Gamma = (\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$, is the so called support formed by pairwise distinct elements of $\mathbb{F}_{2^m}$ and E_i, $i = 1, \ldots, w$ denote the indexes of bits having value one in $\boldsymbol{e}$ in arbitrary ordering. A lookup table representing the support is part of the code-based private key. If it becomes known, the whole key is compromised [12]. From the determination of the roots of $\sigma(Y)$, the error positions, i.e. those bits in $\boldsymbol{e}$ that have value one, are found: if α_{E_i} is a root of $\sigma(Y)$, then $e_{E_i} = 1$. If $w > t$, then $\sigma(Y)$ will have degree less or equal than t, where the probability for the latter is very high, but it will not be of the form given in Eq. (1). In Sec. 4, we will present a number of concrete algorithms for the task of finding the roots of the error locator polynomial $\sigma(Y)$.

3 Remarks about the $\mathbb{F}_{2^m}$ Operations

Before we start with the descriptions of the root-finding algorithms we compare in this work, we want to point out some details concerning the costs of the basic $\mathbb{F}_{2^m}$ operations that are involved, i.e. addition and multiplication.

While $C_{\mathrm{gf\_add}}$, the cost of an addition in $\mathbb{F}_{2^m}$, is given by a simple XOR operation, the multiplication in $\mathbb{F}_{2^m}$ is much more complex and has a number of variants. An efficient software implementation of finite field arithmetics with characteristic 2 and small extension degrees is realized by the use of one lookup

table for the logarithm of each non-zero element to the base of some primitive element, and the corresponding anti-logarithm table.

The standard multiplication, as it is for instance implemented by the "C" macro `gf_mul()` in HyMES [15], which is used throughout their code, takes arguments in the normal representation and outputs the result in normal representation. This type of multiplication, we refer to as *mul_nnn* . Its cost is two conditional branches to check whether the arguments are zero, three table lookups, one arithmetic ADD, and reduction of the result modulo the fields multiplicative order, which in turn consists of several instructions. In the general case, this multiplication is needed, as in most places in the algorithms involved in the decoding with the Patterson Algorithm, multiplication and addition in $\mathbb{F}_{2^m}$ are intermixed, and moreover, operands having value zero cannot be excluded.

However, when operands are known to be non-zero, and multiplications are carried out subsequently, other forms of the multiplication, which have results (a in the algorithm description *mul_abc*) or operands (b and c) in the logarithmic representation, are more efficient:

- *mul_lll* consists only of one arithmetic ADD (a certain number of these multiplications can be carried out before a reduction modulo the multiplicative order becomes necessary to avoid overflowing the register)
- *mul_nln* saves one conditional branch and one table lookup compared with *mul_nnn*

This rough review of the finite field arithmetic implementations in software makes it obvious that it is not sufficient to simply count the instances of multiplication in $\mathbb{F}_{2^m}$, but it has to be considered how the multiplication is embedded into the algorithm and what variant of the multiplication can be used.

4 Variants of Root Finding

In the following subsections we give brief descriptions of the root-finding algorithm variants analyzed in this work.

4.1 Exhaustive Evaluation with and without Division

The most straightforward implementation of the root finding is to simply evaluate the polynomial $\sigma(Y)$ for each element of the code.

The complexity of this algorithm is given as

$$C_{\text{eval}-\text{rf}} = nt(C_{\text{gf\_add}} + C_{\text{mul\_nln}})$$

Remember that n is the code length and t is the error correcting capability. Taking a look at the Horner Scheme evaluation used here, we see that when evaluating $\sigma(Y)$, we can transform $x \neq 0$ to the logarithmic representation, avoiding some unnecessary table lookups, i.e. make use of *mul_nln* .

The algorithm can be sped up by dividing the polynomial $\sigma(Y)$ by each root found. Such a division has basically the same complexity as the evaluation of the polynomial for one single element of $\mathbb{F}_{2^m}$. In the following, we will call these two variants *eval-rf* and *eval-div-rf*.

4.2 Berlekamp Trace Algorithm

As stated in the introduction, our implementation is based on the HyMES implementation [3, 15]. There, the root finding is achieved by the so called Berlekamp Trace Algorithm [16]. For completeness, we provide the description of this algorithm as originally given in [3] in Alg. 1. The initial call to this recursive algorithm is given as $\mathrm{BTA}(\sigma(Y), 1)$, which we will refer to as *BTA-rf* for the remainder of this work. The trace function is defined as $\mathrm{Tr}(Y) = Y + Y^2 + Y^{2^2} + \ldots + Y^{2^{m-1}}$, and $\{\beta_1, \beta_2, \ldots, \beta_m\}$ is a standard basis of $\mathbb{F}_{2^m}$.

Algorithm 1. The recursive Berlekamp Trace Algorithm $\mathrm{BTA}(\sigma(Y), i)$.

Require: the error locator polynomial $\sigma(Y)$
Ensure: the set of roots of $\sigma(Y)$
1: **if** $\deg(\sigma(Y) \leqslant 1)$ **then**
2: **return** root of $\sigma(Y)$
3: **end if**
4: $\sigma_0(Y) \leftarrow \gcd(\sigma(Y), \mathrm{Tr}(\beta_i, Y))$
5: $\sigma_1(Y) \leftarrow \gcd(\sigma(Y), 1 + \mathrm{Tr}(\beta_i, Y))$
6: **return** $\mathrm{BTA}(\sigma_0(Y), i+1) \cup \mathrm{BTA}(\sigma_1(Y), i+1)$

In [19], the complexity of the BTA is given as $\mathcal{O}(mt^2)$. In order to make fair comparison of the various root-finding variants in terms of performance, we optimized the existing implementation of the algorithm by applying the more cost-efficient versions of multiplication in $\mathbb{F}_{2^m}$ as discussed in Sec. 3 where possible. As a result, the running time was reduced by about 10%.

Furthermore, *BTA-rf* can be sped up by using specific root finding algorithms for polynomials of low degree [19]. We only implemented the variant where the roots of polynomials of degree two are determined through the use of a lookup table of size $2n$ bytes (supporting $m = 15$ at most) presented in the referenced work. Then, in the recursion, this algorithm is invoked instead of Alg. 1 whenever the degree of $\sigma(Y)$ is two. In the following, we refer to this algorithm by *BTZ$_2$-rf*.

4.3 Root Finding with Linearized Polynomials

In this section, we explain a root-finding method based on decomposing a polynomial in $\mathbb{F}_{2^m}[Y]$ into linearized polynomials [14]. The idea of this approach is based on the fact that the exhaustive evaluation of a linearized polynomial can be done with much less computational complexity than for general polynomials.

Definition 1. *A polynomial $L(Y)$ over $\mathbb{F}_{2^m}$ is called a linearized polynomial if $L(Y) = \sum_i L_i Y^{2^i}$, where $L_i \in \mathbb{F}_{2^m}$.*

As shown in [14], an affine polynomial of the form $A(Y) = L(Y) + \beta$ with $\beta \in \mathbb{F}_{2^m}$ can be evaluated for the value $Y = x_i$ as

$$A(x_i) = A(x_{i-1}) + L(\Delta_i), \Delta_i = x_i - x_{i-1} = \alpha^{\delta(x_i, x_{i-1})}, \tag{2}$$

where $\{\alpha^0, \alpha^1, \ldots, \alpha^{m-1}\}$ is a standard basis of $\mathbb{F}_{2^m}$ and $\mathrm{wt}\,(x_i \oplus x_{i-1}) = 1$, i.e. their Hamming distance is 1. A generic decomposition of a polynomial $f(Y) = \sum_{i=0}^{t} f_i Y^i$, also given in [14], is

$$f(Y) = f_3 Y^3 + \sum_{i=0}^{\lceil (t-4)/5 \rceil} Y^{5i} A_i(Y), \tag{3}$$

where

$$A_i(Y) = f_{5i} + \sum_{j=0}^{3} f_{5i+2^j} Y^{2^j}. \tag{4}$$

The evaluation of each $A_i(x_i)$ is done efficiently according to Eq. (2). To this end, the exhaustive evaluation of Eq. (3) is done with the x_i being in Gray-Code ordering, i.e. for all i we have that x_i and x_{i+1} differ only in one single bit. Specifically, we use the Gray Code generated by $x_i = (\mathrm{i} >> 1) \oplus i$, where ">>" denotes logical right shift. The actual computation cost is given by the sum of the precomputations, i.e. the computation of the $A_i(Y)$. This cost is given in [14], it is however negligible for secure code parameters. The dominating cost is that of computing $f(Y)$ for all n code elements:

$$C_{\mathrm{dcmp-rf}} = (n-1)(C_{\mathrm{gf\_log}} + C_{\mathrm{squ\_ll}} + 2C_{\mathrm{mul\_lll}} + C_{\mathrm{mul\_nll}} + \lceil (t+1)/5 \rceil \, (2C_{\mathrm{gf\_add}} + C_{\mathrm{mul\_lll}} + C_{\mathrm{mul\_nln}}))$$

where $C_{\mathrm{gf\_log}}$ refers to the cost of converting a $\mathbb{F}_{2^m}$ element from normal to logarithmic representation and $C_{\mathrm{squ\_ll}}$ is the cost for squaring in the logarithmic representation.

4.4 New Hybrid Variants

We also implemented two new hybrid variants. The first we label *dcmp-div-rf(a,b)*. It is given simply by restarting the whole *dcmp-rf* after through divisions by found roots the degree of sigma $\sigma(Y)$ has been reduced by at least $5a$ to $5k+4$ for some positive integer k, and where once $\deg(\sigma(Y)) = b$ no more divisions are performed and standard *dcmp-rf* is used henceforth.

A furhter variation of this is given through *dcmp-div-BTZ_2-rf(a,b)*. It is equal to *dcmp-div-rf(a,b)* until $\deg(\sigma(Y)) = b$. Then, when $\sigma(Y)$ has degree b, *BTZ_2-rf* is invoked to find the remaining roots.

5 Security Aspects of the Root Finding in Code-Based Cryptosystems

In this section, we show that a dependency of the root finding algorithm's running time on the number of the roots of the Error Locator Polynomial $\sigma(Y)$ introduces vulnerabilities to timing attacks against the cleartext, and that other effects threaten the secrecy of the secret support Γ.

In order to be able to judge the relevance of the timing results provided in this section, it is important to know that the syndrome decoding with the Patterson Algorithm, which we consider to include the root-finding, is at least for the McEliece cryptosystem the only source of variable running time, under some assumptions: multiplication of the ciphertext with the parity check matrix is either constant time of linear in the ciphertext's Hamming weight (these are the straightforward implementation choices) and the CCA2 conversion (necessary for both the McEliece and the Niederreiter cryptosystem, see for instance [20]) is constant time (see for instance [21]).

In this security analysis, for the sake of brevity, we restrict ourselves to the main variants *eval-rf*, *eval-div-rf*, *BTA-rf*, and *dcmp-rf*. The side channel problems found for these algorithms naturally extend to the other hybrid variants.

5.1 Security against Message-Aimed Attacks

The first important fact to know is that $\sigma(Y)$ output by Patterson's Algorithm has $\mathrm{wt}(\boldsymbol{e})$ roots in case $\mathrm{wt}(\boldsymbol{e}) \leqslant t$, and only a fraction of t roots if $\mathrm{wt}(\boldsymbol{e}) > t$ (refer to Eq. (1)). For instance, for $n = 2048$ and $t = 50$, $\sigma(Y)$ typically has less than five roots in the latter case.

A dependence of the running time on the number of roots thus potentially creates the following problem: if the case $w > t$ can be inferred from the running time, an attack similar to that described in [9] is possible. In such an attack, the attacker flips a bit in ciphertext he wishes to decrypt, observes the decryption, and from the running time tries to guess whether $t + 1$ or $t - 1$ errors resulted from his bit flip. This clearly gives him information about the error positions piece by piece.

Note that the case of $w < t$ is covered by the countermeasures proposed in [11]. In the presence of these countermeasures, the decryption of a ciphertext with $\mathrm{wt}(e) < t$ also results in $\sigma(Y)$ with degree t and very few roots. But it is important to be aware that even if $w < t$ and $w > t$ are not distinguishable based on the timings, but $w = t$ can be distinguished from $w \neq t$, an attack is still possible: by flipping two bits in a ciphertext and trying to find those cases where $w = t$, the attacker will learn whenever he flipped one non-error and one error position.

In the Patterson Algorithm of our implementation, the countermeasure proposed in [11] is included, so that for $w < t$ we still have $\deg(\sigma(Y)) = t$, but the number of roots of $\sigma(Y)$ is much less than t, as already mentioned. This happens automatically for $w > t$ in Patterson's Algorithm, so that we can expect to find major differences in the running time of the root-finding algorithm only for the cases $w \neq t$ and $w = t$.

In Figures 1(b), 1(a), 1(c) and 1(d), we give plots of the running time of the root-finding for the four different algorithms. The timings were taken on an Atmel ATMega1284P Microcontroller. We chose this platform, as it provides far more deterministic cycle counts than a modern x86 CPU, and thus is more suited to identify possible timing vulnerabilities. We used a Goppa Code with parameters $n = 512$ and $t = 33$ for the syndrome decoding performed on the

microcontroller and created 30 different syndromes for each value of w between 20 and 40. The cycle counts apply to the running time of the respective root-finding algorithm. For each value of w the center mark indicates the mean of the set of the 30 different syndromes, and the bar shows the minimal and maximal values from this set.

The *eval-rf* algorithm's running time, depicted in Fig. 1(a), shows the mean running time of $w = t$ in line with the those of $w \neq t$. However, there seem to be cases of $w = t$ with considerably lower running time than for $w \neq t$, as can be seen by the depicted minimal value. Neither did we find the reason for this, nor did we analyze whether this effect can be used for actual attacks. We justify these omissions by the fact that, as already apparent from the results given in this section, *eval-rf* is not a competitive candidate for root-finding in code-based cryptosystems.

Fig. 1(b) shows clearly the speedup by a factor of two by *eval-div-rf* compared to *eval-rf*. However, also the inherent timing vulnerability [13] of this algorithm cannot be overlooked: the case $w = t$, where the benefit of the divisions has its real impact, is almost twice as fast as for $w \neq t$. This renders it an insecure choice.

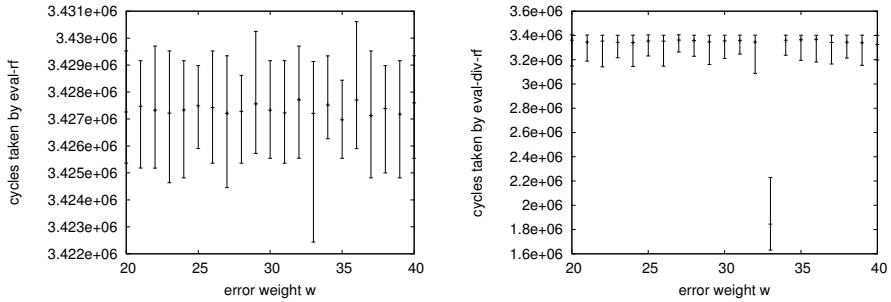
The timing results of *BTA-rf*, as implemented in HyMES, are shown in Fig. 1(c). Here, we can realize that already the mean of the running times for $w = t$ is below most of the minimal values of sets for $w \neq t$, clearly indicating a vulnerability. Obviously, the recursive algorithm behaves different when $\sigma(Y)$ has considerably fewer than t roots.

Finally, Fig. 1(d) shows the results for *dcmp-rf*. There is no apparent difference between the cases $w = t$ and $w \neq t$.

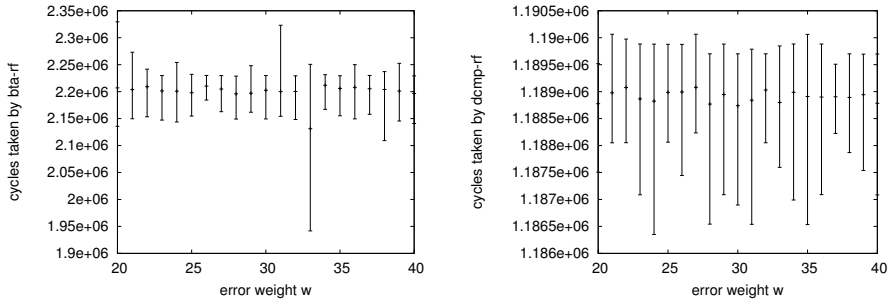
5.2 Security with Respect to Attacks Aiming at the Secret Support

We now show that other vulnerabilities can arise in the root-finding algorithm, which allow attacks against the secret support of the code-based scheme. This is for instance the case, when the running time of the root-finding algorithm depends on the values of the roots found. To understand that this is a vulnerability one has to consider that an attacker can create ciphertexts with $\boldsymbol{e}$ known to him. Then, according to Eq. (1) any information about the roots is information about the support Γ.

One possible vulnerability arises if in *eval-div-rf*, the evaluation of $\sigma(Y)$ is done with Y being substituted in lexicographical order; in this case the found roots are later mapped to the corresponding E_i values by using a table for Γ^{-1}: Fig. 2(a) and 2(b) given in App. A show running times on the AVR platform of the syndrome decoding with *eval-div-rf* for $n-(t-1)$ error vectors created in the following way: a random error pattern of weight $t-1$ was fixed, and the position of the last error, E_t was varied over the remaining free positions, resulting in error vectors with Hamming weight t. On the x-axis, E_t is shown. We will refer to this type of plot as "support scan" henceforth. The result is a relatively clear linear ascend, which is not surprising when considering the *eval-div-rf* algorithm: Starting evaluation at $Y = 0$, the earlier a root is found, the more beneficial is



(a) *eval-rf* (b) *eval-div-rf*



(c) *BTA-rf* (d) *dcmp-rf*

Fig. 1. Cycle counts taken on an ATMega1284P for the various root-finding algorithm variants with parameters $n = 512$ and $t = 33$.

the reduction of the degree of $\sigma(Y)$ by one through the subsequent division. Thus, the task for an attacker amounts to bringing the measured timings into an ascending ordering, giving him Γ. Obviously, there is some distortion of this ordering in Fig. 2(a), which stems from other operations of variable duration in the syndrome decoding. We leave it open whether in this manner the support Γ becomes known to the attacker in its entirety, it is however clearly obvious, that a large amount of information about Γ becomes available.

This vulnerability can be avoided by performing the evaluation of $\sigma(Y)$ with Y being substituted in the order α_0, α_1, ..., α_{n-1}. Note however that the vulnerable version is slightly faster, since there only t table lookups in Γ^{-1} for the found roots are done, whereas in the secure version n such lookups in Γ are necessary. Thus the described problem is realistic.

We also wish to point out that an attack exploiting this vulnerability, in contrast to other previously published timing attacks [10, 11], cannot be detected: The ciphertext carries t errors and will pass the CCA2 integrity test. (Note that a CCA2 conversion is necessary for any Niederreiter or McEliece like code-based encryption scheme, see for instance [20].) This is important, because the other attacks, which cannot be carried out in a clandestine manner in this sense, can be thwarted by countermeasures which detect the irregularity of the ciphertext,

and for instance add an enormous delay or enforce constant running time if possible on the respective platform. In the presence of the threat of power analysis attacks, however, such countermeasures would in most cases not suffice as adding delays after the actual computation will most likely be detectable in the power trace.

We also analyzed the *dcmp-rf* and *BTA-rf* algorithms with respect to these vulnerabilities. As one should expect from an algorithm that performs an exhaustive search, *dcmp-rf* does not exhibit any dependency of the running time on the root values, except for a single pitfall that has to be avoided. This is discussed in some detail in App. A. Though for *BTA-rf* no concrete attack could be derived, the question of its security with respect to key-aimed attacks remains unclear, we give the analysis also in App. A.

6 Performance of the Root-Finding Variants

In this section, we give a comparison of the performance of the root-finding algorithms given in Sec. 4. The code was compiled with GCC version 4.5.2 with the optimization options

```
-finline-functions -O3 -fomit-frame-pointer -march=i686 -mtune=i686
```

and run on a Intel(R) Core(TM)2 Duo CPU U7600 CPU.

In the following we give results for two parameter sets based on the propositions given in [22] for 128 and 256 bit security, which are based on code parameter choices aiming at the minimization of the public key size, which is known to be the most problematic feature of code-based cryptosystems. The only deviation of our parameter choices are with respect to the number of errors added during encryption: in [22], List Decoding [23] which allows for the correction of more than t errors is assumed during decryption. For the smaller parameter set, they choose $t+1$ errors and for the larger $t+2$ errors. The reduction of security of the smaller parameter set in our implementation using only t errors, however, can easily be bounded by understanding that an attacker can get from a ciphertext with $t+1$ errors to t errors by guessing one error position correctly, the success probability of which is $(t+1)/n = 0.02$. Accordingly, the security of the scheme with t errors cannot be smaller than $128 - \log_2(1/0.02) > 122$ bits. An according calculation for the larger parameter set gives a lower bound of 244 bits.

It is noteworthy that these parameter sets optimized for minimal public key size for a given security level use codes with $n < 2^m$, and that this has different effects for our four candidate algorithms. *eval-rf* and *eval-div-rf* both are faster for $n < 2^m$ in contrast to $n = 2^m$, however for the latter the speedup is less than for the former, as there the roots found at the end of the support cause less effort. *dcmp-rf* also benefits from $n < 2^m$, since also then support can be build from a Gray Code. *BTA-rf* however has the same running time no matter whether $n < 2^m$ or $n = 2^m$.

Tab. 1 gives the results for the mentioned parameters. We clearly see that BTZ_2*-rf* and *dcmp-rf* are almost equally fast and that *dcmp-div-rf*, the parameters of which were experimentally optimized for the given code parameters, has

Table 1. Comparison of the average root-finding algorithm performance on an x86 Intel Intel(R) Core(TM)2 Duo CPU U7600 for code parameters as suggested in [22]. All given values are the average of 50 decryptions.

parameters	security level	root-finding algorithm	running time / 10^5 cycles
$n = 2960$, $t = 56$	> 122 bit	*eval-rf*	21.16
		eval-div-rf	16.26
		BTA-rf	8.89
		BTZ$_2$-rf	6.33
		dcmp-rf	6.45
		dcmp-div-rf(1,19)	5.42
		dcmp-div-BTZ$_2$-rf(1,19)	5.12
$n = 6624$, $t = 115$	> 244 bit	*eval-rf*	141.86
		eval-div-rf	71.48
		BTA-rf	32.59
		BTZ$_2$-rf	26.10
		dcmp-rf	25.55
		dcmp-div-rf(3,19)	18.38

even better performance. For the smaller parameter set, *dcmp-div-BTZ$_2$-rf* has a small edge on *dcmp-div-rf*, for the larger code parameter set no parameters of *dcmp-div-BTZ$_2$-rf* allowing an improvement over *dcmp-div-rf* were found.

7 Conclusion and Outlook

In this work we have evaluated a number of different root-finding algorithms with respect to their performance and timing side-channel security in code-based cryptosystems. We have shown that timing vulnerabilities can be present in all of these variants. The variant *eval-rf* and *eval-div-rf* can be ruled out as they are both not competitive in terms of computation speed. The latter, which has at least considerable performance advantages over the former, exhibits a timing side-channel vulnerability with respect to message-aimed attacks, which is difficult to prevent.

Considering the remaining two candidates, we find that for code parameters that minimize the public key size, *dcmp-rf* is clearly faster than the *BTA-rf*, however the latter can achieve much faster results for code parameters with small t resulting in large public keys. Since timing side-channel security of *BTA-rf* is problematic at least with respect to message-aimed attacks, and the fact that the public key size is the much greater challenge in code-based encryption schemes than the computation times, *dcmp-rf* could be seen as the winner of this evaluation, because *dcmp-div-rf* suffers from the same problems as *eval-div-rf*.

But as we stated in the introduction, we do not want to postulate this as the definitive answer concerning the choice of root-finding algorithms in code-based cryptosystems. If one would achieve a timing side-channel secure variant of the *BTA-rf*, running time advantages could be achieved at the expense of public key size, which might be desirable in certain applications. Furthermore, the

most important task certainly is the implementation of code-based cryptosystems on smart cards and related platforms. To achieve competitive performance on such resource constrained platforms, hardware support certainly would have to be present, as it is the case for RSA and elliptic curve based algorithms today. Thus, the real question is that of an optimal choice of algorithms and hardware support, achieving both good performance and side-channel security on these platforms. In this context, among other aspects, it will become relevant how easily an algorithm can be parallelized. Note that *eval-rf*, *eval-div-rf*, and *dcmp-rf* can easily be parallelized by starting independent evaluations at 2^x different equally distant offsets into $\mathbb{F}_{2^m}$ (in the Gray-Code order for *dcmp-rf*). However, the circuitry for any single instance of an *eval-rf* evaluator would be considerably simpler than for *dcmp-rf*. The parallelization of *BTA-rf* seems the most complicated, it would have to be applied to the recursive structure of the algorithm. In view of these open questions we encourage future research investigating implementations with efficient hardware support on resource constrained platforms. Pure software implementations on embedded systems, however, would in the case of a widespread adoption of code-based encryption schemes also remain of great importance, as it is the case for RSA today. Thus the results of this work clearly suggest the superiority of *dcmp-rf* at least in this context.

References

1. McEliece, R.J.: A public key cryptosystem based on algebraic coding theory. DSN Progress Report 42-44, 114–116 (1978)
2. Niederreiter, H.: Knapsack-type cryptosystems and algebraic coding theory. Problems Control Inform. Theory 15(2), 159–166 (1986)
3. Biswas, B., Sendrier, N.: McEliece Cryptosystem Implementation: Theory and Practice. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 47–62. Springer, Heidelberg (2008)
4. Heyse, S.: Low-Reiter: Niederreiter Encryption Scheme for Embedded Microcontrollers. In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 165–181. Springer, Heidelberg (2010)
5. Eisenbarth, T., Güneysu, T., Heyse, S., Paar, C.: MicroEliece: McEliece for Embedded Devices. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 49–64. Springer, Heidelberg (2009)
6. Shoufan, A., Wink, T., Molter, G., Huss, S., Strenzke, F.: A Novel Processor Architecture for McEliece Cryptosystem and FPGA Platforms. In: ASAP 2009: Proceedings of the 2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, pp. 98–105. IEEE Computer Society, Washington, DC (2009)
7. Strenzke, F.: A Smart Card Implementation of the McEliece PKC. In: Samarati, P., Tunstall, M., Posegga, J., Markantonakis, K., Sauveron, D. (eds.) WISTP 2010. LNCS, vol. 6033, pp. 47–59. Springer, Heidelberg (2010)
8. Molter, H.G., Stöttinger, M., Shoufan, A., Strenzke, F.: A Simple Power Analysis Attack on a McEliece Cryptoprocessor. Journal of Cryptographic Engineering (2011)

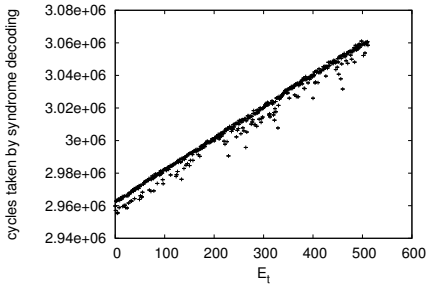
9. Strenzke, F., Tews, E., Gregor Molter, H., Overbeck, R., Shoufan, A.: Side Channels in the McEliece PKC. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 216–229. Springer, Heidelberg (2008)
10. Strenzke, F.: A Timing Attack against the Secret Permutation in the McEliece PKC. In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 95–107. Springer, Heidelberg (2010)
11. Shoufan, A., Strenzke, F., Gregor Molter, H., Stöttinger, M.: A Timing Attack against Patterson Algorithm in the McEliece PKC. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 161–175. Springer, Heidelberg (2010)
12. Heyse, S., Moradi, A., Paar, C.: Practical Power Analysis Attacks on Software Implementations of McEliece. In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 108–125. Springer, Heidelberg (2010)
13. Strenzke, F.: Message-aimed Side Channel and Fault Attacks against Public Key Cryptosystems with homomorphic Properties. Journal of Cryptographic Engineering (2011), doi:10.1007/s13389-011-0020-0; a preliminary version appeared at COSADE 2011
14. Federenko, S., Trifonov, P.: Finding Roots of Polynomials over Finite Fields. IEEE Transactions on Communications 20, 1709–1711 (2002)
15. Biswas, B., Sendrier, N.: HyMES - an open source implementation of the McEliece cryptosystem (2008), http://www-rocq.inria.fr/secret/CBCrypto/index.php?pg=hymes
16. Berlekamp, E.R.: Factoring polynomials over large finite fields. Mathematics of Computation 24(111), 713–715 (1970)
17. Goppa, V.D.: A new class of linear correcting codes. Problems of Information Transmission 6, 207–212 (1970)
18. Patterson, N.: Algebraic decoding of Goppa codes. IEEE Trans. Info. Theory 21, 203–207 (1975)
19. Biswas, B., Herbert, V.: Efficient Root Finding of Polynomials over Fields of Characteristic 2. WEWoRK (2009), http://hal.inria.fr/hal-00626997/PDF/tbz.pdf
20. Kobara, K., Imai, H.: Semantically Secure McEliece Public-Key Cryptosystems - Conversions for McEliece PKC. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 19–35. Springer, Heidelberg (2001)
21. Overbeck, R.: An Analysis of Side Channels in the McEliece PKC (2008), https://www.cosic.esat.kuleuven.be/nato_arw/slides_participants/Overbeck_slides_nato08.pdf
22. Bernstein, D.J., Lange, T., Peters, C.: Attacking and Defending the McEliece Cryptosystem. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 31–46. Springer, Heidelberg (2008)
23. Bernstein, D.J.: List Decoding for Binary Goppa Codes. In: Chee, Y.M., Guo, Z., Ling, S., Shao, F., Tang, Y., Wang, H., Xing, C. (eds.) IWCC 2011. LNCS, vol. 6639, pp. 62–80. Springer, Heidelberg (2011)

A Further Results to the Running Time Dependencies on the Root Values

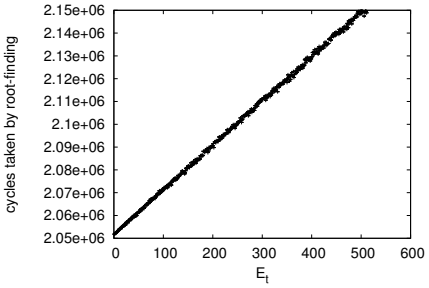
Fig. 2 shows the plots of the dependencies of the running time of the root-finding on the position of a single error bit. See Sec. 5.2 for the discussion of the results for *eval-div-rf*.

For *BTA-rf*, we see some "clouding" effect in the running times, which is also apparent for timings of the whole syndrome decoding, as shown in Fig. 2(c). It is obvious, that these running times are neither constant nor random. There seems to be a tendency to build "clouds"; by which we mean that it seems that an attacker should be able to build hypotheses that if for two different values of E_1 and E_2 the timings are close to each other, then also α_{E_1} and α_{E_2} have close values in their lexicographical interpretation as numbers.

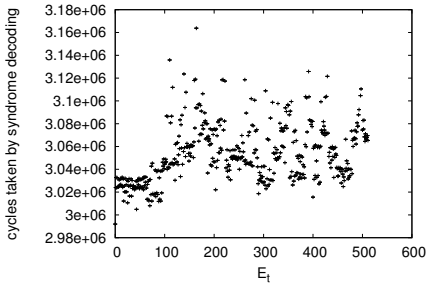
Note for instance the values of E_t below 100 in 2(c), which have consequently lower timings than $3.04 \cdot 10^6$. Though such a dramatic effect was not obvious in all support scans we conducted, it corroborates the notion of "clouding" effects in the timings for *BTA-rf*. Thus we strongly suggest that the running time properties of the *BTA-rf* be subject to thorough analysis before considering its use in real world implementations of code-based schemes.



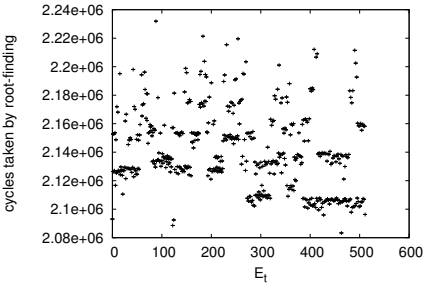
(a) Timing for the syndrome decoding with *eval-div-rf*.



(b) Timing for the root-finding with *eval-div-rf*.



(c) Timings for syndrome decoding with *BTA-rf*.



(d) Timings for root-finding with *BTA-rf*.

Fig. 2. Running times of *eval-div-rf* and *BTA-rf* for $n-(t-1)$ ciphertexts, where $t-1$ error positions are fixed and the $t-th$ position varies, with code parameters $n = 512$ and $t = 33$.

The pitfall concerning the implementation of *dcmp-rf* mentioned in Sec. 5.2 is given through the multiplication by f_3, i.e. σ_3, in Eq. (3). In our implementation,

we precompute the logarithmic representation of σ_3 to subsequently use *mul_nll* for the computation $\sigma_3 Y^3$. In the unprotected variant of our implementation we cover the case $\sigma_3 = 0$ by a conditional branch that bypasses this multiplication. However, as experimental results showed, in this case, the timings clearly allow identification of a syndrome decoding where $\sigma_3 = 0$. The information gained by such an observation is, according to Eq. (1):

$$0 = \sigma_3 = \alpha_{E_1}\alpha_{E_2}\ldots\alpha_{E_{w-3}} \oplus \alpha_{E_1}\alpha_{E_2}\ldots\alpha_{E_{w-4}}\alpha_{E_w} \oplus \ldots,$$

i.e. the sum of products of all possible combinations of $w-3$ different support elements associated with the respective error positions, where w is the error vector's Hamming Weight, usually $w = t$. It is certainly not trivial to exploit this information, however, in combination with other vulnerabilities it might be useful to provide a means of verifying guesses for Γ. The countermeasure to protect against this vulnerability is trivial and comes at a low computational cost, it is described in the following.

The countermeasure is realized by assigning the precomputed value of the logarithm of σ_3 a dummy value during the initialization phase of *dcmp-rf* if $\sigma_3 = 0$, and carrying out the multiplication $Y^3\sigma_3$ with both operands in logarithmic representation regardless of the value of σ_3. Afterwards, a logical AND operation is performed on the result with a mask having all bits set in case of $\sigma_3 \neq 0$ and having value 0 otherwise.

B Further Vulnerabilities in HyMES Syndrome Decoding Implementation

While working with the HyMES implementation [15], we encountered a number of vulnerabilities, potentially enabling both timing and fault attacks. We list them here, because this is good example showing what problems can arise when the syndrome decoding is implemented without implementation security in mind (which was not in the scope of that work).

All code relevant to the syndrome decoding in HyMES is found in the file `decrypt.c`, all line numbers given in the following refer to this file. In line 270, when $\deg(\sigma(Y)) \neq t$, decryption is aborted with an error. This is only a problem if the countermeasures proposed in [11] are not implemented, in this case it allows message-aimed fault-attacks of the type explained in Sec. 5.1 (highly likely that $w > t$ leads to $\deg(\sigma(Y)) = t$, and always that $w < t$ leads to $\deg(\sigma(Y)) = w$).

In line 276, if the root-finding did not return t roots decryption is also aborted with an error. This is clearly allowing message-aimed fault attacks with the two-bit-flip attack described earlier in Sec. 5.1, such a check must not be present in a secure implementation.

In line 285, a Quick Sort algorithm is applied to the set of roots to sort the array. Though we did not seek for attacks against this algorithm, it is certainly clear that the running time of Quick Sort depends on the number of roots, and in general also on their positions.

Strong Privacy for RFID Systems from Plaintext-Aware Encryption

Khaled Ouafi[1] and Serge Vaudenay[2]

[1] IP Video SA, CH-1205 Geneva, Switzerland
[2] EPFL, CH-1015 Lausanne, Switzerland
http://lasec.epfl.ch

Abstract. The Vaudenay model for RFID privacy from Asiacrypt 2007 suffers from the impossibility to address strong privacy. It has however been shown by Ng et al. at ESORICS 2008 that the impossibility result leads to no practical threat, so that the definition from 2007 may be unnecessarily strong. This paper proposes a slight change in the definition of privacy from the Vaudenay model (Asiacrypt 2007). Then, we show that by adding a plaintext-aware assumption on the public-key cryptosystem, the proposed protocol always achieves strong privacy with our new definitions.

1 Introduction

An RFID system consists of 3 components: a *back-end database*, a number of *readers* and *tags*. Tags communicate with readers through a *wireless link* to authenticate themselves. While tags can only maintain one session, readers can communicate with several tags in parallel. Without loss of generality, we assume that the RFID system has only one reader. So, during the authentication process, the reader queries the back-end database through a secure link. Clearly, RFID tags face two contradictory requirements: on the one hand, they must securely identify to a reader; on the other hand, they must hide any traceable information to observers or adversaries.

RFID tags are identified by a unique ID. Cheap tags may be corruptible: it may be possible to open them and read the content of their non-volatile memory. Additionally, they have no internal clock. Such a tag has limited memory and computational power. It can perform symmetric-key based operations: pseudo-random generation, hash computations [10], symmetric-key encryption [9,19] and message authentication codes [33]. However, these limitations vary depending on the application and the allocated budget. Best scenarios allow the tag to use elliptic-curve cryptography [21].

Privacy Models. Several efforts were put in transposing the notion of privacy for RFID, which resulted in several models [3,28,2,25,30,34,15,16,17,11,27,13,31].

Arguably, the definition given by Vaudenay [34] is the most general one. It considers concurrence, tampering (i.e., getting the internal state of an anonymous tag), and the return channel from the reader (i.e., whether a protocol session on the reader side is accepting or not). Contrarily to several other models, it allows adversaries to interact with many concurrent anonymous tags sampled with arbitrary distributions. One difficulty

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 247–262, 2012.

is to identify non-trivial leakage. For instance, let us assume an adversary interacting concurrently with four different anonymous tags. The first three are known to be thrown in a set of three tags numbered 1, 2, and 3, and the last anonymous tag is known to be thrown among a pair of tags numbered 1 and 4. In this case, the adversary trivially infers that this last anonymous tag must be the tag numbered 4. Following simulation-based notions, an information is trivial if the same one could be obtained when the protocol messages are simulated by an additional process called a blinder. The blinder is separate from the adversary and the system. It has therefore neither secrets. However, it knows all interactions.

In [34], Vaudenay shows that Strong privacy (i.e., privacy when adversaries can corrupt any anonymous tags and read the return channel) cannot be achieved. Intuitively, if an adversary creates a legitimate tag then corrupts it, he can then simulate either this tag or an illegitimate one to a reader and the return channel will tell them apart. However, no blinder should be able to do it (otherwise, we would define another adversary from it). So, it may be considered as some non-trivial information. This was quite puzzling since this adversary would by no mean be any threat in practice. Indeed, this example heavily relies on the adversary knowing the expected behavior of the environment and the impossibility to simulate it without guessing what the adversary expects. For this reason, several papers were dedicated to fixing the shortcomings of this model.

At first, Ng et al. [29] proposed at ESORICS 2008 the notion of a "wise adversary", modeling adversaries who cannot guess the behavior of the environment. This fix consists of not allowing adversaries to ask questions for which they know the answer. Canard et al. [12] imposed a different restriction on the adversary called "future-untraceability". This requires, for every adversary, the existence of a simulator for which the output of the adversary is unaffected.

At ESORICS 2011, Hermans et al. [22] proposed a simpler reformulation of Vaudenay's privacy definition that would allow Strong privacy from being achievable by getting rid of the simulation-based approach.

Vaudenay's privacy model was also extended to the case of mutual authentication in a work by Paise and Vaudenay [32]. However, some results were flawed, as discussed by Armknecht et al. [1]. Actually, they show that no RFID protocol with mutual authentication can achieve strong privacy and security at the same time for reasons which are essentially similar to the ones in [34].

In this paper, we use knowledge extractors from plaintext-aware encryption schemes [4,5,6]. Loosely speaking, plaintext-aware encryption schemes are public-key cryptosystems in which the only way for an adversary to produce a valid ciphertext is to choose a plaintext and to encrypt it. So, by reading the adversary's mind, one could extract the corresponding plaintext.

Our contributions. We propose to update the Vaudenay model by changing the definition of the blinder. In short, we allow the blinder to access the random coins used by the adversary so that he could "read his mind" and predict the behavior of the environment as well. This could fix the impossibility result from [34] and [1]. Then, we show by using plaintext aware encryption techniques that Strong privacy can be achieved in our model with the simple protocol (called PKC protocol herein) of [34]. Our result provides strong confidence in the privacy protection deployed by the PKC protocol.

In Appendix, we further show that IND-CCA security is not enough for the PKC protocol to reach strong privacy in the sense that the system may leak some non-simulatable information. To show this, we construct a cryptosystem which is IND-CCA secure but not plaintext-aware.

2 Preliminaries

A function $f(k)$ is said to be polynomial if there exists a constant $n \in \mathbb{N}$ such that $f(k)$ is $O(k^n)$. Similarly, $f(k) = \mathsf{negl}(k)$ if, for every $n \in \mathbb{N}$, $f(k)$ is $O(k^{-n})$.

For an algorithm $\mathcal{A}$, $\mathcal{A}(y;r) \to x$ represents the output after running $\mathcal{A}$ on input y with coins r. The view of $\mathcal{A}$, denoted $\mathsf{view}_{\mathcal{A}}$, is defined to include all the inputs and random coins of $\mathcal{A}$ along with the list of the messages $\mathcal{A}$ received. The ability of an algorithm to query an oracle O is denoted $\mathcal{A}^O$.

Given two algorithms $\mathcal{A}_0$ and $\mathcal{A}_1$ of same input/output domains, we define a probabilistic polynomial-time algorithm $\mathcal{D}$ and its advantage

$$\mathsf{Adv}_{\mathcal{D}}^{\mathcal{A}_0,\mathcal{A}_1}(k) = \left|\Pr[\mathcal{D}^{\mathcal{A}_0}(1^k) \to 1] - \Pr[\mathcal{D}^{\mathcal{A}_1}(1^k) \to 1]\right|,$$

with the probability being taken over the random tape of all the algorithms. $\mathcal{A}_0$ and $\mathcal{A}_1$ are said to be computationally indistinguishable, if for every distinguisher $\mathcal{D}$, we have $\mathsf{Adv}_{\mathcal{D}}^{\mathcal{A}_0,\mathcal{A}_1}(k) = \mathsf{negl}(k)$.

Sampling Algorithms. An efficient sampling algorithm for a probability distribution p is a polynomial-time probabilistic algorithm, in k, denoted Samp, that, on input random coins $\rho \in \{0,1\}^{\ell(k)}$, with $\ell(\cdot)$ being a polynomial function, outputs vector elements from X such that $|\Pr_\rho[\mathsf{Samp}(\rho) = x] - p(x)| = \mathsf{negl}(k)$. We say that a sampling algorithm Samp is inverse-samplable if it is invertible and some conditions on the distributions are fulfilled.

Definition 1 (Inverse-Sampling Algorithm [23]). *We say that an efficient sampling algorithm* Samp *is inverse-samplable if there exists a polynomial-time inverter algorithm* Samp^{-1} *such that* $(\rho, \mathsf{Samp}(\rho))$ *and* $(\mathsf{Samp}^{-1}(x), x) | x = \mathsf{Samp}(\rho)$ *are indistinguishable*

Public-Key Encryption Schemes. A public-key encryption scheme consists of three polynomial-time probabilistic algorithms denoted KeyGen, Enc, and Dec such that for all $k \in \mathbb{N}$, $\Pr[\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m | \mathsf{KeyGen}(1^k) \to (sk, pk)] = 1$. The decryption algorithm may output $\perp$ if it could not decrypt a ciphertext c. We use the standard notions of IND-CPA and IND-CCA security. In the security game, the advantage of the adversary is denoted $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}$ resp. $\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}$.

3 A Model for RFID Security and Privacy

Throughout this section, we recall the definitions in the Vaudenay model and our proposed updates. Most of what follows is taken from [34].

An RFID system is defined by a pair of two probabilistic polynomial-time algorithms and one two-party protocol to be executed between the reader and a tag. A first algorithm SetupReader is used to initialize the reader. It creates a pair of secret/public key (sk, pk) (typically, no public-key cryptography is used and $pk = \perp$). The second algorithm is for the creation of the tags and is $\mathsf{SetupTag}_{pk}(\mathsf{ID}) \rightarrow (K_{\mathsf{ID}}, S_{\mathsf{ID}})$, where ID refers to the identifier of the new tag. When the tag is *legitimate*, the tag secret K_{ID} is stored along with ID in the database; while the tag's initial state S_{ID} is always put inside the tag. An *illegitimate* tag has no entry in the database. Finally, a polynomial-time interactive protocol between the reader and a tag ID in which the reader ends up with a tape Output and the tag ends up with a tape $\mathsf{Output}_{\mathsf{ID}}$ completes the definition of an RFID system. By convention, if the protocol fails from the reader's perspective, we set $\mathsf{Output} = \perp$. When the protocol does not feature reader authentication, $\mathsf{Output}_{\mathsf{ID}}$ is void.

Simple RFID Protocols. We focus on a relevant class of RFID schemes called *simple*. These are 2-path protocols in which the reader sends a challenge and receives an answer. Then, it looks for a $(\mathsf{ID}, K_{\mathsf{ID}})$ database entry satisfying a predicate Ψ. The found pair identifies the tag and may be updated.

Definition 2 (Simple RFID Scheme). *An RFID scheme is said to be simple if the following conditions are fulfilled:*

- *The reader sends a query to the database with its secret key sk and the (possibly partial) transcript τ_p obtained from a protocol session.*
- *There exists a predicate Ψ, i.e., a deterministic polynomial-time algorithm that outputs a single bit, that takes as input sk, τ_p, and a database entry $(\mathsf{ID}, K_{\mathsf{ID}})$ such that the response from the database is computed by returning a database entry, picked uniformly, that satisfies the predicate.*
- *Once a tag ID has been identified in the database, its corresponding secret in the database, K_{ID}, may be updated to a new value. When it takes place, this procedure is carried out by an algorithm* Update *taking as input sk,* ID*, K_{ID}, and the full transcript of the protocol instance τ. This algorithm outputs a new K_{ID} and the database entry $(\mathsf{ID}, K_{\mathsf{ID}})$ is updated.*

It is straightforward to check that simple RFID schemes following Def. 2 satisfy the more general definition from [34].

Fig. 1 represents a simple RFID scheme from [34] which is based on a public-key cryptosystem. In what follows we call it the PKC protocol. In this scheme, the state of the tags is composed of their ID and a uniformly distributed κ-bit string K_{ID}. Upon reception of an α-bit string challenge a, a tag sends the encryption of $\mathsf{ID} \| K_{\mathsf{ID}} \| a$ under the public key pk to the reader. The latter decrypts the received ciphertext using its secret key sk and checks that it is well formed, that a is correctly recovered and that (ID, K) exists in the database.

Adversaries. Adversaries can request the creation of legitimate and illegitimate RFID tags. Furthermore, adversaries have the ability to draw one or more anonymous RFID tags, according to a chosen probability distribution. All interactions with the reader

Tag		**System**
State: $pk, \mathsf{ID}, K_{\mathsf{ID}}$		**Secret key:** sk **DB:** $\{\ldots, (\mathsf{ID}, K_{\mathsf{ID}}), \ldots\}$
	$\xleftarrow{a}$	Choose $a \in_R \{0,1\}^{\alpha}$
$c = \mathsf{Enc}_{pk}(\mathsf{ID} \Vert K_{\mathsf{ID}} \Vert a)$	$\xrightarrow{c}$	Search $(\mathsf{ID}, K_{\mathsf{ID}}) \in \mathsf{DB} : \mathsf{Dec}_{sk}(c) = \mathsf{ID} \Vert K_{\mathsf{ID}} \Vert a$ **Output:** ID or $\perp$

Fig. 1. PKC Protocol: an RFID scheme based on a public-key cryptosystem

and the drawn tags is controlled by the adversary. Moreover, the adversary has also the ability to tamper with any drawn tag and to retrieve its internal state. After a while, the adversary has also the possibility to release the tag so that it can be drawn again.

Definition 3 (Adversary [34]). *An adversary is a probabilistic polynomial-time algorithm. It takes a public key pk as input and has access to the following interfaces:*

- CREATETAG$^b(\mathsf{ID})$*: create a tag with unique identifier* ID*. Depending on the bit b submitted by the adversary, the tag may be legitimate, when* $b = 1$*, or illegitimate, when* $b = 0$*. After calling upon* $\mathsf{SetupTag}_{pk}(\mathsf{ID}) \rightarrow (K_{\mathsf{ID}}, S_{\mathsf{ID}})$ *for both type of tags, the pair* $(\mathsf{ID}, K_{\mathsf{ID}})$ *is inserted into the database if the adversary queried for a legitimate tag.*
- DRAWTAG$(\mathsf{Samp}) \rightarrow ((\mathsf{vtag}_1, b_1), \ldots, (\mathsf{vtag}_n, b_n))$*: select a vector of tags following a polynomial-time sampling algorithm* Samp*. During the period in which a tag is drawn, the adversary has complete control over its interactions. Along* vtag*, a bit b, set to 1 whenever the drawn tag is legitimate and to 0 otherwise, is returned. When a tag is drawn, it is designated by a unique virtual fresh identifier* vtag*. Drawing a tag that was already drawn makes the oracle output* $\perp$*.*[1] *Additionally, this interface keeps a private table* $\mathcal{T}$ *that keeps track of the real identifier of each drawn tag, i.e., it is such that* $\mathcal{T}(\mathsf{vtag})$ *is the real identifier of the virtual tag* vtag*.*
- FREE(vtag)*: release the RFID tag* vtag*. Once* vtag *is released, the adversary can no longer communicate with it (except under another pseudonym if it may be drawn again). Furthermore, its temporary memory is cleared to prevent a protocol session to span under several* vtag *pseudonyms.*[2]
- LAUNCH $\rightarrow \pi$*: make the reader launch a new protocol instance* π*. The returned* π *is a session identifier which can be assumed to be based on a counter.*
- SENDREADER$(m, \pi) \rightarrow m'$*: send a message m to a protocol instance* π *for the reader.*
- SENDTAG$(m, \mathsf{vtag}) \rightarrow m'$*: send a message m for the drawn tag* vtag *and receive the answer* m'*.*
- RESULT$(\pi) \rightarrow x$*: return the result of the completed protocol instance* π*. Namely, it yields* 0 *when* Output $= \perp$ *and* 1 *otherwise.*

[1] Definition 3 only differs from the original one in [34] in the introduction of the sampling algorithm Samp in DRAWTAG queries. Vaudenay [34] uses the term of "distribution" for the input of DRAWTAG although its representation may have exponential length.

[2] The clearance of the temporary memory upon a FREE call was introduced in Paise-Vaudenay [32]. It also meets the notion of "clean tag" by Deng et al. [17].

– CORRUPT(vtag) $\rightarrow S$: *return the current state* S *of the tag* $\mathcal{T}(\mathsf{vtag})$. *It does not return the content of the temporary memory of the tag.*

We consider several classes of adversaries. Weak adversaries do not use the CORRUPT interface. Forward adversaries can only use the CORRUPT interface at the end. Namely, no other interface can be used after the CORRUPT one. All other adversaries are Strong adversaries. We clearly have Weak $\subseteq$ Forward $\subseteq$ Strong. Adversaries that do not have access to the side channel information on the output of the protocol, i.e. to the RESULT oracle, are called NARROW. For any class P of adversaries, we define its Narrow counterpart for which we clearly have $\mathsf{Narrow\text{-}P} \subseteq \mathsf{P}$.

In the sequel, we restrict to adversaries who use distributions to the DRAWTAG such that, at any step, the table $\mathcal{T}$ can be successfully simulated by an algorithm that is only given the view of the adversary as input. That is, we require adversaries to only submit sampling algorithms that are inverse-samplable and allow them to compute a plausible guess for the identity of drawn tags in polynomial-time. For this we introduce a new notion: simulatable adversaries.

Definition 4 (Simulatable adversary). *Let* $\mathcal{A}$ *be an adversary interacting with an RFID system. Let* $\mathsf{view}^t_{\mathcal{A}}$ *be the view of* $\mathcal{A}$ *at its* t*-th step and let* $\mathcal{T}^t$ *denote the table* $\mathcal{T}$ *of the* DRAWTAG *oracle at step* t *of* $\mathcal{A}$. *We say that the adversary* $\mathcal{A}$ *is simulatable if all her sampling algorithms submitted to* DRAWTAG *are inverse-samplable and, for all* t*, there exists a polynomial-time algorithm* $\mathcal{A}'$*, such that* $(view^t_{\mathcal{A}}, \mathcal{T}^t)$ *and* $(view^t_{\mathcal{A}}, \mathcal{A}'(view^t_{\mathcal{A}}))$ *are indistinguishable.*

We note that when the adversary only draws one tag at the time (or in general, a vector of logarithmic length), then our restrictions do not affect the original definition as any sampling algorithm over such a set is inverse-samplable. So, we believe that our restriction to simulatable adversaries is harmless.

Correctness. Basically, correctness formalizes the fact that whenever the reader and a tag ID participate in an undisturbed protocol session, the reader authenticates the tag, that is, it ends up with $\mathsf{Output} = \mathsf{ID}$, except with a small negligible probability. We include all malicious behaviors as it was done in [17].

Definition 5 (Correctness). *A scheme is correct if for any Strong adversary* $\mathcal{A}$*, whenever there is a matching conversation between a tag of identity* ID *produced by* DRAWTAG $\rightarrow (\mathsf{vtag}, b)$ *and a reader instance* π*, except with negligible probability,* π *ends up with output* $\perp$ *if* $b = 0$ *and* ID *if* $b = 1$.

Clearly, the PKC protocol is correct.

Security. Security formalizes the fact that no adversary should be able to make the reader accept on a protocol session in which the adversary has been actively involved. Roughly, an RFID scheme is said to be secure if no adversary is able to make a reader protocol instance recognize an uncorrupted tag ID even if she corrupts all the other tags, unless π and the tag have a matching conversation.

It has been shown in [34] that, for the case of a simple RFID scheme, the notion of security reduces to an adversary playing the following game: create (and draw) a single

tag ID; use LAUNCH, SENDREADER, SENDTAG; use an oracle who checks the predicate $\Psi(sk,\cdot,\cdot,\cdot)$ on inputs different from ID; end on a final SEND command to complete the instance for the reader and the tag. The adversary wins the simple security game if one protocol instance on the reader identified tag ID but had no matching conversation. If the success probability of any adversary in wining the security experiment is negligible, then the scheme is simply secure. For simple schemes, simple security implies security.

It was shown in [34] that the PKC protocol is secure when the cryptosystem is IND-CCA secure.

When the protocol includes reader authentication, a security notion for the reader, in which the adversary's goal is to make the tag accept the reader, also needs to be defined. This was done in [32].

Privacy. An RFID scheme is private if no adversary can learn any information about the identity of drawn tags which is non-trivial. The information is trivial if the protocol messages could be simulated without interacting with the tags or reader and without affecting the output of the adversary. The simulation is performed by a process called a blinder.

Definition 6 (Blinder). *A blinder B for an adversary $\mathcal{A}$ is a polynomial-time algorithm which sees the same view as $\mathcal{A}$ (i.e, all the incoming messages and used coins), records all the adversary's Oracle queries and simulates all the* LAUNCH, SENDREADER, SENDTAG, RESULT *oracles to $\mathcal{A}$. A blinded adversary $\mathcal{A}^B$ is an adversary who does not produce any* LAUNCH, SENDREADER, SENDTAG, RESULT *oracles query but have them simulated by B.*

This definition changes from [34] by letting the blinder see the random tape of the adversary. This is a crucial change as the impossibility result in the Vaudenay model came from that adversaries could ask questions to the system for which they knew the answer but such that it could not be simulated. Providing used coins to the blinder allows it to "read the adversary's mind" and simulate the answer from the system.

Definition 7 (Privacy). *We consider simulatable adversaries who start with an attack phase consisting of* only *oracle queries and some computations then pursuing an analysis phase with no oracle query. In between phases, the adversary receives the hidden table $\mathcal{T}$ of the* DRAWTAG *oracle then outputs* true *or* false*. The adversary wins if the output is* true*. We say that the RFID scheme is P-private if for such adversary $\mathcal{A}$ which belongs to class P there exists a blinder B for which we have $|\Pr[\mathcal{A}\ \mathsf{wins}] - \Pr[\mathcal{A}^B\ \mathsf{wins}]|$ is negligible.*

Again, we only introduced that adversaries must be simulatable in this definition. Clearly, all positive results from [34] hold with these new definitions since privacy is defined by some property of form $\forall\mathcal{A}\ \exists\mathcal{B}$, our new adversaries are compatible with the old definition, and old blinders are compatible with the new definition. Namely:

- Weak privacy can be achieved using pseudo-random functions;
- The PKC protocol with IND-CCA encryption is Forward private;
- The randomized OSK protocol is Narrow-Forward private in ROM.

However, the impossibility of strong privacy may not hold anymore since it is a property of form $\exists\mathcal{A}\ \forall\mathcal{B}$. Actually, we will show that it no longer holds. This is similarly the case for the impossibility result by Armknecht et al. [1] for Narrow-Strong privacy and reader authentication, when mutual authentication is considered.

4 Plaintext-Awareness

Plaintext-awareness states that if an adversary is able to produce a valid ciphertext, then she should know the corresponding plaintext. Formalizing this notion has proven to be a non-trivial task [4,5,6,8,18]. In the end, several notions of plaintext-awareness were defined, such as, PA1, PA2, PA1+, and PA2+.

The difference between PA1 and PA2 lies in the attacker's ability to get ciphertexts from external sources. In the settings of PA2, there is an oracle $\mathcal{P}(\mathsf{aux})$, called plaintext creator and such that, on each query, it picks a message at random (or possibly according to a distribution partially defined by its input aux). The adversary can query $\mathsf{Enc}_{pk}(\mathcal{P}(\mathsf{aux}))$. Any ciphertext obtained through this oracle is added to a list CList, the list of ciphertexts for which the adversary does not know the corresponding plaintexts. The essence of plaintext-awareness is the existence of a polynomial-time algorithm $\mathcal{A}^\star$ (which construction may depend on $\mathcal{A}$), called plaintext extractor that successfully decrypts any ciphertext given by the adversary that was not returned by $\mathsf{Enc}_{pk}(\mathcal{P}(\mathsf{aux}))$. To carry out the extraction, $\mathcal{A}^\star$ is given the view of $\mathcal{A}$ (which includes CList and the random coins of $\mathcal{A}$) and the target ciphertext c to be decrypted for $c \notin \mathsf{CList}$.

To formalize information coming from external sources, Dent [18] extended PA1 to PA1+ for adversaries who can get hold of uniformly distributed bits from an external source. Later, Birkett and Dent [8] introduced the analog notion of PA2+ for PA2 plaintext-awareness. These last two notions were proven to be equivalent under the condition that the encryption scheme is IND-CPA [8]. PA1+ was also shown to imply PA2+ for simulatable encryption schemes [7]. We extend them to PA1++ and PA2++ when using any inverse-simulatable source.

Definition 8 (PA encryption). *Let O_1 denote an oracle that returns a single uniformly distributed bit. Let O_S be an oracle who takes as input an inverse-sampling algorithm and executes it using his own random tape. Given $* \in \{\mathsf{PA1}, \mathsf{PA1+}, \mathsf{PA1++}, \mathsf{PA2}, \mathsf{PA2+}, \mathsf{PA2++}\}$, we say that a public key cryptosystem $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is $*$-plaintext-aware if $\forall\mathcal{A}$, $\exists\mathcal{A}^\star$, $\forall\mathcal{P}$, all probabilistic polynomial-time, $(pk, \mathcal{A}^{O_*(\mathsf{Dec}_{sk}(\mathcal{P}(\cdot)))}(pk))$ and $(pk, \mathcal{A}^{O_*(\mathcal{A}^\star(pk,\cdot,\mathsf{view}_\mathcal{A}))}(pk))$ are indistinguishable, where*

$$\begin{aligned} O_{\mathsf{PA1}}(o) &= (o) & O_{\mathsf{PA2}}(o) &= (\mathsf{Enc}_{pk}(\mathcal{P}(\cdot)), o) \\ O_{\mathsf{PA1+}}(o) &= (O_1, o) & O_{\mathsf{PA2+}}(o) &= (\mathsf{Enc}_{pk}(\mathcal{P}(\cdot)), O_1, o) \\ O_{\mathsf{PA1++}}(o) &= (O_S, o) & O_{\mathsf{PA2++}}(o) &= (\mathsf{Enc}_{pk}(\mathcal{P}(\cdot)), O_S, o) \end{aligned}$$

Note that PA1++ (resp. PA2++) plaintext-awareness trivially implies PA1+ (resp. PA2+). Actually, we can even show equivalence.

Theorem 9. *PA1+ and PA1++ (resp. PA2+ and PA2++) are equivalent.*

Proof. We prove the theorem for the case of PA1++. It can be easily modified so that it applies to PA2++. We thus assume PA1+ plaintext awareness. Let $\mathcal{A}$ be a PA1++ ciphertext creator. We want to construct a plaintext extractor $\mathcal{A}^\star$.

We construct a PA1+ ciphertext creator $\mathcal{B}$ as follows: $\mathcal{B}$ takes input pk and simulates $\mathcal{A}$, forwarding all its decryption queries to the decryption oracle. In order to answer $\mathcal{A}$'s queries to the randomness oracle, $\mathcal{B}$ runs the provided sampling algorithm and query its randomness oracle, that we denote O_1, every time a new random bit is asked for. Clearly, $\mathcal{B}$ terminates in polynomial-time if all samplings can be performed in polynomial-time. Remark that $\mathcal{B}$ does not use any internal randomness besides the one used to initialize $\mathcal{A}$.

Since $\mathcal{B}$ is a valid PA1+ ciphertext creator, we can assert the existence of a plaintext extractor $\mathcal{B}^\star$ indistinguishable from a decryption oracle. We use $\mathcal{B}^\star$ to construct a plaintext extractor $\mathcal{A}^\star$ for $\mathcal{A}$. In the following, we assume that $\mathcal{A}^\star$ maintains a state view' initialized to $\mathsf{view}_{\mathcal{A}}$ that will be used to simulate $\mathcal{B}$'s view. To answer $\mathcal{A}$'s decryption queries, $\mathcal{A}^\star$ proceeds as follow:

1. If $\mathcal{A}$ queried the randomness oracle with an inverse-sampling algorithm Samp and received x since the last invocation of $\mathcal{A}^\star$, then $\mathcal{A}^\star$ computes $\rho_S \leftarrow \mathsf{Samp}^{-1}(x)$. After that, $\mathcal{A}^\star$ updates the simulated view of $\mathcal{B}$ to include the random bits ρ_S, i.e., it sets $\mathsf{view}' \leftarrow \mathsf{view}' \| \rho_S$. Due to the property of inverse-sampling algorithms, $(\rho_S, \mathsf{view}_{\mathcal{A}})$ is indistinguishable from $(\rho, \mathsf{view}_{\mathcal{A}})$, where ρ is the random string returned by O_S for the sampling request. Thus, by induction we show that $\mathsf{view}_{\mathcal{B}}$ and view' are indistinguishable. This procedure is repeated for every new sampling query.
2. $\mathcal{A}^\star$ then calls upon $\mathcal{B}^\star(pk, c, \mathsf{view}')$ and forwards its output to $\mathcal{A}$.

Since $\mathsf{view}_{\mathcal{A}}$ is included in view' which is indistinguishable from $\mathsf{view}_{\mathcal{B}}$,

$$\left|\Pr[\mathcal{D}^{\mathcal{A}^{\mathcal{B}^\star(pk,\cdot,\mathsf{view}'),O_S(pk)}}(1^k) \to 1] - \Pr[\mathcal{D}^{\mathcal{A}^{\mathcal{B}^\star(pk,\cdot,\mathsf{view}_{\mathcal{B}}),O_S(pk)}}(1^k) \to]\right| = \mathsf{negl}(k).$$

Recalling that $\mathcal{B}^\star(pk, \cdot, \mathsf{view}_{\mathcal{B}})$ is indistinguishable from a decryption oracle to $\mathcal{A}$, we deduce that $\mathcal{A}^\star$ is a valid plaintext extractor. In other words,

$$\left|\Pr[\mathcal{D}^{\mathcal{A}^{\mathcal{B}^\star(pk,\cdot,\mathsf{view}'),O_S(pk)}}(1^k) \to 1] - \Pr[\mathcal{D}^{\mathsf{Dec}_{sk}(\cdot),O_S(pk)}(1^k) \to 1]\right| = \mathsf{negl}(k).$$

This concludes the proof. □

The following corollary results from the combination of Theorem 9 with the equivalence result between PA2+ and PA2 [7], under the assumption that the scheme is IND-CPA secure.

Corollary 10. *If an encryption scheme is IND-CPA and PA2 plaintext-aware, then it is PA1++ plaintext-aware.*

5 Strong Privacy Is Possible

In this section, we show that using the new definition of blinders, we can achieve Strong privacy using plaintext-aware encryption schemes.

We consider the PKC protocol in Fig. 1. It has already been used by Vaudenay [34] to achieve Narrow-Strong privacy under the assumption that the underlying encryption scheme is IND-CPA secure, our result requires PA1+ plaintext-awareness from the encryption scheme. Naturally, since our definition of security is unchanged from the original model, IND-CCA security for the encryption scheme is sufficient to prove security [34].

Theorem 11 (PKC protocol achieves strong Privacy). *Assume a cryptosystem which is correct, PA1+ plaintext-aware, and IND-CCA secure. If* $2^{-\kappa}$ *and* $2^{-\alpha}$ *are negligible, then the PKC protocol is correct, secure, and Strong private.*

In Section A, we have shown that IND-CCA security alone is insufficient to prove this kind of result.

Note that in light of Corollary 10, we can implement the encryption scheme by a simulatable, PA1+ plaintext-aware, and IND-CPA secure public-key encryption scheme. Since the Cramer-Shoup [14] and Kurosawa-Desmedt [26] encryption schemes satisfy all these notions [7,24] (under certain extractor-based assumptions), any of these two schemes can be used.

Proof. First note that by Theorem 9, the encryption scheme is PA1++ plaintext-aware. Correctness is trivially induced by the correctness of the encryption scheme while security follows from IND-CCA security and [34, Theorem 19].

Therefore, we only need to prove privacy. To conduct the proof, we consider a Strong adversary $\mathcal{A}$ and construct a blinder iteratively. That is, we construct a sequence of partial blinders $B_1,\ldots,B_5$ and let $\mathcal{A}_i = \mathcal{A}_{i-1}^{B_i}$ with $\mathcal{A}_0 = \mathcal{A}$. The final blinder for $\mathcal{A}$ is $B = B_1 \circ \cdots \circ B_5$. By showing that the outcome of $\mathcal{A}_i$ and $\mathcal{A}_{i+1}$ are computationally indistinguishable, we deduce that B is indeed a full blinder for $\mathcal{A}$. So, the scheme is Strong private.

Game 0. Let Game 0 be the privacy game played by the adversary $\mathcal{A}_0$.

Game 1. We let Game 1 denote the privacy game performed by an adversary who simulates every RESULT on a session π with a transcript (a,c), such that c that has been obtained by a previous $c' = \text{SENDTAG}(\mathsf{vtag}, a')$ query. If $a \neq a'$, for sure c does not decrypt to something containing a, so the answer to RESULT(π) must be 0. The simulation is easy and perfect. In the other case, that is, if $a = a'$, the decryption of c will be parsed to a matching challenge a and some entry $\mathsf{ID}\|K_{\mathsf{ID}}$ which is in the database if and only if vtag is legitimate. Fortunately, the blinder has access to this latter information as it is returned in the response of the DRAWTAG oracle query drawing vtag. Again, the simulation is easy and perfect. This fully defines B_1 and we deduce that $\Pr[\mathcal{A}_0 \text{ wins}] = \Pr[\mathcal{A}_0^{B_1} \text{ wins}]$. We can thus define the adversary $\mathcal{A}_1$ that never queries RESULT on an instance π in which the response c was produced by a previous SENDTAG query.

Game 2. In this game, we make all SENDTAG queries being simulated by a partial blinder B_2. To achieve this, we let r be number of SENDTAG queries and make a sequence of hybrid blinders $B_2^0,\ldots,B_2^r$ in which B_2^i simulates the i first SENDTAG queries. Note that B_2^0 does not make any simulation so $\mathcal{A}_1^{B_2^0}$ is exactly $\mathcal{A}_1$ and that B_2^r is a partial blinder

for all SENDTAG queries. We define the hybrid B_2^i by simulating the i first encountered SENDTAG queries by encrypting random strings of same length as $\mathsf{ID}\|K_{\mathsf{ID}}\|a$.

To prove that $\mathcal{A}_1^{B_2^{i-1}}$ and $\mathcal{A}_1^{B_2^i}$ have computationally indistinguishable distributions, we construct an adversary $\mathcal{C}$ playing the IND-CCA game. Adversary $\mathcal{C}$ receives the public key and runs $\mathcal{A}_1^{B_2^{i-1}}$ or $\mathcal{A}_1^{B_2^i}$, depending on the bit of the indistinguishability game, while simulating the RFID system, except the i-th SENDTAG query. For that, $\mathcal{C}$ must simulate the environment for $\mathcal{A}_1^{B_2^{i-1}}/\mathcal{A}_1^{B_2^i}$. Since all algorithms and oracles of the scheme, except for RESULT, do not require the secret key, $\mathcal{C}$ can easily perform the simulation by itself. Regarding the RESULT interface, $\mathcal{C}$ just queries a decryption oracle and checks whether the decrypted message matches.

The first $i-1$ SENDTAG queries are made to the IND-CCA challenger in a real-or-random version. The challenge ciphertext c in the IND-CCA game is the answer from the challenger. It is either a real answer (as in the $\mathcal{A}_1^{B_2^{i-1}}$ simulation) or a simulated one (as in the $\mathcal{A}_1^{B_2^i}$ simulation). Note that no RESULT query is made on the session in which the adversary sent c (this case has been taken care of in Game 1). So, $\mathcal{C}$ perfectly simulates either the game for $\mathcal{A}_1^{B_2^{i-1}}$ or the game for $\mathcal{A}_1^{B_2^i}$ and is an IND-CCA adversary. Since $\mathcal{C}$ produces the output of $\mathcal{A}_1^{B_2^{i-1}}/\mathcal{A}_1^{B_2^i}$, we obtain that $\left|\Pr[\mathcal{A}_1^{B_2^i} \text{ wins}] - \Pr[\mathcal{A}_1^{B_2^{i+1}} \text{ wins}]\right| \leq \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}(k)$, and it results that $\left|\Pr[\mathcal{A}_1 \text{ wins}] - \Pr[\mathcal{A}_1^{B_2} \text{ wins}]\right| \leq r \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}(k)$, which is negligible as r is polynomially bounded and the scheme is IND-CCA secure.

At this point, we can legitimately consider an adversary $\mathcal{A}_2$ who makes no SENDTAG queries.

Game 3. We now simulate all remaining RESULT queries. To do so, we construct an adversary $\mathcal{E}$ playing the PA1++ game. The way $\mathcal{B}_3$ simulates RESULT will come from the $\mathcal{E}$ construction.

$\mathcal{E}$ takes the public key then simulates $\mathcal{A}_2$ interacting with the RFID system. Recall that, like in Game 2, the algorithms and oracles of the scheme do not depend on the secret key, except for the RESULT queries that will be treated hereafter. We let $\mathcal{E}$ simulate the RFID system to $\mathcal{A}_2$, handling her queries as follow:

- Assuming w.l.o.g. that session identifiers are based on a counter, LAUNCH is deterministically computed by $\mathcal{E}$.
- Upon a CREATETAG(ID) query from $\mathcal{A}_2$, $\mathcal{E}$ inserts $(\mathsf{ID}, -)$ in a table DB_1 if the query asks for a legitimate tag. Otherwise, it inserts $(\mathsf{ID}, -)$ in a table DB_0. This is deterministic.
- $\mathcal{E}$ simulates SENDREADER $\rightarrow a$ queries by asking the oracle O_S to sample from the uniform distribution over $\{0,1\}^\alpha$. It then forwards the received answer a to $\mathcal{A}_2$. This is non-deterministic but only requires uniformly distributed independent bits.
- DRAWTAG(Samp) queries are handled by asking the randomness oracle O_S to sample from the distribution specified by Samp to get one or more random ID. If any of the returned identifiers corresponds to a drawn tag, $\mathcal{E}$ outputs $\perp$. Otherwise, it generates, deterministically and for each returned ID_i, a fresh vtag_i and inserts

the pair $(\mathsf{vtag}_i, \mathsf{ID}_i)$ in the table $\mathcal{T}$. After that, it sets the bit b_i to 1 if ID_i is legitimate, or to 0 otherwise. At last, it returns $(\mathsf{vtag}_1, b_1, \ldots, \mathsf{vtag}_n, b_n)$ to $\mathcal{A}_2$. This is non-deterministic but requires inverse samplable distributions.

- CORRUPT(vtag) makes $\mathcal{E}$ reveal $\mathsf{ID} = \mathcal{T}(\mathsf{vtag})$. Moreover, $\mathcal{E}$ looks for the entry $(\mathsf{ID}, K_{\mathsf{ID}})$ in DB_0 and DB_1. If that corresponding entry contains a K_{ID} different from $'-'$, then it returns it. Otherwise, it queries O_S to sample from the uniform distribution over $\{0,1\}^\kappa$ and assigns the answer to K_{ID}. It subsequently updates the entry $(\mathsf{ID}, -)$ to $(\mathsf{ID}, K_{\mathsf{ID}})$ and returns this last pair as its answer. We further assume that whenever the tag ID is a legitimate one, $\mathcal{E}$ inserts the entry $(\mathsf{ID}, K_{\mathcal{T}(\mathsf{vtag})})$ in a table $\mathcal{T}_{\mathcal{E}}$. This is non-deterministic but only requires uniformly distributed independent bits. Note that non-corrupted tags have no preset K_{ID} key.
- To simulate the RESULT(π) oracle for a reader instance π with transcript (a,c), $\mathcal{E}$ sends c to the decryption oracle, checks that the recovered plaintext is of the form $\mathsf{ID}\|K_{\mathsf{ID}}\|a$, that it matches a $\mathsf{ID}\|K_{\mathsf{ID}} \in \mathsf{DB}_1$. (Note that this implies that tag ID, has been corrupted, and has key K_{ID}.) If this is the case, the answer to RESULT must be 1, otherwise, the simulated answer is 0. Note that when the output of the $\mathcal{E}$ regarding a RESULT query is 1, the genuine RESULT query would also have answered 1. So, this simulation is correct. Errors in the simulation only occur when $\mathcal{E}$ predicts 0 and the genuine RESULT query would also have outputted 1 in a session without matching conversation. Clearly, the failure of one of $\mathcal{E}$'s simulations corresponds to the happening of the event that there is a legitimate and uncorrupted tag which was identified by a session π which received a c which was not produced by any SENDTAG query. This implies that the event E that $\mathcal{A}_2$ wins the security game holds. In other words, $\left|\Pr[\mathcal{A}_2 \text{ wins}] - \Pr\left[\mathcal{A}_2^{\mathcal{E}} \text{ wins}\right]\right| \leq \Pr[E]$. Since it was shown that the PKC protocol is secure, this is negligible.

Since we assumed the encryption scheme to be PA1++ plaintext-aware, we can use the plaintext extractor $\mathcal{E}^\star$ of $\mathcal{E}$ to replace the decryption oracle without significantly altering the outcome distribution. However, $\mathcal{E}^\star$ requires the view of $\mathcal{E}$ instead of the view of $\mathcal{A}_2$, so we cannot use it as an extractor for $\mathcal{A}_2$. Fortunately, it is possible to reconstruct that view given the adversary's random tape and its queries. At first, we note that $\mathcal{E}$'s random coins are only used to initialize $\mathcal{A}_2$. Furthermore, all the randomness $\mathcal{E}$ obtains from O_S to process CORRUPT queries is revealed to $\mathcal{A}_2$. Moreover, since $\mathcal{A}_2$ is simulatable, we can use the algorithm $\mathcal{A}_2'$, induced by Definition 4, to reconstruct, from $\mathcal{A}_2$'s view, a table $\mathcal{T}'$ indistinguishable from $\mathcal{T}$. Since this table lists all the mappings between real and virtual identifiers, it is straightforward to reconstruct a randomness for $\mathcal{E}$ that she received to process the DRAWTAG queries using the Samp^{-1} algorithms corresponding to the sampling queries of $\mathcal{A}_2$. We let this whole operation be carried by a polynomial-time algorithm $\mathcal{V}$ that takes as input the view of $\mathcal{A}_2$ and uses $\mathcal{A}_2'$ to reconstruct a view of $\mathcal{E}$, i.e., it is such that $\mathcal{V}(\mathsf{view}_{\mathcal{A}_2})$ and $\mathsf{view}_{\mathcal{E}}$ are indistinguishable. It follows that $\mathcal{E}^\star(pk, \cdot, \mathcal{V}(\mathsf{view}_{\mathcal{A}_2}))$ and $\mathcal{E}^\star(pk, \cdot, \mathsf{view}_{\mathcal{E}})$ are indistinguishable.

At this point, we are able to define B_3, the partial blinder for RESULT queries. Similarly to $\mathcal{E}$, we assume that B_3 maintains a table $\mathcal{T}_{B_3}$ containing a list of pairs $(\mathsf{ID}, K_{\mathsf{ID}})$ for corrupted legitimate tags. In order to simulate a RESULT query on an instance π of transcript (a,c), the blinder proceed as follow.

1. First, the blinder calls $\mathcal{E}^{\star}(pk, c, \mathcal{V}(\mathsf{view}_{\mathcal{A}_2}))$ to get $\mathsf{Dec}_{sk}(c) = \mathsf{ID}\|K_{\mathsf{ID}}\|a'$.
2. Then it verifies that $a = a'$ and outputs 0 in case of failure. Otherwise, it continues.
3. At last, it outputs 1 if the pair $\mathsf{ID}\|K_{\mathsf{ID}}$ is listed in $\mathcal{T}_{B_3}$, and 0 otherwise.

The probability that Step 1 fails can be expressed as a distinguisher advantage of the PA1++ game or between $\mathcal{V}(\mathsf{view}_{\mathcal{A}_2})$ and $\mathsf{view}_{\mathcal{E}}$, so

$$\left|\Pr\left[\mathcal{A}_2^{B_3} \text{ wins}\right] - \Pr\left[\mathcal{A}_2^{\mathcal{E}} \text{ wins}\right]\right| \leq \mathsf{Adv}^{\mathsf{PA1++}}(k) + \mathsf{negl}(k).$$

At the same time, Step 3 fails when the event E occurs, so using triangle inequalities we conclude that

$$\left|\Pr[\mathcal{A}_2 \text{ wins}] - \Pr\left[\mathcal{A}_2^{B_3} \text{ wins}\right]\right| \leq \mathsf{Adv}^{\mathsf{PA1++}}(k) + \Pr[E] + \mathsf{negl}(k).$$

Recalling that E occurs with negligible probability and that the scheme is PA1++ plaintext-aware, the quantity above becomes negligible. Hence, B_3 describes a successful blinder for the RESULT oracle.

Game 4. In this game, we get rid of $\textsc{SendReader}(\pi) \to a$ queries. This can easily be achieved by constructing a blinder B_4 that returns uniformly distributed values from the set $\{0,1\}^{\alpha}$. We further get rid of the $\textsc{SendReader}(\pi, c)$ queries in a trivial way as they return nothing and are not followed by any $\textsc{Result}(\pi)$ query. Clearly, simulation is perfect as both distributions are perfectly indistinguishable. Hence, $\Pr[\mathcal{A}_3 \text{ wins}] = \Pr[\mathcal{A}_3^{B_4} \text{ wins}]$.

Game 5. Finally, we have an adversary $\mathcal{A}_4$ who only produces LAUNCH queries. We can trivially simulate the Them. It follows that $\Pr[\mathcal{A}_4 \text{ wins}] = \Pr[\mathcal{A}_4^{B_5} \text{ wins}]$. In the end, we have obtained an adversary $\mathcal{A}_5 = \mathcal{A}^B$, with $B = B_1 \circ \cdots \circ B_5$, who does not produce any oracle query that is such that $\left|\Pr[\mathcal{A} \text{ wins}] - \Pr[\mathcal{A}^B \text{ wins}]\right| = \mathsf{negl}(k)$. The scheme is thus Strong private. □

6 Conclusion

We updated the Vaudenay model for RFID privacy. Our model now makes it possible to achieve strong privacy. Actually, we proved that the regular PKC protocol with an IND-CCA and PA1+ secure cryptosystem achieves it. We have further shown that IND-CCA security alone could fail to reach this level of privacy. This shows a separation between our privacy model and the one from [22]. However, the question whether this separation is significant remains open.

Acknowledgement. The authors would like to than Sherman Chow for his valuable help in the final version of this paper.

References

1. Armknecht, F., Sadeghi, A.-R., Scafuro, A., Visconti, I., Wachsmann, C.: Impossibility Results for RFID Privacy Notions. In: Gavrilova, M.L., Tan, C.J.K., Moreno, E.D. (eds.) Transactions on Computational Science XI, Part II. LNCS, vol. 6480, pp. 39–63. Springer, Heidelberg (2010)

2. Avoine, G.: Cryptography in radio frequency identification and fair exchange protocols. PhD thesis, EPFL, Lausanne, Switzerland. Thesis N° 3407 (2005)
3. Avoine, G., Dysli, E., Oechslin, P.: Reducing Time Complexity in RFID Systems. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 291–306. Springer, Heidelberg (2006)
4. Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations among Notions of Security for Public-Key Encryption Schemes. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 26–45. Springer, Heidelberg (1998)
5. Bellare, M., Palacio, A.: Towards Plaintext-Aware Public-Key Encryption Without Random Oracles. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 48–62. Springer, Heidelberg (2004)
6. Bellare, M., Rogaway, P.: Optimal Asymmetric Encryption. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 92–111. Springer, Heidelberg (1995)
7. Birkett, J.: On Plaintext-Aware Public-Key Encryption Schemes. PhD thesis, Royal Holloway, University of London (2010)
8. Birkett, J., Dent, A.W.: Relations Among Notions of Plaintext Awareness. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 47–64. Springer, Heidelberg (2008)
9. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
10. Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash Functions and RFID Tags: Mind the Gap. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 283–299. Springer, Heidelberg (2008)
11. Burmester, M., van Le, T., de Medeiros, B.: Provably Secure Ubiquitous Systems: Universally Composable RFID Authentication Protocols. In: SecureComm 2006, Baltimore, Maryland, USA. IEEE Press (2006)
12. Canard, S., Coisel, I., Etrog, J., Girault, M.: Privacy-preserving RFID systems: Model and constructions. Cryptology ePrint Archive, Report 2010/405 (2010), http://eprint.iacr.org/
13. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2000), http://eprint.iacr.org/
14. Cramer, R., Shoup, V.: A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998)
15. Damgård, I., Pedersen, M.Ø.: RFID Security: Tradeoffs between Security and Efficiency. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 318–332. Springer, Heidelberg (2008)
16. Deng, R.H., Li, Y., Yao, A.C., Yung, M., Zhao, Y.: A new framework for RFID privacy. Cryptology ePrint Archive, Report 2010/059 (2010), http://eprint.iacr.org/
17. Deng, R.H., Li, Y., Yung, M., Zhao, Y.: A New Framework for RFID Privacy. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 1–18. Springer, Heidelberg (2010)
18. Dent, A.W.: The Cramer-Shoup Encryption Scheme Is Plaintext Aware in the Standard Model. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 289–307. Springer, Heidelberg (2006)
19. Feldhofer, M., Dominikus, S., Wolkerstorfer, J.: Strong Authentication for RFID Systems Using the AES Algorithm. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 357–370. Springer, Heidelberg (2004)
20. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences 28(2), 270–299 (1984)
21. Hein, D., Wolkerstorfer, J., Felber, N.: ECC Is Ready for RFID – A Proof in Silicon. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 401–413. Springer, Heidelberg (2009)

22. Hermans, J., Pashalidis, A., Vercauteren, F., Preneel, B.: A New RFID Privacy Model. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 568–587. Springer, Heidelberg (2011)
23. Ishai, Y., Kumarasubramanian, A., Orlandi, C., Sahai, A.: On Invertible Sampling and Adaptive Security. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 466–482. Springer, Heidelberg (2010)
24. Jiang, S., Wang, H.: Plaintext-Awareness of Hybrid Encryption. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 57–72. Springer, Heidelberg (2010)
25. Juels, A., Weis, S.A.: Defining strong privacy for RFID. In: PerCom Workshops 2007, pp. 342–347. IEEE Computer Society (2007)
26. Kurosawa, K., Desmedt, Y.: A New Paradigm of Hybrid Encryption Scheme. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 426–442. Springer, Heidelberg (2004)
27. van Le, T., Burmester, M., de Medeiros, B.: Universally composable and forward-secure RFID authentication and authenticated key exchange. In: ASIACCS 2007, pp. 242–252. ACM (2007)
28. Molnar, D., Wagner, D.: Privacy and security in library RFID: issues, practices, and architectures. In: CCS 2004, pp. 210–219. ACM (2004)
29. Ng, C.Y., Susilo, W., Mu, Y., Safavi-Naini, R.: RFID Privacy Models Revisited. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 251–266. Springer, Heidelberg (2008)
30. Ohkubo, M., Suzuki, K., Kinoshita, S.: RFID privacy issues and technical challenges. Commun. ACM 48(9), 66–71 (2005)
31. Ouafi, K., Phan, R.C.-W.: Traceable Privacy of Recent Provably-Secure RFID Protocols. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 479–489. Springer, Heidelberg (2008)
32. Paise, R.-I., Vaudenay, S.: Mutual authentication in RFID: security and privacy. In: Proceedings of the ASIACCS 2008, pp. 292–299. ACM (2008)
33. Shamir, A.: SQUASH – A New MAC with Provable Security Properties for Highly Constrained Devices Such as RFID Tags. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 144–157. Springer, Heidelberg (2008)
34. Vaudenay, S.: On Privacy Models for RFID. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 68–87. Springer, Heidelberg (2007)

A IND-CCA Security Is Not Sufficient for Strong Privacy

We define $(\mathsf{KeyGen}^1, \mathsf{Enc}^1, \mathsf{Dec}^1)$, a variant of the Goldwasser-Micali cryptosystem [20] as follows.

- $\mathsf{KeyGen}^1(1^k)$. Pick an RSA modulus $N = pq$, i.e, s.t. p and q are primes, and $y, z \in \mathbb{Z}_N^\star$ such that $\left(\frac{y}{N}\right) = +1$, $\left(\frac{y}{p}\right) = -1$, and $\left(\frac{z}{N}\right) = +1$. The scheme's key pair is $pk^1 = (N, y, z)$ and $sk^1 = p$.
- $\mathsf{Enc}^1_{pk^1}(b) = y^b r^2 \mod N$ where $b \in \{0,1\}$ and $r \in_R \mathbb{Z}_N^\star$.
- $\mathsf{Dec}^1_{sk^1}(c) = b$ such that $(-1)^b = \left(\frac{c}{p}\right)$.

Note that z in the public key is unused. Further note that $z \cdot \mathsf{Enc}^1{}_{pk^1}(b) \bmod N$ is a valid encryption of either b or $\bar{b} = 1 - b$ depending on $\left(\frac{z}{p}\right)$ which is unknown for someone holding the public key. Let $(\mathsf{KeyGen}^0, \mathsf{Enc}^0, \mathsf{Dec}^0)$ denote an IND-CCA secure encryption scheme, we define $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as follows.

- **KeyGen.** Run $(sk^0, pk^0) \leftarrow \mathsf{KeyGen}^0(1^k)$ and $(sk^1, pk^1) \leftarrow \mathsf{KeyGen}^1(1^k)$. The scheme's key pair is $pk = (pk^0, pk^1)$ and $sk = (sk^0, sk^1)$.
- **Encrypt.** To encrypt, set $\mathsf{Enc}_{pk}(x) = \mathsf{Enc}^0_{pk^0}\left(\mathsf{Enc}^1_{pk^1}(x_1), \ldots, \mathsf{Enc}^1_{pk^1}(x_n)\right)$ where $x_1, \ldots, x_n$ is the binary decomposition of x.
- **Decrypt.** To decrypt, compute $\mathsf{Dec}_{sk}(c) = \mathsf{Dec}^1_{sk^1}(t_1), \ldots, \mathsf{Dec}^1_{sk^1}(t_n)$ where $t_1, \ldots, t_n = \mathsf{Dec}^0_{sk^0}(c)$.

We can easily see that (KeyGen, Enc, Dec) is IND-CCA secure. It is not plaintext-aware since $\mathsf{Enc}^0_{pk^0}\left(z \cdot \mathsf{Enc}^1_{pk^1}(x_0) \bmod N, \ldots, z \cdot \mathsf{Enc}^1_{pk^1}(x_{n-1}) \bmod N\right)$ is a valid encryption of either $x_0, \ldots, x_{n-1}$ or $\overline{x_0}, \ldots, \overline{x_{n-1}}$ depending on $\left(\frac{z}{p}\right)$. To figure out whether this encrypts x or $\bar{x}$ would require to solve the quadratic residuosity problem, which is supposedly a hard problem.

Consider the PKC protocol of Fig. 1 using the above IND-CCA secure public-key encryption scheme.

Finally, the following Strong adversary defeats privacy.

1: CREATETAG(ID)
2: $\mathsf{v} \leftarrow$ DRAWTAG(ID)
3: $S_{\mathsf{ID}} \leftarrow$ CORRUPT(v)
4: $\pi \leftarrow$ LAUNCH
5: $a \leftarrow$ SENDREADER$(\emptyset, \pi)$
6: Set $x = S_{\mathsf{ID}} \| a = x_1, \ldots, x_n$
7: $s = z \cdot \mathsf{Enc}^1_{pk^1}(x_1), \ldots, z \cdot \mathsf{Enc}^1_{pk^1}(x_n)$
8: $c \leftarrow \mathsf{Enc}^0_{pk}(s)$
9: SENDREADER(c, π)
10: $b \leftarrow$ RESULT(π)
11: Output b

Clearly, an adversary outputs 1 if and only if $\left(\frac{z}{p}\right) = +1$. Therefore, a blinder that follows the same distribution would break the quadratic residuosity problem, i.e., the problem of distinguishing quadratic residues from non-quadratic residues. So, the PKC protocol based on this cryptosystem is strong-private in the model from [22] but is not in our model, assuming that quadratic residuosity is a hard problem. This proves the separation between privacy from [22] and our strong privacy. Indeed, we have shown that the PKC protocol based on an IND-CCA cryptosystem may still leak some non-simulatable information although it is strong private in the sense of [22].

How to Enhance the Security on the Least Significant Bit

Atsuko Miyaji* and Yiren Mo

Japan Advanced Institute of Science and Technology
miyaji@jaist.ac.jp

Abstract. Scalar multiplication, which computes dP for a given point P and a scalar d, is the dominant computation part of Elliptic Curve Cryptosystems (ECC). Recently, Side Channel Attacks (SCA) on scalar multiplication have become real threats. This is why secure and efficient scalar multiplication is important for ECC, and many countermeasures have been proposed so far. The Montgomery Ladder and the Regular right-to-left algorithm are the simplest and the most elegant algorithms. However, they are vulnerable to an SCA on the Least Significant Bit (LSB). In this paper, we investigate how to enhance the LSB security without spoiling the original features of simplicity. Our elegant techniques make the previous schemes secure against the SCA on LSB, while maintaining original performances.

Keywords: Elliptic Curve Cryptography, Scalar Multiplication, Side Channel Attack.

1 Introduction

The Elliptic Curve Cryptosystem (ECC), which uses a group of rational points of an elliptic curve over a finite field, was independently proposed by Miller and Koblitz in the mid 1980s. The security of ECC is based on the Elliptic Curve Discrete Logarithm Problem (ECDLP) and the scalar multiplication, which computes dP for a given point P and a scalar $d = \sum_{i=0}^{\ell-1} d_i 2^i (d > 0)$, is the dominant computation part of ECC. The simplest scalar multiplication is the so-called binary algorithm. The ECC has been attracting the attention of various applications on small devices because the ECC yield security with a compact memory and little computational cost. Especially, for the use of smart card, the ECC needs to be resistant to side channel attacks on scalar multiplication, such as Simple Power Analysis (SPA)[16], Differential Power Analysis (DPA)[16], Zero-value Point Attack (ZPA)[1], Refined Power Analysis (RPA)[7], Safe-Error Attack (SEA)[25], and etc.; and must still work with a compact memory and little computational cost. To address these two issues, many scalar

* This study is partly supported by Grant-in-Aid for Exploratory Research, 19650002.

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 263–279, 2012.

multiplications have been proposed so far. However, there is still room for improvement from the viewpoint of security, memory amount, and computational cost.

Let us review some previous results on right-to-left algorithms. The regular right-to-left algorithm [12] is secure against both SPA and SEA, while it works without extra computation and just repeats both doubling and additions. This is a simple and elegant algorithm. However, it unfortunately does not work regularly for an arbitrary scalar, and needs a special treatment to force the parity of d, that is, d_0 to be an odd number. In fact, not only the right-to-left algorithm but also other right-to-left algorithms (Algorithms 1 and 3 in [11]) do not work regularly for an arbitrary scalar. Two other right-to-left algorithms (Algorithms $1'$ and 2 in [11]) work for an arbitrary d, but they are vulnerable to SEA. In a sense, those right-to-left algorithms fail to achieve the security on the Least Significant Bit (LSB), which is called *LSB security* in this paper. The typical special treatments are: computing $(d-d_0+1)P$ and subtracting P at the end; or adding an appropriate multiple of $\mathrm{ord}(P)$ to d. The former method needs one additional subtraction at the end, and the latter method is useful for only scalars $d \ll \mathrm{ord}(P)$. Those special treatments seem to be rather exaggerated and spoil the simplicity of regular right-to-left algorithm. The problem is just on the LSB security, but is nevertheless unavoidable to execute any d.

On the other hand, from the viewpoint of countermeasure to ZPA, a Random-Initial-Point algorithm (RIP), which works regularly in the right-to-left way, was proposed in [10]. It is called IIT-RIP in this paper. RIP in the left-to-right way was proposed in [19], which is called MMM-RIP in this paper. Both algorithms enhance the security by just repeating both doubling and additions, without extra computation. In a sense, both IIT-RIP and MMM-RIP are also elegant and simple. Compared with MMM-RIP resistant to both SEA and ZPA, IIT-RIP, however, is vulnerable to SEA and needs one more register of points than MMM-RIP, although it is secure against ZPA. In order to enhance the security to SEA, error detection steps are introduced in [2,13]. These, however, need additional steps, and, thus, spoils the original simplicity.

Next, let us review some previous results on left-to-right algorithms. The Montgomery Ladder [23] works regularly in the right-to-left way with only 2 registers of points. However, it is vulnerable to SEA, and leaks LSB (See Alg. 7 in Section 2.5). The signed-digit algorithm proposed in [9] can also work regularly in the left-and-right way with only 2 registers of points in the same way as the Montgomery Ladder. The signed-digit algorithm is secure against SEA, but it is not available for even d. There is room for improvement for both the Montgomery Ladder and the signed-digit algorithm from the viewpoint of security and availability.

In this paper, we improve the right-to-left and left-to-right algorithms from the viewpoint of security, memory amount, computational cost, and availability. First, we improve Joye's regular right-to-left algorithm to work for an arbitrary scalar d, while maintaining the same memory amount as the original.

In this paper, our improved algorithm is called the subtracting-doubling algorithm. Next, we improve the IIT-RIP from the viewpoint of security and memory amount. Our improved algorithm can reduce one register of points from the IIT-RIP, and security is further enhanced. Then, we improve the Montgomery Ladder to be secure against SEA for an arbitrary scalar d, while maintaining the same memory amount as the original algorithm. Finally, we improve the signed-digit algorithm to work for an arbitrary scalar d with only 2 registers of points in the same way as the original.

This paper consists of 6 sections. Section 2 describes some known side channel attacks and the previous right-to-left and left-to-right algorithms. Section 3 presents two new right-to-left algorithms, which improve the IIT-RIP or Joye's regular right-to-left algorithms. Section 4 presents two new left-to-right algorithms, which improve the Montgomery Ladder or the signed-digit algorithm. Section 5 compares our proposed algorithms with the previous algorithms [10,12,23,9]. Section 6 concludes this paper.

2 Previous Results

2.1 Elliptic Curve, Coordinate System, and Scalar Multiplication

We can use several different coordinate systems to represent an elliptic curve. In this work, we assume that an elliptic curve E is defined over $\mathbb{F}_p$ with $p > 3$, and choose the Jacobian coordinate. Then, an elliptic curve is given by $E : y^2 = x^3 + ax + b$ $(a, b \in \mathbb{F}_p)$. The Jacobian coordinate and its variants are described in [5], whose doubling and addition can be slightly improved by changing multiplication to square such as $2Z_1Z_2 = (Z_1+Z_2)^2-Z_1^2-Z_2^2$. The latest addition and doubling formulae are available from [5], and the latest iterated doubling formulae are presented in [22]. The co-Z addition, ZADDU, deals with points having the same Z-coordinate [21], where $(R, P) \leftarrow \mathrm{ZADDU}(P, Q)$ is defined as: $R \leftarrow P + Q = (X_3 : Y_3 : Z_3)$ and $P \leftarrow (\lambda^2 X_1 : \lambda^3 Y_1 : Z_3)$ with $Z_3 = \lambda Z_1$ for input of $P \leftarrow (X_1 : Y_1 : Z)$ and $Q \leftarrow (X_2 : Y_2 : Z)$. The conjugate addition, which outputs $(P + Q, P - Q)$ from P and Q [18], is further improved to ZADDC by combining the co-Z [8], where $(R, S) \leftarrow \mathrm{ZADDC}(P, Q)$ is defined as $R \leftarrow P + Q = (X_3 : Y_3 : Z_3)$ and $S \leftarrow (\overline{X_3} : \overline{Y_3} : Z_3)$ for input of $P \leftarrow (X_1 : Y_1 : Z)$ and $Q \leftarrow (X_2 : Y_2 : Z)$.

Let us summarize the computational cost of the formulae. Here we denote the computational cost of multiplication, square, and inversion over a definition field by M, S and I. Then, the costs of the EC point addition, doubling, k-iterated doublings, and conjugate addition in Jacobian coordinate are $11M+5S$, $2M+8S$, $(3k-1)M+(5k+3)S$ and $12M+6S$, respectively. Note that k-iterated doublings are used in right-to-left algorithms, and the conjugate addition is used in both right-to-left and left-to-right algorithms. We also give the left-to-right and the right-to-left binary algorithms.

Addition formula (Jacobian coord.)	Doubling formula (Jacobian coord.)
$U_1 = X_1Z_2^2$, $U_2 = X_2Z_1^2$, $S_1 = Y_1Z_2^3$, $S_2 = Y_2Z_1^3$ $H = U_2 - U_1$, $I = (2H)^2$, $J = HI$, $R = 2(S_2 - S_1)$, $V = U_1I$ $X_3 = R^2 - J - 2V$, $Y_3 = R(V - X_3) - 2S_1J$, $Z_3 = ((Z_1 + Z_2)^2 - Z_1^2 - Z_2^2)H$	$S = 2((X_1 + Y_1^2)^2 - X_1^2 - Y_1^4)$, $M = 3X_1^2 + aZ_1^4$, $X_3 = M^2 - 2S$, $Y_3 = M(S - X_3) - 8Y_1^4$, $Z_3 = (Y_1 + Z_1)^2 - Y_1^2 - Z_1^2$

Iterated Doubling Formulae to compute 2^kP in Jacobian Coordinate

$Y_0' = 2Y_0, W_0 = aZ_0^4, T_0 = Y_0'^4$, $S = ((X_0 + Y_0'^2)^2 - X_0^2 - T_0)$, $M = 3X_0^2 + W_0$
$X_1 = M^2 - 2S$, $Y_1' = 2M(S - X_1) - T_0, Z_1 = ((Y_0' + Z_0)^2 - Y_0'^2 - Z_0^2)/2$
For $i = 1$ to $k - 1$: {
$W_i = W_{i-1}T_{i-1}, T_i = Y_i'^4, S = ((X_i + Y_i'^2)^2 - X_i^2 - T_i), M = 3X_i^2 + W_i$
$X_{i+1} = M^2 - 2S, Y_{i+1}' = 2M(S - X_{i+1}) - T_i, Z_{i+1} = Y_i'Z_i$
}
$Y_k = Y_k'/2$

Conjugate Addition Formulae in Jacobian Coordinate

$P = (X_1, Y_1, Z_1), Q = (X_2, Y_2, Z_2), P + Q = (X_3, Y_3, Z_3), P - Q = (X_4, Y_4, Z_4)$
$X_3 = A^2 - (4B^3 + 8Z_2^2X_1B^2), Y_3 = A(Z_2^2X_1B^2 - X_3) - Z_2^3Y_1B^3, Z_3 = DB$
$X_4 = C^2 - (4B^2 + 8Z_2^2X_1B^2), Y_4 = C(Z_2^2X_1B^2 - X_4) - Z_2^3Y_1B^3, Z_4 = Z_3$
$A = 2(Z_1^3Y_2 - Z_2^3Y_1), B = Z_1^2X_2 - Z_2^2X_1, C = -2(Z_1^3Y_2 + Z_2^3Y_1), D = (Z_1 + Z_2)^2 - Z_1^2 - Z_2^2$

Algorithm 1. Left-to-Right Binary Alg.

Input: P and $d = \sum_{i=0}^{\ell-1} d_i 2^i = d_{\ell-1}d_{\ell-2}\dots d_0$
Output: dP

1: $R[0] \leftarrow \mathcal{O}, R[1] \leftarrow P$
2: **for** $i = \ell - 1$ to 0 **do**
3: $\quad R[0] \leftarrow 2R[0]$
4: $\quad$ **if** $d_i = 1$ **then**
5: $\quad\quad R[0] \leftarrow R[0] + R[1]$
6: $\quad$ **end if**
7: **end for**
8: **return** $R[0]$

Algorithm 2. Right-to-Left Binary Alg.

Input: P and $d = \sum_{i=0}^{\ell-1} d_i 2^i = d_{\ell-1}d_{\ell-2}\dots d_0$
Output: dP

1: $R[0] \leftarrow \mathcal{O}, R[1] \leftarrow P$
2: **for** $i = 0$ to $\ell - 1$ **do**
3: $\quad$ **if** $d_i = 1$ **then**
4: $\quad\quad R[0] \leftarrow R[0] + R[1]$
5: $\quad$ **end if**
6: $\quad R[1] \leftarrow 2R[1]$
7: **end for**
8: **return** $R[0]$

2.2 Side Channel Attacks

Side Channel Attacks (SCA) are a type of attack which uses information taken from the physical implementation, such as Timing Analysis Attack [15], Simple Power Analysis (SPA) [16] and Differential Power Analysis (DPA) [16]. These are explained in [4]. Here, we summarize SPA and DPA. SPA observes a suitable side channel, such as the power consumption or electromagnetic emanations, and recovers secret information from the leaked information. In DPA, an attacker not only observes but also statistically analyzes the power consumption of a cryptosystem.

In addition to these attacks, the Doubling Attack, which works only in left-to-right algorithms, is proposed in [24], called DblA in this paper. This is because

left-to-right algorithms usually execute in such a way that a return value is doubled and added P when $d_i = 1$, where $d = \sum_{i=0}^{\ell} 2^i d_i$ is represented by $d_{\ell-1}d_{\ell-2}\ldots d_0$, and d_i is the current bit. Right-to-left algorithms usually execute in such a way that $2^i P$ is added to a return value when $d_i = 1$. Here, let us explain how DblA works in left-to-right algorithms. DblA computes dP and $d(2P)$. If $d_i = 0$, then the i^{th}-round $R[0]$ in the computation of dP is the same as the $(i-1)^{\text{th}}$-round $R[0]$ in the computation of $d(2P)$. In the $(i+1)^{\text{th}}$ round of dP and i^{th} round of $d(2P)$, each round executes a doubling. It is possible for an attacker to check whether result values of two doubling operations are the same.

Safe-Error Attack (SEA) timely induces a fault during the execution of an instruction [25], and, deduces whether a target instruction is dummy or not, because an induced error will be a safe-error when the corresponding operation is dummy. Let us explain SEA by taking the double-and-add always algorithm (Algorithm 3) as an example. Suppose that $R[1]$ in Step 4 for $i = i_0$ $(\ell - 1 \leq i_0 \leq 1)$ is attacked. Let $(R[0], R[1], R[2]) = (a, b, c)$ in the beginning of Step i_0. In the case of $(d_{i_0}, d_{i_0-1}) = (0, *)$, Algorithm 3 works as Table 1, where N/A means that the value is wrong: $R[1]$ has an error in $i = i_0$, while there is no error in either $R[0]$ or $R[2]$. However, the error in $R[1]$ will disappear in Step 4 for $i = i_0 - 1$ by inputting $R[1] \leftarrow R[0] + R[2]$. This is why the error of $R[1]$ for $i = i_0$ is a safe-error. In the case of $(d_{i_0}, d_{i_0-1}) = (1, *)$, Algorithm 3 works as Table 2: $R[1]$ has an error, where the error in $R[1]$ is copied into $R[0]$ by $R[0] \leftarrow R[1]$ in Step 5 for $i = i_0$; and, finally, both $R[0]$ and $R[1]$ have errors in Step 4 for $i = i_0 - 1$. This is why the error of $R[1]$ for $i = i_0$ is a real-error. Thus, we can detect $d_{i_0} = 0$ or 1 $(\ell - 1 \geq i_0 \geq 1)$ by using the fact of whether the output value is correct.

Table 1. Safe-Error

$(i$, Step)	Instruction	Value
$(i_0$, 3)	$R[0] \leftarrow 2R[0]$	$R[0] = 2a$
$(i_0$, 4)	$R[1] \leftarrow R[0] + R[2]$	$R[1] = N/A$
$(i_0$, 5)	$R[0] \leftarrow R[0]$	$R[0] = 2a$
$(i_0 - 1$, 3)	$R[0] \leftarrow 2R[0]$	$R[0] = 4a$
$(i_0 - 1$, 4)	$R[1] \leftarrow R[0] + R[2]$	$R[1] = 4a + c$
$(i_0 - 1$, 5)	$R[*] \leftarrow R[1]$	$R[0] = 4a$ or $4a + c$

Table 2. Real-Error

$(i$, Step)	Instruction	Value
$(i_0$, 3)	$R[0] \leftarrow 2R[0]$	$R[0] = 2a$
$(i_0$, 4)	$R[1] \leftarrow R[0] + R[2]$	$R[1] = N/A$
$(i_0$, 5)	$R[0] \leftarrow R[1]$	$R[0] = N/A$
$(i_0 - 1$, 3)	$R[0] \leftarrow 2R[0]$	$R[0] = N/A$
$(i_0 - 1$, 4)	$R[1] \leftarrow R[0] + R[2]$	$R[1] = N/A$
$(i_0 - 1$, 5)	$R[*] \leftarrow R[1]$	$R[0] = N/A$

2.3 Highly Regular Right-to-Left Scalar Multiplication Algorithm

Joye proposed a highly regular powering ladder [12], whose idea is to use a representation of $d - 1$ instead of d. The representation of $d - 1$ for the binary expansion of $d = \sum_{i=0}^{\ell-1} d_i 2^i (d_{\ell-1} = 1)$ is given as follows: $d - 1 = \sum_{i=0}^{\ell-2} (d_i + 1)2^i$. This follows easily by regarding -1 as $\underbrace{\bar{1}11\ldots11}_{l}$. Algorithm 4 shows his regular right-to-left scalar multiplication algorithm.

Algorithm 3. DBL-and-ADD always alg. (L-R)[6]

Input: P and $d = \sum_{i=0}^{\ell-1} d_i 2^i (d > 0)$
Output: dP

1: $R[0] \leftarrow P, R[2] \leftarrow P$
2: **for** $i = \ell - 1$ to 0 **do**
3: $\quad R[0] \leftarrow 2R[0]$
4: $\quad R[1] \leftarrow R[0] + R[2]$
5: $\quad R[0] \leftarrow R[d_i]$
6: **end for**
7: **return** $R[0]$

Algorithm 4. Regular Right-to-Left alg. [12]

Input: P and $d = \sum_{i=0}^{\ell-1} d_i 2^i (d > 1)$
Output: dP

1: $R[1] \leftarrow d_0 P, R[2] \leftarrow P, R[0] \leftarrow R[2]$
2: **for** $i = 1$ to $\ell - 2$ **do**
3: $\quad R[0] \leftarrow 2R[0]$
4: $\quad R[1 + d_i] \leftarrow R[1 + d_i] + R[0]$
5: **end for**
6: $R[0] \leftarrow R[1] + 2R[2]$
7: **return** $R[0]$

We should note that if $d_0 = 0$, this algorithm induces an addition to $\mathcal{O}$ in the first $i \geq 1$ with $d_i = 0$: $R[1] \leftarrow \mathcal{O} + R[0]$ in Step 4, since $R[1] = \mathcal{O}$ until the i. Thus, Algorithm 4 itself can securely execute only for an odd d. In order to enhance the LSB security of d, some *special treatment*, as described in [12,11], is needed to force the parity of d to 1 such as: computing $(d - d_0 + 1)P$ and subtracting P at the end; or adding an appropriate multiple of $\mathrm{ord}(P)$ to d. The former method needs one additional subtraction at the end, and the latter method is useful for only scalars $d \ll \mathrm{ord}(P)$. Those special treatments seem to be rather exaggerated and spoil the simplicity of Algorithm 4. The problem exists only in the LSB, but it is nevertheless unavoidable to execute any d. We'll propose a simple method to enhance the LSB security, which does not need any additional computation, and works for any $\mathrm{ord}(P)$.

Here, we investigate the LSB security of previous right-to-left scalar multiplication algorithms in [12,11]. Not only Algorithms 4, but also other right-to-left scalar multiplication algorithms (Algorithms 1 and 3 in [11]) have an initial value of $\mathcal{O}$ in spite of $d_0 = 0$ or 1. As a result, these algorithms also need a *special treatment* in order to work on an arbitrary d. Two other right-to-left algorithms (Algorithms $1'$ and 2 in [11]) work for both even and odd d, since they have no initial value with $\mathcal{O}$. They are, however, vulnerable to SEA in the last step: if $k_0 = 1$, then an error induced on $R[b] \leftarrow R[b] - P$ will be a safe error, because a return value is $R[0]$, and the error is in $R[1]$. Therefore, those previous right-to-left algorithms in [12,11] leak LSB of the scalar, need a special treatment for an arbitrary d, or are vulnerable to SEA.

2.4 Left-to-Right and Right-to-Left RIP Algorithms

There are several countermeasures against DPA attacks, such as the Randomized Projective coordinate method (RPC)[6], the Randomized Curve method (RC) [14], the Exponent Splitting method (ES)[3] and the Random Initial Point method (RIP)[19,10]. Both RPC and RC are vulnerable to both the Refined Power Analysis (RPA) and the Zero-value Point Attack (ZPA). ES and RIP are resistant to both RPA and ZPA. There are two algorithms of RIP. Algorithm 5

is the left-to-right RIP [19], called MMM-RIP in this paper, although it is called BRIP in the original paper. Algorithm 6 is the right-to-left RIP algorithm [10], called IIT-RIP, although it is called ADA and RIP in the original paper.

Algorithm 5. MMM-RIP [19,20]

Input: P and $d = \sum_{i=0}^{\ell-1} d_i 2^i (d > 1)$
Output: dP

1: $R \leftarrow randompoint()$
2: $R[0] \leftarrow R$; $R[1] \leftarrow -R[0]$
3: $R[2] \leftarrow P - R[0]$
4: **for** $i = \ell - 1$ to 0 **do**
5: $\quad R[0] \leftarrow 2R[0] + R[1 + d_i]$
6: **end for**
7: $R[0] \leftarrow R[0] + R[1]$
8: **return** $R[0]$

Algorithm 6. IIT-RIP [10]

Input: P and $d = \sum_{i=0}^{\ell-1} d_i 2^i (d > 1)$
Output: dP

1: $R = randompoint()$
2: $R[0] \leftarrow R$; $R[2] \leftarrow P$; $R[3] \leftarrow R[0]$
3: **for** $i = 0$ up to $\ell - 1$ **do**
4: $\quad R[1] \leftarrow R[0] + R[2]$
5: $\quad R[2] \leftarrow 2R[2]$; $R[0] \leftarrow R[d_i]$
6: **end for**
7: $R[0] \leftarrow R[0] - R[3]$
8: **return** $R[0]$

Let us investigate differences between Algorithms 5 and 6 from the viewpoint of security, computational cost and memory amount. As for security, Algorithm 6 is vulnerable to SEA: an error in step 4 will be a safe error when $d_i = 0$. It, however, is secure against SPA, DPA, RPA, ZPA, and DblA described in Section 2.2. On the other hand, Algorithm 5 is secure against SEA, SPA, DPA, RPA, ZPA, and DblA. As for the computational cost, we assume the Jacobian coordinate. Algorithm 6 can use the iterated doubling formulae presented in Section 2.1 in the same way as other right-to-left algorithms, which can reduce the computational cost of each $2^i P$. The computational cost for ℓ doublings, $2\ell M + 8\ell S$, is reduced to $(3\ell - 1)M + (5\ell + 3)S$ in total. However, it needs to keep an intermediate point (X_i, Y_i', Z_i) for the next computation, as well as outputs (X_i, Y_i, Z_i) in each round $1 \leqslant i \leqslant \ell - 1$. On the other hand, Algorithm 5 cannot use the iterated doubling formulae but the doubling add algorithm [17], which can compute directly both double and add with a cost of $14M + 9S$. The doubling add algorithm reduces the computational cost of ordinary computations of double and add $(13M + 13S)$ by $4S - M$, but increases the memory amount by 2 more registers. So, there is no difference in the computational cost between Algorithms 5 and 6, under the ordinary addition formulae without increasing memory amount. As for the memory amount, Algorithm 5 needs 3-point registers, while Algorithm 6 needs one more register to keep R until Step 8, and thus, it needs 4-point registers in total.

In summary, Algorithm 5 can execute with a smaller memory amount and is secure against SEA, SPA, DPA, RPA, ZPA, and DblA, while Algorithm 6 needs more registers and is not secure against SEA. Section 3 will present an elegant technique to improve Algorithm 6.

2.5 Highly Regular Left-to-Right Scalar Multiplication

Montgomery Ladder
The Montgomery Ladder [23] is described in Algorithm 7. In order to reduce the computational cost, the co-Z coordinate can be applied in Algorithm 8, where $(R, P) \leftarrow \mathrm{DBLU}(P)$ in Step 1 is defined as: $R \leftarrow 2P = (X_2 : Y_2 : Z_2)$ and $P \leftarrow (\lambda^2 X_1 : \lambda^3 Y_1 : \lambda)$ with $\lambda = Z_2$ [9]; and $\mathrm{ZACAU}(R[d_i], R[1-d_i])$, that is a combination of ZADDC and ZADDU, and can work in $9M + 7S$ with an extra register of $C = (X_1 - X_2)^2$ in addition to two points $R[d_i] = (X_1, Y_1, Z)$ and $R[1-d_i] = (X_2, Y_2, Z)$.

The Montgomery Ladder can work regularly in the left-and-right way with only 2 registers of points. However, we notice that an operation on $R[1]$ of for-loop for the last round, i.e. $i = 0$, becomes a dummy operation because both $R[0]$ and $R[1]$ are executed in the last round, but only $R[0]$ is returned at Step 6. This is why the Montgomery Ladder is vulnerable to SEA, and leaks LSB. One possible countermeasure is to check the coherency between $R[0]$ and $R[1]$ to detect some fault attack. However, it is rather exaggerated and spoils the simplicity of the Montgomery Ladder. We will present a simple method to enhance the LSB security in Section 4.1.

Algorithm 7. Montgomery Ladder [23]

Input: P, $d = \sum_{i=0}^{\ell-1} d_i 2^i (d > 0)$
Output: dP
1: $R[0] \leftarrow P; R[1] \leftarrow 2P$
2: **for** $i = \ell - 2$ to 0 **do**
3: $\quad R[1-d_i] \leftarrow R[0] + R[1]$
4: $\quad R[d_i] \leftarrow 2R[d_i]$
5: **end for**
6: **return** $R[0]$

Algorithm 8. Montgomery Ladder (co-Z) [8]

Input: P, $d = \sum_{i=0}^{\ell-1} d_i 2^i (d > 0)$
Output: dP
1: $(R[1], R[0]) \leftarrow \mathrm{DBLU}(P)$
2: **for** $i = \ell - 2$ to 0 **do**
3: $\quad (R[d_i], R[1-d_i]) \leftarrow \mathrm{ZACAU}(R[d_i], R[1-d_i])$
4: **end for**
5: **return** $R[0]$

Signed-Digit Algorithm
Signed-digit algorithms, both left-to-right and right-to-left, are proposed in [9] by using the fact that any w-bit binary expansion $00\cdots01$ is equal to a w-bit signed-digit expansion $1\bar{1}\cdots\bar{1}\bar{1}$. Here $\bar{1}$ means -1. In fact, any odd binary-expansion number $d = \sum_{i=0}^{\ell-1} d_i 2^i$ $(d_{\ell-1}, d_0 = 1)$ can be written in a non-zero form, called ZSD expansion, as $d = \sum_{i=0}^{\ell-1} \delta_i 2^i$, where $\delta_i = (-1)^{1+d_{i+1}} (0 \leqslant i \leqslant \ell - 2)$ and $\delta_{\ell-1} = 1$. Here, we focus on only the left-to-right algorithm, which is presented in Algorithm 9. Remarkably, the ZSD expansion can be obtained on the fly, as we will see in Algorithm 9. In order to reduce the computational cost, the co-Z coordinate can be applied in Algorithm 10, where $(R, P) \leftarrow \mathrm{TPLU}(P)$ in Step 1 is defined as: $R \leftarrow 3P = (X_3 : Y_3 : Z_3)$ and $P \leftarrow (\lambda^2 X_1 : \lambda^3 Y_1 : \lambda)$ with $\lambda = Z_3$ [9]; and $\mathrm{ZDAU}(R[0], (-1)^{1+d_i} R[1])$ is a direct computation of ZADDU and ZADDC, which can work in $9M + 7S$.

Note that, in the same way as the Montgomery Ladder, the signed-digit algorithm can work regularly in the left-and-right way with only 2 registers of points

and is secure against SEA. However, it works for only odd d. Section 4.2 will present an elegant method to let the signed-digit algorithm work for any d.

Algorithm 9. Signed-digit Alg. [9]

Input: P, $d = \sum_{i=0}^{\ell-1} d_i 2^i$ $(d_0 = 1)$
Output: dP

1: $R[0] \leftarrow P; R[1] \leftarrow P$
2: **for** $i = \ell - 1$ to 1 **do**
3: $\quad R[0] \leftarrow 2R[0] + (-1)^{1+d_i} R[1]$
4: **end for**
5: **return** $R[0]$

Algorithm 10. Signed-digit Alg. (co-Z) [9]

Input: P, $d = \sum_{i=0}^{\ell-1} d_i 2^i$ $(d_0 = 1, d \geq 3)$
Output: dP

1: $(R[0], R[1]) \leftarrow \mathrm{TPLU}(P)$
2: **for** $i = \ell - 2$ to 1 **do**
3: $\quad (R[0], R[1]) \leftarrow \mathrm{ZDAU}(R[0], (-1)^{1+d_i} R[1])$
4: $\quad R[1] \leftarrow (-1)^{1+d_i} R[1]$
5: **end for**
6: **return** $R[0]$

3 Enhance the LSB Security of Right-to-Left Algorithms

First, we improve Algorithm 4 to Algorithm 11, which works for any scalar d and is resistant to SEA and SPA, while maintaining the performance of Algorithm 4. Next, we improve Algorithm 6 from the point of view of security and memory amount, which is presented in Algorithm 12.

3.1 Subtract-Doubling Algorithm

Let us explain Algorithm 1 in detail. To enhance the LSB security, Algorithm 11 transforms an ℓ-bit binary-expansion $d = \sum_{i=0}^{\ell-1} d_i 2^i$ into an ℓ-bit $\{\bar{1}, \bar{2}\}$-expansion with MSB equal to $3 = d_{\ell-1} + 2$ by changing d to $d + 2$ *on the fly*, and, then computes $(d+2)P - 2P$ by regarding 2 as $2\bar{2}\bar{2}\ldots\bar{2}\bar{2}$ for $\bar{2} = -2$. The idea is an extension of Joye's algorithm that regards -1 as $\bar{1}11\ldots11$ for $\bar{1} = -1$. Thus, Algorithm 11 naturally changes d_0 to "-2" and "-1", and repeats subtraction and doubling. This is why Algorithm 11 is called the *subtract-doubling* algorithm. To further enhance the security of d_1 (next to LSB), Algorithm 11 treats d_1 separately from a for-loop. Theorem 1 proves the correctness of Algorithm 11 and also shows that the final subtraction of $2P$ is executed naturally by setting $R[2] = d_0 P - 2P$ in Step 1.

Theorem 1. *Algorithm 11 computes dP correctly.*

Proof: The initial values of $(R[0], R[1], R[2])$ in Step 1 are: $(R[0], R[1], R[2]) = (2P, -P, d_0 P - 2P)$. Thus, the final subtraction of $2P$, that is, $(d+2)P - 2P$ is implemented in the beginning. From the simple discussion, the values of $R[0], R[1], R[2]$ right after the for-loop satisfies the equations: $R[0] = 2^{\ell-1} P$, and $2R[1] + R[2] = \sum_{i=0}^{\ell-2} (d_i - 2) 2^i P - 2P$. Thus, dP is correctly returned as follow:

$$R[0] + 2(R[0] + R[1]) + R[2] = (d_{\ell-1} + 2) 2^{\ell-1} P + \sum_{i=0}^{\ell-2} (d_i - 2) 2^i P - 2P = dP. \blacksquare$$

As for security, Algorithm 11 works in a highly-regular right-to-left way, and executes the same operations in each iteration of for-loop without any dummy operation. This is why Algorithm 11 is resistant to SPA, DblA, and SEA.

3.2 Modified IIT-RIP

Algorithm 6 uses a register of $R[3]$ to store a random initial point R which is used only in Steps 2 and 8. Our algorithm 12 can execute without this register (See in Steps 2 and 10 of algorithm 12). Let us explain in detail. Algorithm 12 embeds a random initial point $2R$ into Algorithm 4 elegantly, where Algorithm 4 computes $(d-1)P+P$ by setting $(R[0], R[1], R[2]) = (P, d_0P, P)$ in the beginning, repeating doubling and addition, and finally returning $R[1] + 2R[2]$, which includes the final addition to P in $(d-1)P + P$ implicitly. We apply this idea to compute $((d-1)P+2R)+(P-2R)$ as follows: set $(R[0], R[1], R[2]) = (P, d_0P+2R, P-R)$ in the beginning, repeat doubling and addition, and finally return $R[1] + 2R[2]$, which includes the addition to $P - 2R$ in $((d-1)P + 2R) + (P - 2R)$ implicitly. Furthermore, the register $R[0]$ is well re-used in the initialization, which avoids increasing one more register. By using these elegant ideas, no extra register is needed to store the random initial point. Note that the conjugate addition in Section 2 can be applied to Step 3, which can reduce the computational cost. The correctness of Algorithm 12 will be shown in Theorem 2.

As for security, Algorithm 12 works in a highly-regular right-to-left way, executes the same operations in each iteration of for-loop without any dummy operation, and applies the RIP countermeasure at the same time. This is why Algorithm 12 is resistant to SPA, DblA, SEA, RPA, ZPA and DPA.

Algorithm 11. Subtract-Doubling Alg.

Input: P and $d = \sum_{i=0}^{\ell-1} d_i 2^i (d > 3)$
Output: dP

1: $R[0] \leftarrow 2P$; $R[1] \leftarrow -P$
2: $R[2] \leftarrow (-1)^{d_0+1} R[d_0]$
3: $R[1 + d_1 \& d_0] \leftarrow (-1)^{d_1 \& \overline{d_0}} R[1] - R[0]$
4: $R[0] \leftarrow 2R[0]$
5: $R[2] \leftarrow (-1)^{d_1 \& \overline{d_0}} R[1 + (-1)^{d_1 \& \overline{d_0}}]$
6: **for** $i = 2$ to $\ell - 2$ **do**
7: $\quad R[d_i + 1] \leftarrow R[d_i + 1] - R[0]$
8: $\quad R[0] \leftarrow 2R[0]$
9: **end for**
10: $R[0] \leftarrow R[0] + 2(R[0] + R[1]) + R[2]$
11: **return** $R[0]$

($\overline{d_0}$ means the complement of d_0.)

Algorithm 12. Modified IIT-RIP

Input: P and $d = \sum_{i=0}^{\ell-1} d_i 2^i (d > 1)$
Output: dP

1: $R = RandomPoint()$
2: $R[0] \leftarrow R$; $R[2] \leftarrow P$
3: $R[1] \leftarrow R[2] + R[0]$
4: $R[2] \leftarrow R[2] - R[0]$
5: $R[1] \leftarrow R[1] + (-1)^{1+d_0} R[1 - (-1)^{1+d_0}]$
6: $R[0] \leftarrow R[0] + R[2]$
7: **for** $i = 1$ to $\ell - 2$ **do**
8: $\quad R[0] \leftarrow 2R[0]$
9: $\quad R[1 + d_i] \leftarrow R[1 + d_i] + R[0]$
10: **end for**
11: $R[0] \leftarrow R[1] + 2R[2]$
12: **return** $R[0]$

Theorem 2. *Algorithm 12 computes dP correctly.*

Proof:
Values of $(R[0], R[1], R[2])$ before Step 6 are[1]: $(R[0], R[1], R[2]) = (P, d_0P + 2R, P - R)$. From the simple discussion, the values of $(R[1], R[2])$ right after the for-loop are: $R[2] = P - R + \sum_{i=1}^{\ell-2} d_i 2^i P$ and $R[1] = d_0P + 2R + \sum_{i=1}^{\ell-2} \overline{d_i} 2^i P$, where $\overline{d_i}$ means the complement of d_i. Thus, dP is correctly returned as follows:

$$2R[2]+R[1] = \sum_{i=0}^{\ell-2}(d_i+1)2^iP+P = (d-1)P+P = dP. \quad \blacksquare$$

4 Enhance the LSB Security of Left-to-Right Algorithms

First, we improve the Montgomery Ladder (Algorithms 7 and 8) such that it is resistant to SEA. It is called the Modified Montgomery Ladder (Algorithms 13 and 14). We also improve the signed-digit algorithm such that it is available for any scalar. It is called the extended signed-digit algorithm (Algorithms 15 and 16).

4.1 Modified Montgomery Ladder

An operation on $R[1]$ in $i = 0$ of the for-loop in Algorithm 7 is a dummy operation, mentioned in Section 2.5. Let us explain steps in $i = 0$ of the for-loop and Step 6 of Algorithm 7 in detail. A returned value $R[0]$ in Step 6 can be represented by using $(r_0, r_1) = (R[0], R[1])$ in $i = 1$ of the for-loop:

$$R[0] = \begin{cases} 2r_0 & \text{if } d_0 = 0, \\ r_0 + r_1 & \text{if } d_0 = 1. \end{cases}$$

We modify steps in $i = 0$ of the for-loop to use both registers by changing to: compute $R[d_0] = 2r_0 + r_1$ and $R[d_0] = R[d_0] - R[1 - d_0]$, and return $R[d_0]$. Then, the returned value is the same as Algorithm 7, seen below:

$$R[d_0] = \begin{cases} 2r_0 & \text{if } d_0 = 0, \\ r_0 + r_1 & \text{if } d_0 = 1. \end{cases}$$

Our Algorithm 13 actually uses both two registers until Step 8. Thus, our algorithm is resistant to SEA. Furthermore, Algorithm 13 executes the same operations in each iteration of the for-loop, and, thus is resistant to SPA in the same way as Algorithm 7. As for the memory amount, it uses registers of 2 points, which is the same as in the case of Algorithm 7.

As for the computational cost, Algorithm 13 has the same for-loop as Algorithm 7. The difference exists only in steps for $i = 0$, where it is in the for-loop in Algorithm 7, while it is out of the for-loop in Algorithm 13. For a further

[1] To avoid an addition to $\mathcal{O}$, Algorithm 12 does not compute $R[1] = d_0P + 2R$ directly but sets $R[1] = P + 2R$ or $2R$ for an odd or even d, respectively.

reduction of the computational cost, the co-Z coordinate can be applied in the same way as Algorithm 7, which is described in Algorithm 14. The difference is that ZACAU in $i = 0$ of the for-loop in Algorithm 8 is changed to ZDAU and ZADDU in Steps 5 and 6 in Algorithm 14.

Algorithm 13. Modified Montgomery Ladder

Input: P, $d = \sum_{i=0}^{\ell-1} d_i 2^i (d > 0)$
Output: dP

1: $R[0] \leftarrow P; R[1] \leftarrow 2P$
2: **for** $i = \ell - 2$ to 1 **do**
3: $\quad R[1 - d_i] \leftarrow R[0] + R[1]$
4: $\quad R[d_i] \leftarrow 2R[d_i]$
5: **end for**
6: $R[d_0] \leftarrow 2R[0] + R[1]$
7: $R[d_0] \leftarrow R[d_0] - R[1 - d_0]$
8: **return** $R[d_0]$

Algorithm 14. Modified Montgomery Ladder (co-Z)

Input: P, $d = \sum_{i=0}^{\ell-1} d_i 2^i (d > 0)$
Output: dP

1: $(R[1], R[0]) \leftarrow \mathrm{DBLU}(P)$
2: **for** $i = \ell - 2$ to 1 **do**
3: $\quad (R[d_i], R[1 - d_i]) \leftarrow \mathrm{ZACAU}(R[d_i], R[1 - d_i])$
4: **end for**
5: $(R[d_0], R[1 - d_0]) \leftarrow \mathrm{ZDAU}(R[0], R[1])$
6: $(R[d_0], R[1 - d_0]) \leftarrow \mathrm{ZADDU}(-R[1 - d_0], R[d_0])$
7: **return** $R[d_0]$

4.2 Extended Signed-Digit Algorithm

Algorithm 9 is only available for an odd scalar, as mentioned in Section 2.5, while Algorithm 7 can work for any d although it reveals LSB. Both algorithms 7 and 9 have an important similarity such that both work with two registers of points. We will change the last steps of Algorithm 9 in the same way as Algorithm 7 to work for any d.

Let us compare these two algorithms. Let $(R[0]_i, R[1]_i)$ be values of $(R[0], R[1])$ at the end of the for-loop for $1 < i < \ell - 2$. Then, by using the feature that $R[1] - R[0] = P$ holds in Algorithm 7, the next equations hold.

$$R[1 - d_i]_i = R[0]_{i+1} + R[1]_{i+1} = 2R[0]_{i+1} + P = 2R[1]_{i+1} - P \text{ (Step 3, Alg. 7)}, \quad (1)$$

$$R[d_i]_i = 2R[d_i]_{i+1} = R[1 - d_i]_i + (-1)^{1+d_i} P \text{(Step 4, Alg. 7)}, \quad (2)$$

where Eq. (1) is represented by using d_{i+1} as follows:

$$R[1 - d_i]_i = 2R[1 - d_{i+1}]_{i+1} + (-1)^{1+d_{i+1}} P \text{ (Step 3, Algorithm 7)}. \quad (3)$$

This is easily derived from: $R[1 - d_i]_i = 2R[0]_{i+1} + P = 2R[1 - d_{i+1}]_{i+1} + (-1)^{1+d_{i+1}} P$ if $d_{i+1} = 1$, and $R[1 - d_i]_i = 2R[1]_{i+1} - P = 2R[1 - d_{i+1}]_{i+1} + (-1)^{1+d_{i+1}} P$ if $d_{i+1} = 0$. On the other hand,

$$R[0]_i = 2R[0]_{i+1} + (-1)^{1+d_i} P \qquad \text{(Step 3, Algorithm 9)}. \quad (4)$$

Then, the following theorem holds.

Theorem 3. *Let $(R[0]_i, R[1]_i)$ be values of $(R[0], R[1])$ at the end of the for-loop for $1 < i < \ell - 2$ in each Algorithms 7 and 9. Then, for the same scalar d and $\ell - 2 > i > 1$,*

$$R[1-d_i]_i(\textit{Alg. 7}) = R[0]_{i+1}(\textit{Alg. 9}).$$

Proof: The statement follows by induction on i. When $i = \ell - 2$,

$$R[1-d_{\ell-2}]_{\ell-2} = 2R[1-d_{\ell-1}]_{\ell-1} + (-1)^{1+d_{\ell-1}}P = 2R[0] + P = 3P(\text{Alg. 7}),$$
$$R[0]_{\ell-1} = 2R[0]_{\ell} + (-1)^{1+d_{\ell-1}}P = 2P + P = 3P(\text{Alg. 9}),$$

follows. Assume that $R[1-d_i]_i(\text{Alg. 7}) = R[0]_{i+1}(\text{Alg. 9})$ holds for i. Then,

$$R[1-d_{i-1}]_{i-1} = 2R[1-d_i]_i + (-1)^{1+d_i}P(\text{Alg. 7}),$$
$$R[0]_i = 2R[0]_{i+1} + (-1)^{1+d_i}P(\text{Alg. 9}),$$

holds for $i-1$, and, thus statements follows. ∎

A simple example between Algorithms 7 and 9 in Table 3 makes Theorem 3 more clear, where underlined points show the relation.

Table 3. Transit of $(R[0], R[1])$ in Alg. 7 and 9 ($d = 45 = (101101)_2$)

Algorithm	Initial Value		$i=5$	4	3	2	1	0	Return
Alg. 7	$R[0]$	P	-	$2P$	$\underline{5P}$	$\underline{11P}$	$22P$	$45P$	$45P$
	$R[1]$	$2P$	-	$\underline{3P}$	$6P$	$12P$	$\underline{23P}$	$46P$	
Alg. 9	$R[0]$	P	$\underline{3P}$	$\underline{5P}$	$\underline{11P}$	$\underline{23P}$	$45P$	-	$45P$
	$R[1]$	P	P	P	P	P	P	-	

Our algorithm is presented in Algorithm 15: for-loop is the same as that of Algorithm 9; and Step 6 changes $(R[0], R[1])$ to those in $i = 1$ of for-loop of Algorithm 7; and Steps 7 and 8 are the same as Steps 6 and 7 in Algorithm 13 in order to be secure against SEA on LSB. Tables 4, 5, 6, and 7 describe all patterns in Algorithm 15.

Algorithm 15. Extended Signed-digit Alg.

Input: P, $d = \sum_{i=0}^{\ell-1} d_i 2^i$ with $d > 1$
Output: dP

1: $R[1] \leftarrow P; R[0] \leftarrow P$
2: **for** $i = \ell - 1$ to 2 **do**
3: $\quad R[0] \leftarrow 2R[0] + (-1)^{1+d_i} R[1]$
4: **end for**
▷ Finalization
5: $b = \overline{d_1 \oplus d_0}$
6: $R[1] = R[0] + (-1)^{1+d_1} R[1]$
7: $R[b] = 2R[1-d_1] + R[d_1]$
8: $R[b] = R[b] - R[1-b]$
9: **return** $R[b]$

Algorithm 16. Extended Signed-digit Alg. (co-Z)

Input: P, $d = \sum_{i=0}^{\ell-1} d_i 2^i$ with $d_0 = 1$ and $d \geq 3$
Output: dP

1: $(R[0], R[1]) \leftarrow \text{TPLU}(P)$
2: **for** $i = \ell - 2$ to 2 **do**
3: $\quad (R[0], R[1]) \leftarrow \text{ZDAU}(R[0], (-1)^{1+d_i} R[1])$
4: $\quad R[1] \leftarrow (-1)^{1+d_i} R[1]$
5: **end for**
▷ Finalization
6: $b = \overline{d_1 \oplus d_0}$
7: $(R[1], R[0]) \leftarrow \text{ZADDU}(R[0], (-1)^{1+d_1} R[1])$
8: $(R[b], R[1-b]) \leftarrow \text{ZDAU}(R[1-d_1], R[d_1])$
9: $(R[b], R[1-b]) \leftarrow \text{ZADDU}(R[b], -R[1-b])$
10: **return** $R[b]$

Table 4. Transit of $(R[0], R[1])$ in Alg. 15 $(d = 44 = (101100)_2)$

Initial Value		For-loop $(5 \geqslant i \geqslant 2)$				Finalization			Return
		5	4	3	2	Step 6	Step 7	Step 8	
$R[0]$	P	$3P$	$5P$	$11P$	$23P$	$23P$	$23P$	$23P$	
$R[1]$	P	P	P	P	P	$22P$	$67P$	$44P$	$44P$

Table 5. Transit of $(R[0], R[1])$ in Alg. 15 $(d = 45 = (101101)_2)$

Initial Value		For-loop $(5 \geqslant i \geqslant 2)$				Finalization			Return
		5	4	3	2	Step 6	Step 7	Step 8	
$R[0]$	P	$3P$	$5P$	$11P$	$23P$	$23P$	$67P$	$45P$	$45P$
$R[1]$	P	P	P	P	P	$22P$	$22P$	$22P$	

Table 6. Transit of $(R[0], R[1])$ in Alg. 15 $(d = 46 = (101110)_2)$

Initial Value		For-loop $(5 \geqslant i \geqslant 2)$				Finalization			Return
		5	4	3	2	Step 6	Step 7	Step 8	
$R[0]$	P	$3P$	$5P$	$11P$	$23P$	$23P$	$70P$	$46P$	$46P$
$R[1]$	P	P	P	P	P	$24P$	$24P$	$24P$	

Table 7. Transit of $(R[0], R[1])$ in Alg. 15 $(d = 47 = (101111)_2)$

Initial Value		For-loop $(5 \geqslant i \geqslant 2)$				Finalization			Return
		5	4	3	2	Step 6	Step 7	Step 8	
$R[0]$	P	$3P$	$5P$	$11P$	$23P$	$23P$	$23P$	$23P$	
$R[1]$	P	P	P	P	P	$24P$	$70P$	$47P$	$47P$

Thus, our Algorithm 15 can work for any d. As for security, Algorithm 15 executes the same operations in each iteration of the for-loop, and is thus secure against SPA. Algorithm 15 is also secure against SEA without revealing LSB using the same idea as in Algorithm 13. As for the memory amount, it uses registers of 2 points, which is the same as that of Algorithm 9. As for the computational cost, Algorithm 15 has the same for-loop as Algorithm 9. The differences exist only in steps for $i = 1$, where it is in the for-loop in Algorithm 9, while it is out of the for-loop in Algorithm 15. For a further reduction of the computational cost, the co-Z coordinate can be applied in the same way as Algorithm 9, which is described in Algorithm 16. The differences are that ZDAU in $i = 1$ of the for-loop in Algorithm 16 is changed to ZADDU, ZDAU and ZADDU in Steps 7 to 9 in Algorithm 16.

5 Comparison

Table 8 shows comparisons of security, which omit the security with the co-Z coordinate because they are the same as that without the co-Z coordinate.

Table 8. Comparisons of security

Alg.	SEA	SPA	DPA	RPA	ZPA	DblA
Right-to-left algorithms:						
Alg. 4 [12]	S	LW	W	W	W	S
Alg. 11	S	S	W	W	W	S
Alg. 6 [10]	W	S	S	S	S	S
Alg. 12	S	S	S	S	S	S
Left-to-right algorithms:						
Alg. 7 [23]	LW	S	W	W	W	S
Alg. 13	S	S	W	W	W	S
Alg. 9 [9]	S	S	W	W	W	S
Alg. 15	S	S	W	W	W	S

S: Secure, W: Weak, LW: LSB Weak

Table 9. Comparisons of computational cost and memory amount

Alg.	Computational cost(per bit)	Memory amount(# regs.)	Memory amount(# points.)	Work for $\forall d$
Right-to-left algorithms:				
Alg. 4 [12]	$13M + 13S$	14	$R[0], R[1], R[2]$	odd d
Alg. 11	$13M + 13S$	14	$R[0], R[1], R[2]$	$\forall d$
Alg. 6 [10]	$13M + 13S$	17	$R[0], R[1], R[2], R[3]$	$\forall d$
Alg. 12	$13M + 13S$	14	$R[0], R[1], R[2]$	$\forall d$
Left-to-right algorithms:				
Alg. 7 [23]	$13M + 13S$	11	$R[0], R[1]$	$\forall d$
Alg. 8 (co-Z) [8]	$9M + 7S$	8	$R[0], R[1]$	$\forall d$
Alg. 13	$13M + 13S$	11	$R[0], R[1]$	$\forall d$
Alg. 14 (co-Z)	$9M + 7S$	8	$R[0], R[1]$	$\forall d$
Alg. 9 [9]	$13M + 13S$	11	$R[0], R[1]$	odd d
Alg. 10 (co-Z) [9]	$9M + 7S$	8	$R[0], R[1]$	odd d
Alg. 15	$13M + 13S$	11	$R[0], R[1]$	$\forall d$
Alg. 16 (co-Z)	$9M + 7S$	8	$R[0], R[1]$	$\forall d$

Table 9 shows comparisons of computational cost and memory amount. Let ℓ represent the length of a scalar d. The computational cost assumes Jacobian coordinates described in Section 2.1. The memory amount is described in two ways: the first indicates the number of points necessary to implement each algorithm; and the other indicates the precise number of registers necessary to implement, which includes registers of points.

Let us compare the right-to-left algorithms: our Algorithms 11 (resp. 12) with Algorithms 4 (resp. 6) from the viewpoint of security, computational cost, and memory amount. Compared with Algorithm 4, our Algorithm 11 enhances the LSB security for SPA, while maintaining the performances such as computational cost per bit and memory amount. Algorithm 4 needs a special treatment in order to be secure against SPA for an arbitrary scalar. On the other hand, Algorithm 12 also enhances security for SEA, although Algorithm 6 is vulnerable to SEA.

In addition, Algorithm 12 can work with 3 points of $R[0], R[1], R[2]$, and thus, it can reduce memory amount. To reduce the computational cost, all right-to-left algorithms, Algorithms 6, 4, 11, and 12, can use iterated doubling formulae, although this technique increases memory amount.

Let us compare left-to-right algorithms from the viewpoint of security, computational cost, and memory amount. Compared with Algorithm 7, Algorithm 13 enhances the LSB security for SEA, while maintaining the performances such as computational cost per bit and memory amount. Algorithm 7 leaks LSB by SEA. On the other hand, Algorithm 15 enhances the availability of Algorithm 9 that works only for odd d. Algorithm 15 is secure against SEA, SPA, and DblA in the same way as Algorithm 9, and also keeps the performances such as computational cost per bit and memory amount.

6 Conclusion

We have revisited regular right-to-left and left-to-right algorithms, IIT-RIP (Algorithm 6), Joye's regular right-to-left algorithm (Algorithm 4), the Montgomery Ladder (Algorithm 7), and the signed-digit algorithm (Algorithm 9), and we modified them to Algorithm 12, Algorithm 11, Algorithm 13, and Algorithm 15, respectively. Those modified algorithms enhance each LSB security using elegant techniques while maintaining performances such as memory amount and computational cost per bit.

References

1. Akishita, T., Takagi, T.: Zero-Value Point Attacks on Elliptic Curve Cryptosystem. In: Boyd, C., Mao, W. (eds.) ISC 2003. LNCS, vol. 2851, pp. 218–233. Springer, Heidelberg (2003)
2. Baek, Y.-J.: Regular 2^w -ary right-to-left exponentiation algorithm with very efficient DPA and FA countermeasures. International Journal of Information Security 9, 363–370 (2010)
3. Ciet, M., Joye, M.: (Virtually) Free Randomization Techniques for Elliptic Curve Cryptography. In: Qing, S., Gollmann, D., Zhou, J. (eds.) ICICS 2003. LNCS, vol. 2836, pp. 348–359. Springer, Heidelberg (2003)
4. Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., Vercauteren, F. (eds.): Handbook of Elliptic and Hyperelliptic Curve Cryptography (Discrete Mathematics and Its Applications). Chapman and Hall/CRC (July 2005)
5. Cohen, H., Miyaji, A., Ono, T.: Efficient Elliptic Curve Exponentiation Using Mixed Coordinates. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 51–65. Springer, Heidelberg (1998)
6. Coron, J.-S.: Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999)
7. Goubin, L.: A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 199–210. Springer, Heidelberg (2002)

8. Goundar, R.R., Joye, M., Miyaji, A.: Co-Z Addition Formulæ and Binary Ladders on Elliptic Curves (Extended Abstract). In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 65–79. Springer, Heidelberg (2010)
9. Goundar, R.R., Joye, M., Miyaji, A., Rivain, M., Venelli, A.: Scalar multiplication on Weierstraß elliptic curves from co-Z arithmetic. Journal of Cryptographic Engineering 1(2), 161–176 (2011)
10. Itoh, K., Izu, T., Takenaka, M.: Efficient Countermeasures Against Power Analysis for Elliptic Curve Cryptosystems. In: Quisquater, J.-J., Paradinas, P., Deswarte, Y., El Kalam, A.A. (eds.) Smart Card Research and Advanced Applications VI. IFIP, vol. 153, pp. 99–113. Springer, Boston (2004)
11. Joye, M.: Highly Regular Right-to-Left Algorithms for Scalar Multiplication. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 135–147. Springer, Heidelberg (2007)
12. Joye, M.: Highly Regular m-Ary Powering Ladders. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 350–363. Springer, Heidelberg (2009)
13. Joye, M., Karroumi, M.: Memory-Efficient Fault Countermeasures. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 84–101. Springer, Heidelberg (2011)
14. Joye, M., Tymen, C.: Protections against Differential Analysis for Elliptic Curve Cryptography. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 377–390. Springer, Heidelberg (2001)
15. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
16. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
17. Longa, P.: ECC point arithmetic formulae (EPAF), `http://patricklonga.bravehost.com/jacobian.html`
18. Longa, P., Gebotys, C.: Novel Precomputation Schemes for Elliptic Curve Cryptosystems. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 71–88. Springer, Heidelberg (2009)
19. Mamiya, H., Miyaji, A., Morimoto, H.: Efficient Countermeasures against RPA, DPA, and SPA. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 343–356. Springer, Heidelberg (2004)
20. Mamiya, H., Miyaji, A., Morimoto, H.: Secure elliptic curve exponentiation against RPA, ZRA, DPA, and SPA. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 89-A(8), 2207–2215 (2006)
21. Meloni, N.: New Point Addition Formulae for ECC Applications. In: Carlet, C., Sunar, B. (eds.) WAIFI 2007. LNCS, vol. 4547, pp. 189–201. Springer, Heidelberg (2007)
22. Miyaji, A.: Generalized scalar multiplication secure against SPA, DPA, and RPA. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 91-A(10), 2833–2842 (2008)
23. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation 48(177), 243–264 (1987)
24. Okeya, K., Sakurai, K.: On Insecurity of the Side Channel Attack Countermeasure Using Addition-Subtraction Chains under Distinguishability between Addition and Doubling. In: Batten, L.M., Seberry, J. (eds.) ACISP 2002. LNCS, vol. 2384, pp. 420–435. Springer, Heidelberg (2002)
25. Yen, S.-M., Joye, M.: Checking before output may not be enough against fault-based cryptanalysis. IEEE Transactions on Computers 49(9), 967–970 (2000)

Improvement in Non-linearity of Carlet-Feng Infinite Class of Boolean Functions

Mansoor Ahmed Khan[1] and Ferruh Özbudak[2]

[1] Institute of Applied Mathematics, Middle East Technical University, Dumlupınar Bul. No:1, 06800, Ankara, Turkey
[2] Department of Mathematics and Institute of Applied Mathematics, Middle East Technical University, Dumlupınar Bul. No:1, 06800, Ankara, Turkey

Abstract. In this paper we present a Walsh spectrum based method derived from the genetic hill climbing algorithm to improve the non-linearity of functions belonging to Carlet-Feng infinite class of Boolean functions, without degrading other cryptographic properties they possess. We implement our modified algorithms to verify the results and also present a comparison of the resultant cryptographic properties with the original functions.

Keywords: Boolean functions, Symmetric cipher, Non-linearity, Algebraic degree, Algebraic immunity, Optimal algebraic immunity.

1 Introduction

With the introduction of Algebraic and Fast Algebraic attacks on stream ciphers with linear feedback in [1, 2] by N. Courtois and W. Meier and their improved variants [3–7], optimal algebraic immunity is amongst the important cryptographic properties for Boolean functions utilized in symmetric ciphers. In [14], C. Carlet and K. Feng proposed an infinite class of Boolean functions that possessed balanced-ness, high algebraic degree, optimal algebraic immunity, high non-linearity compared and good immunity to fast algebraic attacks. The proposed construction in [14] is based on selecting a primitive element $\alpha \in \mathbb{F}_2^n$ and selecting its consecutive powers from 1 to $(2^{n-1} - 2)$, along with "0" and "1" vector in the support set of the function. Subsequently in [19], X.Zeng, C.Carlet, J.Shan, L.Hu presented three more constructions, achieving either the same or in some cases, even higher non-linearities, while maintaining the degree and algebraic immunity as in [14]. These construction methods also utilized a primitive element $\alpha \in \mathbb{F}_2^n$ and selecting its powers in the support set of the function. However, the powers selected in this case were not consecutive, rather based on some pre-defined sets. The constructions proposed in [14] and [19] clearly demonstrated, and also proved mathematically, that selecting different powers of the primitive elements affected the non-linearity of the functions, along with their algebraic degree and algebraic immunity. We have devised two fairly simple algorithms by modifying the genetic hill climbing algorithm [20] for improvement in the non-linearity of functions constructed using the infinite class proposed

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 280–295, 2012.

in [14] for number of variables $n \geq 8$ (Note: [19] describes a different criteria for selecting the powers of the primitive element $\alpha \epsilon \mathbb{F}_2^n$ as compared to [14]). The improved functions not only possess higher non-linearity than the original functions in [14], but also maintain the high algebraic degree and optimal algebraic immunity. The improvement algorithms have been verified by constructing all Boolean functions for $8 \leq n \leq 11$ and improving their non-linearity. The rest of this paper is organized as follows. In section II, some preliminary foundations are presented related to Boolean functions along with the starting point for improvement, which is construction of functions based on the infinite class presented in [14]. The Walsh spectrum based algorithms derived from genetic hill climbing approach for improvement in non-linearity of the constructed functions are presented in section III. The various advantages obtained by the algorithms are described and demonstrated via practical results in section IV. Finally, section V summarizes computer investigation results for $8 \leq n \leq 11$ and their comparison with those in [14].

2 Preliminaries

2.1 Boolean Function Basics

Let $\mathbb{F}_2$ define the binary field. Consequently $\mathbb{F}_2^n$ can be interpreted as an n-dimensional vector space over $\mathbb{F}_2$. Then any Boolean function f on n-variables can is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. Let $\mathfrak{B}_n$ denote the set of all Boolean functions from $\mathbb{F}_2^n$ into $\mathbb{F}_2$. The Boolean function $f(x_1, ..., x_n)$ can be represented as a binary string of length 2^n with each representing the output of the function with respect to the ordered pair $(x_1, ..., x_n)$ as the input

$$f = \{f(0, 0, , ..., 0), f(0, 0, ..., 1), ..., f(1, 1, ..., 1)\} \tag{1}$$

The above representation is known as the truth table of *f*. The sequence of f denoted by Seq(f) is a $(1, -1)$ valued mapping of the truth table obtained by $Seq(f) = 1 - 2f$. The weight or Hamming Weight of a Boolean function wt(f) is the number of 1s in its truth table representation. The support of f , supp(f) is defined as

$$supp(f) = \{\forall\, x \mid f(x) = 1\} \tag{2}$$

An n-variable Boolean function is called balanced if $wt\,(f) = 2^{(n-1)}$, i.e. it has $2^{(n-1)}$ element in its support set. Now for $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$ and $\omega = (\omega_1, \omega_2, ..., \omega_n)$, define $\alpha{\cdot}\omega$ as the usual inner product $\alpha{\cdot}\omega = (\alpha_1\omega_1, \alpha_2\omega_2, ..., \alpha_n\omega_n)$. Then the Wash transform of *f*, W_f is calculated as

$$W_f(\alpha) = \sum_{\omega \epsilon \mathbb{F}_2^n} (-1)^{f(\omega)+\alpha.\omega} \tag{3}$$

It is evident that each coefficient in the Walsh spectrum has values between 2^n and -2^n. The non-linearity nl(f) of *f*, is given by

$$nl(f) = 2^{(n-1)} - \frac{1}{2} \max_{\alpha \epsilon \mathbb{F}_2^n} |W_f(\alpha)| \tag{4}$$

The annihilator of f , AN(f) is defined as the minimum degree of a Boolean function g such that $f * g = 0$, where $f * g$ is the usual product of functions $f * g = f(x).g(x)$. The algebraic immunity of f, AI(f) is the minimum degree non-zero annihilator of f

$$AI(f) = \min \{ deg(g) \mid \forall\, g \epsilon \mathfrak{B}_n \; st \; f(x).g(x) = 0, \; \forall\, x \epsilon \mathbb{F}_2^n \} \tag{5}$$

A Boolean function is k-resilient if it is balanced ($W_f(0) = 0$) and has correlation immunity = k, while correlation immunity = k implies that $W_f(\alpha) = 0$ for all α with $1 \leq wt(\alpha) \leq k$. A Boolean function to be employed in a symmetric cipher must be of high algebraic degree, high non-linearity and optimal algebraic immunity. High algebraic degree offers resistance to the Berlekamp-Massey attack [21], high non-linearity resists fast correlation attacks [22, 23], while high algebraic immunity is a necessary but not sufficient condition to counter algebraic and fast algebraic attacks [1–7]. Therefore obtaining functions with maximum algebraic degree $(n-1)$ for a balanced boolean function, optimal algebraic immunity $\lceil \frac{n}{2} \rceil$ and highest possible non-linearity is essential.

2.2 The Carlet-Feng Infinite Class of Boolean Functions

We now describe the infinite class of balanced Boolean functions with high algebraic degree, optimal algebraic immunity, good immunity to fast algebraic attacks and good nonlinearity as proposed in [14] by C. Carlet and K. Feng. Let $\alpha \epsilon \mathbb{F}_2^n$ be the primitive element of $\mathbb{F}_2^n$. Then Boolean function f from $\mathbb{F}_2^n$ to $\mathbb{F}_2$ for number of variables n whose support set is $supp(f) = \left\{0, 1, \alpha, \alpha^2, ..., \alpha^{2^{(n-1)}-2}\right\}$ has optimal algebraic immunity i.e.$\lceil \frac{n}{2} \rceil$. The algebraic degree of f is $(n-1)$ and it is balanced. Furthermore the non-linearity of f is given by

$$nl(f) = 2^{(n-1)} + \frac{2^{\frac{n}{2}} + 1}{\pi} \ln \left(\frac{\pi}{4(2^n - 1)} \right) - 1 \approx 2^{(n-1)} - \frac{2 \ln 2}{\pi} n 2^{\frac{n}{2}} \tag{6}$$

The mathematical proofs of the above are presented in [14]. It is mentioned that these functions, owing to their high algebraic degree, optimal algebraic immunity and good non-linearity, behave well against fast correlation attacks [22, 23], algebraic attacks, fast algebraic attacks [1–7] and Berlekamp-Massey attack [21]. It is also highlighted that before this construction [14], no infinite class of Boolean functions with high algebraic degree, good algebraic immunity and good non-linearity was presented. In [19], three more classes of Boolean functions were proposed by X.Zeng, C.Carlet, J.Shan, L.Hu. In this case the powers of the primitive element $\alpha \epsilon \mathbb{F}_2^n$ to be included in the support of the function f were chosen based on some pre-defined sets. While the method improved the Non-Linearity of some constructed functions, the rest had the same Non-Linearity

as in [14]. Hence these constructions did not guarantee a higher Non-Linearity than [14] for all functions. Later in [15–18] more constructions were presented to achieve Optimal Algebraic Immunity, good Non-Linearity and in some cases 1-resiliency. However, none of these attained significant increase in the non-linearity of functions as compared to [14]. Since construction in [19] employs a different criterion for selection of powers of the primitive element $\alpha \epsilon \mathbb{F}_2^n$ than [14] and achieves some functions with better non-linearity, we studied the behavior of functions in detail. While [14] presents an easy selection criteria, the one used in [19] is comparatively intricate. Subsequently, our focus was directed to using the construction in [14] as the starting point and then changing the powers of in the support set based on the affect on Walsh spectrum of the functions. This led to the development of a relatively simple algorithm derived from the hill climbing approach [20], based on the behavior of the Walsh spectrum of Boolean functions. The algorithm improves in the non-linearity of functions while preserving the algebraic degree and algebraic immunity. The non-linearity of most of the functions constructed by [14] was improved using this algorithm, while some could not be improved. Subsequently, a second algorithm was developed by modifying the first one to improve the non-linearity of the remaining functions as well. Therefore as compared to [19], which improves the non-linearity in case of some functions as compared to [14], we achieve better non-linearity for all functions.

3 Algorithms for Improving Non-linearity

3.1 The Behavior of Walsh Spectrum

Before we describe the algorithms developed, we first review the behavior of the Walsh spectrum of a Boolean function and the effects caused by changes in the truth table of a function. Recall that the Sylvester-Hadamard matrix, also known as the Walsh-Hadamard matrix, is defined as follows

$$H_0 = 1,\ H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H_n = H_{n-1} \otimes H_1 = H_{n-1} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix} \tag{7}$$

The symbol $\otimes$ denotes the usual Kronecker product. It is clear that H_n is a matrix of order 2^n. Using this matrix, the Walsh Transform of a function, also called the Walsh-Hadamard transform, can be easily calculated [24]. Given the sequence of a Boolean function $Seq(f) = (y_0, y_1, , y_{2^n-1})$, the Walsh spectrum can be computed as

$$H_f = H_n x[y_0, y_1, , y_{2^n-1}] = H_n[y_0, y_1, , y_{2^n-1}]^T$$

$$= H_n \begin{bmatrix} y_0 \\ y_1 \\ . \\ . \\ . \\ y_{2^n-1} \end{bmatrix} = \begin{bmatrix} A+B \\ A-B \end{bmatrix} \tag{8}$$

$$where\ A = \begin{bmatrix} y_0 \\ y_1 \\ . \\ . \\ . \\ y_{2^{n-1}-1} \end{bmatrix} \ and\ B = \begin{bmatrix} y_{2^{n-1}} \\ y_{2^n} \\ . \\ . \\ . \\ y_{2^n-1} \end{bmatrix}$$

Hence the Walsh spectrum can be calculated recursively by using Eqn.7. Let us demonstrate the process by an example

Example 1. Let f be a 3-variable Boolean function with the truth table $f(x_1, ..., x_3) = (0, 1, 1, 1, 1, 0, 0, 0)^T$. Using the recursion described in Eqn.7, the Walsh spectrum computation can be performed as follows

$$Seq(f) = (1, 1, 1, 1, 1, 1, 1, 1)^T$$
$$W_f = H_3 \begin{bmatrix} 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 \end{bmatrix}^T$$

$$W_f = H_3 \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} A+B \\ A-B \end{bmatrix} = \begin{bmatrix} H_2 \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix} + H_2 \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \\ H_2 \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix} - H_2 \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} H_1 \begin{bmatrix} 1 \\ -1 \end{bmatrix} + H_1 \begin{bmatrix} -1 \\ -1 \end{bmatrix} \\ H_1 \begin{bmatrix} 1 \\ -1 \end{bmatrix} - H_1 \begin{bmatrix} -1 \\ -1 \end{bmatrix} \end{bmatrix} + \begin{bmatrix} H_1 \begin{bmatrix} -1 \\ 1 \end{bmatrix} + H_1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ H_1 \begin{bmatrix} -1 \\ 1 \end{bmatrix} - H_1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} H_1 \begin{bmatrix} 1 \\ -1 \end{bmatrix} + H_1 \begin{bmatrix} -1 \\ -1 \end{bmatrix} \\ H_1 \begin{bmatrix} 1 \\ -1 \end{bmatrix} - H_1 \begin{bmatrix} -1 \\ -1 \end{bmatrix} \end{bmatrix} - \begin{bmatrix} H_1 \begin{bmatrix} -1 \\ 1 \end{bmatrix} + H_1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ H_1 \begin{bmatrix} -1 \\ 1 \end{bmatrix} - H_1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} + \begin{bmatrix} -2 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 2 \end{bmatrix} - \begin{bmatrix} -2 \\ 0 \end{bmatrix} \end{bmatrix} + \begin{bmatrix} \begin{bmatrix} 0 \\ -2 \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0 \\ -2 \end{bmatrix} - \begin{bmatrix} 2 \\ 0 \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} + \begin{bmatrix} -2 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 2 \end{bmatrix} - \begin{bmatrix} -2 \\ 0 \end{bmatrix} \end{bmatrix} - \begin{bmatrix} \begin{bmatrix} 0 \\ -2 \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0 \\ -2 \end{bmatrix} - \begin{bmatrix} 2 \\ 0 \end{bmatrix} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} -2 \\ 2 \\ 2 \\ 2 \end{bmatrix} + \begin{bmatrix} 2 \\ -2 \\ -2 \\ -2 \end{bmatrix} \\ \begin{bmatrix} -2 \\ 2 \\ 2 \\ 2 \end{bmatrix} - \begin{bmatrix} 2 \\ -2 \\ -2 \\ -2 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -4 \\ 4 \\ 4 \\ 4 \end{bmatrix}$$

Now, we observe that a single bit change in the truth table of the function f changes the Seq(f) either from 1 to 1 or vice versa. Hence the Walsh spectrum values will either be unaffected or would increase/decrease by a value of 2. This affect is independent to the number of values since for larger variables, only the number of Walsh spectrum values changed would differ, but the deviation would always by ± 2. Hence according to Eqn.4, the non-linearity of the function is increased or decreased by a value 1 with a single bit change in the truth table. Therefore, if suitable element of support set of the function is interchanged with the set of roots, that is, two suitable values in the truth table are swapped; the maximum coefficients in the Walsh spectrum is reduced by 4. Resultantly, the non-linearity of the function can be increased by a value of 2.

3.2 Algorithm 1

Let us fix some notations before presenting the algorithms. The array TT() holds the truth table of the Boolean Function constructed using [14]. ITT() holds the truth table of the improved function. Walsh() refers to the routine that calculates the Walsh spectrum of the Boolean function from its truth table representation. The algorithm is presented below

Walsh(TT())
$maxWalsh = max \mid Walsh(TT()) \mid$
copy $TT() \rightarrow ITT()$
$LastCount = 0$
ReRun:
$Count = LastCount + 1$
If $Count < (2^n - 1)$ *then* {

$For\ i = \alpha^{count}\ to\ \alpha^{2^{(n-1)}-2}\{$
$ITT(i) = 0$
$LastCount = i$
$Walsh(ITT())$

maxWalsh2 = max | Walsh(ITT()) |
If maxWalsh2 < maxWalsh then exit For
Else ITT(i) = TT(i)
Next i
}
If maxWalsh2 ≥ maxWalsh then GoTo Skipj:

For j = $2^{(n-1)}-1$ *to* 2^n-2 {
ITT(j) = 1
Walsh(ITT())
maxWalsh3 = max | Walsh(ITT()) |
If maxWalsh3 < maxWalsh2 then exit For
Else ITT(j) = TT(j)
Next j
}
If maxWalsh 3 = maxWalsh - 4 then output "Function Improved"
Else GoTo ReRun:

Skipj:
}
If maxWalsh2 ≥ maxWalsh or maxWalsh3 > maxWalsh - 4 then output "Function could not be improved" and exit

It is worth mentioning here that in case of hill climbing algorithm [20], pairs of truth table values are swapped. In contrast, Algorithm 1 changes an element of the support set to a root and determines its suitability based on change in the Walsh spectrum instead of searching all possible pair swaps. If the maximum Walsh value is reduced, it keeps this change and looks for a suitable change of a root to the support set. Once the suitable root to support swap is found, the maximum Walsh value is reduced by 4 and the non-linearity of the function improves by 2. This reduces the number of steps since the swap occurs only if the first change is suitable to increase the non-linearity. The support to root swap is handled in the loop for variable "i" and the loop for "j" handles the root to support swap to complete the improvement in non-linearity. In case improvement is not achieved once a support to root swap is selected, the algorithm is re-run by incrementing the "LastCount" variable within the for loop for "i" so that the next suitable support to root swap is selected.

3.3 Algorithm 2

Algorithm 2 is an iterative application of Algorithm 1. However, its also different to Algorithm 1 in the sense that it accepts a change in the truth table even if the maximum value of Walsh spectrum is not decreased, but the number of maximum Walsh values is decreased in the intermediate steps. The maximum Walsh value is ultimately decreased in the final iteration (the non-linearity of the function was improved in at the most 4 iterations in all cases) increasing

the non-linearity of the function by 2 in similar manner as explained in case of Algorith 1 in the preceeding paragraph. The algorithm is presented below

Walsh(TT())
maxWalsh = max $|\, Walsh(TT()) \,|$
nmaxWalsh = # maxWalsh
copy TT() $\rightarrow$ *ITT()*
LastCount = 0
Count2 = 1

ReRun:
Count = LastCount + 1
If Count $< (2^n - 1)$ *then*{

While Count2 $\leq$ *3*{

For i $= \alpha^{count}$ *to* $\alpha^{2^{(n-1)}-2}$*{*
ITT(i) = 0
LastCount = i
Walsh(ITT())
maxWalsh2 = max $|\, Walsh(ITT()) \,|$
nmaxWalsh2 = # maxWalsh2
If maxWalsh2<maxWalsh or nmaxWalsh2 < nmaxWalsh then
Count2 +=1 and exit For
Else ITT(i) = TT(i)
Next i
}
If maxWalsh2 $\geq$ *maxWalsh then GoTo Skipj:*

For j $= {}^{2^{(n-1)}-1}to^{2^n-2}$ {
ITT(j) = 1
Walsh(ITT())
maxWalsh3 = max $|\, Walsh(ITT()) \,|$
nmaxWalsh3 = # maxWalsh3
If maxWalsh3 < maxWalsh2 or nmaxWalsh3 < nmaxWalsh2 then
Count2 +=1 and exit For
Else ITT(j) = TT(j)
Next j
}
If maxWalsh 3 = maxWalsh - 4 then output "Function Improved"
Else GoTo ReRun:

Skipj:
}
If maxWalsh2 $\geq$ *maxWalsh or maxWalsh3 > maxWalsh - 4 then output "Function could not be improved" and exit*

4 Advantages Obtained

By employing Algorithms 1 and 2, the non-linearity of all the functions constructed using Carlet-Feng infinite class of Boolean functions [14] can be improved by at least 2 for number of variables $n \geq 8$. Additionally the algorithms preserve the balanced-ness, maximal algebraic degree and optimal algebraic immunity of the functions. Table 1 and 2 list some selected results for n = 8 to demonstrate the improvement in non-linearity for Algorithm 1 and 2 respectively, although all defining polynomials and primitive elements have been practically verified for $8 \leq n \leq 11$. Some results for n = 9, 10 and 11 are included in Appendix A, B and C respectively for reference.

Another significant advantage of the method is that the swapping of support set element with the root resulting in improvement in non-linearity of the functions is not unique, i.e more than one such pairs exist. Resultantly, whilst there exists only one function for a fixed defining polynomial and a fixed primitive element in the infinite class [14], more than one function with higher non-linearity can be obtained using algorithm 1 or 2 by just changing the value of variable "LastCount". Same is demonstrated by some examples presented in Table 3 for n = 8, although it has been practically verified for all values.

Table 1. Comparison of non-linearities of Carlet-Feng infinite class functions and our improved functions for n = 8 by Algorithm 1

Defining Polynomial (Integer value)	**Primitive element (Integer value)**	**Non-linearity of function in [14]**	**Elements swapped (root↔support)**	**Non-linearity of improved function**
285	2	112	$\alpha^{104} \leftrightarrow \alpha^{230}$	114
299	128	112	$\alpha^{66} \leftrightarrow \alpha^{147}$	114
301	57	112	$\alpha^{101} \leftrightarrow \alpha^{233}$	114
333	16	112	$\alpha^{87} \leftrightarrow \alpha^{241}$	114
351	4	112	$\alpha^{1} \leftrightarrow \alpha^{238}$	114
355	26	112	$\alpha^{94} \leftrightarrow \alpha^{221}$	114
357	101	112	$\alpha^{74} \leftrightarrow \alpha^{145}$	114
361	119	112	$\alpha^{21} \leftrightarrow \alpha^{200}$	114
369	47	112	$\alpha^{20} \leftrightarrow \alpha^{228}$	114
391	61	112	$\alpha^{109} \leftrightarrow \alpha^{241}$	114
397	5	112	$\alpha^{17} \leftrightarrow \alpha^{143}$	114
425	185	112	$\alpha^{105} \leftrightarrow \alpha^{181}$	114
451	220	112	$\alpha^{32} \leftrightarrow \alpha^{160}$	114
463	97	112	$\alpha^{54} \leftrightarrow \alpha^{253}$	114
487	187	112	$\alpha^{65} \leftrightarrow \alpha^{198}$	114
501	10	112	$\alpha^{46} \leftrightarrow \alpha^{137}$	114

Table 2. Comparison of non-linearities of Carlet-Feng infinite class functions and our improved functions for n = 8 by Algorithm 2

Defining Polynomial (Integer value)	**Primitive element (Integer value)**	**Non-linearity of function in [14]**	**Iterations (root↔support)**	**Non-linearity of improved function**
301	2	108	1st $\alpha^{54} \leftrightarrow \alpha^{226}$ 2nd $\alpha^{108} \leftrightarrow \alpha^{245}$ 3rd $\alpha^{75} \leftrightarrow \alpha^{251}$ 4th $\alpha^{33} \leftrightarrow \alpha^{202}$	114
357	2	112	1st $\alpha^{17} \leftrightarrow \alpha^{253}$ 2nd $\alpha^{16} \leftrightarrow \alpha^{165}$	114
425	2	112	1st $\alpha^{81} \leftrightarrow \alpha^{241}$ 2nd $\alpha^{82} \leftrightarrow \alpha^{210}$ 3rd $\alpha^{83} \leftrightarrow \alpha^{250}$	114

Table 3. Different improved functions by our method from the same parent function in [14]

Defining Polynomial (Integer value)	**Primitive element (Integer value)**	**Non-linearity of function in [14]**	**Different options for swapping elements**	**Non-linearity of improved function**
285	2	112	(i) $\alpha^{104} \leftrightarrow \alpha^{230}$	114
			(ii) $\alpha^{36} \leftrightarrow \alpha^{247}$	114
			(iii) $\alpha^{106} \leftrightarrow \alpha^{153}$	114
351	4	112	(i) $\alpha^{32} \leftrightarrow \alpha^{160}$	114
			(ii) $\alpha^{107} \leftrightarrow \alpha^{238}$	114
			(iii) $\alpha^{54} \leftrightarrow \alpha^{234}$	114
369	47	112	(i) $\alpha^{20} \leftrightarrow \alpha^{228}$	114
			(ii) $\alpha^{5} \leftrightarrow \alpha^{220}$	114
			(iii) $\alpha^{31} \leftrightarrow \alpha^{228}$	114
451	220	112	(i) $\alpha^{20} \leftrightarrow \alpha^{228}$	114
			(ii) $\alpha^{101} \leftrightarrow \alpha^{233}$	114
			(iii) $\alpha^{126} \leftrightarrow \alpha^{235}$	114
501	10	112	(i) $\alpha^{46} \leftrightarrow \alpha^{137}$	114
			(ii) $\alpha^{51} \leftrightarrow \alpha^{254}$	114
			(iii) $\alpha^{58} \leftrightarrow \alpha^{136}$	114

5 Summarized Results

The devised algorithms were implemented in MAGMA computational algebra system for $8 \leq n \leq 11$ and all the functions belonging to Carlet-Feng infinite class of Boolean functions [14] were improved using these. Majority of functions were improved using Algorithm 1 by a single pair swap (an element from the support set with a root). The functions that could not be improved by Algorithm 1 were improved by Algorithm 2 within at the most four pair swaps. Table 4 demonstrates a comparison of non-linearities of the improved functions with their parent functions in [14] for $8 \leq n \leq 11$.

Table 4. Comparison of properties of functions in [14] and our improved functions

n	**Degree** $f_{Carlet-Feng}$	**AI** $f_{Carlet-Feng}$	**Non-linearity** $f_{Carlet-Feng}$	**Degree** $f_{Improved}$	**AI** $f_{Improved}$	**Non-linearity** $f_{Improved}$
8	7	4	112	7	4	114
9	8	5	232	8	5	234
10	9	5	478	9	5	480
11	10	6	980	10	6	982

Note: The values of $f_{Carlet-Feng}$ have been taken from [14]. The non-linearity values are average, $nl(f_{Improved}) = (nl(f_{Carlet-Feng}) + 2)$ in all cases.

It was mentioned in [14], the product of the constructed functions with any linear functions $(f * l)$ reduces the degree of the resultant functions to at most $n - 2$ in case of even number of variables "n" and at most $n - 1$ in case of odd "n". Hence the functions do not fall in the worst case or next to worst case resistance to algebraic and fast algebraic attacks [25]. Similar analysis was performed on the functions improved by Algorithm 1 and 2 for $8 \leq n \leq 10$ and it was ascertained that the improved functions offer identical behavior to the parent functions when the product with the set of all linear functions was obtained. Results of the analysis are presented in Table 5.

Table 5. Comparison of properties of functions in [14] and our improved functions

n	**Degree** $f_{Improved}$	**Degree** $f_{Improved} * l$
8	7	$\geq$6
9	8	$\geq$8
10	9	$\geq$8

6 Conclusion

An effective and efficient method of improving non-linearity of Carlet-Feng infinite class of Boolean functions has been developed. The two algorithms devised have been derived from genetic hill climbing algorithm. Not only do these increase the non-linearity of the parent functions but also preserve other cryptographic properties including maximal algebraic degree for balanced functions, optimal algebraic immunity and good resistance to algebraic and fast algebraic attacks as shown in practical results presented. Moreover, the method also increases the total number of functions that can be obtained for each number of variables, whilst a particular defining polynomial and primitive element is fixed. The algorithms have been implemented practically using MAGMA and results presented verify the efficacy of proposed method.

Acknowledgements. We sincerely thank all the anonymous reviewers for their valuable feedback that resulted in overall improvement of this research work.

References

1. Courtois, N.T., Meier, W.: Algebraic Attacks on Stream Ciphers with Linear Feedback. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 345–359. Springer, Heidelberg (2003)
2. Courtois, N.: Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 176–194. Springer, Heidelberg (2003)
3. Armknecht, F.: Improving Fast Algebraic Attacks. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 65–82. Springer, Heidelberg (2004)
4. Meier, W., Pasalic, E., Carlet, C.: Algebraic Attacks and Decomposition of Boolean Functions. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 474–491. Springer, Heidelberg (2004)
5. Armknecht, F.: Algebraic Attacks and Annihilators. In: WEWoRC 2005. LNI, vol. P-74, pp. 13–21. Gesellschaftfür Informatik (2005)
6. Armknecht, F., Ars, G.: Introducing a New Variant of Fast Algebraic Attacks and Minimizing Their Successive Data Complexity. In: Dawson, E., Vaudenay, S. (eds.) Mycrypt 2005. LNCS, vol. 3715, pp. 16–32. Springer, Heidelberg (2005)
7. Rønjom, S., Helleseth, T.: A new attack on the filter generator. IEEE Trans. on Inform. Th. 53(5), 1752–1758 (2007)
8. Braeken, A., Preneel, B.: On the Algebraic Immunity of Symmetric Boolean Functions. In: Maitra, S., Veni Madhavan, C.E., Venkatesan, R. (eds.) INDOCRYPT 2005. LNCS, vol. 3797, pp. 35–48. Springer, Heidelberg (2005)
9. Dalai, D.K., Gupta, K.C., Maitra, S.: Cryptographically Significant Boolean Functions: Construction and Analysis in Terms of Algebraic Immunity. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 98–111. Springer, Heidelberg (2005)
10. Carlet, C.: A method of construction of balanced functions with optimum algebraic immunity. IACR Cryptology ePrint Archive 2006: 149 (2006)

11. Carlet, C., Dalai, D.K., Gupta, K.C., Maitra, S.: Algebraic immunity for cryptographically significant Boolean functions: analysis and construction. IEEE Trans. Inform. Th. 52(7), 3105–3121 (2006)
12. Dalai, D.K., Maitra, S., Sarkar, S.: Basic theory in construction of Boolean functions with maximum possible annihilator immunity. Des. Codes Cryptogr. 40(1), 41–58 (2006)
13. Carlet, C.: On Bent and Highly Nonlinear Balanced/Resilient Functions and Their Algebraic Immunities. In: Fossorier, M.P.C., Imai, H., Lin, S., Poli, A. (eds.) AAECC 2006. LNCS, vol. 3857, pp. 1–28. Springer, Heidelberg (2006)
14. Carlet, C., Feng, K.: An Infinite Class of Balanced Functions with Optimal Algebraic Immunity, Good Immunity to Fast Algebraic Attacks and Good Nonlinearity. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 425–440. Springer, Heidelberg (2008)
15. Tu, Z., Deng, Y.: A Class of 1-Resilient Function with High Nonlinearity and Algebraic Immunity. IACR Cryptology ePrint Archive 2010, 179 (2010)
16. Yusong, D., Dingyi, P.: Construction of Boolean functions with maximum algebraic immunity and count of their annihilators at lowest degree. Science China, Information Sciences 53(4), 780–787 (2010)
17. Pan, S., Fu, X., Zhang, W.: Construction of 1-Resilient Boolean functions with Optimal Algebraic Immunity and Good Nonlinearity. Journal of Computer Science and Technology 26(2) (March 2011)
18. Wang, Q., Peng, J., Kan, H.: Constructions of Cryptographically Significant Boolean functions Using Primitive Polynomials. IEEE Trans. on Inform. Th. 56(6) (June 2010)
19. Zeng, X., Carlet, C., Shan, J., Hu, L.: More Balanced Boolean functions With Optimal Algebraic Immunity and Good Nonlinearity and Resistance to Fast Algebraic Attacks. IEEE Trans. on Inform. Th. 57(9), 6310–6320 (2011)
20. Millan, W., Clark, A., Dawson, E.: An Effective Genetic Algorithm for Finding Highly Non-linear Boolean Functions. In: Han, Y., Quing, S. (eds.) ICICS 1997. LNCS, vol. 1334, pp. 149–158. Springer, Heidelberg (1997)
21. Ding, C., Xiao, G., Sha, W. (eds.): The Stability Theory of Stream Ciphers. LNCS, vol. 561. Springer, Heidelberg (1991)
22. Meier, W., Staffelbach, O.: Fast Correlation Attacks on Stream Ciphers. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 301–314. Springer, Heidelberg (1988)
23. Canteaut, A., Trabbia, M.: Improved Fast Correlation Attacks Using Parity-Check Equations of Weight 4 and 5. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 573–588. Springer, Heidelberg (2000)
24. Porwik, P.: The Spectral Test of the Boolean Function Linearity. International Journal of Applied Mathematics and Computer Science 13(4), 567–575 (2003)
25. Carlet, C.: On a weakness of the Tu-Deng function and its repair. Cryptology ePrint Archive, Report 2009/606, http://eprint.iacr.org/2009/606

Appendix A: Some Results for n = 9

Table 6. Comparison of non-linearities of Carlet-Feng infinite class functions and our improved functions

Defining Polynomial (Integer value)	Primitive element (Integer value)	Non-linearity of function in [14]	Elements swapped (root↔support)	Non-linearity of improved function
529	23	232	$\alpha^{1} \leftrightarrow \alpha^{470}$	234
539	10	234	$\alpha^{96} \leftrightarrow \alpha^{355}$	236
647	2	234	$\alpha^{220} \leftrightarrow \alpha^{417}$	236
661	17	234	$\alpha^{152} \leftrightarrow \alpha^{335}$	236
731	64	232	$\alpha^{254} \leftrightarrow \alpha^{505}$	234
847	32	234	$\alpha^{27} \leftrightarrow \alpha^{494}$	236
859	197	232	$\alpha^{182} \leftrightarrow \alpha^{441}$	234
949	219	232	$\alpha^{5} \leftrightarrow \alpha^{260}$	234

Table 7. Different improved functions by our method from the same parent function in [14]

Defining Polynomial (Integer value)	Primitive element (Integer value)	Non-linearity of function in [14]	Different options for swapping elements	Non-linearity of improved function
529	3	232	(i) $\alpha^{1} \leftrightarrow \alpha^{470}$	234
			(ii) $\alpha^{201} \leftrightarrow \alpha^{343}$	234
			(iii) $\alpha^{5} \leftrightarrow \alpha^{470}$	234
731	64	232	(i) $\alpha^{254} \leftrightarrow \alpha^{505}$	234
			(ii) $\alpha^{184} \leftrightarrow \alpha^{506}$	234
			(iii) $\alpha^{249} \leftrightarrow \alpha^{506}$	234
901	386	232	(i) $\alpha^{241} \leftrightarrow \alpha^{261}$	234
			(ii) $\alpha^{254} \leftrightarrow \alpha^{259}$	234
			(iii) $\alpha^{144} \leftrightarrow \alpha^{325}$	234

Appendix B: Some Results for n = 10

Table 8. Comparison of non-linearities of Carlet-Feng infinite class functions and our improved functions

Defining Polynomial (Integer value)	Primitive element (Integer value)	Non-linearity of function in [14]	Elements swapped (root↔support)	Non-linearity of improved function
1033	2	478	$\alpha^3 \leftrightarrow \alpha^{571}$	480
1051	16	480	$\alpha^{12} \leftrightarrow \alpha^{856}$	482
1163	399	480	$\alpha^{367} \leftrightarrow \alpha^{1007}$	482
1239	2	478	$\alpha^{92} \leftrightarrow \alpha^{945}$	480
1305	903	478	$\alpha^{72} \leftrightarrow \alpha^{815}$	480
1347	32	478	$\alpha^{205} \leftrightarrow \alpha^{863}$	480
1431	4	478	$\alpha^{392} \leftrightarrow \alpha^{582}$	480
2011	16	482	$\alpha^{1} \leftrightarrow \alpha^{630}$	484

Table 9. Different improved functions by our method from the same parent function in [14]

Defining Polynomial (Integer value)	Primitive element (Integer value)	Non-linearity of function in [14]	Different options for swapping elements	Non-linearity of improved function
1033	2	478	(i) $\alpha^3 \leftrightarrow \alpha^{571}$	480
			(ii) $\alpha^5 \leftrightarrow \alpha^{761}$	480
			(iii) $\alpha^6 \leftrightarrow \alpha^{853}$	480
1305	903	478	(i) $\alpha^{72} \leftrightarrow \alpha^{815}$	480
			(ii) $\alpha^{170} \leftrightarrow \alpha^{793}$	480
			(iii) $\alpha^{202} \leftrightarrow \alpha^{642}$	480
1431	4	478	(i) $\alpha^{392} \leftrightarrow \alpha^{582}$	480
			(ii) $\alpha^{501} \leftrightarrow \alpha^{749}$	480
			(iii) $\alpha^{27} \leftrightarrow \alpha^{590}$	480

Appendix C: Some Results for n = 11

Table 10. Comparison of non-linearities of Carlet-Feng infinite class functions and our improved functions

Defining Polynomial (Integer value)	Primitive element (Integer value)	Non-linearity of function in [14]	Elements swapped (root↔support)	Non-linearity of improved function
2053	6	982	$\alpha^{631} \leftrightarrow \alpha^{1584}$	984
2071	2044	982	$\alpha^{467} \leftrightarrow \alpha^{1886}$	984
2119	7	980	$\alpha^{545} \leftrightarrow \alpha^{1352}$	982
2147	16	982	$\alpha^{430} \leftrightarrow \alpha^{1531}$	984
2421	2	982	$\alpha^{1} \leftrightarrow \alpha^{1122}$	984
2955	7	986	$\alpha^{529} \leftrightarrow \alpha^{1375}$	988
3573	596	986	$\alpha^{141} \leftrightarrow \alpha^{1388}$	988
3851	746	982	$\alpha^{473} \leftrightarrow \alpha^{1057}$	984

Table 11. Different improved functions by our method from the same parent function in [14]

Defining Polynomial (Integer value)	Primitive element (Integer value)	Non-linearity of function in [14]	Different options for swapping elements	Non-linearity of improved function
2119	7	980	(i) $\alpha^{545} \leftrightarrow \alpha^{1352}$	982
			(ii) $\alpha^{316} \leftrightarrow \alpha^{1764}$	982
			(iii) $\alpha^{162} \leftrightarrow \alpha^{1862}$	982
2421	2	982	(i) $\alpha^{1} \leftrightarrow \alpha^{1122}$	984
			(ii) $\alpha^{501} \leftrightarrow \alpha^{1025}$	984
			(iii) $\alpha^{985} \leftrightarrow \alpha^{1253}$	984
3851	746	982	(i) $\alpha^{473} \leftrightarrow \alpha^{1057}$	984
			(ii) $\alpha^{650} \leftrightarrow \alpha^{1677}$	984
			(iii) $\alpha^{857} \leftrightarrow \alpha^{2012}$	984

Some Representations of the S-Box of Camellia in $GF(((2^2)^2)^2)$

Alberto F. Martínez-Herrera, J. Carlos Mex-Perera, and Juan A. Nolazco-Flores

Tecnológico de Monterrey, Monterrey Campus, 2501, Garza Sada Sur, Tecnológico, Monterrey, Mexico
{a00798620,carlosmex,jnolazco}@itesm.mx

Abstract. Substitution Box (S-box) is usually the most complex module in some block ciphers. Some prominent ciphers such as AES and Camellia use S-boxes, which are affine equivalents of a multiplicative inverse in small finite fields. This manuscript describes mathematical representations of the Camellia S-box by using composite fields such as polynomial, normal or mixed. An optimized hardware implementation typically aims to reduce the number of gates to be used. Our theoretical design with composite normal bases allows saving gates in the critical path by using 19 XOR gates, 4 AND gates and 2 NOT gates. With composite mixed bases, the critical path has 2 XOR gates more than the representation with composite normal bases. Redundancies found in the affine transformation matrix that form the composite fields were eliminated. For mixed bases, new Algebraic Normal Form identities were obtained to compute the inner composite multiplicative inverse, reducing the critical path of the complete implementation of the Camellia S-box. These constructions were translated into transistor-gate architectures for hardware representations by using Electric VLSI [29] under MOSIS C5 process [17], [18], thus obtaining the corresponding schematic models.

Keywords: S-box, composite fields, number of gates, critical path.

1 Introduction

Nowadays, there is sensitive information that might be conveyed through different unreliable environments. Then, it is desirable to protect such information against eavesdropping or impersonation. Cryptography has become in an invaluable tool to counteract these issues. Nevertheless, several hardware implementations of ciphers generate a huge area usage, power consumption and processing time. These aspects describe the basic footprint of a VLSI implementation. Techniques such as Look-Up Tables (LUTs) or the use of public key cryptography are not possible in environments where a small footprint is mandatory (wireless, near field communication, RFID). Thus, a solution is to use block ciphers. However, some block ciphers have a footprint that cannot be fitted in constrained environments [7]. Therefore, to find mathematical representations of the ciphers that help saving footprint is needed. These representations lead to have different ways of implementing a component of the VLSI architecture. Some representations can be more feasible than others depending on the area usage and processing time. For instance, a particular aspect to consider is the critical path (the largest number of gates

J. Pieprzyk, A.-R. Sadeghi, and M. Manulis (Eds.): CANS 2012, LNCS 7712, pp. 296–309, 2012.

that an input signal travels from the input to the output of a circuit [8]); if an implementation has a large critical path, it will have a negative impact on the processing time even though such implementation contains the best area usage. Thus, a good trade-off should be found among all aspects that form the footprint of a VLSI implementation in a particular environment.

This manuscript is formed by the following sections. In Sect. 2, composite bases are described. In Sect. 3, previous work with composite fields is treated. In Sect. 4, a brief description of the Camellia S-box is given [1]. In Sect. 5, results and comparison tables that show the number of gates used to implement the Camellia S-box are presented. In Sect. 6, the conclusions of this work are summarized.

2 Mathematical Background

An octet can be represented as a vector of bits $\mathbf{b} = [b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7]$, $b_i \in GF(2)$, $i \in \{0, ..., 7\}$ (See notation used by Canright [4] and Nogami et al. [34]). A way of representing $\mathbf{b}$ in the composite field $GF(((2^2)^2)^2)$ can be by using composite polynomial bases as $\mathbf{b} = b(x, y, w)$, where $b(x, y, w) = [(b_0 + b_1 w) + (b_2 + b_3 w)y] + [(b_4 + b_5 w) + (b_6 + b_7 w)y]$; or by using composite normal bases as $\mathbf{b} = b(X, Y, W)$, where $b(X, Y, W) = [(b_0 W + b_1 W^2)Y + (b_2 W + b_3 W^2)Y^4]X + [(b_4 W + b_5 W^2)Y + (b_6 W + b_7 W)Y^4]X^{16}$. An example with mixed bases can be written as $b(X, Y, w) = [(b_0 + b_1 w)Y + (b_2 + b_3 w)Y^4]X + [(b_4 + b_5 w)Y + (b_6 + b_7 w)Y^4]X^{16}$. Lowercase and uppercase letters represent composite polynomial and normal bases, respectively. We use the fields $\{1 + x, X + X^{16}\} \in GF(((2^2)^2)^2)$, $\{1 + y, Y + Y^4\} \in GF((2^2)^2)$ and $\{1 + w, W + W^2\} \in GF(2^2)$. With these bases, there are eight possible representations: PPP, PPN, PNP, PNN, NPP, NPN, NNP, NNN, P meaning polynomial basis and N meaning normal basis. The order of the acronyms indicates how a composite field is formed. For instance, PPP refers to $b(x, y, w)$, NNN refers to $b(X, Y, W)$ and $b(X, Y, w)$ refers to NNP. Nevertheless, there are 432 different composite field representations [4]. For instance, $X^{16} + 1$ instead of $X^{16} + X$, $Y^8 + Y^2$ instead of $Y^4 + Y$. Being $f(z)$ defined in $GF(2^8)$, a Boolean matrix of dimension $8\mathbf{x}8$ is generated by a root $\beta = \mathbf{b}$ that nulls $f(z)$. Thus

$$\mathbf{M} = [\mathbf{b}_0^T\ \mathbf{b}_1^T\ \mathbf{b}_2^T\ \mathbf{b}_3^T\ \mathbf{b}_4^T\ \mathbf{b}_5^T\ \mathbf{b}_6^T\ \mathbf{b}_7^T] \tag{1}$$

Moreover, there are $\mathbf{M}_j$, $j \in \{0, ..., 7\}$ different Boolean matrix transformations, because $f(z)$ has eight different roots according to the *Fundamental Algebra Theorem* [23], [16]. The reader should take care of the composite field where the root is represented, because the octet representations change. Additionally, let h and l the subindexes that indicate the most significant and the least significant component of the composite field, respectively; τ, ν, Δ, α_h, α_l, b_h, $b_l \in GF((2^2)^2)$. The multiplicative inverse can be computed as [4]

– Polynomial bases representation

$$\Delta = (\alpha_h)^2 \nu + (\alpha_h \tau + \alpha_l)\alpha_l \tag{2}$$

$$b_l = (\alpha_h \tau + \alpha_l)\Delta^{-1} \tag{3}$$

$$b_h = \alpha_h \Delta^{-1} \tag{4}$$

– Normal bases representation

$$\Delta = \alpha_h \alpha_l \tau^2 + (\alpha_h + \alpha_l)^2 \nu \tag{5}$$
$$b_l = \alpha_h \Delta^{-1} \tag{6}$$
$$b_h = \alpha_l \Delta^{-1} \tag{7}$$

Then, the complete operation can be represented as

$$\alpha = \mathbf{Ma} \tag{8}$$
$$\gamma = \alpha^{-1} \tag{9}$$
$$\mathbf{b} = \mathbf{M}^{-1}\gamma \tag{10}$$

where the Boolean inverse matrix $\mathbf{M}^{-1}$ is applied to reverse the transformation.

Some S-boxes are constructed by using the multiplicative inverse on finite fields and then included in block ciphers [15]. However, this operation is complex for hardware. A solution to minimize its complexity is by using composite fields, where the multiplicative inverse can be split into different small and simultaneous operations as shown in (2-4) and (5-7). Each operation can be expressed recursively until the simplest field $GF(2^2)$. It is important to highlight that the rules given above can also be applied to $GF((2^4)^2)$, but now the simplest field is $GF(2^4)$. Additionally, there are only two levels instead of three levels. For instance, $\mathbf{b}$ can be represented as $b(x, y) = [b_0 + b_1 y + b_2 y^2 + b_3 y^3] + [b_4 + b_5 y + (b_6 y^2 + b_7 y^3]x$ (PP representation) or $b(X, y) = [b_0 + b_1 y + b_2 y^2 + b_3 y^3]X + [b_4 + b_5 y + (b_6 y^2 + b_7 y^3]X^{16}$ (NP representation).

On the other hand, the recursive versions of the operations for composite fields can be combined when composite mixed bases are used. Nevertheless, the properties for each base should be preserved when they are applied on each level.

3 Related Studies

Mastrovito provided several hardware architectures for optimization purposes [14]. Paar proposed the use of composite fields to minimize the complexity of operations in a finite field. Paar found optimization algorithms for eliminating redundancy in isomorphic matrix transformations [23]. Oloffson generalized Paar's and Mastrovito's studies related to optimization of the multiplicative inverse [22]. Boyar et al. presented a new technique and additional representations of AES S-box [3]. That technique is based on minimizing Hamming Boolean distance among a set of Boolean functions, thus obtaining the most compact representation of AES S-box known in the open literature. However, the critical path is increased. Hence, Boyar et al. provided a modification of Paar's technique to maintain an acceptable trade-off between the number of gates and the critical path [10]. Such technique is denoted as Low-Deep-Greedy (LDG) technique. On the other hand, different algorithms have been proposed to look for roots on irreducible polynomials because Paar's technique is only applicable to primitive polynomials [26], [36].

Rijmen, one of the designers of Rijndael (now AES), suggested the use of composite fields for optimization purposes [25]. This paradigm was used by Satoh et al. on AES by mapping $GF(2^8) \mapsto GF(((2^2)^2)^2)$ [28] and Camellia by mapping $GF(2^8) \mapsto$

$GF((2^4)^2)$ [27]. At least for the knowledge of the authors, this is the unique available reference that describes a way to use the S-box of Camellia by using composite fields. Some examples of this claim can be found in [32], [33] and [9]. Barkan et al. gave other ways to represent AES S-box by using dual bases and also by using different irreducible polynomials [2]. Similar efforts were performed by Lu [12], Lyu [13] and Chen et al. [11]. Wolkerstorfer et al. provided additional ways to construct AES S-box by mapping $GF(2^8) \mapsto GF((2^4)^2)$. They simplified the expressions for Δ^{-1} by using Algebraic Normal Forms (ANFs) [30]. Additional representations for Δ^{-1} were also performed by Zhang et al. [35], [36] and Wong et al. [31].

Mentens et al. exploited the eight roots on the irreducible polynomial of AES by mapping $GF(2^8) \mapsto GF(((2^2)^2)^2)$, where the authors chose those matrices with low Hamming weight [16]. Canright published a comprehensive and detailed study on the S-box of AES. He did his studies by using all possible available bases: polynomial, normal and mixed by mapping $GF(2^8) \mapsto GF(((2^2)^2)^2)$. He proved that the complete number of available combinations among composite bases representations is 432, being the best case those represented by normal bases for all levels [6,5,4]. Nikova et al. showed a way to extend Canright's work by mapping $GF(2^8) \mapsto GF((2^4)^2)$ on normal bases. The authors described the ways to represent Δ^{-1} in $GF(2^4)$ by using ANFs and implementing these ways in MOSIS technologies [20]. Nogami et al. looked for the best isomorphic representation under *mixture* bases, with low Hamming weight and small critical path. These constructions can be achieved by *mixing* polynomial and normal operation properties at the same composite field level [21], [34]. Nogami et al.'s way of representing composite field operations is different than the ones reported by Canright.

4 Mathematical Representation of the S-Box of Camellia

Camellia has 4 different S-boxes [1], but the last 3 S-boxes are derived from the first S-box as shown in the identities listed in (11-14). Letting $\mathbf{a}$ and $\mathbf{b}$ the input and output of the Camellia S-box, then

$$\mathbf{b} = S_1(\mathbf{a}) \tag{11}$$

$$S_2(\mathbf{a}) = [b_7\ b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6] \tag{12}$$

$$S_3(\mathbf{a}) = [b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7\ b_0] \tag{13}$$

$$S_4(\mathbf{a}) = S_1([a_7\ a_0\ a_1\ a_2\ a_3\ a_4\ a_5\ a_6]) \tag{14}$$

Each S-box is formed by 4 components (see Fig.1): a Boolean addition with $\mathbf{c}_i$; a transformation by $\mathbf{F}$; a composite inverse operation and a transformation by $\mathbf{H}$ plus a Boolean addition with $\mathbf{c}_o$.

Camellia primitive polynomial is $f(z) = z^8 + z^6 + z^5 + z^3 + 1$ and the composite polynomial is $q(w) = w^4 + w + 1$ [1]. Satoh et al. specified the composite polynomial $p(x) = x^2 + \tau w + \nu$ and their corresponding values for $\tau,\ \nu \in GF(2^4)$ [27]. They found that the corresponding values for each constant is $\tau = 1 = [1\ 0\ 0\ 0]$ and $\nu = 1 + w^3 = [1\ 0\ 0\ 1]$. $\mathbf{F}$ and $\mathbf{H}$ contain the isomorphic matrix $\mathbf{M}$ and its inverse

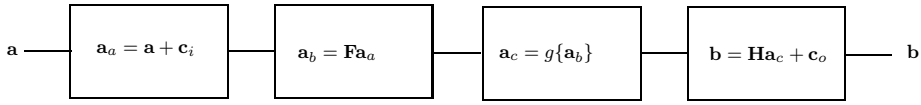


Fig. 1. Simplified S-box architecture of Camellia

$\mathbf{M}^{-1}$, respectively. $\mathbf{M}$ and $\mathbf{M}^{-1}$ were obtained by using $p(x)$, $q(w)$ and the constants τ and ν. The S-box can be described as

$$\mathbf{b} = \mathbf{H}[compositeinverse(\mathbf{F}(\mathbf{a} + \mathbf{c}_i))] + \mathbf{c}_o \ \in \ GF((2^4)^2) \tag{15}$$

where the values of the constants are $\mathbf{c}_i = [1\ 0\ 1\ 0\ 0\ 0\ 1\ 1]$ and $\mathbf{c}_o = [0\ 1\ 1\ 1\ 0\ 1\ 1\ 0]$. It is needed to obtain equivalent matrices $\mathbf{H}_o$ and $\mathbf{F}_o$ in $GF(2^8)$. The equations to obtain $\mathbf{F}$ and $\mathbf{H}$ are $\mathbf{F} = \mathbf{M}\mathbf{F}_o$ and $\mathbf{H} = \mathbf{H}_o\mathbf{M}^{-1}$. Thus, $\mathbf{H}_o$ and $\mathbf{F}_o$ can be computed as

$$\mathbf{F}_o = \mathbf{M}^{-1}\mathbf{F} \tag{16}$$

$$\mathbf{H}_o = \mathbf{H}\mathbf{M} \tag{17}$$

Descriptions given for $\mathbf{F}_o$ and $\mathbf{H}_o$ were tested to obtain the first and the fourth S-boxes of Camellia, as shown in (16-17). The second and the third S-boxes of Camellia are trivial to be obtained. Therefore, these S-boxes were omitted when experiments were carried out to obtain new matrices $\mathbf{F}$ and $\mathbf{H}$ by using $\mathbf{H}_o$ and $\mathbf{F}_o$.

5 Results and Comparisons

Paar's matrix generation algorithm [23] and concepts given in Sect. 2 were applied to compute new representations of $\mathbf{F}$ and $\mathbf{H}$. Those matrices with the fewer number of ones were chosen (see Table 1) by using the criterion $wt(\mathbf{F}) + wt(\mathbf{H})$, where wt is the Hamming weight . Variables $\tau,\ \nu,\ T,\ N$ are represented in their decimal representation, obtained from their binary representation. In similar ways to previous works available in the literature, identities $\tau = 1,\ T = 1$ were considered for normal and mixed bases. For PPP representation, $T \neq 1,\ N = 1$ were considered because an acceptable result is obtained. The same consideration was applied to PP representation for $\tau = 2,\ \nu = 2$. Case based on Satoh et al. parameters is also included in the table. In this work, $S_1(\mathbf{a})$ was optimized.

The results shown in Table 2 were obtained from each composite field. LDG technique provided the number of XOR gates and the critical path for $\mathbf{F}$ and $\mathbf{H}$. The critical path (cp) is computed by $cp(\mathbf{U}) = log_2\lceil\ (\#ones(\mathbf{u}_i))\rceil$, where $\mathbf{u}_i$ is the i-th row of matrix $\mathbf{U}$ with the largest number of non-zero elements. This operation is applied to $\mathbf{F}$ and $\mathbf{H}$. For instance, cases 1 and 2 have only one row with 3 ones, so the longest critical path is $\lceil\ log_2(3)\rceil = 2$. This value is consistent with values given in Table 2. LDG technique provided automatically how many XOR gates are needed to implement $\mathbf{F}$ and $\mathbf{H}$. As an example, reductions for case 6 are shown in Appendix C. The best cases

Table 1. Hamming weight of $\mathbf{F}$ and $\mathbf{H}$

Case	Field	$wt(\mathbf{F})$	$wt(\mathbf{H})$	$wt(\mathbf{F}) + wt(\mathbf{H})$
1	PP($\beta = 16,\ \tau = 1,\ \nu = 9$), (Satoh et al. [27])	17	17	34
2	PP($\beta = 144,\ \tau = 2,\ \nu = 2$)	15	19	34
3	PPP($\beta = 17,\ \tau = 1,\ \nu = 5,\ T = 3,\ N = 1$)	28	24	52
4	PPP($\beta = 16,\ \tau = 1,\ \nu = 8,\ T = 1,\ N = 3$)	24	30	54
5	PPP($\beta = 17,\ \tau = 1,\ \nu = 15,\ T = 1,\ N = 2$)	30	23	53
6	NNN($\beta = 60,\ \tau = 15,\ \nu = 14,\ T = 3,\ N = 2$)	24	28	52
7	NNN($\beta = 195,\ \tau = 15,\ \nu = 14,\ T = 3,\ N = 2$)	24	28	52
8	PNP($\beta = 86,\ \tau = 5,\ \nu = 2,\ T = 1,\ N = 2$)	26	25	51
9	PNP($\beta = 89,\ \tau = 5,\ \nu = 8,\ T = 1,\ N = 2$)	26	25	51
10	NPP($\beta = 1,\ \tau = 1,\ \nu = 8,\ T = 1,\ N = 3$)	28	23	51
11	NPP($\beta = 16,\ \tau = 1,\ \nu = 8,\ T = 1,\ N = 3$)	28	23	51
12	NP($\beta = 1,\ \tau = 1,\ \nu = 9$)	23	17	40
13	NP($\beta = 16,\ \tau = 1,\ \nu = 9$)	23	17	40

for mixed bases were found with $(1 + x)$, $(Y + Y^4)$, $(1 + z)$ (PNP) and $(X + X^{16})$, $(1 + y)$, $(1 + z)$ (NPP). For all the tables shown in this manuscript, X means XOR, No means NOT, A means AND, N means Normal and P means polynomial. Additionally, XOR means to have two-input XOR gate, AND means to have two-input AND gate.

Table 2. Number of gates obtained by applying LDG technique

Case	1	2	3	4	5	6	7	8	9	10	11	12	13
# X $\mathbf{F}$	8	7	13	11	13	13	13	11	11	13	13	12	12
$cp(\mathbf{F})$	2	1	3	3	3	2	2	3	3	3	3	2	2
# X $\mathbf{H}$	8	9	12	12	12	13	13	12	12	11	11	9	9
$cp(\mathbf{H})$	2	2	3	3	3	3	3	3	3	3	3	2	2

For computing Δ^{-1}, Wong et al. [31], and Wolkerstorfer et al. [30] identities were considered. Wolkerstorfer et al.'s identities were applied to $GF((2^4)^2)$ and such representation is simplified in a straightforward way. Wong et al.'s identities were applied to $GF(((2^2)^2)^2)$. Nevertheless, we perform our own reductions. These identities can be applied to NPP and PPP. Nevertheless, for PNP the ANF representations of Δ^{-1} are expressed as shown in (18-21). Such identities can be implemented by using 12 XOR and 8 AND gates.

$$x_0^{-1} = x_2 + x_1x_3 + x_0x_2x_3 + x_1x_2x_3 \tag{18}$$
$$x_1^{-1} = x_2 + x_0x_2 + x_1x_2 + x_3 + x_1x_2x_3 \tag{19}$$
$$x_2^{-1} = x_0 + x_0x_1x_2 + x_1x_3 + x_0x_1x_3 \tag{20}$$
$$x_3^{-1} = x_0 + x_1 + x_0x_2 + x_0x_3 + x_0x_1x_3 \tag{21}$$

Identities listed in (18-21) were obtained by applying ANF transformation [24]

$$\mathbf{A}_0 = [1] \tag{22}$$

$$\mathbf{A}_n = \begin{pmatrix} \mathbf{A}_{n-1} & 0 \\ \mathbf{A}_{n-1} & \mathbf{A}_{n-1} \end{pmatrix} \tag{23}$$

$$f_j(x_0, x_1, x_2, x_3) = \mathbf{A}_4 \mathbf{f}_j \tag{24}$$

where $\mathbf{f}_j, j \in \{0, 1, 2, 3\}$ is the truth table for each one of the outputs in Δ^{-1}, $\mathbf{A}_4$ is a $16\mathbf{x}16$ matrix and $f_j(x_0, x_1, x_2, x_3), j \in \{0, 1, 2, 3\}$ is the ANF representation of the Boolean function. Table 3 shows the number of gates and critical path to be used for each representation of Δ^{-1}. These results were obtained by looking for redundancies on each ANF expression. For the purposes of this manuscript, our results were utilized for implementing Δ^{-1}. An example of the reduction of gates for Δ^{-1} is shown in Appendix B.

Table 3. Comparison of the number of gates and critical path for Δ^{-1}

Δ^{-1}	Wong et al.	Ours	cp Wong et al.	Ours
P	-	18 X/10 A	-	3 X/2 A
PP $(T = 1, N = 2)$	15 X/8 A	13 X/8 A	4 X/2 A	4 X/2 A
PP $(T = 1, N = 3)$	14 X/8 A	13 X/8 A	3 X/2 A	4 X/2 A
PP $(T = 3, N = 1)$	15 X/8 A	15 X/8 A	3 X/2 A	4 X/2 A
NN $(T = 3, N = 2)$	13 X/8 A	12 X/8 A	3 X/1 A	3 X/2 A
NP $(T = 1, N = 2)$	-	12 X/8 A	-	3 X/2 A

To obtain the complete quantity of gates, $\mathbf{c}_i$ and $\mathbf{c}_o$ constants must be considered. There are 4 non-zero elements in $\mathbf{c}_i$ and 5 non-zero elements in $\mathbf{c}_o$. Thus, there are 9 NOT gates. The total amount of gates and the complete critical path are shown in Table 4.

Table 4. Total amount of gates and cp for each case of the Camellia S-box

Case	$\#Gates(\mathbf{F},\mathbf{a}_c,\mathbf{H})$	cp $(\mathbf{F},\mathbf{a}_c,\mathbf{H})$
1	89 X/58 A/9 No	2 X/1 No+ 11 X/4 A+2 X = 15 X/4 A/1 No
2	91 X/58 A/9 No	1 X/1 No+ 12 X/4 A+2 X/1 No = 15 X/4 A/2 No
3	108 X/56 A/9 No	3 X/1 No+ 14 X/4 A+3 X/1 No = 20 X/4 A/2 No
4	104 X/56 A/9 No	3 X/1 No+ 14 X/4 A+3 X/1 No = 20 X/4 A/2 No
5	107 X/56 A/9 No	3 X/1 No+ 14 X/4 A+3 X/1 No = 20 X/4 A/2 No
6	113 X/35 A/9 No	2 X/1 No+ 14 X/4 A+3 X/1 No = 19 X/4 A/2 No
7	113 X/35 A/9 No	2 X/1 No+ 14 X/4 A+3 X/1 N = 19 X/4 A/2 No
8	100 X/44 A/9 No	3 X/1 No+ 15 X/4 A+3 X/1 No = 21 X/4 A/2 No
9	100 X/44 A/9 No	3 X/1 No+ 15 X/4 A+3 X/1 No = 21 X/4 A/2 No
10	105 X/56 A/9 No	3 X/1 No+ 13 X/4 A+3 X/1 No = 19 X/4 A/2 No
11	105 X/56 A/9 No	3 X/1 No+ 13 X/4 A+3 X/1 No = 19 X/4 A/2 No
12	94 X/58 A/9 No	2 X/1 No+ 10 X/4 A+2 X/1 No = 14 X/4 A/2 No
13	94 X/58 A/9 No	2 X/1 No+ 10 X/4 A+2 X/1 No = 14 X/4 A/2 No

where $\mathbf{a}_c$ represents the composite inverse (see Fig. 1). An example of the method to reduce gates on $\mathbf{a}_c$ is shown in Appendix A.

For validation purposes, it is needed to provide an implementation by using a model of transistor gates. Ahmad et al.'s 6T XOR gate model [19] was implemented on Electric VLSI [29]. Ahmad et al.'s model is based on concatenated inverters with a feedback stage. In addition, canonical 6T AND CMOS gate was also implemented [8]. MOSIS C5 process was used with $V_{dd} = 3.0$V [18], [17]. Each 6T XOR gate has 3 PMOS and 3 NMOS transistors, each NOT gate has 1 PMOS and 1 NMOS transistors, and each 6T AND gate has 3 PMOS and 3 NMOS transistors. Table 5 shows the number of PMOS and NMOS transistors used for each case. For instance, case 1 uses 450 PMOS and 450 NMOS transistors.

Table 5. Number of transistors used for each Camellia S-box construction

Case	1	2	3	4	5	6-7	8-9	10-11	12-13
PMOS	450	456	501	489	498	453	441	492	465
NMOS	450	456	501	489	498	453	441	492	465

6 Conclusions

As seen in Table 4, Satoh et al.'s case uses fewer XOR gates (113 XOR-89 XOR=24 XOR) than cases 6 and 7, but cases 6 and 7 utilize fewer AND gates than Satoh et al.'s case (58 AND - 35 AND = 23 AND). On the other hand, Satoh et al.'s case uses 15 XOR/4 AND /1 NOT gates in its critical path, while cases 6 and 7 utilize 19 XOR/4 AND /2 NOT gates. Cases 8 and 9 utilize 100 XOR/44 AND gates and 21 XOR/4 AND /2 NOT as the critical path. For three levels, the representation with all composite normal bases is the nearest best case to Satoh et al.'s solution for the critical path with a difference of 4 XOR/1 NOT gates, reducing the number of AND gates (23 AND gates less) but increasing the number of XOR gates (24 XOR gates more). The best critical path is achieved by the combination NP, with 14 XOR gates, 4 and gates and 2 NOT gates. Compared to Satoh et al.'s case, this means to have 2 less equivalent NOT gates (inverters) in the critical path. Considering the number of transistors to be used, cases 8 and 9 use the fewer number of transistors, with 441 PMOS and 441 NMOS transistors.

Satoh et al. wrote that the four S-boxes of Camellia can be implemented by using 256 equivalent NAND gates. They implemented Δ^{-1} by generating its corresponding Sum-Of-Product representation [27]. For comparison purposes, here the ANF representation is considered for all the cases. Further optimization can be achieved by obtaining AND gate factorization for non-linear terms. In addition, multiplication operation for composite finite fields under normal and polynomial bases representation can be improved to obtain fewer XOR and AND gates. The use of normal bases at the higher level of the composite field representation reduces 1 XOR gate in the critical path, as it can be seen for cases 3, 4, 5 against cases 10 and 11. In this work, $S_1(\mathbf{a})$ was optimized. Each case shown in Table 4 was simulated in VHDL language to test the feasibility for each construction. In addition, each construction was carried out on Electric VLSI under MOSIS C5 process [18], [17]. As future work, the authors will work on the merge of some block ciphers with S-boxes based on multiplicative inverse.

Acknowledgments. The authors wish to give acknowledgments to anonymous reviewers for their suggestions and recommendations in order to improve this manuscript.

References

1. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Specifications of Camellia, a 128 bits block cipher. Technical Report 1, Mitsubishi and NTT DoCoMo, Tokio, Japan (August 2001)
2. Barkan, E., Biham, E.: In How Many Ways Can You Write Rijndael? In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 160–175. Springer, Heidelberg (2002)
3. Boyar, J., Peralta, R.: A New Combinational Logic Minimization Technique with Applications to Cryptology. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 178–189. Springer, Heidelberg (2010)
4. Canright, D.: A very compact Rijndael S-box. Technical report, Naval Postgraduate School, Monterey, CA, USA (May 2005)
5. Canright, D.: A Very Compact S-Box for AES. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 441–455. Springer, Heidelberg (2005)
6. Canright, D., Batina, L.: A Very Compact "Perfectly Masked" S-Box for AES. In: Bellovin, S.M., Gennaro, R., Keromytis, A., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 446–459. Springer, Heidelberg (2008)
7. Feldhofer, M., Wolkerstorfer, J.: Hardware Implementation of Symmetric Algorithms for RFID Security. In: Kitsos, P., Zhang, Y. (eds.) RFID Security, pp. 373–415. Springer US (2009), doi:10.1007/978-0-387-76481-8_15
8. Hodges, D., Jackson, H., Saleh, R.: Analysis and design of digital integrated circuits: in deep submicron technology. McGraw-Hill series in electrical engineering. McGraw-Hill Higher Education (2003)
9. Cheng, H., Heys, H.: Compact Hardware Implementation of the Block Cipher Camellia with Concurrent Error Detection. In: Canadian Conference on Electrical and Computer Engineering, CCECE 2007, pp. 1129–1132 (April 2007)
10. Boyar, J., Peralta, R.: A depth-16 circuit for the AES S-box. Cryptology ePrint Archive, Report 2011/332 (2011), http://eprint.iacr.org/
11. Chen, J.-H., Huang, S.-J., Lin, W.-C., Lu, Y.-K., Shieh, M.-D.: Exploration of Low-Cost Configurable S-Box Designs for AES Applications. In: International Conference on Embedded Software and Systems, ICESS 2008, pp. 422–428 (July 2008)
12. Lu, S.-C.: On the design of AES Based on dual cipher and composite field. Master Thesis, National Cheng Ku University, Taiwan (June 2003)
13. Lyu, J.-W.: Design and implementation of composite-dual cipher based on AES. Master thesis, National Cheng Ku University, Taiwan (June 2006)
14. Mastrovito, E.: VLSI architectures for computations in Galois fields. Dissertation, Linköping University, Linköping, Sweden (1991)
15. Menezes, A., Van Oorschot, P., Vanstone, S.: Handbook of applied cryptography. CRC Press series on discrete mathematics and its applications. CRC Press (1997)
16. Mentens, N., Batina, L., Preneel, B., Verbauwhede, I.: A Systematic Evaluation of Compact Hardware Implementations for the Rijndael S-Box. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 323–333. Springer, Heidelberg (2005)
17. MOSIS. Description of MOSIS library C5, http://www.mosis.com/vendors/view/on-semiconductor/c5
18. MOSIS. MOSIS library C5, http://cmosedu.com/jbaker/courses/ece5410/s09/C5_models.txt

19. Ahmad, N., Rezaul Hasan, S.: Low-power compact composite field AES S-Box/Inv S-Box design in 65nm CMOS using Novel XOR Gate. Integration, the VLSI Journal (2012)
20. Nikova, S., Rijmen, V., Schläffer, M.: Using Normal Bases for Compact Hardware Implementations of the AES S-Box. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 236–245. Springer, Heidelberg (2008)
21. Nogami, Y., Nekado, K., Toyota, T., Hongo, N., Morikawa, Y.: Mixed Bases for Efficient Inversion in $\mathbb{F}((2^2)^2)^2$ and Conversion Matrices of SubBytes of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 234–247. Springer, Heidelberg (2010)
22. Olofsson, M.: VLSI Aspects on Inversion in Finite Fields. Dissertation. Linköping University, Linköping, Sweden (2002)
23. Paar, C.: Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields. Dissertation, Institute for Experimental Mathematics. Universität Essen, Essen (June1994)
24. Preneel, B., Van Leekwijck, W., Van Linden, L., Govaerts, R., Vandewalle, J.: Propagation Characteristics of Boolean Functions. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 161–173. Springer, Heidelberg (1991)
25. Rijmen, V.: Efficient Implementation of the Rijndael S-Box
26. Rudra, A., Dubey, P.K., Jutla, C.S., Kumar, V., Rao, J.R., Rohatgi, P.: Efficient Rijndael Encryption Implementation with Composite Field Arithmetic. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 171–184. Springer, Heidelberg (2001)
27. Satoh, A., Morioka, S.: Unified Hardware Architecture for 128-Bit Block Ciphers AES and Camellia. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 304–318. Springer, Heidelberg (2003)
28. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A Compact Rijndael Hardware Architecture with S-Box Optimization. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 239–254. Springer, Heidelberg (2001)
29. Electric VLSI: Electric VLSI software, http://www.staticfreesoft.com/
30. Wolkerstorfer, J., Oswald, E., Lamberger, M.: An ASIC Implementation of the AES SBoxes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 67–78. Springer, Heidelberg (2002)
31. Wong, M., Wong, M., Nandi, A., Hijazin, I.: Composite field GF($((2^2)^2)^2$) Advanced Encryption Standard (AES) S-box with algebraic normal form representation in the subfield inversion. Circuits, Devices Systems, IET 5(6), 471–476 (2011)
32. Gao, X., Lu, E., Li, L., Lang, K.: LUT based FPGA implementation of SMS4,AES,Camellia. In: Fifth IEEE International Symposium on Embedded Computing, SEC 2008, pp. 73–76 (October 2008)
33. Yalla, P., Kaps, J.-P.: Compact FPGA implementation of Camellia. In: International Conference on Field Programmable Logic and Applications, FPL 2009, August 31-September 2, pp. 658–661 (2009)
34. Yasuyuki, N., Kenta, N., Tetsumi, T., Naoto, H., Yoshitaka, M.: Mixed Bases for Efficient inversion in F$((2^2)^2)^2$ and conversion Matrices of SubBytes of AES. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 94(6), 1318–1327 (2011)
35. Zhang, X., Parhi, K.: High-speed VLSI architectures for the AES algorithm. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 12(9), 957–967 (2004)
36. Zhang, X., Parhi, K.: On the Optimum Constructions of Composite Field for the AES Algorithm. IEEE Transactions on Circuits and Systems II: Express Briefs 53(10), 1153–1157 (2006)

A Number of Logical Gates for $\mathbf{a}_c$

This Appendix describes the way to compute the number of gates and critical path for each component with respect to $\mathbf{a}_c$ in Fig. 1. *Not considered* means that such component is not included in the critical path and *twice* means that the component appears twice in the critical path, ss_ν means square-and-scale with ν and s_τ means scale with τ.

For $GF((2^4)^2)$, Table 6 summarizes how many XOR/AND gates and critical path are needed for each component. Each sub-operation is given at $GF(2^4)$. Our implementation of Δ^{-1} from Wolkerstorfer et al.'s identities [30] utilize 18 X/10 A gates and 3 X/2 A in the critical path. Multiplication operation $MULT$ has the largest critical path, so such component is considered for obtaining the complete critical path of the system. For case 2, s_τ (scale with τ) is also considered in the critical path. For cases NP 12 and 13, the same sub-operations are applied. Nevertheless, only $MULT$ and ADD are considered in the critical path, as shown in (2-4).

For $GF(((2^2)^2)^2)$, Table 7 summarizes how many XOR/AND gates and critical path are needed for each component related to cases 3, 4, 5 , 6, 7, 10 and 11. $mult$, add and $scale$ are in $GF(2^2)$, $MULT$, ss_ν, ADD, Δ^{-1} and inv are in $GF((2^2)^2)$. $\mathbf{a}_c$ is in $GF(((2^2)^2)^2)$. For polynomial bases, Δ^{-1} is implemented with 15 X/8 A gates and 4 X/2 A in the critical path. For normal bases, Δ^{-1} is implemented with 12 X/8 A gates and 4 X/2 A in the critical path. Additionally in normal bases, $swap$ means to exchange the inputs, so it is a costless operation. Each operation can be deducted from (2-4) and (5-7).

Table 6. Number of gates and cp for each operation in $GF((2^4)^2)$

Cases	Operation	# Components	cp
1, 2, 12, 13	$MULT$	15 X/16 A	3 X/1 A
1, 12, 13	ss_ν	2 X/0 A	1 X/0 A (Not considered)
2	ss_ν	3 X/0 A	2 X/0 A (Not considered)
2	s_τ	1 X/0 A	1 X/0 A
1, 2	ADD	4 X/0 A	1 X/0 A (Twice)
12, 13	ADD	4 X/0 A	1 X/0 A
1, 2, 12, 13	Δ^{-1}	18 X/10 A	3 X/2 A
1	inv	43 X/26 A	8 X/3 A
2	inv	45 X/26 A	9 X/3 A
12, 13	inv	43 X/26 A	7 X/3 A
1	$\mathbf{a}_c$	73 X/58 A	11 X/4 A
2	$\mathbf{a}_c$	75 X/58 A	12 X/4 A
12, 13	$\mathbf{a}_c$	73 X/58 A	10 X/4 A

Table 7. Number of gates and cp for each operation in $GF(((2^2)^2)^2)$

Cases	Operation	#Components	cp
3, 4, 5, 10, 11	$mult$	3 X/4 A	2 X/1 A
3, 4, 5, 10, 11	$scale$	1 X/0 A	1 X/0 A (Not considered)
3, 4, 5, 10, 11	add	2 X/0 A	1 X/0 A (Twice)
3, 4, 5, 10, 11	$MULT$	19 X/16 A	4 X/1 A
3, 4, 10, 11	ss_ν	3 X/0 A	2 X/0 A (Not Considered)
5	ss_ν	4 X/0 A	2 X/0 A (Not Considered)
3, 4, 5	ADD	4 X/0 A	1 X/0 A (Twice)
10, 11	ADD	4 X/0 A	1 X/0 A
3	Δ^{-1}	15 X/8 A	4 X/2 A
4, 5, 10, 11	Δ^{-1}	13 X/8 A	4 X/2 A
3	inv	45 X/24 A	10 X/3 A
4	inv	43 X/24 A	10 X/3 A
5	inv	44 X/24 A	10 X/3 A
10, 11	inv	43 X/24 A	9 X/3 A
3	$\mathbf{a}_c$	83 X/56 A	14 X/4 A
4	$\mathbf{a}_c$	81 X/56 A	14 X/4 A
5	$\mathbf{a}_c$	82 X/56 A	14 X/4 A
10, 11	$\mathbf{a}_c$	81 X/56 A	13 X/4 A
6, 7	$mult$	4 X/3 A	2 X/1 A
6, 7	$scale$	1 X/0 A	1 X/0 A
6, 7	add	2 X/0 A	1 X/0 A (twice)
6, 7	$swap$	0 X/0 A	0 X/0 A
6, 7	ADD	4 X/0 A	1 X/0 A
6, 7	$MULT$	21 X/9 A	5 X/1 A
6, 7	ss_ν	4 X/0 A	2 X/0 A (Not considered)
6, 7	$1\Delta^{-1}$	12 X/8 A	3 X/2 A
6, 7	inv	45 X/17 A	9 X/3 A
6, 7	$\mathbf{a}_c$	87 X/35 A	14 X/4 A

Table 8. Number of gates and cp, cases 8 and 9

Cases	Operation	#Gates	cp
8, 9	$mult$	3 X/4 A	2 X/1 A
8, 9	$scale$	1 X/0 A	1 X/0 A
8, 9	add	2 X/0 A	1 X/0 A (Twice)
8, 9	$MULT$	18 X/12 A	5 X/1 A
8, 9	ss_ν	3 X/0 A	2 X/0 A (Not considered)
8, 9	ADD	4 X/0 A	1 X/0 A (Twice)
8, 9	Δ^{-1}	12 X/8 A	3 X/2 A
8, 9	inv	41 X/20 A	10 A/3 A
8, 9	$\mathbf{a}_c$	77 X/44 A	15 X/4 A

For cases 8 and 9, (2-4) are considered, but normal operation properties are applied in the second level and polynomial operations are applied at the lowest level. The corresponding results are summarized in Table 8.

B Number of Gates for Δ^{-1} in $GF(((2^2)^2)^2)$ Normal Bases, with $T = 3, N = 2$

The original equations are ([31])

$$x_0^{-1} = x_2 + x_0x_3 + x_1x_3 + x_0x_2 + x_1x_2x_3 \tag{25}$$
$$x_1^{-1} = x_2 + x_3 + x_0x_3 + x_1x_3 + x_0x_2x_3 \tag{26}$$
$$x_2^{-1} = x_0 + x_1x_2 + x_1x_3 + x_0x_2 + x_0x_1x_3 \tag{27}$$
$$x_3^{-1} = x_0 + x_1 + x_1x_2 + x_1x_3 + x_0x_1x_2 \tag{28}$$

Each term is considered as a single element. For each bit product, 8 AND gates are obtained. The critical path for AND is equal to 2, because of the ternary terms. Equations (25-28) can be written as

$$x_a = (x_2 + (x_0x_3 + x_1x_3)) \tag{29}$$
$$x_b = (x_0 + (x_1x_2 + x_1x_3)) \tag{30}$$
$$x_0^{-1} = x_a + (x_0x_2 + x_1x_2x_3) \tag{31}$$
$$x_1^{-1} = x_a + (x_3 + x_0x_2x_3) \tag{32}$$
$$x_2^{-1} = x_b + (x_0x_2 + x_0x_1x_3) \tag{33}$$
$$x_3^{-1} = x_b + (x_1 + x_0x_1x_2) \tag{34}$$

Counting the number of $+$ in (29-34), 12 XOR gates are obtained. Each output contains the same critical path, which is equal to 3 XOR gates.

C LDG Applied to F

Case 6: Normal $GF((2^2)^2)^2,\ \beta = 60,\ \tau = 15,\ \nu = 14,\ T = 3,\ N = 2$

$$\mathbf{F} = \begin{pmatrix} 1&1&0&1&0&0&0&1 \\ 0&0&0&1&0&1&1&0 \\ 1&0&0&0&1&1&1&0 \\ 0&1&1&0&0&0&1&0 \\ 1&0&1&0&0&0&1&0 \\ 0&0&1&0&0&1&0&0 \\ 0&0&0&0&1&0&1&0 \\ 1&0&0&0&0&1&0&1 \end{pmatrix} \qquad \mathbf{M} = \begin{pmatrix} 1&0&0&1&1&0&0&0 \\ 1&0&0&1&0&1&1&1 \\ 1&1&0&0&1&0&0&1 \\ 1&1&1&0&0&1&0&1 \\ 1&1&1&0&0&0&0&1 \\ 1&1&1&0&1&0&0&1 \\ 1&0&1&0&0&1&0&0 \\ 1&0&0&1&1&1&0&1 \end{pmatrix} \tag{35}$$

$$t_8 = x_2 + x_5 = y_5 \tag{36}$$
$$t_9 = x_4 + x_6 = y_6 \tag{37}$$
$$t_{10} = x_0 + x_5 \tag{38}$$
$$t_{11} = x_2 + x_6 \tag{39}$$
$$t_{12} = x_0 + x_1 \tag{40}$$
$$t_{13} = x_3 + x_5 \tag{41}$$
$$t_{14} = x_3 + x_7 \tag{42}$$
$$t_{15} = x_0 + t_{11} = (x_0 + (x_2 + x_6)) = y_4 \tag{43}$$
$$t_{16} = x_1 + t_{11} = (x_1 + (x_2 + x_6)) = y_3 \tag{44}$$
$$t_{17} = x_6 + t_{13} = (x_6 + (x_3 + x_5)) = y_1 \tag{45}$$
$$t_{18} = x_7 + t_{10} = (x_7 + (x_0 + x_5)) = y_7 \tag{46}$$
$$t_{19} = t_9 + t_{10} = (x_4 + x_6) + (x_0 + x_5) = y_2 \tag{47}$$
$$t_{20} = t_{12} + t_{14} = (x_0 + x_1) + (x_3 + x_7) = y_0 \tag{48}$$

$cp(\mathbf{F}) = \lceil log_2(4) \rceil = 2$, number of XOR gates for $\mathbf{F}$ is equal to 13.

Author Index