Dipanwita Roy Chowdhury
Vincent Rijmen
Abhijit Das (Eds.)

LNCS 5365

Progress in Cryptology – INDOCRYPT 2008

9th International Conference on Cryptology in India
Kharagpur, India, December 2008
Proceedings

Springer

Lecture Notes in Computer Science 5365

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

Dipanwita Roy Chowdhury Vincent Rijmen
Abhijit Das (Eds.)

Progress in Cryptology – INDOCRYPT 2008

9th International Conference on Cryptology in India
Kharagpur, India, December 14-17, 2008
Proceedings

Springer

Volume Editors

Dipanwita Roy Chowdhury
Abhijit Das
Dept. of Computer Science and Engineering
Indian Institute of Technology, Kharagpur 721 302, India,
E-mail: {drc, abhij@cse.iitkgp.ernet.in}

Vincent Rijmen
K.U. Leuven, ESAT/COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
E-mail: Vincent.Rijmen@esat.kuleuven.be

Library of Congress Control Number: 2008940079

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

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-540-89753-4 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-89753-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12572953 06/3180 5 4 3 2 1 0

Message from the General Chairs

The 2008 International Conference on Cryptology in India (INDOCRYPT 2008) was the ninth event in this series. It was organized by the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, in co-operation with the Cryptology Research Society of India (CRSI). Over the years, INDOCRYPT has become a leading forum for disseminating the latest research results in cryptology. This year's conference brought together leading and eminent researchers worldwide in Kharagpur (India), during December 14–17, 2008, to present and discuss a wide variety of aspects on cryptology and security.

The program of the conference spanned over four days and included, in addition to a high-quality technical program, two tutorials delivered by the very best in the field, giving young researchers and students an excellent opportunity to learn about the latest trends in cryptography and cryptanalysis.

A conference of this magnitude would not have been possible without the hard and excellent work of all the members of the Organizing Committee. Our special thanks are due to Dipanwita Roy Chowdhury and Vincent Rijmen (Program Co-chairs) for coordinating and leading the effort of the Program Committee, culminating in an excellent technical program. We are grateful to the Tutorial Chair, Debdeep Mukhopadhyay, for arranging two high-quality tutorial talks by eminent leaders in the field.

We are indebted to all other members of the Organizing Committee for their excellent work. Dilip Kumar Nanda (Organizing Chair) along with his team coordinated all the local arrangements with elan. Abhijit Das (Publication Chair) managed the publication of the conference proceedings through his tireless efforts. We also take this opportunity to acknowledge the contributions of the Publicity Chair (Soumen Maity) and of the Finance Chair (Raja Datta) to the success of the conference. No amount of thanks is sufficient for the omnipresent team of enthusiastic volunteers who did their best for the smooth sailing of the conference.

Last but not the least, we extend our heartfelt thanks to the authors, the reviewers, the participants, and the sponsors of the conference, for their vital contributions to the success of the event.

December 2008

Indranil Sen Gupta
Bimal K. Roy

Message from the Technical Program Chairs

Welcome to the Proceedings of the 9th International Conference on Cryptology, INDOCRYPT 2008. This annual event started off eight years ago in the year 2000 by the Cryptology Research Society of India and has gradually matured into one of the topmost international cryptology conferences.

This year we received 111 papers from all over the world. After a rigorous review process, the Program Committee selected 33 papers out of the 111 submissions. Most of the papers received at least three independent reviews made by the Program Committee members and also by additional external experts. The papers along with the reviews were scrutinized by the Program Committee members during a two-week discussion phase. We would like to thank the authors of all the papers for submitting their quality research work to the conference. Special thanks go to the Program Committee members and the external reviewers who gave their precious time in reviewing and selecting the best set of papers.

We are fortunate to have several eminent researchers as keynote and invited speakers. The main conference program was preceded by a day of tutorial presentations. We would like to thank Debdeep Mukhopadhyay, the Tutorial Chair, for his active initiation and enthusiasm to make the tutorial sessions a success. We would like to express our thanks to Abhijit Das, the Publication Chair, who gave his precious time to compile the conference proceedings. Further, we thank Anirban Sarkar, who helped with the setting up and maintenance of the conference Web server.

We hope that you will find the INDOCRYPT 2008 proceedings technically rewarding.

December 2008

Dipanwita Roy Chowdhury
Vincent Rijmen

Organization

General Chairs

Indranil Sen Gupta	Indian Institute of Technology, Kharagpur, India
Bimal K. Roy	Indian Statistical Institute, Kolkata, India

Program Chairs

Dipanwita Roy Chowdhury	Indian Institute of Technology, Kharagpur, India
Vincent Rijmen	KU Leuven, Belgium and Graz University of Technology, Austria

Tutorial Chair

Debdeep Mukhopadhyay	Indian Institute of Technology, Kharagpur, India

Publication Chair

Abhijit Das	Indian Institute of Technology, Kharagpur, India

Organizing Chair

Dilip K. Nanda	Indian Institute of Technology, Kharagpur, India

Publicity Chair

Soumen Maity	Indian Institute of Technology, Kharagpur, India

Finance Chair

Raja Datta	Indian Institute of Technology, Kharagpur, India

Program Committee

Abhijit Das	IIT Kharagpur, India
Alex Biryukov	Univ. du Luxembourg, Luxembourg
Alfred Menezes	University of Waterloo, Canada
Anne Canteaut	INRIA, France
Arjen K. Lenstra	EPFL, Switzerland and Alcatel-Lucent Bell Laboratories, USA
Bimal K. Roy	ISI Kolkata, India

C. Pandu Rangan	IIT Madras, India
C.E. Veni Madhavan	IISC Bangalore, India
Çetin Kaya Koç	Oregon State University, USA
Chandan Mazumdar	Jadavpur University, Kolkata, India
Charanjit S. Jutla	IBM T.J. Watson Research Center, USA
Christian Rechberger	Graz University of Technology, Austria
Dan Page	University of Bristol, UK
Debdeep Mukhopadhyay	IIT Kharagpur, India
Dipanwita Roy Chowdhury	IIT Kharagpur, India
Helger Lipmaa	Cybernetica AS, Estonia
Indranil Sen Gupta	IIT Kharagpur, India
Ingrid Verbauwhede	ESAT, KU Leuven, Belgium
Jennifer Seberry	University of Wollongong, Australia
Joan Daemen	ST Microelectronics, Belgium
Josef Pieprzyk	Macquarie University, Australia
Jovan Golic	Security Innovation, Telecom Italia, Turin, Italy
Keith Martin	University of London, UK
Kolin Paul	IIT Delhi, India
Matt Robshaw	Orange Labs, France
Matthew Parker	University of Bergen, Norway
Paulo Barreto	University of Sao Paulo, Brazil
Pramod K. Saxena	SAG, New Delhi, India
R. Balasubramanium	IMSc, Chennai, India
Ramarathnam Venkatesan	Microsoft, Redmond, USA
Rei Safavi-Naini	University of Wollongong, Australia
Sanjay Barman	CAIR, Bangalore, India
Shiho Moriai	Sony Computer Entertainment Inc., Japan
Soumen Maity	IIT Kharagpur, India
Subhamoy Maitra	ISI Kolkata, India
Svetla Nikova	KU Leuven, Belgium
Tanja Lange	Technische Universiteit Eindhoven, The Netherlands
Tor Helleseth	University of Bergen, Norway
Vincent Rijmen	KU Leuven, Belgium and Graz University of Technology, Austria
Willi Meier	FHNW, Switzerland

Additional Referees

Andrey Bogdanov	Benoit Libert	Goutam Paul
Angela Piper	Berry Schoenmakers	Håvard Raddum
Arpita Patra	Christophe Clavier	Jaydeb Bhowmik
Arun K. Majumdar	Christian Kraetzer	Jeff Hoffstein
Ashish Choudhary	Florian Mendel	Jean-Philippe Aumasson
Avishek Adhikari	Geong Sen Poh	Joonsang Baek

Juraj Sarinay
Kanta Matsuura
Kazue Sako
Kenny Paterson
Kristian Gjøsteen
Lejla Batina
Mahabir Prasad Jhanwar
Marc Stevens
Matrin Gagné
Martin Schläffer
Maura Paterson
Miroslav Knežević
Mridul Nandy
Michael Naehrig
Nathan Keller
Nele Mentens
Nicolas Sendrier
Nicolas Gama
Nick Howgrave-Graham
Noboru Kunihiro
Onur Ozen
Pim Tuyls
Sanjit Chatterjee
Safuat Hamdy
Sébastien Canard
Sebastiaan Faust
Somitra Kumar Sanadhya
Sourav Mukhopadhyay
Shahram Khazaei
Takashi Satoh
Thomas Popp
Tomislav Nad
Tomoyuki Asano
Toshihiro Ohigashi
Tor Erling Bjørstad
Vipul Goyal
Yannick Seurin
Yong Ki Lee
Yunlei Zhao

Table of Contents

Stream Ciphers

Cryptographic Hash Functions

Public-Key Cryptography – I

Security Protocols

Hardware Attacks

Block Ciphers

Public-Key Cryptography – II

Cryptographic Hardware

Elliptic Curve Cryptography

Threshold Cryptography

Slid Pairs in Salsa20 and Trivium

Deike Priemuth-Schmid and Alex Biryukov

FSTC, University of Luxembourg
6, rue Richard Coudenhove-Kalergi,
L-1359 Luxembourg
(deike.priemuth-schmid,alex.biryukov)@uni.lu

Abstract. The stream ciphers Salsa20 and Trivium are two of the finalists of the eSTREAM project which are in the final portfolio of new promising stream ciphers. In this paper we show that initialization and key-stream generation of these ciphers is *slidable*, i.e. one can find distinct (Key, IV) pairs that produce identical (or closely related) key-streams. There are 2^{256} and more then 2^{39} such pairs in Salsa20 and Trivium respectively. We write out and solve the non-linear equations which describe such related (Key, IV) pairs. This allows us to sample the space of such related pairs efficiently as well as detect such pairs in large portions of key-stream very efficiently. We show that Salsa20 does not have 256-bit security if one considers general birthday and related key distinguishing and key-recovery attacks.

Keywords: Salsa20, Trivium, eSTREAM, stream ciphers, cryptanalysis.

1 Introduction

In 2005 Bernstein [2] submitted the stream cipher Salsa20 to the eSTREAM-project [5]. Original Salsa20 has 20 rounds, later 8 and 12 rounds versions were also proposed. The cipher Salsa20 uses the hash function Salsa20 in a counter mode. Its 512-bit state is initialized by copying into it 128 or 256-bit key, 64-bit nonce and counter and 128-bit constant. Previous attacks on Salsa used differential cryptanalysis exploiting a truncated differential over three or four rounds. The first attack was presented by Crowley [4] which could break the 5 round version of Salsa20 within claimed 3^{165} trials. Later a four round differential was exploited by Fischer et al. [6] to break 6 rounds in 2^{177} trials and by Tsnunoo et al. [12] to break 7 rounds in about 2^{190} trials. The currently best attack by Aumasson et al. [1] covers 8 round version of Salsa20 with estimated complexity of 2^{251}.

The stream cipher Trivium was submitted by De Cannière and Preneel [3] in 2005 to the eSTREAM-project [5]. Trivium has an internal state of 288 bits and uses an 80-bit key and an 80-bit initial value (IV). The interesting part of Trivium is the nonlinear update function of degree 2. In [10] Raddum presented and attacked simplified versions of Trivium called Bivium but the attack on Trivium had a complexity higher than the exhaustive key search. Bivium was

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 1–14, 2008.

completely broken by Maximov and Biryukov [8] and an attack on Trivium with complexity about 2^{100} was presented which showed that key-size of Trivium can not be increased just by loading longer keys into the state. In [9] McDonald et al. attacked Bivium using SatSolvers. Another approach that gained attention recently is to reduce the key setup of Trivium as done by Turan and Kara [13] and Vielhaber [14]. So far no attack faster than exhaustive key search was shown for Trivium.

In this paper we start with our investigation of Salsa20 followed by a description of the attacks. We show that the following observation holds: suppose that you are given two black boxes, one with Salsa20 and one with a random mapping. The attacker is allowed to chose a relation $\boldsymbol{\mathcal{F}}$ for a pair of inputs, after which a secret initial input x is chosen and a pair $(x, \boldsymbol{\mathcal{F}}(x))$ is encrypted either by Salsa20 or by a random mapping. We stress that only the relation $\boldsymbol{\mathcal{F}}$ is known to the attacker. The goal of the attacker is given a pair of ciphertexts to tell whether they were encrypted by Salsa20 or by a random mapping. To make the life of the attacker more difficult the pair may be hidden in a large collection of other ciphertexts. It is clear that for a truly random mapping no useful relation $\boldsymbol{\mathcal{F}}$ would exist and moreover there is no way of checking a large list except for checking all the pairs or doing a birthday attack. On the other hand Salsa20 can be easily distinguished from random in both scenarios if $\boldsymbol{\mathcal{F}}$ is a carefully selected function related to the round-structure of Salsa20. Moreover it is not only a distinguishing but also a complete key-recovery attack via discovering the initial state. Our attacks are independent of the number of rounds in Salsa and thus work for all the 3 versions of Salsa. We also show a general birthday attack on 256-bit key Salsa20 with complexity 2^{192} which can be further sped up twice using sliding observations.

In the second part of this paper we describe our results about Trivium which show a large related key-class (2^{39} out of 2^{80} keys) which produce identical keystreams up to a shift. We solve the resulting non-linear sliding equations using Magma and present several examples of such slid key-IV pairs. The interesting observation is that for a shift of 111 clocks 24-key-bits do not appear in these equations and thus for a fixed IV there is a 2^{24} freedom of choice for the key that may have a sliding property.

2 Slid Pairs in Salsa20

2.1 Brief Description of Salsa20

The Salsa20 encryption function uses the Salsa20 hash function in a counter mode. The internal state of Salsa20 is a 4×4 - matrix of 32-bit words. A vector (y_0, y_1, y_2, y_3) of four words is transformed into (z_0, z_1, z_2, z_3) by calculating[1]

$$z_1 = y_1 \oplus ((y_0 + y_3) \lll 7) \qquad z_3 = y_3 \oplus ((z_2 + z_1) \lll 13)$$
$$z_2 = y_2 \oplus ((z_1 + y_0) \lll 9) \qquad z_0 = y_0 \oplus ((z_3 + z_2) \lll 18) \ .$$

[1] In the complete Salsa20 section the symbol "+" denotes the addition modulo 2^{32}, the other two symbols work at the level of the bits with "$\oplus$" as XOR-addition and "$\lll$" as a shift of bits.

This nonlinear operation called **quarterround** is the basic part of the **columnround** where it is applied to columns as well as of the **rowround** to transform rows. A so called **doubleround** consists of a *columnround* followed by a *rowround*. The *doubleround* function of Salsa20 is repeated 10 times. If Y denotes the matrix a key-stream block is defined by

$$Z = Y + \text{doubleround}^{10}(Y) \ .$$

One *columnround* as well as one *rowround* has 4 *quarterrounds* which means 48 word operations in total. Thus the 10 *doublerounds* of Salsa20 give 960 word operations and result with the 16 word operations from the feedforward in 976 word operations in total for one encryption.

The cipher takes as input a 256-bit key $(k_0, \ldots, k_7)$, a 64-bit nonce (n_0, n_1) and a 64-bit counter (c_0, c_1). A 128-bit key version of Salsa20 copies the 128-bit key twice. In this paper we mainly concentrate on the 256-bit key version. The remaining four words are set to fixed publicly known constants, denoted with $\sigma_0, \sigma_1, \sigma_2$ and σ_3.

2.2 Slid Pairs

The structure of a *doubleround* can be rewritten as *columnround* then a matrix transposition another *columnround* followed by a second transposition. We define $\boldsymbol{\mathcal{F}}$ to be a function which consists of a *columnround* followed by a transposition. Now the 10 *doublerounds* can be transferred into 20 times function $\boldsymbol{\mathcal{F}}$. If we have 2 triples (key1, nonce1, counter1) and (key2, nonce2, counter2) so that

$$\begin{aligned} &\boldsymbol{\mathcal{F}}\,[\,1^{\text{st}} \text{ starting state (key1, nonce1, counter1)}] \\ &= 2^{\text{nd}} \text{ starting state (key2, nonce2, counter2)} \end{aligned}$$

then this property holds for each point during the round computation and especially its end. Pay attention that the feedforward at the end of Salsa20 destroys this property. We call such a pair of a 1^{st} and 2^{nd} starting state a slid pair and show their relation in Fig. 1.

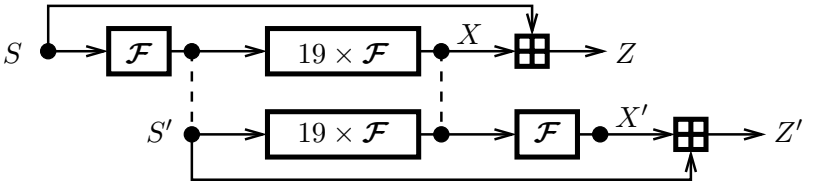


Fig. 1. Relation of a slid pair

In a starting state four words are constants and 12 words can be chosen freely which leads to a total amount of 2^{384} possible starting states. If we want that a starting state after applying function $\boldsymbol{\mathcal{F}}$ results in a 2^{nd} starting state we

obtain four wordwise equations. This means we can choose eight words of the 1st starting state freely whereas the other four words are determined by the equations as well as the words for the 2nd starting state. This leads to a total amount of 2^{256} possible slid pairs.

For the 128-bit key version no such slid pair exists due to the additional constrains of four fewer words freedom in the 1st starting state and four more wordwise equations in the 2nd starting state.

With function $\boldsymbol{\mathcal{F}}$ we get two equations $S' = \boldsymbol{\mathcal{F}}(S)$ and $X' = \boldsymbol{\mathcal{F}}(X)$. The words for these matrices we denote as

$$S = \begin{pmatrix} \sigma_0 & k_0 & k_1 & k_2 \\ k_3 & \sigma_1 & n_0 & n_1 \\ c_0 & c_1 & \sigma_2 & k_4 \\ k_5 & k_6 & k_7 & \sigma_3 \end{pmatrix} \quad X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} \quad Z = \begin{pmatrix} z_0 & z_1 & z_2 & z_3 \\ z_4 & z_5 & z_6 & z_7 \\ z_8 & z_9 & z_{10} & z_{11} \\ z_{12} & z_{13} & z_{14} & z_{15} \end{pmatrix}$$

$$S' = \begin{pmatrix} \sigma_0 & k'_0 & k'_1 & k'_2 \\ k'_3 & \sigma_1 & n'_0 & n'_1 \\ c'_0 & c'_1 & \sigma_2 & k'_4 \\ k'_5 & k'_6 & k'_7 & \sigma_3 \end{pmatrix} \quad X' = \begin{pmatrix} x'_0 & x'_1 & x'_2 & x'_3 \\ x'_4 & x'_5 & x'_6 & x'_7 \\ x'_8 & x'_9 & x'_{10} & x'_{11} \\ x'_{12} & x'_{13} & x'_{14} & x'_{15} \end{pmatrix} \quad Z' = \begin{pmatrix} z'_0 & z'_1 & z'_2 & z'_3 \\ z'_4 & z'_5 & z'_6 & z'_7 \\ z'_8 & z'_9 & z'_{10} & z'_{11} \\ z'_{12} & z'_{13} & z'_{14} & z'_{15} \end{pmatrix}.$$

The set up of the system of equations for a whole Salsa20 computation is too complicated but the equations for the computation of $\boldsymbol{\mathcal{F}}$ are very clear. For a complete description of the equations see the appendix A. The structure of both systems of equations coming from the relation $\boldsymbol{\mathcal{F}}$ is the same especially all the known variables are at the same place. Due to the eight words freedom we have in a 1st or 2nd starting state there are some relations in the 12 non-fixed words. For the 2nd starting state these relations are very clear as they deal only with words

$$0 = k'_2 + k'_1, \quad 0 = k'_3 + n'_1, \quad 0 = c'_1 + c'_0 \quad \text{and} \quad 0 = k'_7 + k'_6 \ , \tag{1}$$

whereas for the 1st starting state these relations depend on the bits and thus are more complicated. Sliding by the function $\boldsymbol{\mathcal{F}}$ is applicable to any version of Salsa20/r where r is even. For r odd there would be no transposition at the end of the round computation, equations are a bit different, though still solvable.

2.3 Sliding State Recovery Attack on the Davies-Meyer Mode

In this subsection we consider a general state-recovery slide attack on a Davies-Meyer construction. We demonstrate it on an example of Davies-Meyer feedforward used with the iterative permutation from Salsa20. The feedforward breaks the sliding property and makes slide attack more complicated to mount. We consider the following scenario:

1. The oracle chooses a secret 512-bit state S (here we assume that there is no restriction of 128-bit diagonal constants and the full 512 bits can be chosen at random).
2. The oracle computes $\boldsymbol{\mathcal{F}}(S) = S'$.

3. The oracle computes Salsa20(S), Salsa20(S') and gives them to the attacker.
4. The goal of the attacker is to recover the secret state S.

Due to the weak diffusion of $\boldsymbol{\mathcal{F}}$ the attacker can write separate systems of equations for each column of S. If one combines for one column the *quarterround* coming from $S' = \boldsymbol{\mathcal{F}}(S)$ the corresponding *quarterround* from $X' = \boldsymbol{\mathcal{F}}(X)$ and the feedforward one gets a system with 16 equations shown below. We assume all 16 variables are unknown.

$$\begin{aligned}
s'_1 &= s_4 \oplus ((s_0 + s_{12}) \lll 7) & x'_1 &= x_4 \oplus ((x_0 + x_{12}) \lll 7)\\
s'_2 &= s_8 \oplus ((s'_1 + s_0) \lll 9) & x'_2 &= x_8 \oplus ((x'_1 + x_0) \lll 9)\\
s'_3 &= s_{12} \oplus ((s'_2 + s'_1) \lll 13) & x'_3 &= x_{12} \oplus ((x'_2 + x'_1) \lll 13)\\
s'_0 &= s_0 \oplus ((s'_3 + s'_2) \lll 18) & x'_0 &= x_0 \oplus ((x'_3 + x'_2) \lll 18)\\
z_0 &= x_0 + s_0 \quad z_8 = x_8 + s_8 & z'_0 &= x'_0 + s'_0 \quad z'_2 = x'_2 + s'_2\\
z_4 &= x_4 + s_4 \quad z_{12} = x_{12} + s_{12} & z'_1 &= x'_1 + s'_1 \quad z'_3 = x'_3 + s'_3
\end{aligned}$$

This system can be reduced to four equations. In the first equation two variables must be guessed to solve it. In the remaining three equations always two variables are known either the guessed s-variable or the calculated s'-variable. Thus they can be solved without guessing any more variables. Depending on which variables are guessed or known some of the equations can be used to check the guess.

$$\begin{aligned}
z'_1 &= \big[(z_4 - s_4) \oplus \big([(z_0 - s_0) + (z_{12} - s_{12})] \lll 7\big)\big] + \big[s_4 \oplus ((s_0 + s_{12}) \lll 7)\big]\\
z'_2 &= \big[(z_8 - s_8) \oplus \big([(z'_1 - s'_1) + (z_0 - s_0)] \lll 9\big)\big] + \big[s_8 \oplus ((s'_1 + s_0) \lll 9)\big]\\
z'_3 &= \big[(z_{12} - s_{12}) \oplus \big([(z'_2 - s'_2) + (z'_1 - s'_1)] \lll 13\big)\big] + \big[s_{12} \oplus ((s'_2 + s'_1) \lll 13)\big]\\
z'_0 &= \big[(z_0 - s_0) \oplus \big([(z'_3 - s'_3) + (z'_2 - s'_2)] \lll 18\big)\big] + \big[s_0 \oplus ((s'_3 + s'_2) \lll 18)\big]
\end{aligned}$$

Therefore the system of equations for one column with complete unknown variables can be solved by guessing only two variables. With the four guesses of 2^{64} steps each the attacker can completely recover the 512-bit secret state S. This shows that Salsa20 without the diagonal constants is easily distinguishable from a random function, for which a similar task would require about 2^{511} steps.

The addition of the diagonal constants reduces the flexibility of the oracle in a choice of the initial states to 2^{256} but the attack works even better:

1. The oracle chooses a starting state S' with key k', nonce n' and counter c' satisfying equations (1). The attacker does not know this state.
2. The oracle applies $\boldsymbol{\mathcal{F}}^{-1}(S')$ to compute the related key k, nonce n and counter c.
3. The oracle computes Salsa20(S), Salsa20(S') and gives them to the attacker.
4. The goal of the attacker is to recover the secret state S.

The knowledge of the diagonal makes the previous attack even faster and allows the full 384-bit (256-bit entropy) state recovery with complexity of $4 \cdot 2^{32}$ because the known words appear at the same place in the system of equations for the columns. If in a system of equations for one column two known variables appear at different places this system is solvable immediately.

If the attacker chooses the nonce and the counter n', c' (160-bits of entropy) then the complexity drops to $2 \cdot 2^{32}$. Furthermore if nonce and counter n, c are known (128-bits of entropy left). The state can be recovered immediately (with

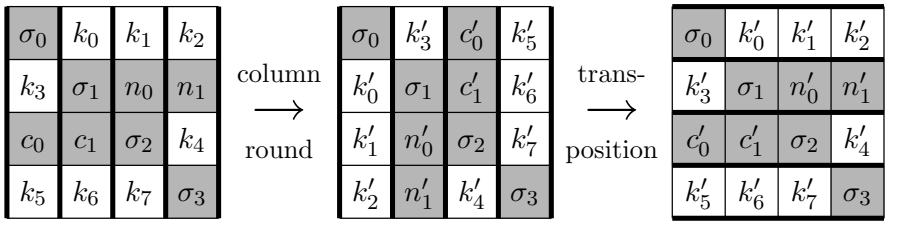


Fig. 2. Relation of the 1st and 2nd starting state

or without knowing counter c' and nonce n'). Figure 2 shows the relation for the starting states with the known words indicated as grey squares.

Due to the fact that we are able to recover the full internal state this attack also works as a related key key-recovery attack on Salsa20 because the key is loaded directly into the internal state. A detailed description how to recover both keys for a slid pair is given in the extended version [11]. Table 1 shows the time complexities for the described attacks, memory complexity is negligible.

Table 1. Time complexities for state-recovery attacks

known words of the starting states	sliding on Salsa20	random oracle
nothing	2^{66}	2^{511}
only diagonal	2^{34}	2^{255}
diagonal, nonce and counter n', c'	2^{33}	2^{159}
diagonal, nonce and counter n, c	$O(1)$	2^{127}
diagonal, nonce and counter n, c and n', c'	$O(1)$	2^{63}

2.4 A Generalized Related Key Attack on Salsa20

Suppose we are given a (possibly large) list of ciphertexts with the corresponding nonces and counters and we are told that in this list the slid pair is hidden. The question is, can we find slid pairs in a large list of ciphertexts efficiently? As we saw in the previous section, given such slid ciphertext pair it is easy to compute both keys. The task is made more difficult by the feedforward of Salsa20, which destroys the sliding relationship. Nevertheless in this section we show that given a list of ciphertexts of size $O(2^l)$ it is possible to detect a slid pair with memory and time complexity of just $O(2^l)$[2]. The naive approach which would require to check for each possible pair the equations from function $\mathcal{F}$ will have complexity $O(2^{2l})$ which is too expensive. Our idea is to reduce the amount of potential pairs by sorting them by eight precomputed words, so that only elements where these eight words match have the possibility to yield a slid pair. After decreasing the number of possible pairs in that way we can check the remaining pairs using additional constraints coming from the sliding equations.

[2] Sorting is done via Bucket sort so we save the logarithmic factor l in complexity.

For the sorting we use Bucket sort because each word has only 2^{32} possibilities. The number of words we sort by is equal to the number of runs of Bucket sort.

We have a set M of ciphertexts with corresponding nonces and counters. Each ciphertext can be either a 1st or a 2nd starting state to regard this the set is stored twice first under M_1 to check for possible 1st starting states and second under M_2 to check for possible 2nd starting states.

Step 1: **Sort the first list**
For each element in set M_1 undo the feedforward for the four words on the diagonal and $x_9 = z_9 - c_1$. Then sort M_1 by the specified eight words $x_0, x_5, x_{10}, x_{15}, x_9$ and c_1, z_1, z_{13}.

Step 2: **Sort the second list**[3]
Select only elements of M_2 that satisfy equation $0 = c'_0 + c'_1$ since only such an entry can be a 2nd starting state. For each element undo the feedforward for the four words on the diagonal and $x'_6, \ldots, x'_9$ because nonces and counters are known. Then compute for each element the words marked in bold in the equations

$$
\begin{aligned}
\boldsymbol{x_0} &= x'_0 \oplus ((z'_2 + z'_3) \lll 18) & \qquad \boldsymbol{k'_3} &= -n'_1 \\
\boldsymbol{x_5} &= x'_5 \oplus ((z'_4 + n'_1 + x'_7) \lll 18) & \qquad \boldsymbol{x_{10}} &= x'_{10} \oplus ((x'_9 + x'_8) \lll 18) \\
\boldsymbol{x_{15}} &= x'_{15} \oplus ((z'_{13} + z'_{14}) \lll 18) & \qquad \boldsymbol{x_1} &= (z'_4 + n'_1) \oplus ((x'_7 + x'_6) \lll 13) \\
\boldsymbol{x_9} &= x'_6 \oplus ((x_5 + x_1) \lll 7) & \qquad \boldsymbol{k_0} &= k'_3 \oplus ((n'_1 + n'_0) \lll 13) \\
\boldsymbol{c_1} &= n'_0 \oplus ((\sigma_1 + k_0) \lll 7) & \qquad \boldsymbol{z_1} &= x_1 + k_0 \\
\boldsymbol{k_6} &= n'_1 \oplus ((n'_0 + \sigma_1) \lll 9) & \qquad \boldsymbol{z_{13}} &= (k_6 + x'_7) \oplus ((x'_6 + x_5) \lll 9) \ .
\end{aligned}
$$

During this computation we calculate three key words k_0, k_6 and k'_3. Sort the set M_2 by the calculated eight words $x_0, x_5, x_{10}, x_{15}, x_9$ and c_1, z_1, z_{13} for the potential 1st starting states.

Step 3: **Check each possible pair**
Cross check all the possible pairs that match in the eight words and thus satisfy the 256-bit filtering. For the conforming pairs we can continue the check, using the following equations. If a test condition is wrong this pair can not be a slid pair. For each pair undo for the ciphertext of the 1st starting state the feedforward for the word $x_6 = z_6 - n_0$. Then compute the bold variables and check the three conditions below

$$
\begin{aligned}
\text{compute} \quad & \boldsymbol{k'_4} = -c'_0 + ((n_0 \oplus c'_1) \ggg 13) & \qquad \boldsymbol{x'_{11}} &= -x'_8 + ((x_6 \oplus x'_9) \ggg 13) \\
& \boldsymbol{k_1} = c'_0 \oplus ((k'_4 + \sigma_2) \lll 9) & \qquad \boldsymbol{x_2} &= x'_8 \oplus ((x'_{11} + x_{10}) \lll 9) \\
& \boldsymbol{k_7} = k'_4 \oplus ((\sigma_2 + n_0) \lll 7) & \qquad \boldsymbol{x_{14}} &= x'_{11} \oplus ((x_{10} + x_6) \lll 7) \\
\text{check} \quad & z'_{11} = x'_{11} + k'_4 \qquad z_2 = x_2 + k_1 \qquad z_{14} = x_{14} + k_7 \ .
\end{aligned}
$$

During this computation we calculate the three key words k_1, k_7 and k'_4.

[3] If the number of rounds of Salsa is odd then such simple sorting would not be possible, since Salsa equations are easier to solve in reverse direction. In our approach we know 2 words at the input and 3 words at the output of the *columnround* which is easier to solve than the opposite (3 words at the input vs. 2 at the output). Nevertheless the system is still solvable.

For the rest of the pairs we have two similar systems of equations to check. We first solve the following equations

$$z'_2 = [(z_8 - c_0) \oplus ((z'_1 - \boldsymbol{k'_0} + x_0) \lll 9)] + [c_0 \oplus ((\boldsymbol{k'_0} + \sigma_0) \lll 9)]$$
$$z'_{13} = [(z_7 - n_1) \oplus ((z'_{12} - \boldsymbol{k'_5} + x_{15}) \lll 9)] + [n_1 \oplus ((\boldsymbol{k'_5} + \sigma_3) \lll 9)]$$

and if there is no solution for k'_0 or k'_5 this pair can not be a slid pair. Otherwise we use k'_0 as well as k'_5 to compute four more key words while we check two more conditions in each system

with k'_0	$\boldsymbol{k'_1} = c_0 \oplus ((k'_0 + \sigma_0) \lll 9)$	$\boldsymbol{k'_2} = -k'_1$
	$\boldsymbol{k_5} = k'_2 \oplus ((k'_1 + k'_0) \lll 13)$	$\boldsymbol{x'_1} = z'_1 - k'_0$
	$\boldsymbol{x_{12}} = (z'_3 - k'_2) \oplus ((z'_2 - k'_1 + x'_1) \lll 13)$	
	$\boldsymbol{k_3} = k'_0 \oplus ((\sigma_0 + k_5) \lll 7)$	$\boldsymbol{x_4} = x'_1 \oplus ((x_0 + x_{12}) \lll 7)$
check	$z_{12} = x_{12} + k_5$	$z_4 = x_4 + k_3$
with k'_5	$\boldsymbol{k'_6} = n_1 \oplus ((k'_5 + \sigma_3) \lll 9)$	$\boldsymbol{k'_7} = -k'_6$
	$\boldsymbol{k_4} = k'_7 \oplus ((k'_6 + k'_5) \lll 13)$	$\boldsymbol{x'_{12}} = z'_{12} - k'_5$
	$\boldsymbol{x_{11}} = (z'_{14} - k'_7) \oplus ((z'_{13} - k'_6 + x'_{12}) \lll 13)$	
	$\boldsymbol{k_2} = k'_5 \oplus ((\sigma_3 + k_4) \lll 7)$	$\boldsymbol{x_3} = x'_{12} \oplus ((x_{15} + x_{11}) \lll 7)$
check	$z_{11} = x_{11} + k_4$	$z_3 = x_3 + k_2$.

For the checking of the potential slid pairs we have nine extra test conditions while expecting only seven but due to the different arithmetic operations the dependencies of the equations are not clear. In total we have at least filtering power of 32×7 bits. Thus we expect that only the correct slid pairs survive this check. The remaining pairs are the correct slid pairs for which we completely know both keys.

Complexity. Assume we are given a list of 2^l ciphertexts with corresponding nonces and counters. Instead of storing the list twice we use two kinds of pointers, one for the potential 1st starting states and the other one for the potential 2nd starting states. For the pointers we need $l/32 \times 2^l$ words of memory. The larger the list of the random states in which our target is hidden – the larger would be the complexity of the attack. However the time complexity of the attack grows only linearly with the size of the list. A summary for the complexity of different lists is given in Table 2 with memory in words and time in Salsa encryptions.

The number of slid pairs is 2^{256} which gives for a random starting the probability of 2^{-128} to be a 1st or a 2nd starting state. Via the birthday paradox we

Table 2. Complexities for different list sizes

list size	memory	time
2^{128}	28×2^{128}	2^{122}
2^{192}	32×2^{192}	2^{186}
2^{256}	36×2^{256}	2^{250}

expect in an amount of 2^{256} random ciphertexts for one slid pair both starting states. We have described how to search in a big list efficiently for a slid pair and recover both secret keys.

2.5 Time-Memory Tradeoff Attacks on Salsa

Salsa20 has 2^{384} possible starting states. We notice that the square root of 2^{384} is less than the keyspace size for keys longer than 192-bits. Thus a trivial birthday attack on 256-bit key Salsa20 would proceed as follows:

In the preprocessing stage a list of randomly chosen starting states together with their ciphertexts is generated and sorted by the ciphertexts. During the on-line stage ciphertexts are captured and checked for a match with an entry of the list. The corresponding key is retrieved from the entry in the list.

Of course due to very high memory complexity this attack can be only viewed as a certificational weakness. The complexities are summarized in Table 3 where R stands for a complete run of Salsa20 and M for a matrix of Salsa (16 words).

Table 3. Complexities for the birthday attack

attack for 256-bit key size	state space	precomp.	memory	time	captured ciphertexts
chosen nonce and counter	2^{256}	$R \times 2^{128}$	$2M \times 2^{128}$	2^{128}	2^{128}
chosen nonce or counter	2^{320}	$R \times 2^{160}$	$2M \times 2^{160}$	2^{160}	2^{160}
general	2^{384}	$R \times 2^{192}$	$2M \times 2^{192}$	2^{192}	2^{192}
using sliding property	2^{384}	$R \times 2^{192}$	$2.5M \times 2^{192}$	2^{192}	2^{191}

Improved Birthday Using the Sliding Property. We can use the sliding property to increase the efficiency of the birthday attack twice (which can be translated into reduction of memory, time or increase of success probability of the birthday attack).

Salsa20 has 2^{384} possible starting states in total and the sliding property reduces the number of possible starting states to 2^{257} (a slid pair has two starting states) which gives for a random starting state the probability of 2^{-127} to be a starting state for a slid pair (either 1^{st} or 2^{nd} one).

During the preprocessing stage we generate a sample of 2^{192} 2^{nd} starting states by using equation (1) from section 2.2 and choose the remaining eight words at random. We compute the corresponding ciphertexts for these states as well as the eight specified words for the corresponding 1^{st} starting states mentioned in section 2.4 Step 2. We use two kinds of pointers to sort this generated list by the ciphertexts for the 2^{nd} starting states and by the eight words for the corresponding 1^{st} starting state. We capture ciphertext from the key-stream where we also know the nonce and the counter. The amount of 2^{191} captured ciphertexts contains about either 2^{64} 1^{st} starting states or 2^{64} 2^{nd} starting states. We check if the ciphertext is a correct one for a 2^{nd} starting state from our list (direct birthday) or is matching the eight words for a 1^{st} starting state for one

of the states from our collection (then proceed as described in section 2.4 Step 3 to check the remaining eight words) (indirect birthday). In both cases we learn the key for this ciphertext.

3 Slid Pairs for Trivium

3.1 Brief Description of Trivium

The designers introduced the stream cipher Trivium with a state size of 288 bits. This internal state can be split into three registers. The first register which we call A has length 93, the second one called B has length 84 and the last register named C has 111 bits. The internal state is denoted in the following way

$$\text{A: } (s_1, s_2, \ldots, s_{93}) \quad \text{B: } (s_{94}, s_{95}, \ldots, s_{177}) \quad \text{C: } (s_{178}, s_{279}, \ldots, s_{288}) \ .$$

Update and Key-Stream Production. The nonlinear update function of degree 2 uses 15 bits of the internal state to compute three new bits each for one register and the key-stream bit z_i is calculated by adding only 6 of these 15 bits together. In the following pseudo-code all computations are over GF(2).

$$\begin{array}{llll} 1. & t_1 & \longleftarrow & s_{66} + s_{93} \\ 2. & t_2 & \longleftarrow & s_{162} + s_{177} \\ 3. & t_3 & \longleftarrow & s_{243} + s_{288} \\ 4. & z_i & \longleftarrow & t_1 + t_2 + t_3 \end{array} \qquad \begin{array}{llll} 5. & t_1 & \longleftarrow & t_1 + s_{91} \cdot s_{92} + s_{171} \\ 6. & t_2 & \longleftarrow & t_2 + s_{175} \cdot s_{176} + s_{264} \\ 7. & t_3 & \longleftarrow & t_3 + s_{286} \cdot s_{287} + s_{69} \end{array}$$

$$\begin{array}{llll} 8. & \text{A:} & (s_1, s_2, ..., s_{93}) & \longleftarrow (t_3, s_1, \ldots, s_{92}) \\ 9. & \text{B:} & (s_{94}, s_{95}, \ldots, s_{177}) & \longleftarrow (t_1, s_{94}, \ldots, s_{176}) \\ 10. & \text{C:} & (s_{178}, s_{279}, \ldots, s_{288}) & \longleftarrow (t_2, s_{178}, \ldots, s_{287}) \end{array}$$

Key and IV Setup. In register A the 80-bit key is loaded and in register B the 80-bit IV. All remaining positions in the three registers are set to zero except for the last three bits in register C which are set to one

$$\begin{array}{lll} \text{A:} & (s_1, s_2, ..., s_{93}) & \longleftarrow (K_{80}, \ldots, K_1, 0, \ldots, 0) \\ \text{B:} & (s_{94}, s_{95}, \ldots, s_{177}) & \longleftarrow (IV_{80}, \ldots, IV_1, 0, \ldots, 0) \\ \text{C:} & (s_{178}, s_{279}, \ldots, s_{288}) & \longleftarrow (0, \ldots, 0, 1, 1, 1) \ . \end{array}$$

In this paper we refer to this state with key, IV and 128 fixed positions as starting state. After the registers are initialized in the described way the cipher is clocked 4×288 times using the update function without producing any key-stream bits. This will finish the key setup. Now each following clock will produce a key-stream bit.

3.2 Slid Pairs

We start with the observation made by Jin Hong on the eSTREAM forum [7], that it is possible to produce sliding states in Trivium. We searched for pairs

of key and IV which produce another starting state after a few clocks. If we have a key and IV pair (K_1, IV_1) which produce another starting state with a key and IV (K_2, IV_2), the created key-stream by (K_2, IV_2) will be the same as the one created by (K_1, IV_1) except for a shift of some bits. The number of shifted bits is equal the number of clocks needed to get from the 1st to the 2nd starting state. We call such a pair of two key and IV pairs a slid pair and denote this with $[(K_1, IV_1), (K_2, IV_2), c]$ whereas c stands for the number of clocks-shifts.

Due to the special structure of the third register with 108 zeros and the last three ones the first possibility of a 2nd starting state to occur is after 111 clocks. Each following clock gives the chance for a 2nd starting state. Two examples for slid pairs written in hexadecimal numbers are given below. The bits for keys and IVs are ordered from 1 to 80 but in the key and IV setup they are used the other way around.

$[(K_1, IV_1), (K_2, IV_2), 111]$

K_1 :	70011000001E00000000
IV_1 :	AF9D635BCEF9AE376CF7
key-stream[4]:	2E7338CB404272ABEE3F7BEC2F8D
	55E27536D29AFFFF15DFDFD711AECC78D13D7B61 ...
K_2 :	780000001DA2000003C1
IV_2 :	1DF35CF6D4FFF4E3A6C0
key-stream:	55E27536D29AFFFF15DFDFD711AECC78D13D7B61 ...

$[(K_3, IV_3), (K_4, IV_4), 112]$

K_3 :	02065B9C001730000000
IV_3:	609FC141828705160A3C
key-stream:	A48BCA9143685F03DE646F83AB52
	88BC9542798983349A959503E63BBF29C4755DE6 ...
K_4 :	B98000003E96E70005CE
IV_4 :	2B7C1483BC476A62E4CB
key-stream:	88BC9542798983349A959503E63BBF29C4755DE6 ...

3.3 Systems of Equations

We describe the 2nd starting state as polynomial equations in the 80 key and 80 IV variables of the 1st pair. The 128 fixed positions in a starting state yield a system of equations with 160 variables and 128 equations. We have more variables than equations which gives us freedom in 32 variables. To solve these systems we tried the F4 algorithm implemented in the computer algebra system Magma to get a Gröbner basis and the solutions for (K_1, IV_1) but gave up after $c = 115$ because of the long computation time. A more brute force approach guessing a part of the variables, check this guess and print the solution which

[4] The shift is $c = 111$ which means the first 111 bits are a prefix. When rewriting these prefix from hexadecimal to binary numbers the leading zero must be omitted because 111 is not a multiple of 4.

Table 4. Some facts for the systems of equations

clock-shift c	111	112	113	114	115	116	...	124
variables in equations	136	137	138	139	140	141	...	149
last key bits not in the equation	24	23	22	21	20	19	...	11
a priori given bits	16	15	14	13	13	13	...	13
computing time magma (days)	2.5	2.5	10	32.5	64	-	-	-
guess bits for magma[5]	0	4	6	8	10	-	-	-

we implemented for each individual c worked much better. To get the 2^{nd} key and IV one can use the systems of equations which describe the key and IV in the 2^{nd} starting state just inserting the known values of the 1^{st} key and IV pair or simply clock Trivium c times starting from (K_1, IV_1) to get (K_2, IV_2).

Some Facts about these Systems. The system of equations for the first instance which appears after 111 clocks contains only 136 variables because the last 24 bits of the key do not occur in this system. Furthermore 16 bits are given a priori due to the 13 zeros in register A and 3 ones in register C. The degree of the monomials in the equations raised from 1 to 3. Due to the missing of the 24 key bits in the equations these bits can be chosen arbitrarily. This leads us to 2^{24} different keys for one IV in the 1^{st} key and IV pair of a slid pair. Table4 collects some facts for the systems which we solved with our brute force approach. We found that we have some times slightly less but most times slightly more solutions that we would have expected. This is described in detail in [11].

Each clock-shift yields in another but related system of equations and the higher the clock-shift the more complicated the system of equations will be. Due to the length of register C which defines the occurrence of a 2^{nd} starting state we have at least 111 clock-shifts. Thus we have minimum $111 \times 2^{32} \approx 2^{39}$ slid pairs, just within a shift of 221 bits of each other. There are much more slid pairs for longer shifts, but the equations would be much more complicated.

Nonexistence of Special Slid Pairs. We searched for slid pairs with additional constraints. The first type applies when the keys in both key and IV pairs are the same for any clock-shift c: $([(K, IV_1), (K, IV_2)], c)$ and the second type applies when both times the same IV is used for any clock-shift c: $([(K_1, IV), (K_2, IV)], c)$. In both cases the fixed 2^{nd} key or IV leads to 80 additional equations which account for the occurrence of all 80-bit of key or IV resulting in overdefined systems with 208 equations and 160 variables. For both types the systems are not likely to be solvable for any reasonably small amount of shift. As a result of the 48 extra equations the chance for such system to have a solution is about 2^{-48}. We computed that for the first 31 instances (clock-shifts 111 up to 142) these systems have no solution.

[5] We guessed these bits to get a solution from Magma in a reasonable amount of time.

4 Conclusion

In this paper we have described sliding properties of Salsa20 and Trivium which lead to distinguishing, key recovery and related-key attacks on these ciphers. We also show that Salsa20 does not offer 256-bit security due to a simple birthday attack on its 384-bit state. Since the likelihood of falling in our related key classes by chance is relatively low (2^{256} out of 2^{384} for Salsa20, 2^{39} out of 2^{80} for Trivium) these attacks do not threaten most of the real-life usage of these ciphers. However designer of protocols which would use these primitives should be definitely aware of these non-randomness properties, which can be exploited in certain scenarios.

References

1. Aumasson, J.-P., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New Features of Latin Dances: Analysis of Salsa, ChaCha and Rumba. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086. Springer, Heidelberg (2008), Full version as IACR eprint, http://eprint.iacr.org/2007/472
2. Bernstein, D.J.: Salsa20. eSTREAM, Report 2005/025 (2005)
3. De Cannière, C., Preneel, B.: TRIVIUM - a stream cipher construction inspired by block cipher design principles. eSTREAM, Report 2005/030 (2005)
4. Crowley, P.: Truncated differential cryptanalysis of five rounds of Salsa20. In: SASC 2006 - Stream Ciphers Revisited (2006)
5. eSTREAM: The ECRYPT Stream Cipher Project, http://www.ecrypt.eu.org/stream/
6. Fischer, S., Meier, W., Berbain, C., Biasse, J.-F., Robshaw, M.J.B.: Non-randomness in eSTREAM Candidates Salsa20 and TSC-4. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 2–16. Springer, Heidelberg (2006)
7. Hong, J.: Discussion Forum. certain pairs of key-IV pairs for Trivium, created (September 13, 2005), http://www.ecrypt.eu.org/stream/phorum/read.php?1,152
8. Maximov, A., Biryukov, A.: Two Trivial Attacks on Trivium. In: SASC 2007 - The State of the Art of Stream Ciphers (2007)
9. McDonald, C., Charnes, C., Pieprzyk, J.: Attacking Bivium with MiniSat. eSTREAM, Report 2007/040 (2007)
10. Raddum, H.: Cryptanalytic Results on TRIVIUM. eSTREAM, Report 2006/039 (2006)
11. Priemuth-Schmid, D., Biryukov, A.: Slid Pairs in Salsa20 and Trivium. Cryptology ePrint Archive, Report 2008/405 (2008), http://eprint.iacr.org/2008/405
12. Tsunoo, Y., Saito, T., Kubo, H., Suzaki, T., Nakashima, H.: Differential Cryptanalysis of Salsa20/8. In: SASC 2007 - The State of the Art of Stream Ciphers (2007)
13. Turan, M.S., Kara, O.: Linear Approximations for 2-round Trivium. In: SASC 2007 - The State of the Art of Stream Ciphers (2007)
14. Vielhaber, M.: Breaking ONE.Fivium by AIDA an Algebraic IV Differential Attack. Cryptology ePrint Archive, Report 2007/413 (2007), http://eprint.iacr.org/2007/413

A Salsa20 System of Equations for a Slid Pair

Word equations given by the equations $S' = \boldsymbol{\mathcal{F}}(S)$ and $X' = \boldsymbol{\mathcal{F}}(X)$.

1. $k'_0 = k_3 \oplus ((\sigma_0 + k_5) \lll 7)$
2. $k'_1 = c_0 \oplus ((k'_0 + \sigma_0) \lll 9)$
3. $k'_2 = k_5 \oplus ((k'_1 + k'_0) \lll 13)$
4. $\sigma_0 = \sigma_0 \oplus ((k'_2 + k'_1) \lll 18)$

5. $n'_0 = c_1 \oplus ((\sigma_1 + k_0) \lll 7)$
6. $n'_1 = k_6 \oplus ((n'_0 + \sigma_1) \lll 9)$
7. $k'_3 = k_0 \oplus ((n'_1 + n'_0) \lll 13)$
8. $\sigma_1 = \sigma_1 \oplus ((k'_3 + n'_1) \lll 18)$

9. $k'_4 = k_7 \oplus ((\sigma_2 + n_0) \lll 7)$
10. $c'_0 = k_1 \oplus ((k'_4 + \sigma_2) \lll 9)$
11. $c'_1 = n_0 \oplus ((c'_0 + k'_4) \lll 13)$
12. $\sigma_2 = \sigma_2 \oplus ((c'_1 + c'_0) \lll 18)$

13. $k'_5 = k_2 \oplus ((\sigma_3 + k_4) \lll 7)$
14. $k'_6 = n_1 \oplus ((k'_5 + \sigma_3) \lll 9)$
15. $k'_7 = k_4 \oplus ((k'_6 + k'_5) \lll 13)$
16. $\sigma_3 = \sigma_3 \oplus ((k'_7 + k'_6) \lll 18)$

17. $x'_1 = x_4 \oplus ((x_0 + x_{12}) \lll 7)$
18. $x'_2 = x_8 \oplus ((x'_1 + x_0) \lll 9)$
19. $x'_3 = x_{12} \oplus ((x'_2 + x'_1) \lll 13)$
20. $x'_0 = x_0 \oplus ((x'_3 + x'_2) \lll 18)$

21. $x'_6 = x_9 \oplus ((x_5 + x_1) \lll 7)$
22. $x'_7 = x_{13} \oplus ((x'_6 + x_5) \lll 9)$
23. $x'_4 = x_1 \oplus ((x'_7 + x'_6) \lll 13)$
24. $x'_5 = x_5 \oplus ((x'_4 + x'_7) \lll 18)$

25. $x'_{11} = x_{14} \oplus ((x_{10} + x_6) \lll 7)$
26. $x'_8 = x_2 \oplus ((x'_{11} + x_{10}) \lll 9)$
27. $x'_9 = x_6 \oplus ((x'_8 + x'_{11}) \lll 13)$
28. $x'_{10} = x_{10} \oplus ((x'_9 + x'_8) \lll 18)$

29. $x'_{12} = x_3 \oplus ((x_{15} + x_{11}) \lll 7)$
30. $x'_{13} = x_7 \oplus ((x'_{12} + x_{15}) \lll 9)$
31. $x'_{14} = x_{11} \oplus ((x'_{13} + x'_{12}) \lll 13)$
32. $x'_{15} = x_{15} \oplus ((x'_{14} + x'_{13}) \lll 18)$

Word equations given by the feedforward for the key-stream words of the 1st and 2nd starting state.

33. $z_0 = x_0 + \sigma_0$
34. $z_1 = x_1 + k_0$
35. $z_2 = x_2 + k_1$
36. $z_3 = x_3 + k_2$

37. $z_4 = x_4 + k_3$
38. $z_5 = x_5 + \sigma_1$
39. $z_6 = x_6 + n_0$
40. $z_7 = x_7 + n_1$

41. $z_8 = x_8 + c_0$
42. $z_9 = x_9 + c_1$
43. $z_{10} = x_{10} + \sigma_2$
44. $z_{11} = x_{11} + k_4$

45. $z_{12} = x_{12} + k_5$
46. $z_{13} = x_{13} + k_6$
47. $z_{14} = x_{14} + k_7$
48. $z_{15} = x_{15} + \sigma_3$

49. $z'_0 = x'_0 + \sigma_0$
50. $z'_1 = x'_1 + k'_0$
51. $z'_2 = x'_2 + k'_1$
52. $z'_3 = x'_3 + k'_2$

53. $z'_4 = x'_4 + k'_3$
54. $z'_5 = x'_5 + \sigma_1$
55. $z'_6 = x'_6 + n'_0$
56. $z'_7 = x'_7 + n'_1$

57. $z'_8 = x'_8 + c'_0$
58. $z'_9 = x'_9 + c'_1$
59. $z'_{10} = x'_{10} + \sigma_2$
60. $z'_{11} = x'_{11} + k'_4$

61. $z'_{12} = x'_{12} + k'_5$
62. $z'_{13} = x'_{13} + k'_6$
63. $z'_{14} = x'_{14} + k'_7$
64. $z'_{15} = x'_{15} + \sigma_3$

New Directions in Cryptanalysis of Self-Synchronizing Stream Ciphers

Shahram Khazaei[1] and Willi Meier[2]

[1] EPFL, Lausanne, Switzerland
[2] FHNW, Windisch, Switzerland

Abstract. In cryptology we commonly face the problem of finding an unknown key K from the output of an easily computable keyed function $F(C, \mathsf{K})$ where the attacker has the power to choose the public variable C. In this work we focus on self-synchronizing stream ciphers. First we show how to model these primitives in the above-mentioned general problem by relating appropriate functions F to the underlying ciphers. Then we apply the recently proposed framework presented at AfricaCrypt'08 by Fischer *et. al.* for dealing with this kind of problems to the proposed T-function based self-synchronizing stream cipher by Klimov and Shamir at FSE'05 and show how to deduce some non-trivial information about the key. We also open a new window for answering a crucial question raised by Fischer *et. al.* regarding the problem of finding weak IV bits which is essential for their attack.

Keywords: Self-synchronizing Stream Ciphers, T-functions, Key Recovery.

1 Introduction

The area of stream cipher design and analysis has made a lot of progress recently, mostly spurred by the eStream [6] project. It is a common belief that designing elegant strong synchronizing stream ciphers is possible, however, it is harder to come up with suitable designs for self-synchronizing ones. Despite numerous works on self-synchronizing stream ciphers in the literature, there is not yet a good understanding of their design and cryptanalytic methods. Many self-synchronizing stream ciphers have shown not to withstand cryptanalytic attacks and have been broken shortly after they have been proposed. In this work we show how to model a self-synchronizing stream cipher by a family of keyed functions $F(C, \mathsf{K})$. The input parameter C, called the *public variable*, can be controlled by the attacker while the input K is an unknown parameter to her called the extended key; it is a combination of the actual key used in the cipher and the unknown internal state of the cipher. The goal of the attacker would be to recover K or to get some information about it. The problem of finding the unknown key K, when access is given to the output of the function $F(C, \mathsf{K})$ for every C of the attacker's choice, is a very common problem encountered in cryptography. In general, when the keyed function F looks like a random function, the best way to solve the problem is to exhaust the key space. However, if F is far from being a random function there might be more efficient methods. Recently, Fischer *et. al.* [7] developed a method to recover the key faster than by exhaustive search in case F does not properly mix its

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 15–26, 2008.

input bits. The idea is to first identify some bits from C referred to as *weak public variable bits* and then to consider the coefficient of a monomial involving these weak bits in the algebraic normal form of F. If this coefficient does not depend on all the unknown bits of K, or it weakly depends on some of them, it can be exploited in an attack. Having modeled the self-synchronizing stream ciphers as the above-mentioned general problem, we consider the T-function based self-synchronizing stream cipher proposed by Klimov and Shamir [8] and use the framework from [7] to deduce some information about the key bits through some striking relations. Finding the weak public variable bits was raised as a crucial open question in [7] which was done mostly by random search there. In the second part of our work we try to shed some light in this direction in a more systematic way. The recently proposed cube attack by Dinur and Shamir [4], which has a strong connection to [7] and the present work, also includes some systematic procedure to find weak public variable bits.

The rest of the paper is organized as follows. In section 2 we review the method from [7] and try to make the connection between [4] and [7] clearer. In section 3 we describe the self-synchronizing stream ciphers and explain how to derive keyed functions $F(C, \mathsf{K})$ which suit the framework from [7]. Section 4 covers the description of the Klimov-Shamir T-function based self-synchronizing stream cipher along with its reduced word-size versions which will later be attacked in section 5 and 6. Section 6 also includes our new direction of finding weak bits in a systematic way.

2 An Approach for Key Recovery on a Keyed Function

Notations. We use $\mathbb{B} = \{0, 1\}$ for the binary field with two elements. A general m-bit vector in $\mathbb{B}^m$ is denoted by $C = (c_1, c_2, \ldots, c_m)$. By making a partition of C into $U \in \mathbb{B}^l$ and $W \in \mathbb{B}^{m-l}$, we mean dividing the variables set $\{c_1, c_2, \ldots, c_m\}$ into two disjoint subsets $\{u_1, ..., u_l\}$ and $\{w_1, ..., w_{m-l}\}$ and setting $U = (u_1, \ldots, u_l)$ and $W = (w_1, \ldots, w_{m-l})$. However, whenever we write $(U; W)$ we mean the original vector C. For example $U = (c_2, c_4)$ and $W = (c_1, c_3, c_5)$ is a partition for the vector $C = (c_1, c_3, c_4, c_5)$ and $(U; W)$ is equal to C and not to $(c_2, c_4, c_1, c_3, c_5)$. We also use the notation $U = C \setminus W$ and $W = C \setminus U$. A vector of size zero is denoted by $\emptyset$.

In this section we review the framework from [7] which was inspired by results from [1, 5, 11]. Let $F : \mathbb{B}^m \times \mathbb{B}^n \rightarrow \mathbb{B}$ be a keyed Boolean function which maps the m-bit public variable C and the n-bit secret variable K into the output bit $z = F(C, \mathsf{K})$. An oracle chooses a key K uniformly at random over $\mathbb{B}^n$ and returns $z = F(C, \mathsf{K})$ to an adversary for any chosen $C \in \mathbb{B}^m$ of adversary's choice. The oracle chooses the key K only once and keeps it fixed and unknown to the adversary. The goal of the adversary is to recover K by dealing with the oracle assuming that he has also the power to evaluate F for all inputs, *i.e.* all secret and public variables. To this end, the adversary can try all possble 2^n keys and filter the wrong ones by asking enough queries from the oracle. Intuitively each oracle query reveals one bit of information about the secret key if F mixes its input bits well enough to be treated as a random Boolean function with $n + m$ input bits. Therefore, assuming $\log_2 n \ll m$, then n key bits can be recovered by sending $\mathcal{O}(n)$ queries to the oracle. More precisely if the adversary asks the oracle $n + \beta$ queries for some integer $\beta \gg 0$, then the probability that only the unknown

chosen key by the oracle (*i.e.* the correct candidate) satisfies these queries while all the remaining $2^n - 1$ keys fail to satisfy all the queries is $(1 - 2^{-(n+\beta)})^{2^n - 1} \approx 1 - e^{-2^{-\beta}}$ (for $\beta = 10$ it is about $1 - 10^{-3}$). The required time complexity is $\mathcal{O}(n2^n)$. However, if F extremely deviates from being treated as a random function, the secret key bits may not be determined uniquely. It is easy to argue that F divides $\mathbb{B}^n$ into J *equivalence classes* $\mathcal{K}_1,\ \mathcal{K}_2, \ldots, \mathcal{K}_J$ for some $J \leq 2^n$, see Lemma 1 from [7]. Two keys K' and K'' belong to the same equivalence class iff $F(C, \mathsf{K}') = F(C, \mathsf{K}'')$ for all $C \in \mathbb{B}^m$. Let n_i denotes the number of keys which belong to the equivalence class $\mathcal{K}_i$. Note that we have $\sum_{i=1}^{J} n_i = 2^n$. A random key lies in the equivalence class $\mathcal{K}_i$ with probability $n_i/2^n$ in which case $(n - \log_2 n_i)$ bits of information can be achieved about the key. The adversary on average can get $\sum_{i=1}^{J}(n - \log_2 n_i)\frac{n_i}{2^n}$ bits of information about the n key bits by asking enough queries. It is difficult to estimate the minimum number of needed queries due to the statistical dependency between them. It highly depends on the structure of F but we guess that $O(n)$ queries suffice again. However, in case where F does not properly mix its input bits, there might be faster methods than exhaustive search for key recovery. We are interested in faster methods of recovering the unknown secret key in this case.

If one derives a weaker keyed function $\Gamma(W, \mathsf{K}) : \mathbb{B}^{m-l} \times \mathbb{B}^n \rightarrow \mathbb{B}$ from F which depends on the same key and a part of the public variables, the adversary-oracle interaction can still go on through Γ this time. The idea of [7] is to derive such functions from the algebraic expansion of F by making a partition of the m-bit public variable C into $C = (U; W)$ with l-bit vector U and $(m - l)$-bit vector W. Let $F(C, \mathsf{K}) = \sum_{\alpha} \Gamma_\alpha(W, \mathsf{K}) U^\alpha$ where $U^\alpha = u_1^{\alpha_1} u_2^{\alpha_2} \ldots u_l^{\alpha_l}$ for the multi-index $\alpha = (\alpha_1, \ldots, \alpha_l)$. In other words, $\Gamma_\alpha(W, \mathsf{K})$ is the coefficient of U^α in the algebraic expansion of F. For every $\alpha \in \mathbb{B}^l$, the function $\Gamma_\alpha(W, \mathsf{K})$ can serve as a function Γ derived from F. The function corresponding to $\alpha = (1, \ldots, 1)$ is the coefficient of the maximum degree monomial. Previous works [5, 7] suggest that this function is usually more useful. We also only focus on the maximum degree monomial coefficient. Hence we drop the subscript α and write $\Gamma(W, \mathsf{K})$ instead of $\Gamma_\alpha(W, \mathsf{K})$ for $\alpha = (1, \ldots, 1)$. Inspired by the terminology of [4] we refer to U as *cube vector* and to $\Gamma(W, \mathsf{K})$ as *superpoly* corresponding to cube vector U. Thanks to the relation $\Gamma(W, \mathsf{K}) = \bigoplus_{U \in \mathbb{B}^l} F((U; W), \mathsf{K})$, the adversary can still evaluate the superpoly for any W of his choice and for the same chosen key K by the oracle. This demands that the adversary sends 2^l queries to the oracle for each evaluation of Γ.

In order to have an effective attack we need to have a weak superpoly function. In [7] several conditions were discussed under which the superpoly can be considered as a weak function and potentially lead to an attack. Refer to [4] for more scenarios and generalizations. In this paper we look for cube vectors U such that their superpoly does not depend on a large number of key bits. We refer to those key bits which $\Gamma(W, \mathsf{K})$ does not depend on as *neutral key bits*. This is a special case of the third scenario in [7] where *probabilistic neutral bits* were used instead. If the superpoly effectively depends on $t_k \leq n$ key bits and $t_w \leq m - l$ public key bits, assuming these $t_k + t_w$ bits are mixed reasonably well, the involved t_k secret bits can be recovered in time $t_k 2^{l+t_k}$ by sending $O(t_k 2^l)$ queries to the oracle. However, if the superpoly extremely deviates from being treated as a random function, as we already argued, it may even happen

that the t_k key bits can not be determined uniquely. In this case one has to look at the corresponding equivalence classes to see how much information one can achieve about the involved t_k key bits. In sections 5 and 6 we will provide some examples by considering Klimov-Shamir's self-synchronizing stream cipher.

2.1 Connection with Previous Works

The attack is closely related to differential [2,9] and integral [10,3] kind of attacks, and the recent cube attack [4]. For $l = 0$ we have $U = \emptyset$ and $W = C$ and hence $\Gamma = F$, that is we are analyzing the original function. For $l = 1$ let's take $U = (c_i)$ and $W = C \setminus (c_i)$ for some $1 \leq i \leq m$. In this case we are considering a variant of (truncated) differential cryptanalysis, that is we have $\Gamma(W, \mathsf{K}) = F(C, \mathsf{K}) \oplus F(C \oplus \Delta C, \mathsf{K})$ where ΔC is an m-bit vector which is zero in all bit positions except the i-th one. For bigger l, this approach can be seen as an adaptive kind of higher order differential cryptanalysis. A more precise relation between the framework in [7] and (higher order) differential cryptanalysis seems to be as follows: The superpoly $\Gamma(W, \mathsf{K})$, which computes the coefficient of the maximum degree monomial, is computed as the sum of all outputs $F(C, \mathsf{K})$ where $C = (U; W)$ has a fixed part W and U varies over all possible values. This is what is also done in (higher order) differential cryptanalysis. However, in applications of the framework in [7], the values for W are often chosen adaptively. By adaptively we mean that a stronger deviation from randomness is observed for some specific choices for W (*e.g.* low weight W's) or even a specific value for W (*e.g.* W = 0). Whereas in most applications of (higher order) differential cryptanalysis, specific input values are of no favour. The recently proposed cube attack by Dinur and Shamir [4] still lies in the second scenario (condition) proposed in [7], having had been inspired by the earlier work by Vielhaber [12]. In [7] the public variable C was the Initial Vector of a stream cipher and the cube variables were called *weak IV bits* whenever the derived function Γ turned out to be weak enough to mount an attack. This concept can be adapted according to each context depending on the public variable (weak ciphertext bits, weak plaintext bits, weak message bits, *etc*). In general the terminology *weak public variables* can be used. On the whole, it is not easy to find week public variables. While [4] uses a more systematic procedure, [7] uses random search over cube vectors. In section 6, we will also take kind of systematic method. Another point which is worth mentioning is that cube attack [4] nicely works with complexity $O(n2^{d-1})$ if F is a random function of degree d in its $m + n$ input bits. In this case the superpoly corresponding to any cube vector of size $d - 1$ is weak, since it is a random linear function in key bits and remaining public variables.

3 Self-Synchronizing Stream Ciphers

A self-synchronizing stream cipher is built on an output filter $\mathcal{O} : \mathcal{K} \times \mathcal{S} \rightarrow \mathcal{M}$ and a self-synchronizing state update function (see Definition 1) $\mathcal{U} : \mathcal{M} \times \mathcal{K} \times \mathcal{S} \rightarrow \mathcal{M}$, where $\mathcal{S}$, $\mathcal{K}$ and $\mathcal{M}$ are the cipher state space, key space and plaintext space. We suppose that the ciphertext space is the same as that of the plaintext. Let $K \in \mathcal{K}$ be the secret key, and $\{S_i\}_{i=0}^{\infty}$, $\{p_i\}_{i=0}^{\infty}$ and $\{c_i\}_{i=0}^{\infty}$ denote the sequences of cipher state, plaintext and ciphertext respectively. The initial state is computed through the initialization procedure

as $S_0 = \mathcal{I}(K, IV)$ from the secret key K and a public initial value IV. The ciphertext (in an additive stream cipher) is then computed according to the following relations:

$$c_i = p_i \oplus \mathcal{O}(K, S_i), \tag{1}$$

$$S_{i+1} = \mathcal{U}(c_i, K, S_i). \tag{2}$$

Definition 1. *[8] (SSF) Let $\{c_i\}_{i=0}^{\infty}$ and $\{\hat{c}_i\}_{i=0}^{\infty}$ be two input sequences, let S_0 and $\hat{S}_0$ be two initial states, and let K be a common key. Assume that the function $\mathcal{U}$ is used to update the state based on the current input and the key: $S_{i+1} = \mathcal{U}(c_i, K, S_i)$ and $\hat{S}_{i+1} = \mathcal{U}(\hat{c}_i, K, \hat{S}_i)$. The function $\mathcal{U}$ is called a self-synchronizing function (SSF) if equality of any r consecutive inputs implies the equality of the next state, where r is some integer, i.e.:*

$$c_i = \hat{c}_i, \ldots, c_{i+r-1} = \hat{c}_{i+r-1} \Rightarrow S_{i+r} = \hat{S}_{i+r}. \tag{3}$$

Definition 2. *The "resynchronization memory" of a function $\mathcal{U}$, assuming it is a SSF, is the least positive value of r such that Eq. 3 holds.*

3.1 Attack Models on Self-Synchronizing Stream Ciphers

There are two kinds of attack on synchronizing stream ciphers: *distinguishing attacks* and *key recovery attacks*[1]. The strongest scenario in which these attacks can be applied is a *known-keystream* attack model or a *chosen-IV-known-keystream* attack if the cipher uses an IV for initialization. It is not very clear how applying distinguishing attacks make sense for self-synchronizing stream ciphers. However, in the strongest scenario, one considers key recovery attacks in a *chosen-ciphertext* attack model or in a *chosen-IV-chosen-ciphertext* attack if the cipher uses an IV for initialization.

In this paper we only focus on chosen-ciphertext attacks. Our goal as an attacker is to efficiently recover the unknown key K by sending to the decryption oracle chosen ciphertexts of our choice. More precisely, we consider the family of functions $\{\mathcal{H}_i : \mathcal{M}^i \times \mathcal{K} \times \mathcal{S} \rightarrow \mathcal{M} | i = 1, 2, \ldots r\}$, where r is the resynchronization memory of the cipher and $\mathcal{H}_i(c_1, \ldots, c_i, K, S) = \mathcal{O}(K, \mathcal{G}_i(c_1, \ldots, c_i, K, S))$, where $\mathcal{G}_i : \mathcal{M}^i \times \mathcal{K} \times \mathcal{S} \rightarrow \mathcal{S}$ is recursively defined as $\mathcal{G}_{i+1}(c_1, \ldots, c_i, c_{i+1}, K, S) = \mathcal{U}(c_{i+1}, K, \mathcal{G}_i(c_1, \ldots, c_i, K, S))$ with initial condition $\mathcal{G}_1 = \mathcal{U}$.

Note that due to the self-synchronizing property of the cipher $\mathcal{H}_r(c_1, \ldots, c_r, K, S)$ is actually independent of the last argument S, however, all other $r-1$ functions depend on their last input. The internal state of the cipher is unknown at each step of operation of the cipher but because of the self-synchronizing property of the cipher it only depends on the last r ciphertext inputs and the key. We take advantage of this property and force the cipher to get stuck in a fixed but unknown state $\mathsf{S}^\star$ by sending the decryption oracle ciphertexts with some fixed prefix $(\mathsf{c}^\star_{-r+1}, \ldots, \mathsf{c}^\star_{-1})$ of our choice. Having forced the cipher to fall in the unknown fixed state $\mathsf{S}^\star$, we can evaluate any of the functions $\mathcal{H}_i$, $i = 1, 2, \ldots, r$, at any point $(c_1, \ldots, c_i, K, \mathsf{S}^\star)$ for any input $(c_1, \ldots, c_i)$ of our choice

[1] One could also think of *state recovery attack* in cases in which the synchronizing stream cipher is built based on a finite state machine and the internal state does not easily reveal the key.

by dealing with the decryption oracle. To be clearer let $z = \mathcal{H}_i(c_1, \ldots, c_i, K, \mathsf{S}^\star)$. In order to compute z for an arbitrary $(c_1, \ldots, c_i)$, we choose an arbitrary $\mathsf{c}^\star_{i+1} \in \mathcal{M}$ and ask the decryption oracle for $(p_{-r+1}, \ldots, p_{-1}, p_0, \ldots, p_{i+1})$– the decrypted plaintext corresponding to the ciphertext $(\mathsf{c}^\star_{-r+1}, \ldots, \mathsf{c}^\star_0,\ c_1, \ldots, c_i, \mathsf{c}^\star_{i+1})$. We then set $z = p_{i+1} \oplus \mathsf{c}^\star_{i+1}$.

To make notations simpler, we merge the unknown values K and $\mathsf{S}^\star$ in one unknown variable $\mathsf{K} = (K, \mathsf{S}^\star) \in \mathcal{K} \times \mathcal{S}$, called *extended unknown key*. We then use the simplified notation $\mathcal{F}_i(C, \mathsf{K}) = \mathcal{H}_i(c_1, \ldots, c_i, K, \mathsf{S}^\star) : \mathcal{M}^i \times (\mathcal{K} \times \mathcal{S}) \rightarrow \mathcal{M}$ where $C = (c_1, \ldots, c_i)$.

4 Description of the Klimov-Shamir T-Function Based Self-Synchronizing Stream Cipher

Shamir and Klimov [8] used the so-called multiword T-functions for a general methodology to construct a variety of cryptographic primitives. No fully specified schemes were given, but in the case of self-synchronizing stream ciphers, a concrete example construction was outlined. This section recalls its design. Let $\lll$, $+$, $\times$, $\oplus$ and $\vee$ respectively denote left rotation, addition modulo 2^{64}, multiplication modulo 2^{64}, bit-wise XOR and bit-wise OR operations on 64-bit integers. The proposed design works with 64-bit words and has a 3-word internal state $S = (s_0, s_1, s_2)^T$. A 5-word key $K = (k_0, k_1, k_2, k_3, k_4)$ is used to define the output filter and the state update function as follows:

$$\mathcal{O}(K, S) = \big((s_0 \oplus s_2 \oplus k_3) \lll 32\big) \times \big(((s_1 \oplus k_4) \lll 32) \vee 1\big), \tag{4}$$

and

$$\mathcal{U}\big(c,\ K,\ S\big) = \begin{pmatrix} \big(((s'_1 \oplus s'_2) \vee 1) \oplus k_0\big)^2 \\ \big(((s'_2 \oplus s'_0) \vee 1) \oplus k_1\big)^2 \\ \big(((s'_0 \oplus s'_1) \vee 1) \oplus k_2\big)^2 \end{pmatrix}, \tag{5}$$

where

$$\begin{aligned} s'_0 &= s_0 \oplus c \\ s'_1 &= s_1 - (c \lll 21) \\ s'_2 &= s_2 \oplus (c \lll 43). \end{aligned} \tag{6}$$

Generalized Versions. We also consider generalized versions of this cipher which use ω-bit words (ω even and typically $\omega = 8, 16, 32$ or 64). For ω-bit version the number of rotations in the output filter, Eq. 4, is $\frac{\omega}{2}$ and those of the state update function, Eq. 6, are $\lfloor \frac{\omega}{3} \rceil$ and $\lfloor \frac{2\omega}{3} \rceil$, $\lfloor x \rceil$ being the closest integer to x.

It can be shown [8] that the update function $\mathcal{U}$ is actually a SSF whose resynchronization memory is limited to ω steps and hence the resulting stream cipher is self-synchronizing indeed. Our analysis of the cipher for $\omega = 8,\ 16,\ 32$ and 64 shows that it resynchronizes after $r = \omega - 1$ steps (using $\omega(\omega - 1)$ input bits). It is an open question if this holds in general.

Remark 1. In [8] the notation $(k_0, k_1, k_2, k_{\mathcal{O}}, k'_{\mathcal{O}})$ is used for the key instead of the more standard notation $(k_0, k_1, k_2, k_3, k_4)$. The authors possibly meant to use a 3-word key (k_0, k_1, k_2) by deriving the other two key words ($k_{\mathcal{O}}$ and $k'_{\mathcal{O}}$ in their notations corresponding to k_3 and k_4 in ours) from first three key words. However, they do not specify how this must be done if they meant so. Also they did not introduce an initialization procedure for their cipher. In any case, we attack a more general situation where the cipher uses a 5-word secret key $K = (k_0, k_1, k_2, k_3, k_4)$ in chosen-ciphertext attack scenario. Moreover, for the 64-bit version the authors mentioned "the best attack we are aware of this particular example [64-bit version] requires $\mathcal{O}(2^{96})$ time", without mentioning the attack.

5 Analysis of the Klimov-Shamir T-Function Based Self-Synchronizing Stream Cipher

Let ω ($\omega = 8, 16, 32$ or 64) denote the word size and $r = \omega - 1$ be the resynchronization memory of the ω-bit version of the Klimov-Shamir self-synchronizing stream cipher. Let $\mathbb{B} = \{0, 1\}$ and $\mathbb{B}_\omega$ denote the binary field and the set of ω-bit words respectively. Following the general model of analysis of self-synchronizing stream ciphers in section 3.1, we focus on the family of functions $\mathcal{F}_i(C, \mathsf{K}) : \mathbb{B}_\omega^i \times \mathbb{B}_\omega^8 \to \mathbb{B}_\omega$, $i = 1, 2, \ldots, r$ where $C = (c_1, \ldots, c_i)$ and $\mathsf{K} = (K, \mathsf{S}^\star) = (k_0, k_1, k_2, k_3, k_4, \mathsf{s}_0^\star, \mathsf{s}_1^\star, \mathsf{s}_2^\star)$. We also look at a word b as an ω-bit vector $b = (b_0, \ldots, b_{\omega-1})$, b_0 being its LSB and $b_{\omega-1}$ its MSB. Therefore any vector $A = (a_0, a_1, ..., a_{p-1}) \in \mathbb{B}_\omega^p$ could be also treated as a vector in $\mathbb{B}^{p \times \omega}$ where the $(i\omega + j)$-th bit of A is $a_{i,j}$, the j-th LSB of the word a_i, for $i = 0, 1, \ldots, p-1$ and $j = 0, 1, \ldots, \omega - 1$ (we start numbering the bits of vectors from zero).

Now, for any $i = 1, \ldots, r$ and $j = 0, \ldots, \omega - 1$ we consider the family of Boolean functions $\mathcal{F}_{i,j} : \mathbb{B}^{i\omega} \times \mathbb{B}^{8\omega} \to \mathbb{B}$ which maps the $i\omega$-bit input C and the 8ω-bit extended key K into the j-th LSB of the word $\mathcal{F}_i(C, \mathsf{K})$. Any of these keyed functions can be put into the framework from [7] explained in section 2. The next step is to consider a partitioning $C = (U; W)$ with l-bit segment U and $(i\omega - l)$-bit segment W to derive the (hopefully weaker) functions $\Gamma_{i,j}^U : \mathbb{B}^{i\omega - l} \times \mathbb{B}^{8\omega} \to \mathbb{B}$ where $\Gamma_{i,j}^U$ is the superpoly in $\mathcal{F}_{i,j}$ corresponding to the cube vector U. Whenever there is no ambiguity we drop the superscript or the subscripts. We may also use $\Gamma_{i,j}^U[\omega]$ in some cases to emphasize the word-size. We are now ready to give our simulation results.

Note: Instead of giving giving the variables of cube vector U we give the bit numbers. For example for $\omega = 16$, the set $\{0, 18, 31, 32\}$ stands for the cube vector $U = (c_{1,0}, c_{2,2}, c_{2,15}, c_{3,0})$.

Example 1. For all possible common word sizes ($\omega = 8, 16, 32$ or 64) we have been able to find some i, j and U such that Γ is independent of W and only depends on three key bits $k_{0,0}$, $k_{1,0}$ and $k_{2,0}$. Table 1 shows some of these quite striking relations. We also found relations $\Gamma_{1,0}^{\{3\}}[8] = 1 + k_{2,0}$ and $\Gamma_{1,0}^{\{6,7,8,9,10\}}[16] = 1 + k_{0,0}$ involving only one key bit. For $\omega = 64$, the three relations in Table 1 give 1.75 bits of information about $(k_{0,0}, k_{1,0}, k_{2,0})$.

Table 1. Simple relations on three key bits $(k_{0,0}, k_{1,0}, k_{2,0})$

ω	i	j	U	$\Gamma^U_{i,j}[\omega]$
8	2	0	$\{2\}$	$1 + k_{0,0}k_{1,0} + k_{0,0}k_{2,0} + k_{1,0}k_{2,0}$
16	3	0	$\{5\}$	$1 + k_{0,0}k_{1,0} + k_{2,0} + k_{0,0}k_{1,0}k_{2,0}$
16	3	0	$\{10\}$	$1 + k_{0,0} + k_{1,0}k_{2,0} + k_{0,0}k_{1,0}k_{2,0}$
32	5	0	$\{11\}$	$1 + k_{0,0}k_{1,0} + k_{2,0} + k_{0,0}k_{1,0}k_{2,0}$
32	16	0	$\{96, 97, 98\}$	$1 + k_{0,0} + k_{2,0} + k_{0,0}k_{2,0}$
64	11	0	$\{21\}$	$1 + k_{0,0}k_{1,0} + k_{2,0} + k_{0,0}k_{1,0}k_{2,0}$
64	11	0	$\{42\}$	$1 + k_{0,0} + k_{1,0}k_{2,0} + k_{0,0}k_{1,0}k_{2,0}$
64	12	0	$\{20\}$	$1 + k_{0,0}k_{1,0} + k_{0,0}k_{2,0} + k_{1,0}k_{2,0}$

A more detailed analysis of the functions $\Gamma^U_{i,j}[\omega](W, \mathsf{K})$ for different values of i, j and U reveals that many of these functions depend on only few bits of their $(i\omega - l)$-bit and 8ω-bit arguments. Let t_w and t_k respectively denote the number of bits of W and K which Γ effectively depends on. In addition let t'_k out of t_k bits come from K and the remaining $t_s = t_k - t'_k$ bits from $\mathsf{S}^\star$ (remember $\mathsf{K} = (K, \mathsf{S}^\star)$). Table 2 shows these values for some of these functions.

Having in mind what we mentioned in section 2 and being too optimistic, we give the following proposition.

Table 2. Effective number of bits of each argument which Γ depends on. Note that the functions having the same number of effective bits do not necessarily have the same involved variables.

ω	i	j	U	t_k	t_w	t'_k	t_s	comment
8	1	0	$\emptyset$	20	8	9	11	
16	1	0	$\emptyset$	40	16	17	23	
32	1	0	$\emptyset$	80	32	33	47	
64	1	0	$\emptyset$	160	64	65	95	
8	1	0	$\{1\}$	9	5	3	6	
16	1	0	$\{u\}$	18	11	6	12	$8 \le u \le 10$
32	1	0	$\{u\}$	42	23	14	28	$16 \le u \le 20$
64	1	0	$\{u\}$	90	51	30	60	$32 \le u \le 42$
8	3	0	$\{8\}$	4	6	4	0	
8	3	0	$\{18\}$	5	7	5	0	
16	7	0	$\{16\}$	17	58	17	0	
16	7	0	$\{34\}$	16	52	16	0	
16	7	0	$\{33, 34\}$	12	33	12	0	
16	7	0	$\{38, 39\}$	12	30	12	0	
32	15	0	$\{32\}$	41	293	41	0	
32	15	0	$\{66\}$	40	279	40	0	
32	15	0	$\{76, 77\}$	36	231	36	0	
64	31	0	$\{64\}$	89	1274	89	0	
64	31	0	$\{130\}$	88	1243	88	0	
64	31	0	$\{129, 130\}$	84	1158	84	0	
64	31	0	$\{150, 151\}$	84	1155	84	0	

Proposition 1. *If a function $\Gamma^U_{i,j}$ is random-looking enough, recovering the t_k unknown bits of the extended key takes expected time $i \times t_k \times 2^{l+t_k}$.*

The unity of time is processing one ciphertext word of the underlined self-synchronizing stream cipher. The factors 2^l and i come from the following facts: computing Γ from $\mathcal{F}_i$ needs 2^l evaluations of $\mathcal{F}_i$ (remember $\Gamma^U_{i,j}(W,\mathsf{K}) = \bigoplus_{U\in\mathbb{B}^l} \mathcal{F}_{i,j}((U;W),\mathsf{K})$) and computing $\mathcal{F}_i$ needs i iterations of the cipher.

Even if the ideal condition of Proposition 1 is not satisfied, the only thing which is not guaranteed is that the t_k involved unknown bits are uniquely determined. Yet some information about them can be achieved. Refer to the note in section 2 regarding the equivalence classes.

Example 2. Take the relation $\Gamma^{\{18\}}_{3,0}[8](W,\mathsf{K})$ from Table 2. This particular function depends on $t_k = 5$ bits $(k_{0,0}, k_{0,1}, k_{1,0}, k_{2,0}, k_{2,1})$ of the key and on $t_w = 7$ bits $(c_{1,4}, c_{1,5}, c_{1,6}, c_{2,0}, c_{2,1}, c_{2,5}, c_{2,6})$ of the ciphertext. The ANF of this function is:

$$\begin{aligned}\Gamma^{\{18\}}_{3,0}[8] = {} & 1 + k_{0,0}k_{0,1} + k_{0,0}k_{0,1}k_{2,0} + k_{2,0}k_{2,1} + k_{0,0}c_{1,4} + \\ & k_{0,0}k_{2,0}c_{1,4} + k_{0,0}k_{1,0}c_{1,5} + k_{0,0}k_{1,0}k_{2,0}c_{1,5} + \\ & k_{0,0}c_{1,6} + k_{0,0}k_{2,0}c_{1,6} + k_{2,0}c_{2,0} + k_{0,0}k_{2,0}c_{2,0} + \\ & k_{2,0}c_{2,1} + c_{2,0}c_{2,1} + k_{0,0}c_{2,0}c_{2,1} + k_{2,0}c_{2,0}c_{2,1} + \\ & k_{0,0}k_{2,0}c_{2,0}c_{2,1} + k_{1,0}k_{2,0}c_{2,5} + k_{2,0}c_{2,6}.\end{aligned} \tag{7}$$

This equation can be seen as a system of $2^{t_w} = 128$ equations versus $t_k = 5$ unknowns. Our analysis of this function shows that only 48 of the equations are independent which on average can give 3.5 bits of information about the five unknown bits (2 bits of information for 25% of the keys and 4 bits for the remaining 75% of the keys).

Example 3. Take the relation $\Gamma^{\{33,34\}}_{7,0}[16](W,\mathsf{K})$ from Table 2. This particular function depends on $t_k = 12$ key bits and on $t_w = 33$ ciphertext bits. Our analysis of this function shows that on average about 2.41 bits of information about the 12 key bits can be achieved (10 bits of information for 12.5% of the keys, 3 bits for 25% of the keys and 0.67 bits about the remaining 62.5% of the keys).

Example 4. Take the relation $\Gamma^{\{38,39\}}_{7,0}[16](W,\mathsf{K})$ from Table 2. This particular function depends on $t_k = 12$ key bits and on $t_w = 30$ ciphertext bits. Our analysis of this function shows that on average about 1.94 bits of information about the 12 key bits can be achieved (10 bits of information for 12.5% of the keys, 3 bits for another 12.5% of the keys and 0.42 bits for the remaining 75% of the keys).

Example 5. Take the relation $\Gamma^{\{34\}}_{7,0}[16](W,\mathsf{K})$ from Table 2. This particular function depends on $t_k = 16$ key bits and on $t_w = 52$ ciphertext bits. Our analysis of this function shows that on average about 5.625 bits of information about the 16 key bits can be achieved (13 bits of information for 25% of the keys, 11 bits for 12.5% of the keys, 4 bits for another 12.5% of the keys, and 1 bit for the remaining 50% of the keys).

For larger values of i we expect Γ to fit better the ideal situation of Proposition 1. Therefore, we give the following claim about the security of the 64-bit version of Klimov-Shamir's proposal.

Table 3. Finding weak ciphertext bits in a systematic way (for $\Gamma_{1,0}[64]$)

U	K	W	t_k	t_w	t'_k	t_s	Time
$\{41\}$	$\{0-19, 21-30, 384-414, 449-467, 469-478\}$	$\{0-9, 22-40, 42-63\}$	90	51	30	60	$2^{97.5}$
$\{9, 41\}$	$\{0-19, 21-29, 384-413, 449-467, 469-477\}$	$\{0-8, 22-40, 42-63\}$	87	50	29	58	$2^{95.4}$
$\{8, 9, 41\}$	$\{0-19, 21-28, 384-412, 449-467, 469-476\}$	$\{0-7, 22-40, 42-63\}$	84	49	28	56	$2^{93.4}$
$\{7-9, 41\}$	$\{0-19, 21-27, 384-411, 449-467, 469-475\}$	$\{0-6, 22-40, 42-63\}$	81	48	27	54	$2^{91.3}$
$\{6-9, 41\}$	$\{0-19, 21-26, 384-410, 449-467, 469-474\}$	$\{0-5, 22-40, 42-63\}$	78	47	26	52	$2^{89.3}$
$\vdots$							$\vdots$
$\{1-9, 41\}$	$\{0-19, 21, 384-405, 449-467, 469\}$	$\{0, 22-40, 42-63\}$	63	42	21	42	$2^{79.0}$
$\{1-9, 40, 41\}$	$\{0-18, 21, 384-405, 449-466, 469\}$	$\{0, 22-39, 42-63\}$	61	41	20	41	$2^{77.9}$
$\{1-9, 39-41\}$	$\{0-17, 21, 384-405, 449-465, 469\}$	$\{0, 22-38, 42-63\}$	59	40	19	40	$2^{76.9}$
$\vdots$							$\vdots$
$\{1-9, 34-41\}$	$\{0-12, 21, 384-405, 449-460, 469\}$	$\{0, 22-33, 42-63\}$	49	35	14	35	$2^{71.7}$
$\{1-9, 33-41\}$	$\{0-11, 21, 384-405, 449-459, 469\}$	$\{0, 22-32, 42-63\}$	47	34	13	34	$2^{70.5}$
$\{1-9, 32-41\}$	$\{0-10, 21, 384-405, 449-458, 469\}$	$\{0, 22-31, 42-63\}$	45	33	12	33	$2^{69.5}$

Proposition 2. *We expect each of the functions* $\Gamma_{31,0}^{\{129,130\}}[64]$ *and* $\Gamma_{31,0}^{\{150,151\}}[64]$ *to reveal a large amount of information about the corresponding* $t_k = 84$ *involved key bits for a non-negligible fraction of the keys. The required computational time is* $31 \times 84 \times 2^{2+84} \approx 2^{92.8}$.

In [7] the bits of the set U were called *weak IV bits*. With the same terminology, here we call them *weak ciphertext bits*. How to find these weak bits was raised as an open question in [7]. In the next section we present a systematic procedure to find weak ciphertext bits, with the consequence of improving Proposition 2.

6 Towards a Systematic Approach to Find Weak Ciphertext Bits

The idea is to start with a set U and extend it gradually. At each step we examine all the ciphertext bits which Γ^U depends on, to choose an extended U for the next step which results in a Γ which depends on the least number of key bits. Table 3 shows our simulation results by starting from function $\Gamma_{1,0}^{\{41\}}[64]$ from Table 2 which effectively depends on $t_k = 90$ extended key bits and $t_w = 51$ ciphertext bits. Similar to Proposition 2, one expects each of the functions $\Gamma_{1,0}^{U}[64]$ in Table 3 to reveal a large amount of information about the corresponding t_k involved extended key bits (including t'_k effective key bits) for a non-negligible fraction of the keys in time $t_k 2^{l+t_k}$, as indicated in the last column. In particular by starting from the function in the bottom of Table 3, (the promised large amount of information about) the involved $t'_k = 12$ key bits and $t_s = 33$ internal state bits can be gained in time $2^{69.5}$ (for a non-negligible fraction of the keys). Notice, that once we have the correct value for the unknown extended key for some function in Table 3, those of the previous function can be recovered by little effort. Therefore we present the following proposition.

Proposition 3. *We expect that by starting from* $\Gamma_{1,0}^{\{1-9,32-41\}}[64]$ *and going backwards to* $\Gamma_{1,0}^{\{41\}}[64]$ *as indicated in Table 3, a large amount of information about the involved* $t_k = 90$ *unknown bits (including* $t'_k = 30$ *effective key bits) is revealed for a non-negligible fraction of the keys in time* $2^{69.5}$.

Remark 2. By combining the results of different functions Γ one can get better results. Finding an optimal combination demands patience and detailed examination of different Γ's. We make this statement clearer by an example as follows. Detailed analysis of $\Gamma_{31,0}^{\{129,130\}}[64]$ and $\Gamma_{31,0}^{\{150,151\}}[64]$ shows that the key bits which they depend on are $\{0-27, 64-90, 128-156\}$ and $\{0-28, 64-90, 128-155\}$, respectively. These two functions have respectively 27 and 28 bits in common with the 30 key bits $\{0-19, 21-30\}$ involved in $\Gamma_{1,0}^{\{41\}}[64]$. They also include the key bits $\{0, 32, 64\}$ for which 1.75 information can be easily gained according to Ex. 1. Taking it altogether it can be said that a large amount of information about the 88 key bits $\{0-30, 32, 64-90, 128-156\}$ can be achieved in time $2^{69.5}$ with a non-negligible probability.

7 Conclusion

In this work we proposed a new analysis method for self-synchronizing stream ciphers which was applied to Klimov-Shamir's example of a construction of a T-function based

self-synchronizing stream cipher. We did not fully break this proposal but the strong key leakage demonstrated by our results makes us believe a total break is not out of reach. In future design of self-synchronizing stream ciphers one has to take into account and counter potential key leakage.

Acknowledgement. We would like to thank Martijn Stam for his helpful editorial comments.

References

1. Aumasson, J.-P., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 470–488. Springer, Heidelberg (2008)
2. Biham, E., Shamir, E.: Differential Cryptanalysis of DES-like Cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer, Heidelberg (1991)
3. Daemen, J., Knudsen, L.R., Rijmen, V.: The Block Cipher Square. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997)
4. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. Cryptology ePrint Archive, Report 385 (2008)
5. Englund, H., Johansson, T., Turan, M.S.: A Framework for Chosen IV Statistical Analysis of Stream Ciphers. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 268–281. Springer, Heidelberg (2007)
6. eSTREAM - The ECRYPT Stream Cipher Project, http://www.ecrypt.eu.org/stream
7. Fischer, S., Khazaei, S., Meier, W.: Chosen IV Statistical Analysis for Key Recovery Attacks on Stream Ciphers. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 236–245. Springer, Heidelberg (2008)
8. Klimov, A., Shamir, A.: New Applications of T-Functions in Block Ciphers and Hash Functions. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 18–31. Springer, Heidelberg (2005)
9. Knudsen, L.R.: Truncated and Higher Order Differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995)
10. Knudsen, L.R., Wagner, D.: Integral Cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 112–127. Springer, Heidelberg (2002)
11. O'Neil, S.: Algebraic Structure Defectoscopy. Cryptology ePrint Archive, Report 2007/378 (2007), http://www.defectoscopy.com
12. Vielhaber, M.: Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack. Cryptology ePrint Archive, Report 2007/413

Analysis of RC4 and Proposal of Additional Layers for Better Security Margin

Subhamoy Maitra[1] and Goutam Paul[2]

[1] Applied Statistics Unit, Indian Statistical Institute,
Kolkata 700 108, India
subho@isical.ac.in
[2] Department of Computer Science and Engineering,
Jadavpur University, Kolkata 700 032, India
goutam_paul@cse.jdvu.ac.in

Abstract. In this paper, the RC4 Key Scheduling Algorithm (KSA) is theoretically studied to reveal non-uniformity in the expected number of times each value of the permutation is touched by the indices i, j. Based on our analysis and the results available in the literature regarding the existing weaknesses of RC4, few additional layers over the RC4 KSA and RC4 Pseudo-Random Generation Algorithm (PRGA) are proposed. Analysis of the modified cipher (we call it RC4$^+$) shows that this new strategy avoids existing weaknesses of RC4.

Keywords: Bias, Cryptography, Keystream, KSA, PRGA, RC4, Secret Key, Stream Cipher.

1 Introduction and Motivation

RC4 is one of the most popular and efficient stream ciphers. The data structure of RC4 consists of an array S of size N (typically, 256), which contains a permutation of the integers $\{0, \ldots, N-1\}$, two indices i (deterministic) and j (pseudo-random) and a secret key array K. Given a secret key key of l bytes (typically 5 to 32), the array K of size N is such that $K[y] = key[y \bmod l]$ for any y, $0 \leq y \leq N-1$.

There are two components of the cipher: the Key Scheduling Algorithm (KSA) that turns an identity permutation into a random-looking permutation and the Pseudo-Random Generation Algorithm (PRGA) that generates keystream bytes which get XOR-ed with the plaintext bytes to generate ciphertext bytes. All additions in both the KSA and the PRGA are additions modulo N.

KSA

Initialization:
For $i = 0, \ldots, N-1$
$S[i] = i$;
$j = 0$;
Scrambling:
For $i = 0, \ldots, N-1$
$j = (j + S[i] + K[i])$;
Swap($S[i]$, $S[j]$);

PRGA

Initialization:
$i = j = 0$;
Keystream Generation Loop:
$i = i + 1$;
$j = j + S[i]$;
Swap($S[i]$, $S[j]$);
$t = S[i] + S[j]$;
Output $z = S[t]$;

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 27–39, 2008.

The literature on RC4 cryptanalysis is quite rich. There have been several works on the reconstruction of the permutation looking at the keystream output bytes [10,31,19]. Of these, the latest one [19] achieves a complexity of 2^{241}, rendering RC4 insecure with key length beyond 30 bytes. Further, knowing the permutation, it is also possible to get certain information on the secret key [23,2,1].

Apart from these, there exist several other works [5,22,13,14,15,16,26,27] on the weaknesses of the RC4 PRGA. However, all of these exploit the initial keystream bytes only. According to [20], if some amount of initial keystream bytes are thrown away, then RC4 is quite safe to use. Moreover, it is argued in [13,25] that many biases in the PRGA are due to the propagation of the biases in the KSA via Glimpse Theorem [8,15]. These biases in the keystream would disappear, if one could remove the corresponding biases in the permutation during the KSA.

In this paper, we discuss several weaknesses of RC4 and suggest remedies to overcome them. During last few years, there have been efforts, e.g., VMPC [35], RC4A [27], RC4(n, m) [6] etc. on the modification of RC4 towards further improvement and there also exist distinguishing attacks on them [18,32,33]. This shows that there is significant interest in the cryptographic community for analysis and design of RC4 and its modifications. However, in all of these ciphers, the design is modified to a great extent relative to RC4. We keep the RC4 structure as it is and add a few more operations to strengthen the cipher. Thus, we attempt to exploit the good points of RC4 and then provide some additional features for a better security margin.

One may argue that concentrating on the *eSTREAM* candidates [3] is more practical than modifying RC4. However, the *eSTREAM* candidates have complicated structure in general and they work on word (32 bit) oriented manner. Our goal is to keep the simple structure of RC4 and add a few steps to it to have a byte oriented stream cipher with further strength. The existing literature on RC4 reveals that in spite of having a very simple description, the cipher possesses nice combinatorial structures in the shuffle-exchange paradigm. Our design retains this elegant property of RC4 and at the same time removes the existing weaknesses.

2 Movement Frequency of Permutation Values

Before we go into the technicalities, let us introduce a few notations. We denote the initial identity permutation by S_0 and the permutation at the end of the r-th round of the KSA by S_r, $1 \leq r \leq N$. Note that $r = y+1$, when the deterministic index i takes the value y, $0 \leq y \leq N-1$. Thus, the permutation after the KSA will be denoted by S_N. By j_r, we denote the value of the index j after it is updated in round r. Also, let $f_y = \frac{y(y+1)}{2} + \sum_{x=0}^{y} K[x]$, that would be referred frequently in the subsequent discussions.

We observe that many values in the permutation are touched once with a very high probability by the indices i, j during the KSA.

Theorem 1. *The probability that a value v in the permutation is touched exactly once during the KSA by the indices i, j, is given by $\frac{2v}{N} \cdot (\frac{N-1}{N})^{N-1}$, $0 \leq v \leq N-1$.*

Proof. Initially, v is located at index v in the permutation. It is touched exactly once in one of the following two ways.

1. v is not touched by any of $\{j_1, j_2, \ldots, j_v\}$ in the first v rounds. In round $v+1$, when i becomes v, the value v at index v is moved to the left by j_{v+1} due to the swap and remains there until the end of KSA. Thus, the probability contribution of this part is $(\frac{N-1}{N})^v \cdot \frac{v}{N} \cdot (\frac{N-1}{N})^{N-v-1} = \frac{v}{N} \cdot (\frac{N-1}{N})^{N-1}$.
2. For some t, $1 \leq t \leq v$, it is not touched by any of $\{j_1, j_2, \ldots, j_{t-1}\}$; then it is touched for the first time by $j_t = v$ in round t and hence is moved to index $t-1$; and it is not touched by any one of the subsequent $(N-t)$ many j values. The probability contribution of this part is $\sum_{t=1}^{v} (\frac{N-1}{N})^{t-1} \cdot \frac{1}{N} \cdot (\frac{N-1}{N})^{N-t} = \frac{v}{N} \cdot (\frac{N-1}{N})^{N-1}$.

Adding the above two contributions, we get the result. □

Using similar arguments one could compute the probability that a value is touched exactly twice, thrice and in general x times, during the KSA. However, the computation would be tedious and complicated for $x > 1$. A more natural measure of this asymmetric behaviour would be the expected number of times each value in the permutation is touched during the KSA. This is computed in the next theorem.

Theorem 2. *The expected number of times a value v in the permutation is touched by the indices i, j during the KSA is given by $E_v = 1 + (\frac{2N-v}{N}) \cdot (\frac{N-1}{N})^v$, $0 \leq v \leq N-1$.*

Proof. Let $x_{v,y} = 1$, if the value v is touched by the indices i, j in round $y+1$ of the KSA (i.e., when $i = y$); otherwise, let $x_{v,y} = 0$, $0 \leq v \leq N-1$, $0 \leq y \leq N-1$. Then the number of times v is touched by i, j during the KSA is given by $X_v = \sum_{y=0}^{N-1} x_{v,y}$. In any round $y+1$, any value v is touched by j with a probability $\frac{1}{N}$. To this, we need to add the probability of v being touched by i, in order to find $P(x_{v,y} = 1)$. Now, v is touched by the index i in round $y+1$, iff $S_y[y] = v$. We consider three possible ways in which $S_y[y]$ can become v.

1. Case $y < v$. Initially, the value v was situated in index v. In order for v to move from index v to index $y < v$, either v has to be touched by i and y has to be touched by j, or vice versa, during the first y rounds. But this is not possible, giving $P(S_y[y] = v) = 0$.
2. Case $y = v$. We would have $S_v[v] = v$, if v is not touched by any of $\{j_1, j_2, \ldots, j_v\}$ in the first v rounds, the probability of which is $(\frac{N-1}{N})^v$.
3. Case $y > v$. Once $S_v[v] = v$, the swap in the next round moves the value v to a random location j_{v+1}, giving $P(S_{v+1}[y] = v) = (\frac{N-1}{N})^v \cdot \frac{1}{N}$. For all

$y > v$, until y is touched by the deterministic index i, i.e., until round $y+1$, v will remain randomly distributed. Hence, for all $y > v$, $P(S_y[y] = v) = P(S_{v+1}[y] = v) = \frac{1}{N}(\frac{N-1}{N})^v$.

Noting that $E(x_{v,y}) = P(x_{v,y} = 1) = \frac{1}{N} + P(S_y[y] = v)$, we have $E_v = E(X_v) = \sum_{y=0}^{N-1} E(x_{v,y}) = 1 + \sum_{y=0}^{v-1} P(S_y[y] = v) + P(S_v[v] = v) + \sum_{y=v+1}^{N-1} P(S_y[y] = v) = 1 + (\frac{2N-v}{N}) \cdot (\frac{N-1}{N})^v$ (adding the three-part contributions). □

We find that E_v decreases from 3.0 to 1.37, as v increases from 0 to 255. To demonstrate how close the experimental values of the expectations match with our theoretical values, we perform 100 million runs the KSA, with random key of 16 bytes in each run. The experimental results correspond to the theoretical formula, as summarised in the first two rows of Table 1 in Section 3.3.

In [24,1], it is shown that the probabilities $P(j_{y+1} = S_N^{-1}[y])$ increase with increasing y. This is connected to the above decreasing pattern in the expectations. In the first half of the KSA, i.e., when y is small, the values $v = S[y]$ are thrown more to the right with high probability by the index j_{y+1} due to the swap and hence are touched again either by the deterministic index i or by the pseudo-random index j in the subsequent rounds. On the other hand, in the second half of the KSA, i.e., when $y \geq 128$, the values $v = S[y]$ are thrown more to the left by the index j_{y+1} due to the swap and hence are never touched by i in the subsequent rounds, and may be touched by j with a small probability.

Towards designing a key scheduling algorithm in shuffle-exchange paradigm, it is important that each value in the permutation is touched (and therefore moved with probability almost one) sufficient number of times. In such a case, it will be harder to guess the values of j for which a permutation byte is swapped. In RC4 KSA, there are many permutation bytes which are swapped only once with a high probability, leading to information leakage from S_N regarding the secret key bytes. We keep this in mind while designing the modified KSA in the next section.

3 Removing the Weaknesses of KSA

In this section, we first look into what are the existing weaknesses of the RC4 KSA, followed by suggestions to remove them. We propose a new version of the KSA and study its security issues.

3.1 Existing Weaknesses

Many works have explored the RC4 KSA and discovered its different weaknesses. Here we present an overview of these results.

(1) In [28], it was empirically shown that the probabilities $P(S_N[y] = f_y)$ decrease from 0.37 for $y = 0$ to 0.006 for $y = 48$ (with $N = 256$) and beyond that settle down to 0.0039 ($\approx \frac{1}{256}$). Later, in [23], explicit formula for these

probabilities for all $y \in [0, \ldots, N-1]$ were theoretically derived. This result was further used in [23,2,1] to recover the secret key from the final permutation S_N after the KSA.

(2) In RC4 KSA, the update rule is $j = (j + S[i] + K[i])$. The work [23] showed that for a certain class of update functions which update j as a function of "the permutation S and j in the previous round" and "the secret key K", it is always possible to construct explicit functions of the key bytes which the permutation at every stage of the KSA will be biased to.

(3) It has been shown in [13] that the bytes $S_N[y]$, $S_N[S_N[y]]$, $S_N[S_N[S_N[y]]]$, and so on, are biased to f_y. In particular, they showed that $P(S_N[S_N[y]] = f_y)$ decreases from 0.137 for $y = 0$ to 0.018 for $y = 31$ and then slowly settles down to 0.0039 (beyond $y = 48$).

(4) Analysis in [24,1] shows that inverse permutations $S_N^{-1}[y]$, $S_N^{-1}[S_N^{-1}[y]]$, and so on are biased to j_{y+1}, and in turn, to f_y.

(5) It was shown for the first time in [17, Chapter 6] and later investigated further in [20,25] that each permutation byte after the KSA is significantly biased (either positive or negative) towards many values in the range $0, \ldots, N-1$. For each y, $0 \leq y \leq N-2$, $P(S_N[y] = v)$ is maximum at $v = y+1$ and this maximum probability ranges approximately between $\frac{1}{N}(1 + \frac{1}{3})$ and $\frac{1}{N}(1 + \frac{1}{5})$ for different values of y, with $N = 256$.

(6) The work [4] showed for the first time that RC4 can be attacked when used in the IV mode (e.g. WEP [11]). Subsequently, there have been series of improvements [15,9,30,34] in this direction, exploiting the propagation of weak key patterns to the keystream output bytes.

3.2 Proposal for KSA$^+$: A Revised KSA

In this section, we present a modified design (called KSA$^+$) that removes the weaknesses of RC4 KSA discussed in Section 3.1. The evaluation for such a design is presented in Section 3.3. In this case, we will name the permutation after the KSA$^+$ as S_{N^+}.

We propose a three-layer key scheduling followed by the initialization. The initialization and basic scrambling in the first layer are the same as the original RC4 KSA.

Initialization

For $i = 0, \ldots, N-1$
    $S[i] = i$;
$j = 0$;

Layer 1: Basic Scrambling

For $i = 0, \ldots, N-1$
    $j = (j + S[i] + K[i])$;
    Swap($S[i], S[j]$);

In the second layer, we scramble the permutation further using IV's. According to [7], for stream ciphers using IV's, if the IV is shorter than the key, then the algorithm may be vulnerable against the Time Memory Trade-Off attack. Thus, in this effort, we choose the IV size as the same as the secret key length. The deterministic index i moves first from the middle down to the left end and then from the middle upto the right end. In our scheme, an l-byte IV, denoted by an array $iv[0, \ldots, l-1]$, is used from index $\frac{N}{2} - 1$ down to $\frac{N}{2} - l$ during the left-ward movement and the same IV is repeated from index $\frac{N}{2}$ up to $\frac{N}{2} + l - 1$ during

the right-ward movement. Here, we assume that N is even, which is usually the case in standard RC4. For ease of description, we use an array IV of length N with $IV[y] = 0$ for those indices which are not used with IV's.

For $N = 256$ and $l = 16$, this gives a placement of $16 \times 2 = 32$ many bytes in the middle of the IV array spanning from index 112 to 143. This is to note that repeating the IV bytes will create a dependency so that one cannot choose all the 32 bytes freely to find some weakness in the system as one byte at the left corresponds to one byte at the right (when viewed symmetrically from the middle of an N-byte array). Further, in two different directions, the key bytes are added with the IV bytes in an opposite order. Apart from the $2l$ many operations involving the IV, the rest of $N - 2l$ many operations are without the involvement of IV in Layer 2. This helps in covering the IV values and chosen IV kind of attacks will be hard to mount.

Layer 2: Scrambling with IV
For $i = \frac{N}{2} - 1$ down to 0
$j = (j + S[i]) \oplus (K[i] + IV[i])$;
Swap$(S[i], S[j])$;
For $i = \frac{N}{2}, \ldots, N - 1$
$j = (j + S[i]) \oplus (K[i] + IV[i])$;
Swap$(S[i], S[j])$;

Layer 3: Zigzag Scrambling
For $y = 0, \ldots, N - 1$
If $y \equiv 0 \bmod 2$ then $i = \frac{y}{2}$;
Else $i = N - \frac{y+1}{2}$;
$j = (j + S[i] + K[i])$;
Swap$(S[i], S[j])$;

In the third and final layer, we perform more scrambling in a zig-zag fashion, where the deterministic index i takes values in the following order: 0, 255, 1, 254, 2, 253, ..., 125, 130, 126, 129, 127, 128. In general, if y varies from 0 to $N - 1$ in steps of 1, then $i = \frac{y}{2}$ or $N - \frac{y+1}{2}$ depending on y is even or odd respectively. Introducing more scrambling steps definitely increases the cost of the cipher. The running time of the KSA$^+$ is around three times that of RC4 KSA, because there are three similar scrambling layers instead of one, each having N iterations. As the key scheduling is run only once, this will not affect the performance of the cipher much.

3.3 Analysis of KSA$^+$ with Respect to RC4 KSA

In this section, we discuss how the new design avoids many weaknesses of the original RC4 KSA. We performed extensive experiments to verify that KSA$^+$ is indeed free from the weaknesses of the RC4 KSA. In all our experiments that are presented in this section, we use null IV, i.e., $iv[y] = 0$ for all y. We could not find any weakness with such null IV as well as with randomly chosen IV's.

Removal of Secret Key Correlation with the Permutation Bytes: Let us first discuss on Layer 2 of the KSA+. The deterministic index i is moved from the middle to the left end so that the values in the first quarter of the permutation, which were biased to linear combination of the secret key bytes, are swapped. This helps in removing the biases in the initial values of Item (1) described in Section 3.1. This is followed by a similar operation in the second half of the permutation to get rid of the biases of the inverse permutation as

described in Item (4). Next, the XOR operation helps further to wipe out these biases. The biases considering the nested indexing mentioned in Item (3) and Item (4) arise due to the biases of direct indexing. So, the removal of the biases at the direct indices of S_N and S_N^{-1} gets rid of those at the nested indices also.

The bias of Item (2), which is a generalization of the bias of Item (1), originates from the incremental update of j which helps to form a recursive equation involving the key bytes. In the new design, the bit-by-bit XOR operation as well as the zig-zag scrambling in Layer 3 prevents in forming such recursive equations connecting the key bytes and the permutation bytes.

We could not find any correlation between $S_{N^+}[y]$ (also $S_{N^+}[S_{N^+}[y]]$, $S_{N^+}[S_{N^+}[S_{N^+}[y]]]$, ...) with f_y. We believe that with our design, it is not possible to get correlation of the permutation bytes with any function combining the secret key bytes.

In Section 2, the relation between the biases of the inverse permutation and the movement frequency of the permutation values has been discussed in detail. The following experimental results show that, such weaknesses of RC4 KSA are absent in our design. Averaging over 100 million runs of KSA$^+$ with 16 bytes key in each run, we find that as v increases from 0 to 255, E_v decreases from 4.99 to 3.31 after the end of Layer 2 and from 6.99 to 5.31 after the end of Layer 3. Table 1 shows the individual as well as the incremental effect of each of Layer 2 and Layer 3, when they act upon the identity permutation S_0 and the permutation S_N obtained after Layer 1. The data illustrate that the effect of Layer 2 or Layer 3 over identity permutation S_0 is similar as Layer 1. However, after Layer 1 is over (when we have somewhat random permutation S_N coming out of RC4 KSA), each of Layer 2 and Layer 3 individually enforces each value in the permutation to be touched uniformly (approximately twice) when the average is considered over many runs.

Table 1. Average, Standard Deviation, Maximum and Minimum of the expectations E_v over all v between 0 and 255. Here Lr means Layer r, $r = 1, 2, 3$.

		avg	sd	max	min
RC4 KSA (KSA$^+$ L1)	Theory	2.0025	0.4664	3.0000	1.3700
	Experiment	2.0000	0.4655	2.9959	1.3686
KSA$^+$ L2 (Experiment)	L2 on S_0	2.0000	0.4658	2.9965	1.3683
	L2 on S_N	2.0000	0.0231	2.0401	1.9418
	L1 + L2	4.0000	0.4716	4.9962	3.3103
KSA$^+$ L3 (Experiment)	L3 on S_0	2.0000	0.4660	3.0000	1.3676
	L3 on S_N	2.0000	0.0006	2.0016	1.9988
	L1 + L2 + L3	6.0000	0.4715	6.9962	5.3116

Uniform values of the expectations can be achieved easily with normal RC4, by keeping a count of how many times each element is touched and performing additional swaps involving the elements that have been touched less number of times. However, this will require additional space and time. In normal RC4, many permutation elements are touched only once (especially those towards the right end of the permutation), leaking information on j in the inverse permutation. Our target is to prevent this by increasing the number of times each element is touched, without keeping any additional space such as a counter. The data in Table 1 show that this purpose is served using our strategy.

How Random is S_{N^+}: Now we present experimental evidences to show how the biases of Item (5) in RC4 KSA are removed. We compare the probabilities $P(S[u] = v)$ for $0 \leq u, v \leq 255$ from standard KSA and our KSA$^+$. All the experiments are performed with 100 million runs, each with a randomly chosen secret key of length 16 bytes and null IV.

Experimental results show that there exists some non-uniformities after Layer 2, which is completely removed after Layer 3. The maximum and minimum values of the probabilities as well as the standard deviations summarised in Table 2 elaborate this fact further.

Table 2. Average, Standard Deviation, Maximum and Minimum of the Probabilities $P(S[u] = v)$ over all u and v between 0 and 255. Note that $\frac{1}{N} = 0.003906$ for $N = 256$.

		avg	sd	max	min
RC4 KSA	Theory [25, Theorem 1]	0.003901	0.000445	0.005325	0.002878
	Experiment	0.003906	0.000448	0.005347	0.002444
KSA$^+$ (Experiment)	After Layer 2	0.003906	0.000023	0.003983	0.003803
	After Layer 3	0.003906	0.000006	0.003934	0.003879

In [17, Page 67], it was mentioned that the RC4 KSA need to be executed approximately 6 times in order to get rid of these biases. Whereas, in our case, we need to run KSA effectively 3 times.

On Introducing the IV's: The IV-mode attacks, mentioned in Item (6) of Section 3.1, succeed because in the original RC4, IV's are either prepended or appended with the secret key. As the Layer 2 shows, in KSA$^+$, we use the IV's in the middle and also the corresponding key bytes are added in the updation of j. In Layer 2, $2l$ many operations involve IV values, but $N - 2l$ many operations do not. Moreover, after the use of IV, we perform a third layer of zig-zag scrambling where no use of IV is made. This almost eliminates the possibility of chosen IV attack once the key scheduling is complete.

SSL protocol bypasses the WEP attack [4] by generating the encryption keys used for RC4 by hashing (using both MD5 and SHA-1) the secret key and the IV together, so that different sessions have unrelated keys [29]. Since our KSA$^+$ is believed to be free from the IV-weaknesses, it can be used without employing hashing. Thus, the cost of hashing can be utilized in the extra operations in Layer 2 and Layer 3. This conforms to our design motivation to keep the basic structure of RC4 KSA and still avoid the weaknesses.

On Retaining the Standard KSA in Layer 1: One may argue that Layer 1 is not necessary and Layer 2, 3 would have taken care of all the existing weaknesses of RC4. While this may be true, these two layers, when operated on identity permutation, might introduce some new weaknesses not yet known. It is a fact that RC4 KSA has some weaknesses, but it also reduces the key correlation with the permutation bytes and other biases at least to some extent compared to the beginning of the KSA. In the process, it randomizes the permutation to a certain extent. The structure of RC4 KSA is simple and elegant and easy to analyze. We first let this KSA run over the identity permutation, so that we can target the exact biases that are to be removed in the subsequent layers. In

summary, we wanted to keep the good features of RC4 KSA, and remove only the bad ones.

We evaluated the performance of our new design using the *eSTREAM testing framework* [3]. The C-implementation of the testing framework was installed in a machine with Intel(R) Pentium(R) 4 CPU, 2.8 GHz Processor Clock, 512 MB DDR RAM on Ubuntu 7.10 (Linux 2.6.22-15-generic) OS. A benchmark implementation of RC4 is available within the test suite. We implemented our modified RC4, which we call RC4$^+$, that incorporates both KSA$^+$ and PRGA$^+$, maintaining the API compliance of the suite. Test vectors were generated in the NESSIE [21] format.

Tests with 16 bytes secret key and null IV using the *gcc_default_O3-ual-ofp* compiler report 16944.70 cycles/setup for RC4 KSA and 49823.69 cycles/setup for the KSA$^+$ of RC4$^+$. Thus, we can claim that the running time of our KSA$^+$ is approximately $\frac{49823.69}{16944.70} = 2.94$ times than that of the RC4 KSA.

4 PRGA$^+$: Modifications to RC4 PRGA

There are a number of important works related to the analysis of the RC4 PRGA. The main directions of cryptanalysis in this area are

(1) finding correlations between the keystream output bytes and the secret key [28,22,13] and key recovery in the IV mode [4,15,9,30,34] (these exploit the weaknesses of both the KSA and the PRGA),

(2) recovering the RC4 permutation from the keystream output bytes [10,31,19] and

(3) identifying distinguishers [14,27,16].

In Section 3.2, we proposed KSA$^+$ in such a manner that one cannot get secret key correlations from the permutation bytes. This guarantees that the keystream output bytes, which are some combination of the permutation bytes, cannot have any correlation with the secret key. As argued in Section 3.3, IV's are used in such a way, that they cannot be easily exploited to mount an attack. So we target the other two weaknesses, enlisted in Item (2) and (3) above, in our design of PRGA$^+$.

For any byte b, b_R^n (respectively b_L^n) denotes the byte after right (respectively left) shifting b by n bits. For $r \geq 1$, we denote the permutation, the indices i, j and the keystream output byte after round r of the PRGA (or PRGA$^+$) by S_r^G, i_r^G, j_r^G and z_r respectively.

The main idea behind this design of PRGA$^+$ is masking the output byte such that it is not directly coming out from any permutation byte. Two bytes from the permutation are added modulo 256 (a nonlinear operation) and then the outcome is XOR-ed with a third byte (for masking non-uniformity). Introducing additional $S[t'], S[t'']$, over the existing $S[t]$ in RC4, makes the running time of PRGA$^+$ more than that of RC4 PRGA. Note that the evolution of the permutation S in PRGA$^+$ stays exactly the same as in RC4 PRGA. We introduce a constant value $0xAA$ (equivalent to 10101010 in binary) in t', as without this, if

j^G becomes 0 in rounds 256, 512, ... (i.e., when $i^G = 0$), then t and t' in such a round become equal with probability 1, giving an internal bias.

RC4 PRGA
Initialization:
$i = j = 0$;
Keystream Generation Loop:
$i = i + 1$;
$j = j + S[i]$;
Swap($S[i]$, $S[j]$);
$t = S[i] + S[j]$;
Output $z = S[t]$;

PRGA$^+$
Initialization:
$i = j = 0$;
Keystream Generation Loop:
$i = i + 1$;
$j = j + S[i]$;
Swap($S[i]$, $S[j]$);
$t = S[i] + S[j]$;
$t' = (S[i_R^3 \oplus j_L^5] + S[i_L^5 \oplus j_R^3]) \oplus 0xAA$;
$t'' = j + S[j]$;
Output $z = (S[t] + S[t']) \oplus S[t'']$;

Resisting Permutation Recovery Attacks: The basic idea of cryptanalysis in [19] is as follows. Corresponding to a window of $w+1$ keystream output bytes, one may assume that all the j's are known, i.e., $j_r^G, j_{r+1}^G, \ldots, j_{r+w}^G$ are known. Thus w many $S_r^G[i_r^G]$ will be available from $j_{r+1}^G - j_r^G$. Then w many equations of the form $S_r^{G^{-1}}[z_r] = S_r^G[i_r^G] + S_r^G[j_r^G]$ will be found where each equation contains only two unknowns. The idea of [10] (having complexity around 2^{779} to 2^{797}) actually considered four unknowns $j^G, S^G[i^G], S^G[j^G], S^{G^{-1}}[z]$.

Our design does not allow the strategy of [19] as $S^G[S^G[i^G] + S^G[j^G]]$ is not exposed directly, but it is masked by several other quantities. To form the equations as given in [19], one first needs to guess $S^G[t], S^G[t'], S^G[t'']$ and looking at the value of z, there is no other option than to go for all the possible choices. The same permutation structure of S in RC4$^+$ can be similarly exploited to get the good patterns [19, Section 3], but introducing additional t', t'', we ensure the non-detectability of such a pattern in the keystream and thus the idea of [19, Section 4] will not work.

Information on permutation bytes is also leaked in the keystream via the Glimpse Main Theorem [8,15], which states that during any PRGA round, $P(S[j] = i - z) = P(S[i] = j - z) \approx \frac{2}{N}$. The assumption $i = S[i] + S[j]$ holds with a probability $\frac{1}{N}$, leading to the bias $P(S[j] = i - z) = \frac{1}{N} \cdot 1 + (1 - \frac{1}{N}) \cdot \frac{1}{N} = \frac{2}{N} - \frac{1}{N^2} \approx \frac{2}{N}$. To obtain such biases in PRGA$^+$, one need to have more assumptions of the above form. Thus, Glimpse like biases of PRGA$^+$, if at all exist, would be much weaker.

Resisting Distinguishing Attacks: In [14], it was proved that $P(z_2 = 0) = \frac{2}{N}$ instead of the uniformly random case of $\frac{1}{N}$. This originates from the fact that when $S_N[2] = 0$ and $S_N[1] \neq 2$ after the KSA, the second keystream output byte z_2 takes the value 0. Based on this, they showed a distinguishing attack and a ciphertext-only attack in broadcast mode. We avoid this kind of situation in our design. As a passing remark, we like to present an experimental result. Hundred million secret keys of length 16 byte are generated and 1024 rounds of PRGA are executed for each such key. The empirical evidences indicate that $P(z_r = v) = \frac{1}{N}$, $1 \leq r \leq 1024, 0 \leq v \leq N - 1$.

In the work [27], it was observed that $P(z_1 = z_2) = \frac{1}{N} - \frac{1}{N^2}$, which leads to a distinguishing attack. Even after extensive experimentation, we could not observe such bias in the keystream output bytes of PRGA$^+$. The same experiment described above supported that $P(z_r = z_{r+1})$ is uniformly distributed for $1 \leq r \leq 1023$.

In [16], it has been shown that getting strings of pattern $ABTAB$ (A, B are bytes and T is a string of bytes of small length G, say $G \leq 16$) are more probable in RC4 keystream than in random stream. In uniformly random keystream, the probability of getting such pattern irrespective of the length of T is $\frac{1}{N^2}$. It has been shown in [16, Theorem 1] that for RC4, the probability of such an event is $\frac{1}{N^2}(1 + \frac{e^{\frac{-4-8G}{N}}}{N})$, which is above $\frac{1}{N^2}$, but less than $\frac{1}{N^2} + \frac{1}{N^3}$. This result is based on the fact that the permutation values in locations that affect the swaps and the selection of output bytes in both pairs of rounds that are G-round apart, remain unchanged with high probability during the intermediate rounds. The permutation in PRGA$^+$ evolves in the same way as RC4 PRGA, but the keystream output generation in PRGA$^+$ is different, which does not allow the pattern AB to propagate down the keystream with higher probability for smaller interval lengths (G). In [16], 2^{16} keystreams of size 2^{24} each were used to observe these biases effectively. The simulation on PRGA$^+$ reveals that it is free from these biases.

We now present the software performance analysis of PRGA$^+$ using the same specifications as described at the end of Section 3.3. The stream encryption speed for RC4 and RC4$^+$ turned out to be 14.39 cycles/byte and 24.51 cycles/byte respectively. Thus, we can claim that the running time of one round of our PRGA$^+$ is approximately $\frac{24.51}{14.39} = 1.70$ times than that of RC4 PRGA.

5 Conclusion

Though RC4 can be stated in less than ten lines, newer weaknesses are being discovered every now and then even after twenty years of its discovery. This raises the need for a new design of a stream cipher, which would be as simple as the description of RC4, yet devoid of the existing weaknesses of RC4. This is the target of this paper. We present a three-layer architecture of the scrambling phase after the initialization, which removes many weaknesses of the KSA. We also add a few extra steps in the PRGA to strengthen the cipher. Experimental results also support our claim. An extended version of this paper is available in IACR Eprint Server [12], that contains some relevant graphs which could not fit here due to space constraints.

Even after our arguments and empirical evidences, the security claim of RC4$^+$ is a conjecture, as is the case with many of the existing stream ciphers. We could not observe any immediate weakness of the new design and the cipher is subject to further analysis.

References

1. Akgun, M., Kavak, P., Demirci, H.: New Results on the Key Scheduling Algorithm of RC4. In: Indocrypt 2008 (2008)
2. Biham, E., Carmeli, Y.: Efficient Reconstruction of RC4 Keys from Internal States. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 270–288. Springer, Heidelberg (2008)
3. eSTREAM, the ECRYPT Stream Cipher Project (last accessed on July 18, 2008), http://www.ecrypt.eu.org/stream
4. Fluhrer, S.R., Mantin, I., Shamir, A.: Weaknesses in the Key Scheduling Algorithm of RC4. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 1–24. Springer, Heidelberg (2001)
5. Golic, J.: Linear statistical weakness of alleged RC4 keystream generator. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 226–238. Springer, Heidelberg (1997)
6. Gong, G., Gupta, K.C., Hell, M., Nawaz, Y.: Towards a General RC4-Like Keystream Generator. In: Feng, D., Lin, D., Yung, M. (eds.) CISC 2005. LNCS, vol. 3822, pp. 162–174. Springer, Heidelberg (2005)
7. Hong, J., Sarkar, P.: New Applications of Time Memory Data Tradeoffs. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 353–372. Springer, Heidelberg (2005)
8. Jenkins, R.J.: ISAAC and RC4 (1996) (last accessed on July 18, 2008), http://burtleburtle.net/bob/rand/isaac.html
9. Klein, A.: Attacks on the RC4 stream cipher. Designs, Codes and Cryptography 48(3), 269–286 (2008)
10. Knudsen, L.R., Meier, W., Preneel, B., Rijmen, V., Verdoolaege, S.: Analysis Methods for (Alleged) RCA. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 327–341. Springer, Heidelberg (1998)
11. LAN/MAN Standard Committee. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, 1999 edition. IEEE standard 802.11 (1999)
12. Maitra, S., Paul, G.: Analysis of RC4 and Proposal of Additional Layers for Better Security Margin (Full Version). IACR Eprint Server, eprint.iacr.org, number 2008/396, September 19 (2008)
13. Maitra, S., Paul, G.: New Form of Permutation Bias and Secret Key Leakage in Keystream Bytes of RC4. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 253–269. Springer, Heidelberg (2008)
14. Mantin, I., Shamir, A.: A Practical Attack on Broadcast RC4. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 152–164. Springer, Heidelberg (2002)
15. Mantin, I.: A Practical Attack on the Fixed RC4 in the WEP Mode. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 395–411. Springer, Heidelberg (2005)
16. Mantin, I.: Predicting and Distinguishing Attacks on RC4 Keystream Generator. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 491–506. Springer, Heidelberg (2005)
17. Mantin, I.: Analysis of the stream cipher RC4. Master's Thesis, The Weizmann Institute of Science, Israel (2001)
18. Maximov, A.: Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of RC4 Family of Stream Ciphers. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 342–358. Springer, Heidelberg (2005)

19. Maximov, A., Khovratovich, D.: New State Recovery Attack on RC4. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 297–316. Springer, Heidelberg (2008)
20. Mironov, I. (Not So) Random Shuffles of RC4. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 304–319. Springer, Heidelberg (2002)
21. New European Schemes for Signatures, Integrity, and Encryption (last accessed on September 19, 2008), https://www.cosic.esat.kuleuven.be/nessie
22. Paul, G., Rathi, S., Maitra, S.: On Non-negligible Bias of the First Output Byte of RC4 towards the First Three Bytes of the Secret Key. In: Proceedings of the International Workshop on Coding and Cryptography (WCC) 2007, pp. 285–294 (2007); Extended version available at Designs, Codes and Cryptography 49, 1-3 (December 2008)
23. Paul, G., Maitra, S.: Permutation after RC4 Key Scheduling Reveals the Secret Key. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 360–377. Springer, Heidelberg (2007)
24. Paul, G., Maitra, S.: RC4 State Information at Any Stage Reveals the Secret Key. IACR Eprint Server, eprint.iacr.org, number 2007/208 (June 1, 2007); This is an extended version of [23]
25. Paul, G., Maitra, S., Srivastava, R.: On Non-randomness of the Permutation After RC4 Key Scheduling. In: Boztaş, S., Lu, H.-F(F.) (eds.) AAECC 2007. LNCS, vol. 4851, pp. 100–109. Springer, Heidelberg (2007)
26. Paul, S., Preneel, B.: Analysis of Non-fortuitous Predictive States of the RC4 Keystream Generator. In: Johansson, T., Maitra, S. (eds.) INDOCRYPT 2003. LNCS, vol. 2904, pp. 52–67. Springer, Heidelberg (2003)
27. Paul, S., Preneel, B.: A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 245–259. Springer, Heidelberg (2004)
28. A. Roos. A class of weak keys in the RC4 stream cipher. Two posts in sci.crypt, message-id 43u1eh$1j3@hermes.is.co.za and 44ebge$llf@hermes.is.co.za (1995) (last accessed on July 18, 2008), http://groups.google.com/group/sci.crypt.research/msg/078aa9249d76eacc?dmode=source
29. RSA Lab Report. RSA Security Response to Weaknesses in Key Scheduling Algorithm of RC4 (last accessed on July 18, 2008), http://www.rsa.com/rsalabs/node.asp?id=2009
30. Tews, E., Weinmann, R.P., Pyshkin, A.: Breaking 104 bit WEP in less than 60 seconds. IACR Eprint Server, eprint.iacr.org, number 2007/120 (April 1, 2007) (last accessed on July 18, 2008)
31. Tomasevic, V., Bojanic, S., Nieto-Taladriz, O.: Finding an internal state of RC4 stream cipher. Information Sciences 177, 1715–1727 (2007)
32. Tsunoo, Y., Saito, T., Kubo, H., Shigeri, M., Suzaki, T., Kawabata, T.: The Most Efficient Distinguishing Attack on VMPC and RC4A. In: SKEW 2005 (last accessed on July 18, 2008), http://www.ecrypt.eu.org/stream/papers.html
33. Tsunoo, Y., Saito, T., Kubo, H., Suzaki, T.: A Distinguishing Attack on a Fast Software-Implemented RC4-Like Stream Cipher. IEEE Transactions on Information Theory 53(9), 3250–3255 (2007)
34. Vaudenay, S., Vuagnoux, M.: Passive-Only Key Recovery Attacks on RC4. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 344–359. Springer, Heidelberg (2007)
35. Zoltak, B.: VMPC One-Way Function and Stream Cipher. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 210–225. Springer, Heidelberg (2004)

New Results on the Key Scheduling Algorithm of RC4

Mete Akgün, Pınar Kavak , and Hüseyin Demirci

Tübitak UEKAE, 41470 Gebze, Kocaeli, Turkey
{makgun,pinar,huseyind}@uekae.tubitak.gov.tr

Abstract. A new bias is detected in the key scheduling algorithm of RC4 and a novel framework that advantageously combines this new bias with the existing ones is proposed. Using the new bias, a different algorithm is proposed to retrieve the RC4 key given the state table. The new method not only improves the success probability but also provides a more efficient way of calculation in comparison with the previous methods for any key size. The efficiency of the algorithm is demonstrated experimentally. If the key length is 40 bits, the secret key is retrieved with a 99% success rate in 0.007 seconds. The success probability for retrieving the 128 bit RC4 key is also increased significantly. 128-bit key can be retrieved with 3% success rate in 185 seconds and 7.45% success rate in 1572 seconds on a 2.67GHz Intel CPU.

Keywords: RC4, Stream Cipher, Cryptanalysis, Key Scheduling Algorithm, State Table.

1 Introduction

RC4 is one of the most famous stream ciphers which was designed by Ron Rivest. It is introduced in 1987 but the algorithm is kept secret until its description is anonymously published on the Cypherpunks mailing list [1] in 1994.

After its first release, RC4 stream cipher became very popular especially in software. In the past twenty years it is mostly used in some popular protocols such as SSL (Secure Socket Layer) and TLS (Transport Layer Security) to protect internet traffic and some others such as WEP (Wired Equivalent Privacy) and WPA (Wi-Fi Protected Access) to secure wireless networks.

Attacks on RC4 are generally majored in two groups. First group is based on the weaknesses of the PRGA. Second group is based on the weaknesses of the KSA. Additionally, many works try to exploit the special weaknesses existing in the usage of IV (Initial Value).

Our study focuses on analyzing the KSA. In this paper, we present a more efficient algorithm to derive the secret key from a given internal state. Analyzing the KSA has direct consequences in WEP attacks like in [9], [22] and [23]. The algorithm basicly depends on the newly discovered bias in the KSA. Although, the new bias seems symmetrically similar to the previously known biases in terms of structure, it provides an independent piece of information about the internal

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 40–52, 2008.

states of the KSA. We propose a very efficient simple algorithm which combines the Roos' observation [21] and the equations derived in [19], the difference equations in [2] and our new bias advantageously.

This paper is organized as follows: In Section 2, we describe the RC4 algorithm. In Section 3, we present the existing work on RC4. In Section 4, we describe the basic assumptions and equations of the attacks presented in [19] and [2] which use the bias of the first bytes of the initial permutation. In Section 5, we give the description of our new bias on the KSA and related distributions some of which are already known. In Section 6, we explain the key recovering algorithm. In Section 7, we discuss our results along with a comparison with the previous studies. In Section 8, we summarize the work done.

Recently, we have been aware of other two studies on the key retrieval problem from the state table. The first study [13] has many common points with this work. The second group has considered a bit-by-bit key recovery approach [8].

2 The RC4 Stream Cipher

The internal state table of RC4 S consists of a permutation of N possible words where $N = 2^n$. Two n-bit index pointers i and j are used to randomize the state table. The pseudo-random variable j is secret but i is public and its value at any stage of the stream generation is generally known. The running key values are also produced with an address variable formed with the help of i and j.

RC4 consists of two algorithms; KSA (Key Scheduling Algorithm) and PRGA (Pseudo Random Generation Algorithm). KSA initializes the internal state table S with the encryption key K.

The KSA and PRGA are given below. All additions in the algorithms are performed modulo N.

Key Scheduling Algorithm	**Pseudo Random Generation Algorithm**
for $i = 0$ to 2^{n-1}	$i \leftarrow 0$
$S[i] \leftarrow i$	$j \leftarrow 0$
endfor	**loop**
$j \leftarrow 0$	$i \leftarrow i + 1$
for $i = 0$ to 2^{n-1}	$j \leftarrow j + S[i]$
$j \leftarrow j + S[i] + K[i \bmod l]$	swap($S[i]$, $S[j]$)
swap($S[i]$, $S[j]$)	$t \leftarrow S[i] + S[j]$
endfor	**output** $S[t]$
	endloop

In most of the applications, RC4 uses the parameters $n = 8$ and $l = 16$. Hence the table S consists of $N = 256$ elements. In this paper, we also assume these values unless explicitly stated otherwise.

3 Previous Work on RC4

Finney in [3], observed a group of states that RC4 can never enter. These states satisfy the property $j = i+1$ and $S[i+1] = 1$. If RC4 was not designed carefully, one of every 2^{16} keys would fall into a cycle of length $255 \cdot 256$.

Golic showed that RC4 can be distinguished from other keystream generators by using the linear statistical weakness of RC4 [6].

Knudsen et al. showed the intrinsic properties of RC4 which are independent of the key scheduling and the key size [10]. They have developed a backtracking algorithm in which the initial state table S is guessed given a small part of the running key stream.

Fluhrer and McGrew described a method which explicitly computes digraph probabilities [5]. This method can be used to distinguish 8-bit RC4 from a random sequence. Parts of the internal state table can also be determined with this method.

Grosul and Wallach showed that a pair of keys produce running key streams that are very similar in the first 256 bytes when the key size is same with the table size [7].

Mironov proposed an idealized model of RC4 and analyzed it applying the theory of random shuffles [18]. At the end of his analysis, he found a conservative estimate (512 bytes of the running key) that should be discarded for safety.

Mantin's thesis [14] is a valuable resource on RC4 up to its date of writing. Then, Mantin and Shamir described a major statistical weakness in RC4 caused by the first and second bytes of the running key [16].

Fluhrer, Mantin and Shamir showed that RC4 is completely insecure in a common mode of operation which is used in Wired Equivalent Privacy Protocol, in which a fixed secret key is concatenated with known IV modifiers [4]. With these observations, practical attacks were designed and applied on the WEP protocol. Vaudenay and Vuagnoux, described a passive only attack that improves the key recovery process on WEP by the weaknesses they observed in KSA of RC4 [23]. Tews, R.P. Weinmann and Pyshkin demonstrate a new attack on 104 bit WEP [22].

Pudovkina found the number of keys of the RC4 cipher generating initial permutations with the same cycle structure [20]. It is found that the distribution of the initial permutations is not uniform.

Mantin described a new distinguishing attack using the bias in the digraph distribution of the cipher [15]. By this bias, one can predict the next bit or byte of the running key if 2^{45} or 2^{50} output words are known respectively.

Maitra and Paul have observed that the initial bytes of the permutation after the KSA are biased with some linear combination of the key bytes. This leads to a bias in some key stream bytes [11], [12].

Paul and Maitra, discovered the secret key from the initial state table using biases in the first entries of the table [19]. They create some equations by using the first entries of the initial state table. These equations have significant probability. They guess some of the bytes of the secret key and they obtain the rest of the key by using these equations. Existence of many correct equations lead

to the success of their algorithm. Carmeli and Biham presented an algorithm to retrieve the secret key if the internal state table is given [2]. Their study depends on the equations in [19]. They also propose additional equations by considering the differences of the existing equations. They declare that they increased the success rate by filtering and correcting some of the equations and also using the new ones. Combining all these equations into a statistical algorithm lead them obtain better results than [19].

Recently Maximov and Khovratovich proposed an attack which recovers some special internal states of the RC4 from the keystream [17].

We started our study by depending on the biases given in [19] and [2] and by using some of the equations described in [19] and [2]. We observed a different bias and by using this bias we created new equations that lead us to obtain higher success rate than [2].

4 Notations and Basic Assumptions

4.1 Notations

The S_r and j_r denote the values of the state table S and the index value j after r iterations of the KSA respectively. j_0 is the initial value of j and j_N is the last value of j at the end of the KSA. S_0 is the identity permutation and S_N is the initial permutation. S is used to denote the initial permutation instead of S_N.

We denote the number of key bytes by l.

$K[a...b]$ denotes the sum of the key bytes in the range $a, a+1, ..., b$.

By a variable $address(t)$, we mean the index i such that $S[i] = t$ when the KSA has finished.

$e[a]$ means the event a. The events are described in Section 5.

nc denotes the number of candidates, which have the highest weight among a larger group.

4.2 Previous Biases of the KSA

A bias of a linear combination of the secret key bytes is first discovered by Roos in 1995 [21]. This bias is described in Theorem 1. He has given the probability of the bias experimentally and has observed that it has significant probability in the first $40 - 50$ entries of the state table.

Theorem 1. *The most likely value for $S[i]$ at the end of the KSA is*

$$S[i] = K[0...i] + \frac{i(i+1)}{2} mod\ N. \tag{1}$$

Paul and Maitra, in [19] has expressed this bias theoretically. The formula is given in Theorem 2.

Theorem 2. *Assume that during the KSA the index j takes its values uniformly at random from $0, 1, ..., N-1$. Then,*

$$P(S[i] = K[0...i] + \frac{i(i+1)}{2}) \geq (\frac{N-i}{N}).(\frac{N-1}{N})^{\frac{i(i+1)}{2}+N} + \frac{1}{N}. \tag{2}$$

The underlying assumptions of Theorem 2 are the following:

1. $S_r[r] = r$ for $r \in \{0, ..., i\}$, i.e. $S[r]$ is not swapped until the r-th iteration.
2. $S_i[j_{i+1}] = j_{i+1}$.
3. $j_r \neq i$ for $r \in \{i+1, ..., N-1\}$.

Assuming that the first event occurs, only the key bytes and some constant values are needed to calculate the j_{i+1} value.

$$j_{i+1} = \sum_{r=0}^{i}(K[r] + S_r[r]) = \sum_{r=0}^{i}(K[r] + r) = K[0...i] + \frac{i(i+1)}{2}.$$

Assuming that the second event occurs, $S_{i+1}[i] = j_{i+1}$ holds after $i+1^{th}$ iteration of the KSA. Assuming that the third event occurs, the index j does not point to $S[i]$ again, so $S[i]$ is not swapped again once more until the end of the KSA. If all of the events occur then (1) holds

$$S_N[i] = S_{i+1}[i] = j_{i+1} = \sum_{r=0}^{i}(K[r] + S_r[r]) = K[0...i] + \frac{i(i+1)}{2}.$$

Finally, Biham and Carmeli have generalized the assumptions in [2] as follows:

1. $S_r[r] = r$ for $r \in \{i_1+1, ..., i_2\}$, i.e. S[r] is not swapped until the r-th iteration.
2. $S_{i_1}[j_{i_1+1}] = j_{i_1+1}$ and $S_{i_2}[j_{i_2+1}] = j_{i_2+1}$.
3. $j_r \neq i_1$ for $r \in \{i_1+1, ..., N-1\}$ and $j_r \neq i_2$ for $r \in \{i_2+1, ..., N-1\}$.

Under these assumptions, they get

$$S_N[i_2] - S_N[i_1] = K[i_1+1...i_2] + \frac{i_2(i_2+1)}{2} - \frac{i_1(i_1+1)}{2}.$$

They have considered the differences of the state elements. Theorem 3 gives the bias of these differences.

Theorem 3. *Assume that during the KSA the index j takes its values uniformly at random from $0, 1, ..., N-1$ and let $0 \leq i_1 < i_2 < N$. Let $C_i = S[i] - \frac{i(i+1)}{2}$. Then,*

$$P((C_{i_2} - C_{i_1}) = K[i_1+1...i_2]) \geq$$

$$[(1 - \frac{i_2}{N})^2(1 - \frac{i_2 - i_1 + 2}{N})^{i_1}(1 - \frac{2}{N})^{N-i_2-1}\prod_{r=0}^{i_2-i_1-1}(1 - \frac{r+2}{N})] + 1/N. \quad (3)$$

In this way, they have more equations to consider. The difference bias works still after the 50^{th} entry, and their probabilities are much higher. The authors have also suggested assigning a weight to the key candidates, and methods to filter wrong equations and adjust the weights. As a result, their success probabilities of retrieving the key are much higher than [19].

The proofs of theorems use the occurance probability of the underlying events and can be shown by induction.

5 Useful Distributions of the KSA

In this section we present the statistical properties of the KSA which will be helpful in our algorithm.

Definition 1. *After the KSA, if $j_i = S[i]$, we call this as event 1 has occured for index i, and denote event 1 by $e[1]$.*

If the following assumptions hold, event 1 occurs.

1. $j_r \neq j_i$ for $r \in \{0, ..., i-1\}$, i.e. $S[j_i]$ is not swapped until the i-th iteration.
2. $j_i > i$, i.e $S[i]$ is swapped with a greater index.
3. $j_r \neq i$ for $r \in \{i+1, ..., N-1\}$, i.e. $S[i]$ is not swapped after the i-th iteration.

The following theorem for the probability of $e[1]$ exists in Section 2 of [19] in a more generalized framework.

Theorem 4

$$P(S[i] = j_i) \geq (1 - \frac{1}{N})^i (1 - \frac{i-1}{N})(1 - \frac{1}{N})^{N-i-1} + \frac{1}{N}. \quad (4)$$

This property investigates the information obtained from $S[i]$ as in Theorem 2 and 3. But it is related with a single entry of the table, not with a sequence sum. Therefore, it gives information for only the j value, not the key itself. In Section 6, we will exploit this property to obtain information of the key bytes.

The probability of $e[1]$ is still high after the 50^{th} entry.

Table 1. The Probabilities Given by Theorem 4

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Prob.	.371	.369	.368	.366	.365	.363	.362	.360	.359	.358	.356	.355	.353	.352	.350	.349
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Prob.	.348	.346	.345	.343	.342	.340	.339	.337	.336	.335	.333	.332	.330	.329	.327	.326

5.1 New Bias

In this section, we propose a new property of the KSA which seems similar to the observation in [2] in a symmetric structure. Instead of their consideration of $S[i]$'s, we have also exploited the information from the j values, i.e. $j = address(S[i])$'s. It is interesting that they form their bias by only considering one of the swapped variables. This approach provides us an independent bias. The new bias is more significant for the address of latter indexes, in contrast with the previous properties.

Definition 2. *After the KSA, if $j_i = address(i)$, we call this as event 2 has occured for index i, and denote event 2 by $e[2]$.*

$e[2]$ occurs under the following assumptions:

1. $S_i[i] = i$, i.e. $S[i]$ is not swapped until the i-th iteration.
2. $j_i \leq i$.
3. $j_r \neq j_i$ for $r \in \{i_1 + 1, ..., N - 1\}$.

If these conditions hold, $S_i[i]$ is swapped with $S_i[j]$, and then $S_i[j]$ is not swapped with another value till the end of KSA. Therefore, we have $address(i) = j_i$. The probability distribution of this event is the following:

Theorem 5

$$P(address[i] = j_i) \geq (1-\frac{1}{N})^i(\frac{i}{N})(1-\frac{1}{N})^{(N-i-1)} + \frac{1}{N} = (1-\frac{1}{N})^{N-1}(\frac{i}{N}) + \frac{1}{N}.$$

The probabilities of $e[2]$ are higher for the greater index values.

Table 2. The Probabilities Given by Theorem 5

i	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
Prob.	.326	.327	.329	.330	.332	.333	.335	.336	.337	.339	.340	.342	.343	.345	.346	.348
i	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
Prob.	.349	.350	.352	.353	.355	.356	.358	.359	.360	.362	.363	.365	.366	.368	.369	.371

We can generalize this procedure by analyzing the differences as in [2]. We explain the information satisfied by the differences with an example.

Example 1. Assume that the following events happened during KSA:

1. $S_{250}[250] = 250$, $S_{251}[251] = 251$,
2. At step 250, we swap $[250]$ with $S_{250}[j_{250}]$. Therefore $S_{250}[j_{250}] = 250$. Assume that this entry does not change in the later steps.
3. At step 251, we swap $S_{251}[251]$ with $S_{251}[j_{251}]$. Therefore $S_{251}[j_{251}] = 251$. Assume that this entry does not change in the later steps.

If the above assumptions hold, then we have,
$address(251) = j_{251}$ and $address(250) = j_{250}$. Therefore,

$$address(j_{251}) - address(j_{250}) = j_{251} - j_{250} = (S[251] + K[11]) \bmod 256$$

depends only on $K[11]$. Considering the addresses of the values 251 and 250, we gather information about one byte of the key.

This method can be generalized as below:

1. $S_r[r] = r$ for $r \in \{i_1, ..., i_2\}$, i.e. S[r] is not swapped until the r-th iteration.
2. $j_{i_1} \leq i_1$ and $j_{i_2} \leq i_2$.
3. $j_r \neq j_{i_1}$ for $r \in \{i_1 + 1, ..., N - 1\}$ and $j_r \neq j_{i_2}$ for $r \in \{i_2 + 1, ..., N - 1\}$.

If the first event occurs, then the index j is affected in iterations i_1 through i_2 only by the key bytes and constant values. The second event ensures that the index i and the third event ensures that the index j does not point to $S[j_{i_1}]$ and $S[j_{i_2}]$ in later iterations. Therefore, $S[j_{i_1}] = i_1$ and $S[j_{i_2}] = i_2$ remains the same after the KSA. The probability distribution of this bias is given in Theorem 6.

Theorem 6. *Assume that during the KSA the index j takes its values uniformly at random from $\{0, 1, ..., N-1\}$, and let $0 \leq i_1 < i_2 < N$. Then,*

$$P(address(i_2) - address(i_1) = K[i_1+1...i_2] + (i_1+1) + (i_1+2) + ... + i_2) \geq$$

$$(\frac{N-(i_2-i_1+1)}{N})^{i_1} \prod_{r=1}^{i_2-i_1} (1-\frac{r}{N}) \times (\frac{i_1+1}{N})(\frac{i_2+1}{N}) \times (\frac{N-1}{N})^{2N-i_1-i_2-2}. \quad (5)$$

The solutions of equations from Theorem 3 and Theorem 6 can be used to get the encryption key. But instead, we suggest to analyze the journey of j values during the KSA with the help of Theorems 4 and 5.

5.2 More Distributions

Theorems 4 and 5 motivate to consider the probabilities of the following events. Definition 3 and Definition 5 are previously analyzed in [11]. These events have non trivial probabilities to distinguish the j sequence from random.

Definition 3. *After the KSA, if $j_i = S[S[i]]$, we call this as event 3 has occured for index i, and denote event 3 by $e[3]$.*

Definition 4. *After the KSA, if $j_i = address(address(i))$, we call this as event 4 has occured for index i, and denote event 4 by $e[4]$.*

Similarly, we may define the following events:

Definition 5. *After the KSA, if $j_i = S[S[S[i]]]$, we call this as event 5 has occured for index i, and denote event 5 by $e[5]$.*

Definition 6. *After the KSA, if $j_i = address(address(address(i)))$, we call this as event 6, and denote event 6 by $e[6]$.*

We have observed that continuing further after $e[6]$ does not produce helpful bias. Now we have 6 events for guessing j values produced during KSA. In Section 6, we will use them to retrieve information about the key bytes.

6 The Key Recovering Algorithm

In our algorithm, first we get a unique suggestion by using some key guessing methods which hold with a probability for all of the key bytes, but we consider a portion of this group as correct. From the unique suggestion, we assign m bytes of the key with some probabilities. However we do not know exactly which m bytes are correct. Therefore, we have to try $C(n, m)$ number of combinations to match the correct one. After the assignment of partial key bytes, the remaining bytes are updated according to the update mechanism. The pseudo code of the algorithm is given in Section 6.4.

6.1 Key Guessing Methods

This section explains the scenarios which are beneficial for us to guess the j values during the KSA. The KSA produces 256 pseudo-random j values. If we have the information of successive j and $S[i]$ values, we can get the related key bytes. Our assumption is that one of the events have occured for the successive j values and $S[i] = i$ for the second entry. Since $j_i = j_{i-1} + S[i] + K[i \bmod l]$ in KSA, one byte of key information is obtained.

The j values can be guessed by the 6 events in Section 5.

1. e[1]: $j = S[i]$ (The event of Theorem 4.)
2. e[2]: $j = address(i)$ (The event of Theorem 5.)
3. e[3]: $j = S[S[i]]$.
4. e[4]: $j = address(address(i))$.
5. e[5]: $j = S[S[S[i]]]$.
6. e[6]: $j = address(address(address(i)))$.

These 6 events confirm 6 candidates for the j value at each step of the KSA. Then, for two successive values j_i and j_{i+1}, we obtain 36 combinations. For instance, the event $e[11]$ means both j_i and j_{i+1} satisfy event 1 above, and the event $e[41]$ denotes that j_i satisfies event 4 and j_{i+1} satisfies event 1. Additionally, we assume that $S_i[i] = i$ is satisfied. In this way, we get 36 candidates for one key byte. Since the key is repeated (N/l) times during KSA, we get $(N/l) \times 36$ possible values for each key byte. Since these events occur with different probabilities in different parts of the table, the candidates which are below a previously determined threshold value are eliminated.

For each index value, these probabilities are assigned as weights for the candidates. A candidate value can be observed more than once with different probabilities for the same key byte. Total weight of such candidates are calculated in a way that uses their frequency and also the sum of their individual weights.

6.2 Initial Key Guessing

The first 16-byte candidate that has the maximum weight is assigned as the key bytes. We do not trust all bytes of the selected candidate but generally assume that arbitrary m-bytes of it are true with some probability. Then, we are left with $C(l, m)$ possible combinations that should be tried to find the matching key bytes.

For the 16-byte key this method provides $m = 4, 8$ and 12 bytes of the key with probabilities 0.91, 0.65 and 0.06 respectively. Using only this information, the full 16-byte key can be guessed with 0.0001 probability in 0.0063 seconds on 2.67GHz CPU with 2GB RAM. We can use this information to provide a lower bound for the success probability of the algorithm for greater complexities. For instance, if the 8 bytes are obtained correctly by this approach, and the remaining 8 bytes are searched, then the full key can be obtained with 0.65 probability with a complexity of $C(16, 8) \times 2^{8\times 8} \approx 2^{78}$ key trials.

6.3 The Update Mechanism

In [2], the authors have used an iterative process for reviewing and updating the weights. We use a similar structure with a special technique to increase the success probability of obtaining new key bytes from the already known ones.

The longer sequence sums considered in Theorem 3 provide information on more bytes, but have less probability to occur. For our algorithm, we have decided to consider the sums in groups of 4 bytes as the optimal sequence length parameter. We have already guessed some of the bytes in the 4-byte group from the selected combination elements. In addition to these known bytes, the sum informations in the 4-byte sequence are also used. By this way the unknown bytes of this group are determined using the method given in [2]. This method increases the number of candidates, but the weights of the correct ones also increase.

We can explain this with the following example.

Example 2. Assume that there are 36×16 candidates for each of $K[0], K[1], K[2], K[3]$ and the values of $K[0]$ and $K[2]$ are fixed in the selected combination. We can use the following equations to find the possible $K[1]$ values:

1. $K[1] = K[0...1] - K[0]$.
2. $K[1] = K[1...2] - K[2]$.
3. $K[1] = K[0...2] - K[0] - K[2]$.

After obtaining new candidates for $K[1]$, we can fix its value by trying the possible candidates through a selected depth. Then, we need to find possible $K[3]$ values by using the following equations:

1. $K[3] = K[2...3] - K[2]$.
2. $K[3] = K[1...3] - K[1] - K[2]$.
3. $K[3] = K[0...3] - K[0] - K[1] - K[2]$.

From the new candidates, the same procedure is applied to decide on the value of $K[3]$. After fixing the value of $K[3]$ the updating process is finished for this 4-byte group. The same method is applied for the remaining 4-byte groups of the key and a total guess is made for all key bytes.

6.4 The Algorithm

In this section, we will discuss our key retrieval algorithm given the initial state table. This algorithm is based on the observations described in the previous sections. This algorithm utilizes the observations in [19] and [2] with the new ones in a novel framework.

The below algorithm is designed for fixed m and nc values. We can increase the success probability of retrieving the key by using different (m, nc) pairs. In this case, we have to consider the (m_i, nc_i) pairs which have no trivial intersection. The total time complexity of this attack is the sum of individual complexities of the algorithm for each (m_i, nc_i) pair. Since there is no way to avoid the intersection between the pairs, the gained success probability is less than the sum of individual success probabilities. The results of this approach for 12 and $16 - byte$ key are given in Table 3.

KEY_RETRIEVAL_ALGORITHM(S)

1. Compute all C_i values for i and obtain all suggestions for sum such as $K[0...l] = C_{l+1} - C_1$.
2. Among the suggestions select the one with the highest weight as sum value.
3. Reduce all C_i's in which $i > l$ to suggestions for sequences in which $i < l$.
4. Choose the parameter m. For all m-byte combinations of l do:
   – 4.1 Fix the specific bytes of the key that are declared to be true in the selected combination(described in Section 6.2).
   – 4.2 For the remaining $l-m$ bytes, choose the parameter nc, number of candidates.
   – 4.3 For each 4-byte group do:
   –– 4.3.1 Start the update process which chooses the first nc candidates that have been sorted according to the weights for the unknown key bytes(described in Section 6.3).
   – 4.4 Try all combinations of resulting candidates obtained for 4-byte groups. Return the correct key when it is found.
5. Return fail.

The first three lines are from [2].

Table 3. Experimental Results of the Key Retrieval Algorithm

l	m	nc	$P_{Success}$	$T_{[sec]}$	# of Trials	$P_{Success}$ of [2], [19]	$T_{[sec]}$ [2]	$T_{[sec]}$ [19]*
5	2	128	.998	0.011	$2^{24.3}$	.8640	0.02	366
	3	256	.998	0.008	$2^{19.3}$			
12	6	6	.254	3.959	$2^{25.36}$	.0124 .0212	3.04 7.43	100 1000
	8	8	.239	1.808	$2^{20.95}$			
		32	.431	48.939	$2^{28.95}$			
	m_i^{**}	nc_i^{**}	.506	54.390	$2^{30.78}$			
16	10	8	.034	185	$2^{30.96}$	.0005	278	500
	12	16	.020	16.7	$2^{26.82}$			
	m_i^{***}	nc_i^{***}	.0745	1572	$2^{35.91}$			

l =# of key bytes, m =# of selected bytes, nc =selection depth.
*Rough estimation of [2] about [19] to achieve the same success with them.
**$(m_i, nc_i) = \{(4,6),(5,7),(6,8),(7,10),(8,16),(9,32),(10,128)\}$.
***$(m_i, nc_i) = \{(6,4),(7,5),(8,6),(9,7),(10,8),(11,10),(12,22),(13,64),(14,256)\}$.

7 Experimental Results

Table 3 compares our results with previous studies [19] and [2]. We cannot execute the previous algorithms under exactly the same conditions with our algorithm, since we do not have the source code of these algorithms. We have run the algorithm on 2.67GHz Intel CPU with 2GB RAM. The success probabilites are derived from 10000 randomly generated key-state table pairs. Time complexity estimates are empirical. For each chosen l, m and nc values, we try $C(l,m) * (nc)^{(l-m)}$ number of key candidates, but the time cost of each candidate varies with respect to the location of the fixed bytes in the combination and the depth of the trial. For short key sizes such as 5 or 8 bytes, our algorithm

almost retrieves more than 90 percent success in seconds. For longer key sizes, our success probabilities are about 10-50 times better in the same computation time. The results indicate the effectiveness in terms of both the time complexity and success probability.

8 Conclusions

We presented our new observations on the Key Scheduling Algorithm (KSA) of RC4 which point out the structural weaknesses of it. We showed the theoretical distribution of our newly detected bias which is structurally symmetric but independent from the previous studies. We described a framework which uses a simple and efficient algorithm to retrieve the RC4 secret key, given the internal state table. The new observations on the KSA of RC4 significantly increase the success rate of the attack compared to the previous studies. We exploited the Roos' observation [21], its theoretical proof [19] and the difference equations of [2] in addition to the events of our new observations. This work shows that the KSA of RC4 is not perfect, because it leaks information about the secret key if the initial state table is known. It is an open question whether these observations can be used for attacking Pseudo Random Generation Algorithm (PRGA) or improving the attacks on the Wired Equivalent Privacy (WEP) protocol.

References

1. Anonymous, RC4 Source Code, CypherPunks mailing list, September 9 (1994), http://cypherpunks.venona.com/date/1994/09/msg00304.html
2. Biham, E., Carmeli, Y.: Efficient Reconstruction of RC4 Keys from Internal States. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 270–288. Springer, Heidelberg (2008)
3. Finney, H.: An RC4 Cycle That Can't Happen, sci.crypt posting (September 1994)
4. Fluhrer, S.R., Mantin, I., Shamir, A.: Weaknesses in the Key Scheduling Algorithm of RC4. In: Vaudenay, S., Youssef, A.M. (eds.) SAC 2001. LNCS, vol. 2259, pp. 1–24. Springer, Heidelberg (2001)
5. Fluhrer, S.R., McGrew, D.A.: Statistical Analysis of the Alleged RC4 Keystream Generator. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 19–30. Springer, Heidelberg (2001)
6. Golic, J.D.: Linear Statistical Weakness of Alleged RC4 Keystream Generator. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 226–238. Springer, Heidelberg (1997)
7. Grosul, A.L., Wallach, D.S.: A Related-Key Cryptanalysis of RC4, Technical Report-00-358, Department of Computer Science, Rice University (October 2000)
8. Khazaei, S., Meier, W.: On Reconstruction of RC4 Keys from Internal States
9. Klein, A.: Attacks on the RC4 Stream Cipher, February 27 (2006), http://cage.ugent.be/klein/RC4
10. Knudsen, L.R., Meier, W., Prenel, B., Rijmen, V., Verdoolaege, S.: Analysis Methods for (Alleged) RC4. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 327–341. Springer, Heidelberg (1998)

11. Maitra, S., Paul, G.: New Form of Permutation Bias and Secret Key Leakage in Keystream Bytes of RC4. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 253–269. Springer, Heidelberg (2008)
12. Maitra, S., Paul, G.: New Form of Permutation Bias and Secret Key Leakage in Key Stream Bytes of RC4, http://eprint.iacr.org/2007/261.pdf
13. Maitra, S.: Personal Communication
14. Mantin, I.: Analysis of the Stream Cipher RC4, M. Sc. Thesis, The Weizmann Institute of Science, Israel (2001), http://www.wisdom.weizmann.ac.il/~itsik/RC4/Papers/Mantin1.zip
15. Mantin, I.: Predicting and Distinguishing Attacks on RC4 Keystream Generator. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 491–506. Springer, Heidelberg (2005)
16. Mantin, I., Shamir, A.: A Practical Attack on Broadcast RC4. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 152–164. Springer, Heidelberg (2002)
17. Maximov, A., Khovratovich, D.: New State Recovery Attack on RC4. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 297–316. Springer, Heidelberg (2008)
18. Mironov, I.: (Not So) Random Shuffles of RC4. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 304–319. Springer, Heidelberg (2002)
19. Paul, G., Maitra, S.: RC4 State Information at Any Stage Reveals the Secret Key. In: Proceedings of SAC 2007 (2007), http://eprint.iacr.org/2007/208.pdf
20. Pudovkina, M.: The Number of Initial States of the RC4 Cipher with the Same Cycle Structure, Cryptology ePrint Archive, 2002-171, IACR 2002 (2002)
21. Roos, A.: A Class of Weak Keys in the RC4 Stream Cipher, Two posts in sci.crypt (1995), http://marcel.wanda.ch/Archive/WeakKeys
22. Tews, E., Weinmann, R.P., Pyshkin, A.: Breaking 104 Bit WEP in Less than 60 Seconds (2007), http://eprint.iacr.org/2007/120.pdf
23. Vaudenay, S., Vuagnoux, M.: Passive-Only Key Recovery Attacks on RC4. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876. Springer, Heidelberg (2007)
24. Wagner, D.: Weak Keys in RC4, sci.crypt posting (September 1995)

Two Attacks on RadioGatún

Dmitry Khovratovich

University of Luxembourg
dmitry.khovratovich@uni.lu

Abstract. We investigate the security of the hash function design called RadioGatún in a recently proposed framework of sponge functions. We show that previously introduced symmetric trails can hardly be used to construct collisions and to find a second preimage efficiently. As a generalization of truncated differentials, trails with linear and non-linear restrictions on differences are proposed. We use these trails to find semi-free-start collisions and second preimages with the meet-in-the middle approach and the complexity in the gap between claimed security level and the birthday bound. We also provide some observations on lower bounds on the complexity of our methods with respect to the length of the trail used. This is the best attack on RadioGatún.

Keywords: hash functions, cryptanalysis, sponge.

RadioGatún [1], the subject of this paper, is a design of hash functions proposed by Bertoni et al. at the Second Cryptographic Hash Workshop in 2006. Though having been presented as a so called *iterative mangling function* it actually fits the *sponge* framework later proposed by the same authors [2,3]. The hash functions Panama [6] and Grindahl [10] also have much common with RadioGatún and the sponge framework.

The sponge is an iterative construction, which is an alternative to the Merkle-Damgård design. The latter approach consists of the iterated application of the compression function, which gets a message block as the input and assumed to be collision-resistant. The sponge construction operates on smaller message blocks and a round function. After a message is fully processed the sponge generates output of infinite length by just consecutively applying the round function and taking a block in an internal state as a new output block.

Bertoni et al. proved [2] that such a construction is resistant against collision and (second-)preimage attacks of complexity lower than the birthday bound assuming that the round function is a randomly chosen permutation which properties are not exploited by an adversary. However, this assumption is not the case for concrete sponge-based hash functions so the designers claim a reduced security level (see the next section for careful explanation).

RadioGatún is actually a family of hash functions with the size l_w of the building block — word — as a parameter. Although the internal state of the hash function is rather big (58 words), the performance is quite impressive. For example, RadioGatún with $l_w = 32$, which is claimed to be as secure as

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 53–66, 2008.

SHA-256, is twice faster [1]. This makes RadioGatún a very promising design in view of the NIST hash function competition [8].

This paper presents two attacks on RadioGatún: semi-free-start (or chosen IV) collision search and the second preimage search. The outline of the paper is as follows. First we describe the RadioGatún hash function and discuss on the claimed security level: why it is much lower than an intuitive bound with respect to the size of the internal state. In Section 2 we investigate the differential-based collision attacks on RadioGatún using the notion of differential trail. We show that previously introduced symmetric trails [1] do not provide attacks with reasonable complexity. We introduce trails with truncated differentials of linear form, which are extremely suitable for RadioGatún due to its slow diffusion.

Then we present collision and second-preimage attacks based on the trails discussed above. Both attacks use the invertibility of the round function and the abscence of the message scheduling in order to apply the meet-in-the middle approach. Collisions are found in the chosen IV framework though a bit slower second-preimage attack can be also converted to the collision search. The complexities of both attacks are in the gap between the claimed security level and the bound given by the birthday paradox. We also provide some theoretical observations on the lower bound on the complexities of the attacks which can be maintained with the truncated differential trails and the meet-in-the-middle approach.

1 RadioGatún

Description. RadioGatún operates on words of some integer length l_w. The parameter is fixed for a concrete hash function, and we denote by RadioGatún-l_w the corresponding hash function. An internal state of RadioGatún consists of two substates also called *belt* (B) and *mill* (A) of size 39 words and 19 words, respectively. The round function treats them differently. The belt is updated by a simple linear transformation and fed with 12 words of the mill and with 3 words of the message block in a linear way. The mill is fed with 3 words of the belt and 3 words of the message block in a linear way and afterwards is updated by a nonlinear function. The resulting round function is invertible (see Fig. 1).

There is no message scheduling in RadioGatún. First a message to be hashed is appropriately padded and then it is divided by 3-word blocks, which are used only once. The iteration starts with the state full of 0s. One step consists of the message injection and the application of the round function. After the message is fully processed RadioGatún iterates with 16 blank rounds (without any injections) and generates output of infinite length by just consecutively applying the round function and taking a 2-word block in the internal state[1] as a new output block.

We denote the injected block by M. Following this notation, the round function of RadioGatún transforms a state $S = (A, B)$ to a new state $S' = (A', B')$:

[1] More precisely, the second and the third words of the mill.

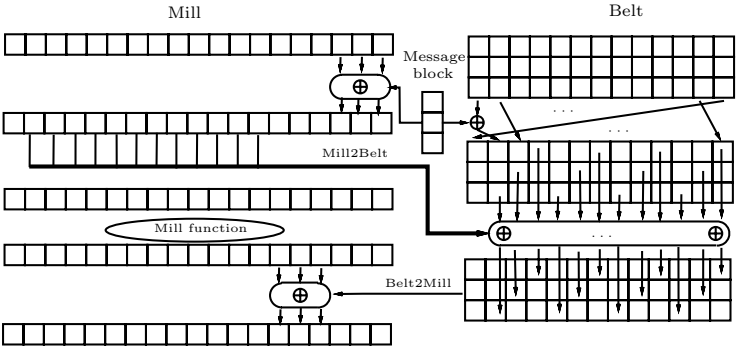


Fig. 1. One round of RadioGatún

- $B \xleftarrow{\text{Message injection } (M) \text{ and shift}} B$;
- $A \xleftarrow{\text{Message injection } (M)} A$;
- $B' \xleftarrow{\text{Mill2Belt feedforward } (A)} B$;
- $A \xleftarrow{\text{Mill function}} A$;
- $A' \xleftarrow{\text{Belt2Mill feedforward } (B')} A$.

Only the Mill function is non-linear. Due to space limits a full description is skipped, so we refer to the original paper [1].

Security. The output of RadioGatún can be considered as a pseudorandom generator, which generates 2 words per step. The designers assume that each application will choose its own length of the hash digest. While for short l-bit outputs the complexity of collision search can be estimated as $2^{l/2}$ RadioGatún calls, this is evidently not for longer ones. As a result, a common security level should have been defined thus providing an upper bound on the complexity of a particular attack.

In the original paper the notion of *capacity* was introduced. The capacity of the ideal iterative mangling function is the size of internal state minus the size of the message block to be injected. However, since the RadioGatún round function is not ideal, the security level of RadioGatún was indicated by a smaller capacity of $19l_w$. This implicitly means that both collision and second-preimage attacks are slower than $2^{9.5l_w}$ though it was not clearly stated. The best non-trivial attack found by the designers requires 2^{46l_w} hash function calls and is substantially slower than the birthday attack, which requires about $2^{27.5l_w}$ hash queries[2]. Now the designers explicitly claim a security level of $2^{9.5l_w}$ operations for both the attacks [9] thus following so called *flat sponge claim* [2]. So we conclude that there is a big gap between the birthday bound with respect to the internal state and the security level. Any attack in this gap, though not breaking

[2] The internal state contain 58 words, but a 3-word flexibility is provided by the injection of a message block not used before. See also Sec. 3.

the security level, could be nevertheless interesting because it should point out weaknesses in the internal transformations.

2 Trails

In order to build a collision we consider so called *differential trails* [1,5,7], or simply *trails*. A trail is a pair of [hash function] iterations with restrictions on internal variables. Such restrictions may be imposed on the *differences* between variables or the *values* of particular variables.

This paper is partly inspired by the attack on Grindahl [12]. In this section some notions from that paper are used (control words, degrees of freedom) so we kindly ask the reader to familiarize with it.

2.1 Symmetric Trails and Trails with Fixed Differences

If a design operates on words of arbitrary length (like RadioGatún) then one may consider so-called *symmetric trails* that deal with word differences of form $000\ldots0$ and $111\ldots1$. In the original paper on RadioGatún [1] and in the attack on Panama [5] symmetric trails were discussed. Such trails are in some sense independent of the word length. However, their probabilities seem to drastically decrease as l_w grows.

Indeed, authors of [1] found a symmetric trail for the RadioGatún with 1-bit words such that a collision search following this trail would require about 2^{46} operations while the birthday bound is $2^{27.5}$. This observation made authors to claim that corresponding symmetric trail for RadioGatún-l_w (RadioGatún with l_w-bit words) would imply 2^{46l_w} as the complexity of the collision search.

However, the following observation make us to disagree with this generalization. Given a trail with fixed (non-truncated) values of differences (not only symmetric ones) an adversary actually knows the input and output differences $(\Delta_{in}, \Delta_{out})$ of the nonlinear function χ, the part of the Mill function. The pair $\langle \Delta_{in}, \Delta_{out} \rangle$ impose a set of conditions on input and output *values*. The number of conditions imposed on the input value is the Hamming weight of the input difference plus the number of 001-patterns[3] in the difference [5][4].

Let us estimate the average number of conditions. The average Hamming weight of the $19l_w$ bit word is $9.5l_w$, the average number of 001-patterns is $17/8l_w$. Thus we have about $11.5l_w$ conditions on the bits of the mill in each round. If there were no injection to the mill, only two rounds would give enough conditions to completely determine the value of the mill. However, $6l_w$ bits are injected to the mill from the belt ($3l_w$) and the message ($3l_w$) thus compensating $6l_w$ conditions so we have about $5.5l_w$ bit conditions per round. As a result, one can define the values of the mill given only about 4 rounds of a trail. However,

[3] The word $\underbrace{001}$ 0000 $\underbrace{001}$ $\underbrace{001}$ 000 contains three 001-patterns.

[4] Full description of function χ and its properties may be found in Daemen's PhD thesis [4, p. 126].

since a full collision trail covers at least 6 rounds, with high probability no message pair fits a given trail.

We can reformulate this result as an informal conjecture.

Conjecture 1. Either a differential trail with fixed values of differences has probability 0 or any 4 consecutive rounds of the trail completely define the message pair.

A counterargument may be that one might find a trail with low Hamming weight of the input difference. However, such a difference is likely to expand to an average one due to diffusion properties in the Mill function. So far there is no example of such low-weight trails.

We conclude that symmetric trails and trails with fixed differences seem to be insufficient to evaluate the security of RadioGatún.

2.2 Truncated Differentials and Linear Space of Differences

The key idea is to consider truncated differentials of linear form and exploit the linearity of transformations in the belt. We take a linear subspace $R \subseteq Z_2^{l_w}$ of dimension r. Let us also consider the first round such that the difference is injected by the message block. Let these injected differences belong to R.

If the Mill function provided an ideal diffusion then the probability that the difference in any word of the mill after applying the Mill function belongs to R would be about 2^{r-l_w}. However, words 0, 3, 6, 10, 11, 14 and 18 of the mill are not affected by the message injection, so there will be zero difference in them after the first round. Thus 8 of the 12 mill words that are feedforwarded to the belt are affected by the message injection. The difference in them is not randomly distributed but one can find R such that the R-difference appears with probability 2^{r-l_w}. The inverse of the Mill function provides the diffusion close to uniform.

An example of R for RadioGatún-8 might be the following space: $R = \{b_7 b_6 \dots b_1 b_0 \mid b_7 = b_6 = b_5 = 0\}$. Let A and A' be random mills such that $A[16] \oplus A'[16]$, $A[17] \oplus A'[17]$, and $A[18] \oplus A'[18]$ belong to R. Apply the Mill function to both mills and compute the difference $\Delta A = \{\Delta_0, \Delta_1, \dots, \Delta_{18}\}$. We made 1000 experiments and observed that the probability that $\Delta_i \in R$ is close to $1/8 = 125/1000$ (see Table 1).

Table 1. Distribution of differences in the output of the Mill function

i	1	2	4	5	7	8	9	12	13	15	16	17
$\#\{\Delta_i \in R\}$	145	134	116	141	115	122	109	134	132	138	129	106

Thus we assume that 8 words enter the Mill2Belt feedforward with the difference from R with probability[5] $2^{8(r-l_w)}$. One of these differences is added to the

[5] We assume the independency of the separate events, which seems to be the case for non-trivial R and quite big (> 7) n.

difference imposed by the message injection. Since any linear space is closed under addition, all 10 non-zero differences (8 from the mill and 2 from the message injection) in the belt belong to R (see also Table 2, round 1).

Now we describe how this idea can be used in attacks.

3 Collision Search

In this section we show how to find a state S and the two different messages m and m' that convert S to the same state. Following the notation from [2] there exist two paths $p \neq q$ from one state to another one. This is usually called *semi-free-start collision attack* [11]. In other words, we build a collision for messages with a chosen IV. Although the IV is fixed to 0 in RadioGatún, the IV that we get in the attack can be any intermediate internal state, which makes the attack interesting.

First we describe a simplified version of the attack, and then introduce several tricks, which lead to a full attack. We apply the meet-in-the-middle approach, because the initial state can be arbitrarily chosen, there is no message schedule, and the round function is invertible. As a result, we can start with a final state and step back.

We start from a set of arbitrary chosen pairs of identical states. We vary injected message blocks during 5 rounds and difference in them and thus get set S_1 of pairs. 5 rounds are required to fill 38 of 39 belt words with differences. The sixth message injection fills the last belt word. We also start from another set of arbitrary chosen pairs of identical states and step *backwards* for 4 rounds varying message blocks and difference in them as well. As a result, we get set S_2 of pairs. If a pair belongs to both sets then we obtain a collision. The complexity of this approach is about 2^{58l_w} hash function queries. This process is briefly illustrated in Figure 2.

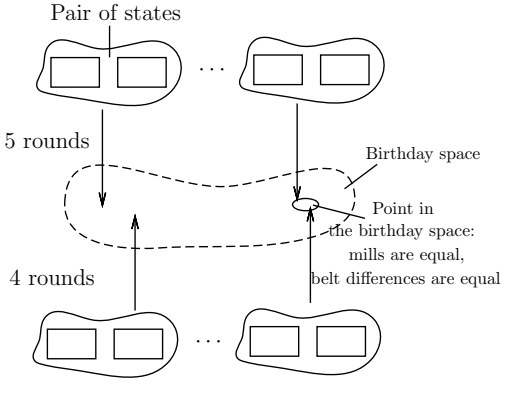


Fig. 2. Outline of the meet-in-the-middle collision search

Table 2. Full trail. Words with differences after the round function is applied.

Round	Words with differences	
	Mill	Belt
1	1, 2, 4, 5, 7, 8, 9, 12, 13, 15, 16, 17	[1,0], [1,1], [1,2]
2	All	[1,0], [1,1], [1,2], [2,0], [2,1], [2,2], [4,2], [5,1], [7,2], [8,1], [9,0], [12,0]
3	All	[0,0], [1,0], [1,1], [1,2], [2,0], [2,1], [2,2], [3,0], [3,1], [3,2], [4,2], [5,1], [5,2], [6,0], [6,1], [7,2], [8,1], [8,2], [9,0], [9,1], [10,0], [10,2], [11,1], [12,0]
4	All	All except [0,1], [0,2], [6,0], [9,0]
5	All	All except [0,2]
6	All	All
7	All	All except [1,2], [2,1], [3,0], [4,2], [5,1], [6,0], [7,2], [8,1], [9,0], [10,2]
8	All	[0,0], [0,1] ,[0,2], [1,1], [2,0], [3,2], [4,1], [5,0], [6,2], [7,1], [8,0], [9,2], [10,1], [11,0], [12,0], [12,1], [12,2]
9	12–18	[0,0], [0,1], [0,2]

Due to space limitations we can not provide here a full graphical representation of the resulting differential trail. However, it can be fully determined by the words with differences after each round. Providing that each message block has difference in all three words we derive the trail described in Table 2. The trail covers 9.5 rounds (after the 10th message injection all the words have zero difference).

In order to compare this approach with the birthday attack we introduce the notion of the *birthday space*. Assume that in order to get a collision we need to obtain two states that fit a particular relation (in the simplest case — two equal states). Then each class of equivalency is a point in the birthday space. The complexity of an attack that uses a birthday paradox is thus the square root of the size of the birthday space. If an adversary seeks an internal collision using the birthday paradox, the coincidence of all the words is not necessary. Since there is no message scheduling, and each message block can be chosen independently, it is enough to obtain two states colliding in all words not affected by the message injection and in three more words, which are the sums of words affected by the corresponding message words. Thus we have the birthday space of dimension $55l_w$, and the complexity of the birthday attack is $2^{27.5l_w}$, which is smaller than our naive meet-in-the-middle attack, where the dimension of the birthday space is $2 * 19l_w + 2 * 39l_w = 116l_w$.

Next we show that pairs may not completely coincide. First we relax restrictions on the belt and show that the equality of the difference in the belt is enough.

Proposition 1. *The belt-to-mill feedforward in $n \leq 13$ consecutive rounds can be considered as injection of n independently chosen 3-word blocks.*

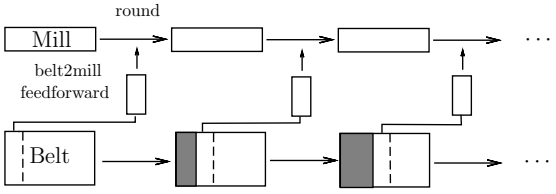


Fig. 3. Scheme of the belt recovery

Proof. Indeed, one can recover the belt from any $n \le 13$ 3-word blocks without contradiction. This can be proved by the following observation. Let us derive the values of the belt words consecutively while iterating one round after another. At each step we derive 3 more known words and use the known message and mill values to carry out the known values to the next step. Due to slow rotation new values cover consecutive columns in the belt. This process is briefly illustrated in Figure 3.

Formally, denote the belt in the beginning of the trail by B and in the end by B'. It is easy to see that belt words are not mixed with each other, only message and mill words are added. Thus $B'[i,j] = B[i-n,j] + f(i,j)$ where n is the trail length and f is a function of the message and the mills. The belt words that are feedforwarded to the mill are derived from distinct B words. Thus giving any set of feedforward blocks, the mill values and message blocks one can recover the original B without contradiction.

Let us return to the 9-round trail. If a state from S_1 and a state from S_2 coincide in the mill and in the difference in the belt then the corresponding parts of the trail can be combined into one trail. At the same time, 27 words of the initial belt are recovered. The other 12 words can be assigned randomly. The dimension of the birthday space is $2 * 19l_w + 39l_w = 77l_w$ so the complexity of the second version of the attack is about $2^{38.5l_w}$.

Linear Truncated Differentials. In order to reduce the birthday space we impose restrictions on the differences that are fed to the belt. We choose integer $r \le l_w$ (the exact value of r will be defined later) and a linear space $R \subset Z_2^{l_w}$ of dimension r that fits the assumptions of uniformity (see section 2.2). In order to obtain a desired difference we vary the injected messages. We choose the first message block in the pair randomly thus having $3l_w$ *degrees of freedom* (see also [12]). The second message should have the R-difference with the first one so we have $3r$ more degrees of freedom. Thus the probability that we find the words to be injected such that a given pair pass through the next round with R-restriction is $2^{8r-8l_w+3r+3l_w} = 2^{11r-5l_w}$ (if this value exceeds 1 then we just obtain more pairs).

The latter value can be also considered as a *multiplier* c such that if N pairs enter the round then $c \cdot N$ pairs with R-difference can be obtained from them after one iteration.

We need 5 rounds and one more message injection to fill the 39 words of the belt (we use the trail presented in Table 2) with R-differences. Let (A, B) denote the internal state as a pair of the mill and the belt in the beginning of sixth round. We also require that the 12 words of A that are feeded to the belt in the sixth round should also have R-difference (this is arranged by the message injection in rounds 4 and 5). To sum up, we need 56 R-difference words in the mills during five rounds while the freedom provided by injections is $5*(3r+3l_w) = 15r+15l_w$. Additionally, we randomly choose the words that are feedforwarded from the belt (see Proposition 1) in rounds 1-4 thus having $12l_w$ more degrees of freedom. As a result, if we start with 2^{n_1} pairs then $2^{n_1+15r+27l_w+56r-56l_w} = 2^{n_1+71r-29l_w}$ pairs pass through five rounds.

Now we consider the second part of the trail and proceed back from the aero-difference state. Only 3 message injections are needed to fill the belt with R-differences. However, the difference in the mill would coincide with the difference in the 12 words of the belt. We add one more round. Thus 48 words with R-difference should be obtained during the process. The multiplier is

$$2^{4*(12r-12l_w+3r+3l_w+3l_w)} = 2^{60r-24l_w}.$$

Finally, let us calculate the dimension of the birthday space. Recall that we need that pairs should coincide in the value of the mill, in the difference of the mill, and in the difference of the belt. The dimension of the resulting birthday space is $19l_w+(12r+7l_w)+39r = 51r+26l_w$. However, we have not used the freedom that is provided by the message injection in round 6 and the bell-to-mill feedforward in round 5 yet. This freedom allows us to further relax the restriction on the coincidence of pairs: we do not care of values of the mill in 6 words and of 3 word differences. Finally, the resulting birthday space is of dimension $20l_w+48r$.

Now we compute r such that the number of pairs throughout the attack is minimal. Let us denote by 2^{n_1} and 2^{n_2} the number of pairs that we start with from the first round and from the last round, respectively. Then the number of pairs and the complexity of the attack[6] is bounded by $\max(2^{n_1}, 2^{n_1+71r-29l_w}, 2^{n_2}, 2^{n_2+60r-24l_w})$. The second requirement is that the number of pairs in the middle round should be enough to perform the birthday attack: $(n_1+71r-29l_w)+(n_2+60r-24l_w) = 20l_w+48r$. The best solution is provided by the r equal to $0.4l_w$. This implies the equation $n_1+n_2 = 39.8l_w$. The resulting complexity is $2^{19.9l_w}$.

Relaxation. Further we note that several words in the belt are updated by the mill words twice during the first 5 rounds. Since we need R-difference only in the middle state, arbitrary difference can be injected at the first time and later converted to the R-difference. As before, we expect the probability of getting an R-difference as 2^{r-l_w}. After this *relaxation* we have no restrictions on the difference in the message injection in the first round (the idea is illustrated in Figure 4). Furthermore, we have no restrictions on the mill difference in the first round. The only difference that should be maintained by the first injection is the difference in 3 mill words after round 2.

[6] We assume that the search for appropriate message blocks and belt words is of negligible cost and can be maintained with a lookup table.

Following this approach we obtain probability 2^{48r-6l_w} for a random pair to come out of the first part of the trail. The probability for the second part of the trail (reverse process) is $2^{48r-12l_w}$. However, the number of pairs is no longer a monotonic function of the round number, so we adjust the value of r in order to keep the number of considered pairs minimal during the attack. The resulting complexity is about 2^{18l_w} hash function queries with $r = 4/13l_w$ and the birthday space of dimension $48r + 20l_w \approx 34l_w$. The number of pairs after every round is given in Table 3.

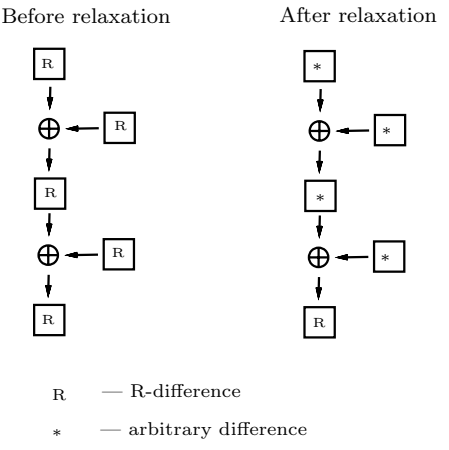


Fig. 4. Idea of the relaxation

Strengthening. The fact that the number of pairs is not a monotonic function of the round number means that degrees of freedom are not properly used. Here we notice that after relaxation most words with R-differences are not added to each other so we can omit the restriction on linearity. One may consider a *group of differences* (instead of a linear space) of arbitrary size between 1 and 2^{l_w}.

In order to flatten the function of the number of pairs we consider particular words in the mill and *strengthen* the restriction on differences in them taking another space R for a particular word. As a result, we deal with several different R's, each with its own size.

The benefit is given as follows. Suppose we work with a two-round trail. The number of pairs is N before the first round, $2^l N$ ($l > 0$) before the second round, and N after the second round. Then the the overall complexity is bigger than both the initial and end values and is equal to $2^l N$. If we follow the idea of strengthening and add l more conditions on the difference after the first round (and in the end) then the number of pairs is reduced to N after the first round and to $\frac{N}{2^l}$ in the middle. The dimension of the middle space is also decreased by

Table 3. The complexity of the collision search after the relaxation ($r = 4/13l_w$)

Round	Degrees of freedom	Words to control	Number of pairs ($\log_2$)
0	-	-	$12.5l_w$
1	$6l_w$	3	$14.5l_w$
2	$9l_w$	10	$16.6l_w$
3	$9l_w$	11	$18l_w$
4	$9l_w$	13	$18l_w$
5	$9l_w$	13	$18l_w$

Round	Deg. of freedom	Words to control	N-r of pairs
10	-	-	$13.8l_w$
9	$9l_w$	10	$15.9l_w$
8	$9l_w$	10	$18l_w$
7	$9l_w$	13	$18l_w$
6	$9l_w$	15	$16l_w$

l. In order to maintain the birthday attack we must increase the initial number of pairs from N to $N2^{l/2}$. The complexity of the attack is thus reduced to $N2^{l/2}$.

Theoretical lower bound. One may ask the question what the smallest complexity is that we can achieve following the ideas of linear differences, relaxation and strengthening. Let us recall that the dimension of the middle space without restrictions on differences is $77l_w$. If we impose P linear restrictions on differences then the dimension will be $77l_w - P$. On the other hand, we have $51l_w$ degrees of freedom (provided by 6 message blocks and belt2mill feedforward blocks) to compensate the restrictions. Thus the multiplier of the first part of the trail is 2^{51l_w-P}. The lowest complexity is achieved if the multiplier is equal to 1, so we obtain $P = 51l_w$ and the dimension of the middle space is $26l_w$. The number of pairs required by the birthday attack is 2^{13l_w} which is the lower bound.

4 Second Preimage Search

The idea of the second-preimage attack is similar to a simple collision one. While we looked for collisions with arbitrary pairs, in the second-preimage attack the first element of every pair is fixed and is equal to the original internal state. We pull a number of states through the iteration process from both ends and look for the coincidence in the middle round. We vary injected messages in order to obtain R-differences in the middle round. The trail is similar to that is given in Table 2, but the zero differences are now arbitrary differences.

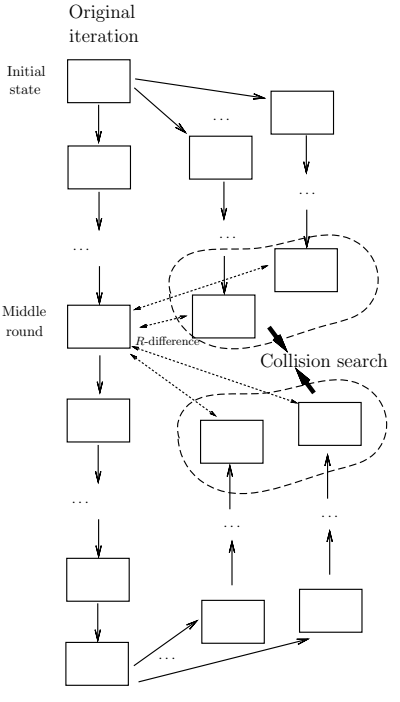


Fig. 5. Outline of the second preimage search

Let us consider 10 rounds of the hash iteration to which we want to find a second preimage (called below the *original iteration*). Where these rounds should be located will be discussed later. Denote the internal states of the original iteration in the beginning of 11 consecutive rounds: $I_0, I_1, \ldots, I_{10}$. Suppose we also have N_1 states (the exact value will be also defined later) that are resulted from iteration of the original zero state with some random message. Then we consider N_1 differences between these states and the state I_0 as the first difference in the 10-round trail, which is obtained from the trail in Table 2 by adding one more round in the beginning and replacing zero differences with arbitrary differences. Next for each

of N_1 states we look for the 6-block messages that provide an R-difference state in the middle round (a state that has an R-difference in every word with the state I_6). As a result, we obtain a set S_1 of internal states. See also Figure 5 as an illustration.

Similarly, suppose we have N_2 states that are resulted from reverse iteration of the last internal state of the original iteration. We treat them in the similar way and look for the 4-block messages that provide an R-difference state in the middle round. Thus we obtain a set S_2 of internal states. Then we look for a state that is presented in both sets. Such a state implies a parallel iteration, which gives the same hash value.

Now let us estimate what are N_1, N_2 and the complexity of the attack. The injection in round 0 controls 4 mill words in the end of round 1 such that the resulting difference belong to R. The injections in rounds 1-4 control 12 words and the last one control 8 words. Thus the probability that a state can be pulled to the middle state with R-differences is $2^{(4+12*4+8)(r-l_w)+3*6r} = 2^{78r-60l_w}$. The same idea holds for reverse steps. We start with N_2 states and the proportion $2^{4*(12r-12l_w+3r)} = 2^{60r-48l_w}$ of them comes out of the iteration.

The dimension of the birthday space in the beginning of 6-th round is $7l_w+51r$ (12 words of the mill and all the words of the belt must have R-difference, and the other 7 mill words may have arbitrary difference). Given $3r$ more degrees of freedom from the message injection in round 6 we derive that $2^{3.5l_w+24r}$ internal states are required to perform the birthday attack.

The optimal complexity is given by the $r = 0.8l_w$ that converts the multiplier $2^{60r-48l_w}$ to one. Thus we derive

$$N_1 = 2^{3.5l_w+24r+60l_w-78r} = 2^{20.3l_w}; \quad N_2 = 2^{3.5l_w+24r+48l_w-60r} = 2^{22.7l_w}.$$

If we follow the method of relaxation and strengthening as described in Section 3 then the complexity about 2^{20l_w} could be achieved. This is actually the lower bound for these meet-in the middle attacks with 10-round trails, which can be checked following the method in Section 3.

5 Implementation of Attacks

Though optimal r might be non-integer, we can take concrete values just to check whether our approach works in real life. Due to high complexity of the attack even with small number bits in a word we can not perform the attack as a whole but we tested the RadioGatún round function on different spaces R and encountered good distribution of differences in the output (see Table 1), especially in reverse steps. We also checked that values in the belt words and message blocks to be injected can be chosen such that the desired differences appear in the output of the non-linear function. Thus we substantiated the main assumptions made throughout the description of the attack.

One may also argue that RadioGatún with reduced number of *words* in the belt and in the mill may be considered as an easier object for the attack. However, the reduced round function and its inverse do not provide good differential

Table 4. Summary of attacks on RadioGatún

Attack	Type	Complexity	Origin
Collisions	Symmetric trails	2^{46l_w}	[1]
	Birthday	$2^{27.5l_w}$	-
	R-difference	$2^{19.9l_w}$	This paper
	After relaxation	2^{18l_w}	This paper
Second preimage search	Birthday	$2^{27.5l_w}$	-
	R-differences	$2^{22.7l_w}$	This paper
	After relaxation	$\sim 2^{20l_w}$	This paper*

* – hypothetical.

characteristics (close to random) anymore. We checked this for the internal state that is reduced threefold. This (non-uniformity) is also the case with small l_w (the number of bits in a word), which makes our attack inefficient.

We also note that the attack becomes trivial for RadioGatún-1 since there are only two options for R, and both of them give high complexity.

6 Conclusions

We investigated the security of RadioGatún using differential trails with linear restrictions on differences. We applied the meet-in-the-middle approach and managed to reduce the complexity with help of new tricks such as relaxation and strengthening. We showed how to find semi-free-start collisions with complexity about 2^{18l_w} hash function calls and the second preimage with about $2^{22.7l_w}$ calls (with a possible improvement up to 2^{20l_w}). We also provided theoretical lower bounds on the complexity of the attack which follow the same approach.

The main weakness of the RadioGatún round function that we exploited is plenty of linear operations and slow diffusion in the belt. We suppose that a compromiss between adding more non-linearity in the primitive transformations and the speed might be found so the design could be seriously strengthened and the security level could be increased (say, up to 2^{16l_w}). As a result, a smaller versions (in terms of l_w) could be used as a 256/384/512-bit hash function.

Regarding RadioGatún itself, though our attacks do not break the claimed security level ($2^{9.5l_w}$), they are faster than the birthday attack and the attack that might be carried out from Grindahl [12]. Thus we conclude that RadioGatún is still resistant against differential-based collision search though this resistance is now provided only by a substantially low security level.

Acknowledgements

The author greatly thanks Alex Biryukov, Ivica Nikolic, Stefan Lucks, Joan Daemen and the RadioGatún team, and the anonymous reviewers for their valuable and helpful comments. The author is supported by PRP "Security & Trust" grant of the University of Luxembourg.

References

1. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Radiogatun, a belt-and-mill hash function. In: NIST Cryptographic Hash Workshop (2006)
2. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: ECRYPT Hash Workshop (2007), http://sponge.noekeon.org/
3. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the indifferentiability of the sponge construction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 181–197. Springer, Heidelberg (2008)
4. Daemen, J.: Cipher and hash function design strategies based on linear and differential cryptanalysis. PhD thesis, K.U.Leuven (March 1995)
5. Daemen, J., Van Assche, G.: Producing collisions for PANAMA, instantaneously. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 1–18. Springer, Heidelberg (2007)
6. Daemen, J., Clapp, C.S.K.: Fast hashing and stream encryption with PANAMA. In: Vaudenay, S. (ed.) FSE 1998. LNCS, vol. 1372, pp. 60–74. Springer, Heidelberg (1998)
7. Daemen, J., Rijmen, V.: The Design of Rijndael. AES — the Advanced Encryption Standard. Springer, Heidelberg (2002)
8. Cryptographic Hash Project, http://csrc.nist.gov/groups/ST/hash/index.html
9. http://radiogatun.noekeon.org/
10. Knudsen, L.R., Rechberger, C., Thomsen, S.S.: The Grindahl hash functions. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 39–57. Springer, Heidelberg (2007)
11. Menezes, A., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
12. Peyrin, T.: Cryptanalysis of Grindahl. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 551–567. Springer, Heidelberg (2007)

Faster Multicollisions*

Jean-Philippe Aumasson

FHNW, Windisch, Switzerland

Abstract. Joux's multicollision attack is one of the most striking results on hash functions and also one of the simplest: it computes a k-collision on iterated hashes in time $\lceil \log_2 k \rceil \cdot 2^{n/2}$, whereas $k!^{1/k} \cdot 2^{n(k-1)/k}$ was thought to be optimal. Kelsey and Schneier improved this to $3 \cdot 2^{n/2}$ if storage $2^{n/2}$ is available and if the compression functions admits easily found fixed-points. This paper presents a simple technique that reduces this cost to $2^{n/2}$ and negligible memory, when the IV can be chosen by the attacker. Additional benefits are shorter messages than the Kelsey/Schneier attack and cost-optimality.

Keywords: hash function, collision.

1 Introduction

Cryptographic hash functions are key ingredients in numerous schemes like public-key encryption, digital signatures, message-authentication codes, or multiparty functionalities. The last past years the focus on hash functions has dramatically increased, because of new attacks on the compression algorithm of MD5 and SHA-1 and on their high-level structure, e.g. *multicollision attacks*. We introduce these attacks below.

Consider an arbitrary function $f : \{0,1\}^n \times \{0,1\}^m \mapsto \{0,1\}^n$. A classic construction [24,25] defines the *iterated hash* of f as the function

$h_{H_0}(M_1 \dots M_\ell)$:
  for $i = 1, \dots, \ell$ **do**
    $H_i \leftarrow f(H_{i-1}, M_i)$
  return H_ℓ

where H_0 is called the *initial value* (IV), and f the *compression function*. Damgård and Merkle [7,18] independently proved in 1989 that h is collision-resistant if f is collision-resistant when the bitlength of the message is appended at its end (a technique referred as *MD-strengthening*). This technique also prevents the *fixed-point attack*—a folklore multicollision attack—whose basic idea is that if M satisfies $f(H_0, M) = H_0$, then $h_{H_0}(M \dots M) = H_0$.

The problem we will focus on is how quickly one can compute k distinct messages mapping by h_{H_0} to the same value, when MD-strengthening is applied (call

* Article previously accepted to SECRYPT 2008, but withdrawn by the author because unable to attend the conference. This author was supported by the Swiss National Science Foundation under project no. 113329.

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 67–77, 2008.

this a *k-collision*). An extension of the birthday attack computes k-collisions[1] within about $k!^{1/k} \cdot 2^{n(k-1)/k}$ calls to f, which was believed to be the optimal until the technique of [10] that requires only $\lceil \log_2 k \rceil \cdot 2^{n/2}$ f-calls. Kelsey and Schneier subsequently reduced this cost to $3 \cdot 2^{n/2}$ [12], provided that storage $2^{n/2}$ is available, and that f admits easily found fixed-points. Though seldom cited, this technique is more powerful than Joux's in the sense that the cost of finding a k-multicollision is independent of k, yet a drawback is the length of the colliding messages, significantly larger.

1.1 Contribution

This paper reviews the previous techniques for computing k-collisions, and presents a novel method whose main features are

- a cost independent of the number of colliding messages k (with $2^{n/2}$ trials)
- short colliding messages (with $\lceil \log_2 k \rceil$ blocks)
- negligible storage requirements

Limitations of the attack are the need for easily found fixed-points, and the IV chosen by the attacker. This means that the IV used for the multicollisions cannot be set to a predefined value, which corresponds to the model called "semi-free-start collisions" in [14], "collision with different IV" in [21], and "collision (random IV)" in [17]. Within this model, our technique is optimal, because k-collisions become as expensive as collisions.

The practical impact of this attack is limited, because it does not break the complexity barrier $2^{n/2}$. However, in terms of price/performance ratio (or "value" [21, §2.5.1]) it outperforms all the previous attacks, since for the same price as a collision, one gets k-collisions.

1.2 Related Work

Multicollisions received a steady amount of attention since Joux's attack: [19,9] generalized them to constructions where a message block can be used multiple times; [30] revisited the birthday attack for multicollision; dedicated multicollision attacks were found for MD2 [13] and MD4 and HAVAL [31]. Finally, [11] used multicollisions for the "Nostradamus attack".

1.3 Notations

Let $f : \{0,1\}^n \times \{0,1\}^m \mapsto \{0,1\}^n$ be the compression function of the iterated hash h_{H_0}, for an arbitrary H_0, where MD-strengthening is applied. If f admits easily found fixed-points, write $\mathsf{FP}_f : \{0,1\}^m \mapsto \{0,1\}^n$ a function such that for all M, $\mathsf{FP}_f(M)$ is a fixed-point for f, i.e. $f(\mathsf{FP}_f(M), M) = \mathsf{FP}_f(M)$.

Then, fix a unit of *time* (e.g. an integer addition, a call to f, a MIPS-year, etc.), and a unit of *space* (e.g. a bit, a 32-bit word, a n-bit chaining value, a

[1] Plural is used because from any k-collision we can derive many other k-collisions, by appending the same arbitrary data at the end of colliding messages.

128 Gb hard drive, etc.), and write the cost of computing f as $\mathbf{T}_f$ time units and $\mathbf{S}_f$ space units (resp. $\mathbf{T}_{\mathsf{FP}}$ and $\mathbf{S}_{\mathsf{FP}}$ for FP_f); we assume these costs input-independent; we disregard the extra cost of auxiliary operations and memory accesses (though of certain practical relevance); we also disregard the constant factor caused by "memoryless" birthday attacks [29,23].

Note that our goal is to find (the description of) many messages with same digest, not to effectively construct them. Hence, the time cost of finding a k-collision is not lower-bounded by k (e.g. k steps of a Turing machine), neither are the space requirements.

2 Joux Multicollisions

This method computes 2^k-collisions for k times the cost of finding a single collision: Assuming $m < n$, first compute a colliding pair (M_1, M_1'), i.e. such that $f(H_0, M_1) = f(H_0, M_1') = H_1$, then compute a second colliding pair (M_2, M_2') such that $f(H_1, M_2) = f(H_1, M_2') = H_2$, and so on until (M_k, M_k') with H_{k-1} as IV. Hence, for a symbol $X \in \{M, M'\}$, any of the 2^k messages of the form $X_1 \dots X_k$ has intermediate hash values $H_1, \dots, H_k$, and 2^k-collisions can be derived from these 2^k messages by appending extra blocks with correct padding. The cost of the operations above is time $k \cdot 2^{n/2} \cdot \mathbf{T}_f$, and negligible space.

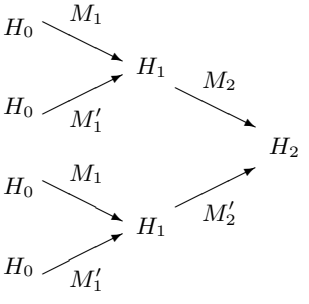


Fig. 1. Illustration of Joux's method for $k = 2$: first a collision $f(H_0, M_1) = f(H_0, M_1') = H_1$ is computed, then a second collision $f(H_1, M_2) = f(H_1, M_2') = H_2$ is found; the 4 colliding messages are M_1M_2, M_1M_2', $M_1'M_2$, and $M_1'M_2'$

Fig. 1 gives an intuitive presentation of the attack; computing a 2^k-collision can be seen as the bottom-up construction of a binary tree, where each collision increases by one the tree depth. Note that a chosen IV does not help the attacker.

3 Kelsey/Schneier Multicollisions

As an aside in their paper on second-preimages, Kelsey and Schneier reported a method for computing k-collisions when f admits fixed-points [12, §5.1]; an advantage over Joux's attack is that the cost no longer depends on k. Here we will detail this result, which benefited of only a few informal lines in [12], and is seldom refered in literature.

3.1 Fixed-Points

A *fixed-point* for a compression function f is a pair (H, M) such that $f(H, M) = H$. For a random f finding a fixed-point requires about 2^n trials, by brute force search. Because it does not represent a security threat *per se*, neither it helps to find preimages or collisions, that property has not been perceived as an undesirable attribute: in 1993, Preneel, Govaerts and Vandewalle considered that "this attack is not very dangerous" [22], and according to Schneier in 1996, this "is not really worth worrying about" [28, p.448]; the HAC is more prudent, writing "Such attacks are of concern if it can be arranged that the chaining variable has a value for which a fixed point is known" [17, §9.102.(iii)].

The typical example is the Davies-Meyer construction for blockcipher-based compression functions, which sets $f(H, M) = E_M(H) \oplus H$. Hence, for any M a fixed point is $(E_M^{-1}(0), M)$:

$$E_M(E_M^{-1}(0)) \oplus E_M^{-1}(0) = 0 \oplus E_M^{-1}(0) = H.$$

Therefore, each message block M has a unique H that gives $f(H, M) = H$ and that is trivial to compute[2].

Note that the functions MD4/5 and SHA-0/1/2 all implicitly follow a Davies-Meyer scheme (where integer addition replaces XOR). More generally, an iterated hash may admit fixed-points for a sequences of compressions rather than a single compression—e.g. for two compressions, defining $f'(H, M, M') = f(f(H, M), M')$. Generic multicollision attacks apply as well to this type of function, up to a redefinition of f and m.

3.2 Basic Strategy

We first consider the simplest case, i.e. when any IV is allowed. Recall the fixed-point attack mentioned in §1, which exploits a fixed-point $f(H, M) = H$ to build the multicollision $h_H(M) = h_H(MM) = h_H(MMM \ldots M) = H$. MD-strengthening protects against this attack, since it forces the last blocks of the messages to be distinct. The idea behind Kelsey/Schneier multicollisions is to bypass MD-strengthening using a *second fixed-point*. This fixed-point will be used to adjust the length of all messages to a similar value, to get the same padding data in all messages. Fig. 2 illustrates this attack: fix $n > 2$; if the first fixed-point is repeated k times, then the second fixed-point is repeated $n - k$ times to have n blocks in total. The last block imposed by MD-strengthening will thus be the same for all messages. The second fixed-point is integrated via a meet-in-the-middle technique (MITM) that goes as follows:

1. Compute a list L_1:

$$(M_1, f(H_0, M_1)), \ldots, (M_{2^{n/2}}, f(H_0, M_{2^{n/2}})).$$

2. Compute a list L_2:

$$(M'_1, \mathsf{FP}(M'_1)), \ldots, (M'_{2^{n/2}}, \mathsf{FP}(M_{2^{n/2}})).$$

[2] Similar fixed-points can be found for the constructions numbered 5 to 12 in [22].

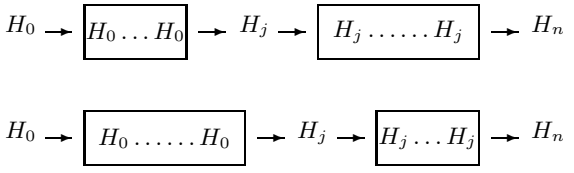


Fig. 2. Schematic view of the Kelsey/Schneier multicollision attack, for an IV chosen by the attacker: a first fixed-point allows to expand the message, while a second one adjust the lengths to a similar value

3. Look for a collision on the second pair element $(M_i, H_j) \in L_1, (M'_j, H_j) \in L_2$.
4. Construct colliding messages of the form $M_i \ldots M_i M'_j \ldots M'_j$, such that the length of the whole message is kept constant.

The attack runs in time $2^{n/2} \cdot \mathbf{T}_f + 2^{n/2} \cdot \mathbf{T}_{\mathsf{FP}}$, and needs storage $\mathbf{S}_f + \mathbf{S}_{\mathsf{FP}} + 2^{n/2} \cdot \mathbf{S}_{(n+m)}$, with $\mathbf{S}_{(n+m)}$ the space used to store a $(n+m)$-bit string. These values are independent of the size of the multicollision. The length of messages is addressed later.

When the IV is restricted to a specific value, the first fixed-point has to be introduced with another MITM; time cost grows to $2 \cdot 2^{n/2} \cdot \mathbf{T}_f + 2^{n/2} \cdot \mathbf{T}_{\mathsf{FP}}$, and storage is similar (the second MITM reuses the space allocated for the first one).

3.3 Multiple Fixed-Points and Message Length

In the above attack, a k-collision contains messages of about k blocks. In comparison, Joux's method produces messages of $\lceil \log_2 k \rceil$ blocks. This gap can be reduced by using more than two fixed-points: Assume that $K > 2$ fixed-points are integrated in the message. The attack now runs in time $(K-1)(2^{n/2} \cdot \mathbf{T}_f + 2^{n/2} \cdot \mathbf{T}_{\mathsf{FP}})$, counting $(K-1)$ MITM's, for a chosen IV. Also suppose a limit of ℓ blocks per message (e.g. a maximum number of blocks allowed by a design, typically 2^{64}), with $\ell > 2K$.

Given the limit ℓ, how large can be a multicollision in terms of K? The number of constructible colliding messages is equal to the number of *compositions* of ℓ having at most K non-null summands[3]. The number we are looking for is $\mathcal{C}_{\ell,K} = \sum_{i=0}^{K-1} \binom{\ell}{i}$ (summing over the number of separators), so we will get a $\mathcal{C}_{\ell,K}$-collision.

For example, consider SHA-256, which admits fixed-points: with $K = 8$ one finds 2^{57}-collisions in time about $14 \cdot 2^{128}$, with 1024-block messages; in comparison Joux's method computes 2^{57}-collisions in time about $57 \cdot 2^{128}$, with 57-block messages, and if we fix the message length to 1024 it finds 2^{1024}-collisions, in time about $1024 \cdot 2^{128}$. This stresses that a small number of fixed-points leads to much longer messages. Performance becomes similar for the two attacks (in terms of time cost, message length, and k) when $K = \lfloor \ell/2 \rfloor$.

[3] A composition (or ordered partition) of a number is a way of writing it as an ordered sum of positive integers. For example, 3 admits four compositions: 3, $2+1$, $1+2$, $1+1+1$.

4 Faster Multicollisions

This section presents a method applicable when the compression function admits easily found fixed-points (like MD5, SHA-1, SHA-256), and when the IV can be chosen by the attacker. Despite its relative simplicity it has not mentioned in the literature, as far as we know.

4.1 Description

The key idea of the attack is that of *fixed-point collision*, i.e. a collision for the function FP_f; since FP_f outputs n-bit this costs time $\mathbf{T}_{\mathsf{FP}} \cdot 2^{n/2}$ and space $\mathbf{S}_{\mathsf{FP}}$. A fixed-point collision is a pair (M, M') such that $\mathsf{FP}_f(M) = \mathsf{FP}_f(M') = H_0$, and thus $f(H_0, M) = f(H_0, M') = H_0$. The distribution of H_0 (as a random variable) depends on f and FP_f; e.g. for Davies-Meyer schemes based on a pseudoranom permutation (PRP), this will be uniform.

Once found a fixed-point collision (M, M'), a 2^k-collision can be constructed by considering all the k-block sequences in the set $\{M, M'\}^k$ followed by an arbitrary sequence of blocks $M^\star$ with convenient padding. For example, a 4-collision will be

$$\begin{aligned}
H_0 &\xrightarrow{M} H_0 \xrightarrow{M} H_0 \xrightarrow{M^\star} H \\
H_0 &\xrightarrow{M} H_0 \xrightarrow{M'} H_0 \xrightarrow{M^\star} H \\
H_0 &\xrightarrow{M'} H_0 \xrightarrow{M} H_0 \xrightarrow{M^\star} H \\
H_0 &\xrightarrow{M'} H_0 \xrightarrow{M'} H_0 \xrightarrow{M^\star} H
\end{aligned}$$

The sole significant computation is for finding a fixed-point collision, hence the whole attack costs time $\mathbf{T}_{\mathsf{FP}} \cdot 2^{n/2}$ and memory $\mathbf{S}_{\mathsf{FP}}$ (with negligible overhead). For instance, for a Davies-Meyer function computing FP_f has the same cost as computing f, thus time cost is $\mathbf{T}_f \cdot 2^{n/2}$. Observe that the attack requires no call to the compression function itself, but just to the derived function FP_f.

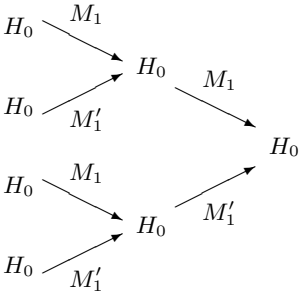


Fig. 3. Illustration of our technique for $k = 2$: a fixed-point collision $f(H_0, M_1) = f(H_0, M_1') = H_0$ is computed, then the four colliding messages are M_1M_1, M_1M_1', $M_1'M_1$, and $M_1'M_1'$. Contrary to Joux's attack, H_0 is here chosen by the attacker.

If computing fixed-points is nontrivial but easier than expected, this attack becomes more efficient than Joux's as soon as $k > \mathbf{T}_{\mathsf{FP}}/\mathbf{T}_f$ (for computing 2^k-collisions).

4.2 Finding Fixed-Point Collisions

For a PRP-based Davies-Meyer compression function, the cost of finding a fixed-point collision (i.e. $\mathsf{FP}_f(M) = \mathsf{FP}_f(M')$) equals the cost of finding a collision (i.e. $f(H_0, M) = f(H_0, M')$); indeed in both cases the function is essentially one query to the PRP, thus the same refined birthday-based methods can be used [29,23].

This suggests that for Davies-Meyer functions (like MD5, SHA-1, SHA-256) finding a fixed-point collision is cost-equivalent to finding a collision: indeed the goal is now to find (M, M') such that $E_M^{-1}(0) = E_{M'}^{-1}(0)$, while classical collisions need $E_M(H) = E_{M'}(H)$. Therefore, if E is a PRP then finding a fixed-point collision with fixed IV is exactly as hard a finding a collision.

For hash functions that don't have obvious fixed-points, finding a fixed-point collision is at least as hard as finding a collision. Contrary to Davies-Meyer schemes, the ability to find fixed-IV collisions does not directly allow to find fixed-point collisions.

The statements above cover other blockcipher-based schemes that allow the easy finding of fixed-points (cf. the 8 schemes in [22]). We conjecture that known techniques for finding collisions on MD5 and SHA-1 can be adapted to find fixed-point collisions within similar complexity.

4.3 Distinct-Length Multicollisions

The attacks of Joux and Kelsey/Schneier find colliding messages of same length. A variant of our technique allows to find sets of messages that collide and do not all have the same block length. The idea is to find a fixed-point collision $f(H, M) = f(H, M') = H$ such that M and M' contain valid padding bits, that is, are of the form $\ldots 10 \ldots 0 \| \ell$. The chosen message bitlength ℓ should be different for M and M', and be consistent with the number of zeros added. Finding a fixed-point collision with these restrictions is not more expensive than in the general case as soon as at least $n/2$ bits in the message blocks are not padding bits.

Once a pair (M, M') with the above conditions is found, we can directly describe multicollisions. Suppose for example that $M = \ldots 10 \ldots 0 \| \ell$ and $M' = \ldots 10 \ldots 0 \| \ell'$, where ℓ encodes the length of a 2-block message, and ℓ' encodes the length of a 3-block message. Then the messages $M \| M$, $M' \| M$, $M \| M \| M'$,..., $M' \| M' \| M'$ all have the same hash value by h_H, and have suitable message length encoding.

4.4 Comparison to Joux and Kelsey/Schneier

Compared to Joux's technique, ours has the advantage of a cost independent of k; optimality of the algorithm follows (with respect to the assumption that

a single collision costs at least $2^{n/2}$ f-calls). Compared to Kelsey/Schneier, our technique benefits of short messages ($\lceil \log_2 k \rceil$ for a k-collision), and no storage requirement. However, our attack is limited by the chosen IV, which makes it irrelevant for many applications of hash functions.

Consider for example an attacker with $2^{130} \cdot \mathbf{T}_f$ power to attack SHA-256: with Joux's technique he finds 4-collisions, with Kelsey/Schneier's he finds k-collisions with k-block messages if memory $2^{128} \cdot \mathbf{S}_{(768)}$ is available, and with our method he finds k-collisions of length $\lceil \log_2 k \rceil$ for 4 different IV's, for any k.

4.5 Application to Concatenated Hash Functions

Let the hash function $\mathcal{H}(M) = h_{H_0}(M) \| h'_{H'_0}(M)$, where h is an iterated hash whose compression function f admits fixed-points, and h' and ideal hash function (in practice, h and h' might be the same function, and use different IV's). Suppose further that both hash to n-bit digests.

A basic birthday attack finds collisions on $\mathcal{H}$ within 2^n calls to h, and as many to h'; Joux reduced this cost to $n/2 \cdot 2^{n/2} \cdot \mathbf{T}_f + 2^{n/2} \cdot \mathbf{T}_{h'}$. Our multicollision technique applies similarly, if the IV of h can be chosen by the attacker: first compute a $2^{n/2}$-collision for h, in time $2^{n/2} \cdot \mathbf{T}_{\mathsf{FP}}$, then look for a collision on h' among these messages, in time $2^{n/2} \cdot \mathbf{T}_{h'}$. Assuming $\mathbf{T}_{h'} = \mathbf{T}_f$, we get an overall cost $2^{n/2+1} \cdot \mathbf{T}_f$, instead of $(n+1) \cdot 2^{n/2} \cdot \mathbf{T}_f$ with Joux's technique. Our method is almost optimal, since it almost reaches the cost of computing a collision on h or h' (up to a factor 2).

4.6 Countermeasures

The foremost question is "do we really need countermeasures?" A pragmatic answer would be negative, arguing that the barrier $2^{n/2}$ remains intact thus the security level is not reduced; however, from a price/performance perspective, security is clearly damaged. So if cheap countermeasures exist there seems to be really few reasons to ignore them.

The first obvious measure against our attacks and Kelsey/Schneier's is to avoid easy-to-find fixed-points. For example by using one of the four blockcipher-based constructions in [22] that have no fixed-points. Another choice is to "dither" the hash function, i.e. adding a stage-dependent input to the compression function, cf. [2,26,6,12,1,4]). For example by adding a counter to the input of f, such that $H_i = f(H_{i-1}, M_i, i)$. Dithering however doesn't protect against Joux's method, since this computes a new collision for every dither value.

Joux's attack can be prevented by a technique like the "wide-pipe" and "double-pipe" of [15] or the similar chop-MD [6] construction, which enlarge the chain values compared to the hash value. This trick also makes our attack unapplicable, because it increases the cost of finding fixed-point collisions. Kelsey/Schneier attacks are applicable when fixed-points are easily found.

Another construction proposed in [16] prevents from all multicollision attacks presented here, including ours. Generally, our attack will work for some hash construction when both Joux's and Kelsey/Schneier do, hence won't work when at least one does not apply.

A construction published in Dean's thesis [8, §5.6.3, credited to Lipton] consists in hashing M as $\tilde{M}\|\tilde{M}$, with $\tilde{M}$ the padded message, to simulate a "variable IV". This prevents all nontrivial multicollision attacks, but is unreasonably inefficient.

5 Conclusions

We presented a multicollision attack applicable to iterated hashes when the IV can be chosen by the attacker, and when fixed-points for the compression function are easy to find. This can be seen as a variant of Joux's attack when some restrictions are put on the hash function (Joux's attack works for any IV and doesn't need fixed-points).

Our attack leaves open two related issues:

1. Can we find other generic attacks on iterated hashes that exploit easily-found fixed-points?
2. How to find fixed-point collisions for dedicated hash functions?

Current known generic attacks using fixed-points are those of Dean for second-preimages [8, 5.3.1], Kelsey/Schneier for multicollision [12], and ours in this paper. Fixed-point collisions are likely to be found using similar techniques as collisions, for blockcipher-based functions. Positive results to those two issues would lead to new generic attacks (finding collisions or preimages) and new dedicated attacks (finding fixed-points).

Acknowledgements

I wish to thank the referees of ICALP 2008 for many helpful comments, and John Kelsey for suggesting the attack of §4.3.

References

1. Andreeva, E., Bouillaguet, C., Fouque, P.-A., Hoch, J.J., Kelsey, J., Shamir, A., Zimmer, S.: Second preimage attacks on dithered hash functions. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 270–288. Springer, Heidelberg (2008)
2. Aumasson, J.-P., Phan, R.C.-W.: How (not) to efficiently dither blockcipher-based hash functions? In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 308–324. Springer, Heidelberg (2008)
3. Biham, E., Dunkelman, O.: A framework for iterative hash functions - HAIFA. In: Second NIST Cryptographic Hash Workshop (2006)
4. Biham, E., Dunkelman, O.: A framework for iterative hash functions - HAIFA. Cryptology ePrint Archive, Report 2007/278 (2007); Extended version of [3]
5. Brassard, G. (ed.): CRYPTO 1989. LNCS, vol. 435. Springer, Heidelberg (1990)
6. Coron, J.-S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård revisited: How to construct a hash function. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 430–448. Springer, Heidelberg (2005)

7. Damgård, I.: A design principle for hash functions. In: Brassard [5], pp. 416–427
8. Dean, R.D.: Formal Aspects of Mobile Code Security. PhD thesis, Princeton University (1999)
9. Hoch, J., Shamir, A.: Breaking the ICE - finding multicollisions in iterated concatenated and expanded (ICE) hash functions. In: Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 179–194. Springer, Heidelberg (2006)
10. Joux, A.: Multicollisions in iterated hash functions. application to cascaded constructions. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 306–316. Springer, Heidelberg (2004)
11. Kelsey, J., Kohno, T.: Herding hash functions and the Nostradamus attack. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 183–200. Springer, Heidelberg (2006)
12. Kelsey, J., Schneier, B.: Second preimages on n-bit hash functions for much less than 2^n work. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 474–490. Springer, Heidelberg (2005)
13. Knudsen, L.R., Mathiassen, J.E.: Preimage and collision attacks on MD2. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 255–267. Springer, Heidelberg (2005)
14. Lai, X., Massey, J.: Hash functions based on block ciphers. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 55–70. Springer, Heidelberg (1993)
15. Lucks, S.: Design principles for iterated hash functions. Cryptology ePrint Archive, Report 2004/253 (2004)
16. Maurer, U.M., Tessaro, S.: Domain extension of public random functions: Beyond the birthday barrier. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 187–204. Springer, Heidelberg (2007)
17. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
18. Merkle, R.: One way hash functions and DES. In: Brassard [5], pp. 428–446
19. Nandi, M., Stinson, D.: Multicollision attacks on generalized hash functions. Cryptology ePrint Archive, Report 2004/330 (2004); Later published in [20]
20. Nandi, M., Stinson, D.: Multicollision attacks on a class of hash functions. IEEE Transactions on Information Theory 53, 759–767 (2007)
21. Preneel, B.: Analysis and Design of Cryptographic Hash Functions. PhD thesis, Katholieke Universiteit Leuven (1993)
22. 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)
23. Quisquater, J.-J., Delescaille, J.-P.: How easy is collision search? Application to DES (extended summary). In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 429–434. Springer, Heidelberg (1990)
24. Rabin, M.: Digitalized signatures. In: Lipton, R., DeMillo, R. (eds.) Foundations of Secure Computation, pp. 155–166. Academic Press, London (1978)
25. Rabin, M.: Digitalized signatures and public-key functions as intractable as factorization. Technical Report MIT/LCS/TR-212, MIT (1979)
26. Rivest, R.: Abelian square-free dithering for iterated hash functions. In: ECRYPT Conference on Hash Functions (2005); Also presented in [27]
27. Rivest, R.: Abelian square-free dithering for iterated hash functions. In: First NIST Cryptographic Hash Workshop (2005)
28. Schneier, B.: Applied Cryptography, 2nd edn. John Wiley & Sons, Chichester (1996)

29. Sedgewick, R., Szymanski, T.G., Chi-Chih Yao, A.: The complexity of finding cycles in periodic functions. SIAM Journal of Computing 11(2), 376–390 (1982)
30. Suzuki, K., Tonien, D., Kurosawa, K., Toyota, K.: Birthday paradox for multi-collisions. In: Rhee, M.S., Lee, B. (eds.) ICISC 2006. LNCS, vol. 4296, pp. 29–40. Springer, Heidelberg (2006)
31. Yu, H., Wang, X.: Multi-collision attack on the compression functions of MD4 and 3-pass HAVAL. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 206–226. Springer, Heidelberg (2007)

A New Type of 2-Block Collisions in MD5*

Jiří Vábek[1], Daniel Joščák[1,2], Milan Boháček[1], and Jiří Tůma[1]

[1] Charles University in Prague,
Department of Algebra,
Sokolovská 83, 186 75 Prague 8, Czech Republic
[2] S.ICZ a.s., Hvězdova 1689/2a, Praha 4
jiri.vabek@centrum.cz, daniel.joscak@i.cz, milan.bohacek@gmail.com,
tuma@karlin.mff.cuni.cz

Abstract. We present a new type of 2-block collisions for MD5. The colliding messages differ in words m_2, m_9, m_{12} in both blocks. The differential paths for the collisions were generated by our implementation of Stevens algorithm [11]. The actual colliding messages were found by a version of Klima's algorithm involving tunnels [3].

Keywords: MD5, differential paths, collisions, Stevens algorithm.

1 Introduction

At rump session of Crypto 2004 X. Wang presented two pairs of colliding messages for MD5 [17]. A more detailed description of the method for constructing colliding pairs of messages was given in the paper [18] presented at Eurocrypt 2005. Each colliding pair consisted of messages of the same length 1024 bits, i.e. two blocks $(M_1||M_2)$ and $(M_1'||M_2')$. Their modular differences were:

$$\begin{aligned}\delta M_1 = M_1' - M_1 &= (0,0,0,0,2^{31},0,0,0,0,0,0,+2^{15},0,0,2^{31},0)\\ \delta M_2 = M_2' - M_2 &= -\delta M_1\\ &= (0,0,0,0,2^{31},0,0,0,0,0,0,-2^{15},0,0,2^{31},0)\end{aligned}$$

The most important part of [18] was the so called differential path for each block. The differential path says how modular differences and xor of registers Q_t, Q_t' evolve during the calculation of the MD5 compression function applied to M_1, M_1' and M_2, M_2' resp. However, the paper [18] gives no information about how the differential paths were found.

Since then, many improvements of the collision search algorithm based on the differential paths of Wang et al. have been published. Two most important developments were the multi-block message modification (already mentioned in [18]) and tunneling [3].These methods decreased the time required for finding a pair of colliding messages to less than one minute on a PC. The theoretical

* The first two authors were supported by GAUK number 301-10/257689 and the fourth author was supported by MSM0021620839.

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 78–90, 2008.

complexity was estimated [2] to 2^{27} calculations of the MD5 compression function for Klima's algorithm [3] and 2^{29} calculations for Stevens' algorithm [12].

A new type of collisions - the so called *chosen prefix collisions* were published in [11]. In case of chosen prefix collisions one starts with different initial vectors IVand IV$'$ with modular difference $\delta\mathrm{IV} = \mathrm{IV} - \mathrm{IV}' = (0, x, x, x)$ and constructs messages M, M' such that $\mathrm{MD5}(\mathrm{IV}, M) = \mathrm{MD5}(\mathrm{IV}', M')$. The number of blocks of M and M' equals the weight of x i.e. the minimal number of non-zero coefficients in any binary signed digit representation (BSDR) of x. The authors used chosen prefix collisions to construct colliding X.509 certificates. A major development of [11] is an algorithm for an automated construction of differential paths.

Another paper to mention in this context is the paper [16] by Yajima et al. The authors point out that there might be colliding pairs of messages with other differences than those of Wang et al. However their estimates of time required for finding colliding pairs with these differences were too high even if the corresponding differential paths were known.

Recently Sasaki et al. [10] made another progress in the study of MD5 collisions. They constructed a differential path that allowed them to find two message blocks M_1, M_1' with modular differences

$$\delta M_1 = M_1' - M_1 = (0,0,0,0,0,0,0,0,0,0,0,+2^{31},0,0,0,0)$$

such that $\mathrm{MD5}(\mathrm{IV}, M_1) - \mathrm{MD5}(\mathrm{IV}, M_1') = (2^{31}, 2^{31}, 2^{31}, 2^{31})$. Then they modify the algorithms of [1] to construct efficiently a message block M_2 such that $\mathrm{MD5}(\mathrm{IV}, M_1||M_2) = \mathrm{MD5}(\mathrm{IV}, M_1'||M_2)$. This is then applied to recover the first 31 letters of the client password used in the APOP. It was significant improvement of the result from [4] and [9], where Wang's et al. collisions enabled recovery of the first three characters of the password.

In this paper we present a new type of 2-block collisions for MD5. We choose one of the differences of messages suggested in [16] and construct the corresponding colliding message pair. In our case the colliding messages $(M_1||M_2)$ and $(M_1'||M_2')$ have differences

$$\begin{aligned}\delta M_1 = M_1' - M_1 &= (0,0,2^{31},0,0,0,0,0,0,+2^{27},0,0,2^{31},0,0,0)\\ \delta M_2 = M_2' - M_2 &= -\delta M_1\\ &= (0,0,2^{31},0,0,0,0,0,0,-2^{27},0,0,2^{31},0,0,0)\end{aligned}$$

We use our own implementation of Stevens differential path searching algorithm (the original implementation has not been published yet) to construct differential paths. We also give some details of our implementation in section 4. As for the algorithm finding colliding messages satisfying a given differential path we also use our own implementation of Klima's algorithm [3]. We do not provide any details since the algorithm based on tunnels is described well in e.g. [3], [13].

Recently Xie et al. announced in [19] a different type of two block colliding messages with differences

$$\delta M_1 = M_1' - M_1 = (0,0,0,0,0,0,+2^8,0,0,2^{31},0,0,0,0,0,2^{31})$$
$$\delta M_2 = M_2' - M_2 = -\delta M_1$$
$$= (0,0,0,0,0,0,-2^8,0,0,2^{31},0,0,0,0,0,2^{31})$$

Their collisions also belong among the collisions forecasted in [16], the case $t = 43$, see section 3.

2 Preliminaries

We follow description and notation from [13]. MD5 can be described as follows:

1. Pad the message with the 1-bit, then as many 0 bits until the resulting length equals 448 mod 512, and the bitlength of the original message expressed as a 64-bit integer. The total bitlength of the padded message is then multiple of 512.
2. Divide the padded message into N consecutive 512-bit blocks $M_1, M_2, \ldots, M_N$.
3. Go through $N + 1$ states IV_i, for $0 \leq i \leq N$, called the intermediate hash values. Each intermediate hash value IV_i consists of four 32-bit words a_i, b_i, c_i, d_i. For $i = 0$ these are initialized to fixed public values: (a_0, b_0, c_0, d_0) = (`0x67452301`, `0xEFCDAB89`, `0x98BADCFE`, `0x10325476`) and for $i = 1, 2, \ldots, N$ intermediate hash value IV_i is computed using the MD5 compression function described below: $\mathrm{IV}_i = H(\mathrm{IV}_{i-1}, M_i)$.
4. The resulting hash value is the last intermediate hash value IV_N.

2.1 MD5 Compression Function

The input for the compression function $H(\mathrm{IV}, M)$ is an intermediate hash value $\mathrm{IV} = (a, b, c, d)$ of length 128bits and a 512-bit message block M. There are 64 steps, each step uses a modular addition, a left rotation, and a non-linear function. Depending on the step t, addition constants C_t and rotation constants s_t (all defined in standard [6]) are used.

The non-linear function f_t is defined by

$$f_t = \begin{cases} F(x,y,z) = (x \wedge y) \vee (\neg x \wedge z), & \text{for } 0 \leq t \leq 15, \\ G(x,y,z) = (x \wedge z) \vee (y \wedge \neg z), & \text{for } 16 \leq t \leq 31, \\ H(x,y,z) = x \oplus y \oplus z, & \text{for } 32 \leq t \leq 47, \\ I(x,y,z) = y \oplus (x \wedge \neg z), & \text{for } 48 \leq t \leq 63, \end{cases}$$

The message block M is divided into 16 consecutive 32-bit words $m_0, m_1, \ldots, m_{15}$ and expanded to 64 words W_t, for $0 \leq t < 64$, of 32 bits each:

$$W_t = \begin{cases} m_t, & \text{for } 0 \leq t \leq 15, \\ m_{1+5t \bmod 16}, & \text{for } 16 \leq t \leq 31, \\ m_{5+3t \bmod 16}, & \text{for } 32 \leq t \leq 47, \\ m_{7t \bmod 16}, & \text{for } 48 \leq t \leq 63, \end{cases}$$

For $0 \leq t < 64$ the compression function algorithm maintains a working register with 4 state words $Q_t, Q_{t-1}, Q_{t-2}, Q_{t-3}$. These are initialized as $(Q_0, Q_{-1}, Q_{-2}, Q_{-3}) = (b, c, d, a)$ and, for $0 \leq t < 64$ in succession, updated as follows:

$$\begin{aligned} F_t &= f_t(Q_t, Q_{t-1}, Q_{t-2}), \\ T_t &= F_t + Q_{t-3} + C_t + W_t, \\ R_t &= RL(T_t, s_t), \\ Q_{t+1} &= Q_t + R_t. \end{aligned}$$

After all steps are computed, the resulting state words are added to the intermediate hash value and returned as output: $H(\mathrm{IV}, M) = (a + Q_{61}, b + Q_{64}, c + Q_{63}, d + Q_{62})$.

2.2 Differential Paths

A differential path for compression function H is a precise description of the propagation of differences through the 64 steps caused by $\delta\mathrm{IV}$ and δM

$$\begin{aligned} \delta F_t &= f_t(Q'_t, Q'_{t-1}, Q'_{t-2}) - f_t(Q_t, Q_{t-1}, Q_{t-2}); \\ \delta T_t &= \delta F_t + \delta Q_{t-3} + \delta W_t; \\ \delta R_t &= RL(T'_t, C_t) - RL(T_t, C_t); \\ \delta Q_{t+1} &= \delta Q_t + \delta R_t. \end{aligned}$$

We use notation of bitconditions (also taken from [13]) on (Q_t, Q'_t) to describe differential paths, where a single bitcondition specifies directly or indirectly the values of the bits $Q_t[i]$ and $Q'_t[i]$.

A *binary signed digit representation* (BSDR) of a word X is a sequence $Y = (k_i)_{i=0}^{31}$, often simply denoted as $Y = (k_i)$, of 32 digits $k_i \in \{-1, 0, +1\}$ for $0 \leq i \leq 31$, where

$$X \equiv \sum_{i=0}^{31} k_i 2^i \mod 2^{32}.$$

A particularly useful BSDR of a word X which always exists is the Non-Adjacent Form (NAF), where no two non-zero k_i's are adjacent. The NAF is not unique since we work modulo 2^{32} (making $k_{31} = -1$ equivalent to $k_{31} = +1$), however we will enforce uniqueness of the NAF by choosing $k_{31} \in \{0, +1\}$. Among the BSDRs of a word, the NAF has minimal weight (see e.g. [14]).

Table 1. Differential bitconditions

$\mathsf{q}_t[i]$	condition on $(Q_t[i], Q'_t[i])$	k_i
$\cdot$	$Q_t[i] = Q'_t[i]$	0
$+$	$Q_t[i] = 0,\ Q'_t[i] = 1$	$+1$
$-$	$Q_t[i] = 1,\ Q'_t[i] = 0$	-1

Table 2. Boolean function bitconditions

$\mathsf{q}_t[i]$	condition on $(Q_t[i], Q'_t[i])$	direct/indirect	direction
0	$Q_t[i] = Q'_t[i] = 0$	direct	
1	$Q_t[i] = Q'_t[i] = 1$	direct	
^	$Q_t[i] = Q'_t[i] = Q_{t-1}[i]$	indirect	backward
v	$Q_t[i] = Q'_t[i] = Q_{t+1}[i]$	indirect	forward
!	$Q_t[i] = Q'_t[i] = \neg Q_{t-1}[i]$	indirect	backward
y	$Q_t[i] = Q'_t[i] = \neg Q_{t+1}[i]$	indirect	forward
m	$Q_t[i] = Q'_t[i] = Q_{t-2}[i]$	indirect	backward
w	$Q_t[i] = Q'_t[i] = Q_{t+2}[i]$	indirect	forward
#	$Q_t[i] = Q'_t[i] = \neg Q_{t-2}[i]$	indirect	backward
h	$Q_t[i] = Q'_t[i] = \neg Q_{t+2}[i]$	indirect	forward
?	$Q_t[i] = Q'_t[i] \wedge (Q_t[i] = 1 \vee Q_{t-2}[i] = 0)$	indirect	backward
q	$Q_t[i] = Q'_t[i] \wedge (Q_{t+2}[i] = 1 \vee Q_t[i] = 0)$	indirect	forward

3 New 2-Block Collisions in MD5

The collisions of Wang et al. [18] make use of a weakness in the message expansion of MD5, in particular in its interplay with the non-linear function in the third round (steps $t = 32, \ldots, 47$). This weakness appears for all $t = 32, \ldots, 44$, not only fot $t = 34$ used in [18]. This had been observed already by Yajima et al. in [16]. They conjectured that any $t = 32, \ldots, 44$ might possible lead to 2-block collisions with similar characteristics.

More specifically there are 13 triplets of possible differences in messages

$$d_t = (\delta W_t, \delta W_{t+1}, \delta W_{t+3}) = (\delta m_{(5+3i) \bmod 16},\ \delta m_{(8+3i) \bmod 16},\ \delta m_{(14+3i) \bmod 16}),$$

where $i = t - 32$, for which one can hope to construct 2-block collision similar to those ones by Wang et al. They are summarized in Table 3.

We believe this pattern must had been already known to Wang et al, because they had chosen for their collisions the triplet defined by $t = 34$ which requires the smallest number of conditions in rounds 2–4 to have manageable computational complexity. An obvious strategy for choosing the t and the corresponding triplet of differences to have a differential path with manageable computational complexity is to obey the following conflicting requirements:

Table 3. Generalized Wang type differentials

step	δQ_t	δF_t	δW_t	δT_t	δR_t	s_t
t	0	0	$\pm 2^{31-s_t}$	$\pm 2^{31-s_t}$	$\cdot 2^{31}$	
$t+1$	$\cdot 2^{31}$	$\cdot 2^{31}$	$\cdot 2^{31}$	0	0	
$t+2$	$\cdot 2^{31}$	0	0	0	0	
$t+3$	$\cdot 2^{31}$	$\cdot 2^{31}$	$\cdot 2^{31}$	0	0	

1. Choose the triplet d_t such that the differences in messages appear in the second round as soon as possible, in particular the difference in $\delta m_{5+3i \mod 16}$.
2. Choose the triplet d_t such that the difference in $\delta m_{5+3i \mod 16}$ appears in the fourth round as late as possible.

However, to find colliding messages with the forecasted differences requires to find a differential path including a partial collision after two rounds, the prescribed differences in the third round and their consequences in the rest of round 3 and in round 4. This is what Wang et al. did in [18]. Yajima et al. in [16] constructed a partial differential path for steps $17, \ldots, 64$ in the case of differences d_{44}, and estimated the number of conditions in rounds 2 to 4 for their partial differential path. They presented a table comparing the number of conditions in rounds 2 to 4 for the differential path for the first block of Wang et al. for the differences d_{34}, and the partial differential path they proposed for the differences d_{44}.

Table 4. Number of conditions in rounds 2 to 4 for the first block

round	Wang: d_{34}	Yajima: d_{44}	ours: d_{44}
2	15	52	50
3	0	0	0
4	20	17	16

In this paper we present full differential paths for both blocks for the differences d_{44} and examples of the 2-block colliding messages with these differences. We constructed the full differential paths using our own implementation of Stevens algorithm described in [11] and [13]. The differential path for the first block we constructed differs in the second round from the partial path proposed by Yajima et al. In table 4 we also present the number of conditions on Q_t without the conditions on T_t for our differential path for the first block. The numbers taken from the paper by Yajima et al. are also the numbers of conditions on Q_t without the conditions on T_t that were completely missing in the paper by Wang et al. [18].

The differential paths we constructed and the corresponding colliding messages are presented in appendix A. The actual colliding messages were found by an algorithm involving Klima's tunnels which is similar to the near-collision block searching algorithm presented by Stevens in [13].

4 On Our Implementation of Stevens Algorithm

In this section, we discuss the details of our implementation of Stevens algorithm for generating differential paths. This algorithm can be divided into two main parts

- extending partial differential paths,
- connecting partial differential paths.

We use the following terminology for partial differential paths. An *upper path* is a partial differential path generated forward from an IV, a *lower path* is a partial differential path generated backward from the registers $64, \ldots, 61$. Note that our terminology differs from the one used by Stevens in [11].

4.1 Extending Partial Differential Paths

We provide more information on backward generation of lower paths. We start with a partial differential path for steps $63, \ldots, 31$. This path is kept fixed through out the whole run of the algorithm. We constructed it by hand in the simplest possible way. This lower path is presented in table 5.

Table 5. Partial lower differential path with δ

t	δQ_t	δF_t	δW_t	δT_t	δR_t	s_t
28	$\cdot 2^{31}$					
29	0					
30	0					
31	0	0	$\cdot 2^{31}$	0	0	20
32–43	0	0	0	0	0	
44	0	0	$\pm 2^{27}$	$\pm 2^{27}$	$\cdot 2^{31}$	4
45	$\cdot 2^{31}$	$\cdot 2^{31}$	$\cdot 2^{31}$	0	0	11
46	$\cdot 2^{31}$	0	0	0	0	16
47	$\cdot 2^{31}$	$\cdot 2^{31}$	$\cdot 2^{31}$	0	0	23
48-51	$\cdot 2^{31}$	$\cdot 2^{31}$	0	0	0	
52	$\cdot 2^{31}$	0	$\cdot 2^{31}$	0	0	6
53-61	$\cdot 2^{31}$	$\cdot 2^{31}$	0	0	0	
62	$\cdot 2^{31}$	0	$\cdot 2^{31}$	0	0	15
63	$\cdot 2^{31}$	$\cdot 2^{31}$	$\pm 2^{27}$	$\pm 2^{27}$	$\pm 2^{16}$	21
64	$\pm 2^{16}$	-	-	-	-	

The Stevens algorithm for extending differential paths uses 3 basic choices at each step t.

1. A choice of a BSDR of δQ_t.
2. A choice of $\delta F_t[i]$ for $i = 0, \ldots, 31$. This choice determines a BSDR of δF_t.
3. A choice of a BSDR of δT_t (in the case of generating upper paths forward) or δR_t (in the case of generating lower paths backward).

To limit the number of possible choices of BSDR's for δQ_t we use the following 4 basic parameters

(a) *max_nbr* is the maximal number of BSDR's of δQ_t,
(b) *max_dif* is the the maximal difference between the weight of a BSDR of δQ and the weight of its NAF,
(c) *max_len* is the maximal length of carry propagation,
(d) *max_prp* is the maximal number of carry propagations.

To choose a BSDR of δQ_t within the limits specified by the parameters one can use different approaches. One possibility is to generate randomly a BSDR satisfying the parameters and then to continue to the next choice. Another possibility is to generate all BSDR's satisfying all four parameters and then either to choose randomly from all generated possibilities or deterministically in some prescribed order.

Stevens mentions in his thesis that he sets up $max\_dif = 2$ and then he chooses δQ_t randomly among all BSDR's satisfying this condition. The advantage of this approach is speed.

In our implementation we have selected the other approach and generate all BSDR's satisfying all four parameters. Then we choose a particular BSDR deterministically. Our approach gives us information about the number of possible choices of BSDR's and therefore it provides us with some information about the tree of all possible extensions of partial differential paths.

To generate all BSDR's of δQ_t (within the limits set up by the four parameters) exactly once we developed our own algorithm. The details of the algorithm and a proof of its correctness will be presented in another paper.

The second choice is to pick $\delta F_t[i]$, for $i = 0, \ldots 31$, and therefore BSDR of δF_t. In what follows we use notations and definitions of sets U_{abc}, V_{abc} and $W_{abc,g}$ from subsection 5.5.2 of [11]. This choice depends on the precomputed values of the functions $FC(t, abc, g)$ and $BC(t, abc, g)$, where $a = \mathsf{q}_t[i]$, $b = \mathsf{q}_t[i-1]$, $c = \mathsf{q}_t[i-2]$ are bitconditions, and $g \in \{0, 1, -1\}$.

There are again two different approaches to choose $\delta F_t[i]$ (that is a BSDR of δF_t). One possibility is to choose $\delta F_t[i]$'s randomly from the set V_{abc} provided $|V_{abc}| > 1$. This is the approach used by Marc Stevens in his thesis [13]. This leads to random selection of BSDR's of δF_t. Our approach is to limit the number of possible choices for a BSDR of δF_t by a parameter max_dF and, if $|V_{abc}| > 1$, we choose $\delta F_t[i] \in V_{abc}$ in the prescribed order 0, 1, -1. We proceed from $i = 0$ to $i = 31$.

The third choice is to pick a BSDR of δR_t. Depending on the choice of BSDR of δR_t there are at most four possibilities for δT_t. Stevens describes in his thesis how he chooses the most probable one. In our implementation we choose either the NAF of δR_t or the BSDR that differs from the NAF of δR_t in the sign at the leading bit.

The algorithm for generating upper paths forward differs from the one for generating lower paths backward in inessential details.

4.2 Connecting Partial Differential Paths

We generate partial upper differential paths forward up to step 12 (the last computed value is δQ_{13}) and partial lower differential paths backward up to step 17 (the last computed value is δQ_{14}). The choice of the bounds is the same as in [13].

We have implemented the algorithm for connecting differential paths described in [11] without any modifications. It should be noted however that the output of the algorithm depends on the order of some steps in the algorithm and on the data structures used to keep the intermediate results.

Table 6. The parameters for partial lower paths

t	max_dif	max_len	max_prp	max_nbr	max_dF	max_con
15	-	-	-	-	-	67
16	-	-	-	-	-	60
17	2	2	2	10	1000	51
18	2	2	2	10	1000	41
19	2	2	2	10	1000	39
20	2	2	2	10	1000	35
21	2	2	2	10	1000	24
22	2	2	2	10	1000	20
23	2	2	2	10	1000	18
24	2	2	2	10	1000	11
30-25	2	1	2	10	1000	10

We supplement the connecting algorithm with the check if the rotation of δT_t, $t = 11, \ldots, 15$, leads to the correct δR_t selected in the extending parts of the algorithm. We try all possibilities for free bits in registers $Q_{11}, \ldots, Q_{16}$ and when there exists the possibility providing correct rotation of δT_t, we fix free bits and continue with the collision generating part of algorithm.

The connecting algorithm seems to have surprisingly high success rate. Stevens in a test run of his improved connecting algorithm successfully connected 52 pairs of upper and lower paths out of $2.5 \cdot 10^5 \times 5 \cdot 10^5$ attempted pairs.

In our implementation the ratio of successfully connected pairs appears to be very sensitive on the choice of parameters for generating partial differential paths, especially the parameter max_dF. The distribution of the number of successfully connected pairs in different runs of our implementation was rather irregular, but on average we constructed about 126 full differential paths out of $8 \cdot 10^4 \times 2 \cdot 10^5$ pairs of upper and lower paths for the first block and 4 full differential paths for the second block. However, without it no reasonable estimate of the success rate of the connecting partial differential path algorithm can be made and the number of test runs is not high enough to make any reasonable conclusions. In any case, this observed phenomenon calls for deeper theoretical investigation.

4.3 Choosing Parameters

The number of generated partial differential paths in a given time appears to be extremely sensitive on the choice of the parameters.

We present in table 6 the parameters we used for generating lower paths in each step. The parameter *max_con* for step t denotes the total number of conditions from the start of generation of partial paths to step t. We used the same parameters for both blocks.

The strategy for generating lower paths was to set the parameters in such a way that the following goals were achieved.

- The number of possible lower paths generated using chosen parameters is sufficient for the next (connecting) part of the algorithm (from about $5 \cdot 10^4$ to $2.5 \cdot 10^5$).
- The time needed to generate sufficient number of possible lower paths using chosen parameters is feasible (less than 1 day on single PC).
- The number of conditions in steps $17, \ldots, 30$ is as small as possible.
- In particular, the number of conditions in steps $2, \ldots, 30$ is as small as possible.

Setting the parameters is not straightforward and the values were obtained after some experimentation. There might be better choices and a theoretical understanding for an automated choice of the parameters is needed.

The strategy for the generating upper paths was not formulated in such detail. The goal was to limit the number of conditions in steps $0, \ldots, 12$ in such a way that the sufficient number of upper differential paths was generated in few hours. The total number of conditions is 80 for the first block and 180 for the second block.

5 Conclusion

We presented a new type of 2-block MD5 collisions. We found them using our implementation of Stevens algorithm. The implementation can be used to construct differential paths for other types of differences in messages stated in [16], i.e. to construct target or 2-block collisions in MD5.

Acknowledgment

The authors thank Marc Stevens for providing them the details and running times of some parts of his algorithm.

References

1. den Boer, B., Bosselaers, A.: Collisions for the Compression Function MD5. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 293–304. Springer, Heidelberg (1994)
2. Joščák, D.: Finding Collisions in Cryptographic Hash Functions Master's thesis, Charles University in Prague (2006), http://cryptography.hyperlink.cz/2006/diplomka.pdf
3. Klima, V.: Tunnels in Hash Functions: MD5 Collisions Within a Minute, Cryptology ePrint Archive: Report 105/2006, http://eprint.iacr.org/2006/105
4. Leurent, G.: Message Freedom in MD4 and MD5 Collisions: Application to APOP. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 320–339. Springer, Heidelberg (2007)
5. Liang, J., Lai, X.: Improved collision attack on hash function MD5, Cryptology ePrint Archive: Report 425/2005, http://eprint.iacr.org/2005/425
6. Rivest, R.: The MD5 Message-Digest Algorithm, Request for Comments: 1321 (April 1992), http://rfc.net/rfc1321.html

7. Sasaki, Y., Naito, Y., Kunihiro, N., Ohta, K.: Improved Collision Attack on MD5, Cryptology ePrint Archive: Report 400/2005, http://eprint.iacr.org/2005/400
8. Sasaki, Y., Naito, Y., Yajima, J., Shimoyama, T., Kunihiro, N., Ohta, K.: How to Construct Sufficient Condition in Searching Collisions of MD5, Cryptology ePrint Archive: Report 074/2006, http://eprint.iacr.org/2006/074
9. Sasaki, Y., Yamamoto, G., Aoki, K.: Practical Password Recovery on an MD5 Challenge and Response, Cryptology ePrint Archive: Report 101/2007, http://eprint.iacr.org/2007/101
10. Sasaki, Y., Wang, L., Ohta, K., Kunihiro, N.: Security of MD5 Challenge and Response: Extension of APOP Password Recovery Attack. In: Malkin, T.G. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 1–18. Springer, Heidelberg (2008)
11. Stevens, M., Lenstra, A., de Weger, B.: Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 1–22. Springer, Heidelberg (2007)
12. Stevens, M.: Fast Collision Attack on MD5, Cryptology ePrint Archive, Report 2006/104 (2006), http://eprint.iacr.org/
13. Stevens, M.: On Collisions for MD5, Master's thesis, Eidhoven University of Technology (2007)
14. Muir, J.A., Stinson, D.R.: Minimality and other properties of the width-w nonadjacent form. Mathematics of Computation 75, 369–384 (2006)
15. Yajima, J., Shimoyama, T.: Wangs sufficient conditions of MD5 are not sufficient, Cryptology ePrint Archive: Report 263/2005, http://eprint.iacr.org/2005/263
16. Yajima, J., Shimoyama, T., Sasaki, Y., Naito, Y., Kunihiro, N., Ohta, K.: How to construct a differential path of MD5 for collision search. In: SCIS 2006 (2006)
17. Wang, X., Feng, D., Lai, X., Yu, H.: Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199 (2004), http://eprint.iacr.org/2004/199
18. 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)
19. Xie, T., Feng, D., Liu, F.: A New Collision Differential For MD5 With Its Full Differential Path, Cryptology ePrint Archive, Report 2008/230 (2008), http://eprint.iacr.org/2008/230

A Differential Paths and Collision Example

Table 7. Differential path for the first and second blocks without tunnels, registers -3 to 30

	first block				second block			
-3	....	....	....	....	....	....	....	....
-2	....	....	....	....	+.......	1	....	....
-1	....	....	....	....	+.......	1	....	....
0	....	....	....	....	+.......	.......+	....	....
1	....	v	....	....	+.......	......v+	....	......v.
2	....	1	....	....	+.......	......1+	....	v.....1.
3	....	.......+	....	....	+.......	v.....+.	....	1.....+.
4	...v....	.......+	....	....	+v.v....	1.....+1	..v.....	+.....+.
5	...1....	.......+	....	.....v..	+1.1..v.	+...v.+.	..1.....	+...v.+.
6	...+....	.......+	....	v....1..	.-.-..1.	+.v.1.+.	.v+...v.	+...1.+.
7	...+...v	.......+	....	1..v.+..	1-.-.v+v	-v..+v+v	.0+..v..	+.v.+.+.
8	.0.+...0	.......+	.v...v.v	+..1.+..	.1.-v.+0	+1+.+0+.	v+.v.0+v	-v.v+v+v
9	.1.+..1+	....v..+	.....1.1	+..+.-..	.+.-1+.+	1-0..-.-	1-1..+0.	-1+11.+.
10	.+.-.v0+	....11.+	.+...+0-	.vv+.-..	.1..-10-	1-1.1+10	-00+.-1+	.+0+1+.-
11	v+0-0.+1	10v1+0v.	v1v0.+1+	1..+000.	y-.1-1.-	11-.1-11	-1-0.-00	0+++-010
12	.11+1++1	0100++11	0-110++0	0--0111.	00v.-1v+	0++0-101	010-v11-	11101000
13	+1+-+0--	-+--11-1	++--1+1-	+11+---.	++0..+.+	-00100+-	--++.++1	--0++-+.
14	1++0+-11	+--1+---	--++++--	0+01000.	+1+.1--+	+-0+++++	+10+-101	0+.+-01.
15	11+10+11	11+01010	01011v11	-1.0110.	y1-..-11	00-000+0	+000010+	0-.1+10.
16	.11.10..	-11.0100	-100+.10	01..0...	..1..110	+0111011	+..1-..-	.0.01...
17	..^..^..	..0...1.	....+..1	1+..-...	..^..^..	..0...1.	1...-..0	.-..+...
18	....	^.....+.	^...-..-	....0...	....	^.....-.	0...+..+	....0...
19	.....0..	..-...0.	....0...	.^..+...	.....0..	..+...0.	....0...	.^..-...
20	.....1..	..-...1.	....-..^	....	.....1..	..+...1.	....+..^	....
21	0....-..	..+...1.	....	....^...	0....+..	..-...1.	....	....^...
22	1....0..	..0...-.	....^...	....	1....0..	..0...+.	....^...	....
23	+....0..	..1.....	....	....	+....0..	..1.....	....	....
24	0....1..	......^.	....	....	1....1..	......^.	....	....
25	-....+..	....	....	....	+....-..	....	....	....
26	-.......	....	....	....	-.......	....	....	....
27	+....^..	....	....	....	+....^..	....	....	....
28	+.......	....	....	....	+.......	....	....	....
29	0.......	....	....	....	0.......	....	....	....
30	!.......	....	....	....	!.......	....	....	....

Table 8. Differential path for both blocks, registers 31 to 64. I, J, K $\in \{0, 1\}$, I $\neq$ K

31-45	.....	.....	.....	.....
46	I.....	.....	.....	.....
47	J.....	.....	.....	.....
48	I.....	.....	.....	.....
49	J.....	.....	.....	.....
50	I.....	.....	.....	.....
51	J.....	.....	.....	.....
52	K.....	.....	.....	.....
53	J.....	.....	.....	.....
54	K.....	.....	.....	.....
55	J.....	.....	.....	.....
56	K.....	.....	.....	.....
57	J.....	.....	.....	.....
58	K.....	.....	.....	.....
59	J.....	.....	.....	.....
60	K.....	.....	.....	.....
61	J.....	.....	.....	.....
62	I.....	.....	.....	.....
63	J.....	.....	.....	.....
64	.....	.....	.....	.....

Table 9. Collision

IV	0x67452301	0x10325476	0x98badcfe	0xefcdab89
M_1	0xCE7E83CA	0xCADE345E	0xB81D83A5	0x562EDF19
	0xB93C9D41	0xF9C4E244	0x5B9B832F	0xE16D2FE5
	0x4B286759	0xF9FE0301	0xA912EF12	0x95A85769
	0x18ADF66C	0x8B1AD802	0x291B44AB	0x732AF6A2
N_1	0xCE7E83CA	0xCADE345E	0x381D83A5	0x562EDF19
	0xB93C9D41	0xF9C4E244	0x5B9B832F	0xE16D2FE5
	0x4B286759	0x01FE0301	0xA912EF12	0x95A85769
	0x98ADF66C	0x8B1AD802	0x291B44AB	0x732AF6A2
IV_1	0xFADBF815	0x1B73566D	0x6BCF3C99	0x5D6E2DFF
IV'_1	0x7ADBF815	0x9B73566D	0xEBCF3C99	0xDD6F2DFF
$IV_1 \oplus IV'_1$	0x80000000	0x80000000	0x80000000	0x80010000
M_2	0x6A9B0D7D	0x9AAEEDA9	0x62255628	0xB6A85040
	0xC7E08FD1	0x077E530A	0xDEDD6809	0xD20A7D80
	0x55DFBE93	0x78571C29	0xC13D746C	0x062792C8
	0x45A152CE	0x69727500	0x351EC8F7	0xCFFFAF73
N_2	0x6A9B0D7D	0x9AAEEDA9	0xE2255628	0xB6A85040
	0xC7E08FD1	0x077E530A	0xDEDD6809	0xD20A7D80
	0x55DFBE93	0x70571C29	0xC13D746C	0x062792C8
	0xC5A152CE	0x69727500	0x351EC8F7	0xCFFFAF73
$IV_2 = IV'_2$	0xA5A29F9F	0xBC622670	0x54E1D520	0xE6FA818E

New Collision Attacks against Up to 24-Step SHA-2

(Extended Abstract)

Somitra Kumar Sanadhya* and Palash Sarkar

Applied Statistics Unit, Indian Statistical Institute,
203, B.T. Road, Kolkata 700108, India
somitra_r@isical.ac.in, palash@isical.ac.in

Abstract. In this work, we provide new and improved attacks against 22, 23 and 24-step SHA-2 family using a local collision given by Sanadhya and Sarkar (SS) at ACISP '08. The success probability of our 22-step attack is 1 for both SHA-256 and SHA-512. The computational efforts for the 23-step and 24-step SHA-256 attacks are respectively $2^{11.5}$ and $2^{28.5}$ calls to the corresponding step reduced SHA-256. The corresponding values for the 23 and 24-step SHA-512 attack are respectively $2^{16.5}$ and $2^{32.5}$ calls. Using a look-up table having 2^{32} (resp. 2^{64}) entries the computational effort for finding 24-step SHA-256 (resp. SHA-512) collisions can be reduced to $2^{15.5}$ (resp. $2^{22.5}$) calls. We exhibit colliding message pairs for 22, 23 and 24-step SHA-256 and SHA-512. This is the *first* time that a colliding message pair for 24-step SHA-512 is provided. The previous work on 23 and 24-step SHA-2 attacks is due to Indesteege et al. and utilizes the local collision presented by Nikolić and Biryukov (NB) at FSE '08. The reported computational efforts are 2^{18} and $2^{28.5}$ for 23 and 24-step SHA-256 respectively and $2^{43.9}$ and 2^{53} for 23 and 24-step SHA-512. The previous 23 and 24-step attacks first constructed a pseudo-collision and later converted it into a collision for the reduced round SHA-2 family. We show that this two step procedure is unnecessary. Although these attacks improve upon the existing reduced round SHA-2 attacks, they do not threaten the security of the full SHA-2 family.

Keywords: Cryptanalysis, SHA-2 hash family, reduced round attacks.

1 Introduction

Cryptanalysis of SHA-2 family has recently gained momentum due to the important work of Nikolić and Biryukov [6]. Prior work on finding collisions for step reduced SHA-256 was done in [4,5] and [8]. These earlier works used local collisions valid for the XOR linearized version of SHA-256 from [2] and [7]. On the other hand, the work [6] used a local collision which is valid for the actual SHA-256.

The authors in [6] developed techniques to handle nonlinear functions and the message expansion of SHA-2 to obtain collisions for up to 21-step SHA-256. The 21-step attack of [6] succeeded with probability 2^{-19}. Using similar

* This author is supported by the Ministry of Information Technology, Govt. of India.

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 91–103, 2008.

techniques, but utilizing a different local collision, [11] showed an attack against 20-step SHA-2 which succeeds with probability one and an attack against 21-step SHA-256 which succeeds with probability 2^{-15}. Further work [9] developed collision attacks against 21-step SHA-2 family which succeeds with probability one. Very recently, Indesteege et al. [3] have developed attacks against 23 and 24 step SHA-2 family. They utilize the local collision from [6] in these attacks.

Our Contributions. Our contributions in terms of the number of steps attacked and the success probability of these attacks are as follows.

- We describe the first *deterministic* attack against 22-step SHA-256 and SHA-512.
- We describe new attacks against 23 and 24-step SHA-256 and SHA-512.
  - The complexity of the 23-step attack for both SHA-256 and SHA-512 is improved in comparison to the existing 23-step attacks of [3].
  - The complexity of 24-step SHA-512 attack is improved in comparison to the existing attack of [3]. In fact, improving the complexity to $2^{32.5}$ from the earlier reported 2^{53} allows us to provide the *first* message pair which collides for 24-step SHA-512.

Table 1. Summary of results against reduced SHA-2 family. Effort is expressed as either the probability of success or as the number of calls to the respective reduced round hash function.

Work	Hash Function	Steps	Effort		Local Collision	Attack	Example
			Prob.	Calls	utilized	Type	provided
[4,5]	SHA-256	18		*	GH [2]	Linear	yes
[8]	SHA-256	18		**	SS_5 [7]	”	yes
[6]	SHA-256	20	$\frac{1}{3}$		NB [6]	Non-linear	yes
		21	2^{-19}		”	”	yes
[11]	SHA-256/512	18,20	1	1	SS [11]	”	yes
	SHA-256	21	2^{-15}		”	”	yes
[9]	SHA-256/512	21	1	1	”	”	yes
[3]	SHA-256	23		2^{18}	NB [6]	”	yes
		24		$2^{28.5}$	”	”	yes
	SHA-512	23		$2^{43.9}$	”	”	yes
		24		2^{53}	”	”	**no**
This work	SHA-256/SHA-512	22	1	1	SS [11]	”	yes
	SHA-256	23		$2^{11.5}$	”	”	yes
		24		$2^{28.5}$	”	”	yes
		24		$2^{15.5}$ †	”	”	no
	SHA-512	23		$2^{16.5}$	”	”	yes
		24		$2^{32.5}$	”	”	**yes**
		24		$2^{22.5}$ ‡	”	”	no

* It is mentioned in [4,5] that the effort is 2^0 but no details are provided.
** Effort is given as running a C-program for about 30–40 minutes on a standard PC.
† A table containing 2^{32} entries, each entry of size 8 bytes, is required.
‡ A table containing 2^{64} entries, each entry of size 16 bytes, is required.

- Using a table lookup, the complexity of the 24-step SHA-256 attack is improved in comparison to the existing 24-step attack of [3]. The table contains 2^{32} entries with each entry of size 8 bytes. Similary, the complexity of the 24-step SHA-512 attack is also improved using a table lookup. For this case, the table lookup has 2^{64} entries each entry of 16 bytes.
- Examples of Colliding message pairs are provided for 22, 23 and 24-step SHA-256 and SHA-512.

Our contributions to the methodology of the attacks are as follows.

– We use a different local collision for our 22, 23 and 24-step attacks. The earlier work [3] uses the local collision from [6] while we use a local collision from [11].
– The work in [3] describes 23 and 24-step collisions as a two-part procedure– first obtain a pseudo-collision and then convert it into a collision. In contrast, our analysis is direct and shows that such a two-part description is unnecessary.
– Details of a required "guess-then-determine algorithm" to solve a non-linear equation arising in the 24-step attack are provided in this work. A suggestion for a similar algorithm is given in [3] but no details are provided. There are two algorithms– one for SHA-256 and the other for SHA-512.

A summary of results on collision attacks against reduced SHA-2 family is given in Table 1.

2 Preliminaries

In this paper we use the following notation:

– Message words: $W_i \in \{0,1\}^n$, $W'_i \in \{0,1\}^n$; n is 32 for SHA-256 and 64 for SHA-512.
– Colliding message pair:$\{W_0, W_1, W_2, \ldots W_{15}\}$ & $\{W'_0, W'_1, W'_2, \ldots W'_{15}\}$.
– Expanded message pair:$\{W_0, W_1, W_2, \ldots W_{r-1}\}$ & $\{W'_0, W'_1, W'_2, \ldots W'_{r-1}\}$. The number of steps r is 64 for SHA-256 and 80 for SHA-512.
– The internal registers for the two messages at step i: $\text{REG}_i = \{a_i, \ldots, h_i\}$ and $\text{REG}'_i = \{a'_i, \ldots, h'_i\}$.
– $\text{ROTR}^k(x)$: Right rotation of an n-bit string x by k bits.
– $\text{SHR}^k(x)$: Right shift of an n-bit string x by k bits.
– $\oplus$: bitwise XOR; $+, -$: addition and subtraction modulo 2^n.
– $\delta X = X' - X$ where X is an n-bit quantity.
– $\delta\Sigma_1(x) = \Sigma_1(e'_i) - \Sigma_1(e_i) = \Sigma_1(e_i + x) - \Sigma_1(e_i)$.
– $\delta\Sigma_0(x) = \Sigma_0(a'_i) - \Sigma_0(a_i) = \Sigma_0(a_i + x) - \Sigma_0(a_i)$.
– $\delta f^i_{MAJ}(x, y, z) = f_{MAJ}(a_i + x, b_i + y, c_i + z) - f_{MAJ}(a_i, b_i, c_i)$.
– $\delta f^i_{IF}(x, y, z) = f_{IF}(e_i + x, f_i + y, g_i + z) - f_{IF}(e_i, f_i, g_i)$.

2.1 SHA-2 Hash Family

Eight registers are used in the evaluation of SHA-2. In Step i, the 8 registers are updated from (a_{i-1}, b_{i-1}, c_{i-1}, d_{i-1}, e_{i-1}, f_{i-1}, g_{i-1}, h_{i-1}) to (a_i, b_i, c_i, d_i, e_i, f_i, g_i, h_i). For more details, see [1].

By the form of the step update function, we have the following relation.

Cross Dependence Equation (CDE)

$$e_i = a_i + a_{i-4} - \Sigma_0(a_{i-1}) - f_{MAJ}(a_{i-1}, a_{i-2}, a_{i-3}). \tag{1}$$

Later, we make extensive use of this relation. Note that a special case of this equation was first utilized in §6.1 of [11]. The equation in the form above was used in [9]. This equation can be used to show that the SHA-2 state update can be rewritten in terms of only one state variable. This fact was later observed in [3] independently.

Table 2. The 9-step Sanadhya-Sarkar local collision [11] used in the present work. Our deterministic 22-step attack and the probabilistic 23 and 24-step attacks use unequal message word differences to achieve the same differential path.

Step	δW_i		Register differences							
	I	II	δa_i	δb_i	δc_i	δd_i	δe_i	δf_i	δg_i	δh_i
$i-1$	0	0	0	0	0	0	0	0	0	0
i	1	1	1	0	0	0	1	0	0	0
$i+1$	-1	δW_{i+1}	0	1	0	0	-1	1	0	0
$i+2$	δW_{i+2}	0	0	0	1	0	-1	-1	1	0
$i+3$	δW_{i+3}	δW_{i+3}	0	0	0	1	0	-1	-1	1
$i+4$	0	0	0	0	0	0	1	0	-1	-1
$i+5$	0	0	0	0	0	0	0	1	0	-1
$i+6$	0	0	0	0	0	0	0	0	1	0
$i+7$	δW_{i+7}	0	0	0	0	0	0	0	0	1
$i+8$	-1	-1	0	0	0	0	0	0	0	0

3 Nonlinear Local Collision for SHA-2

We use two variations of a 9-step non-linear local collision for our attacks. This local collision was given recently by Sanadhya and Sarkar [11]. This local collision starts by introducing a perturbation message difference of 1 in the first message word. Next eight message words are chosen suitably to obtain the desired differential path. Table 2 shows the local collision used. The message word differences are different for the two variations of the local collision. Columns headed I and II under δW_i in Table 2 show the message word differences for the first and the second variations of the local collision respectively.

In the local collision, the registers (a_{i-1}, ..., h_{i-1}) and W_i are inputs to Step i of the hash evaluation and this step outputs the registers (a_i, ..., h_i).

4 The Deterministic 22-Step SHA-2 Attack

In [6], a single local collision spanning from Step 6 to Step 14 is used and a 21-step collision for SHA-256 is obtained probabilistically. We use a similar method for our attack but this time we use the local collision of Table 2 spanning from Step 7 to Step 15. Message words are given by Column (II). The SHA-2 design has freedom of message words W_0 to W_{15}. Since the local collision spans this range only, we can deterministically satisfy all the required conditions. The message words after Step 16 are generated by message expansion. The local collision is chosen in such a way that the message expansion produces no difference in words W_i and W'_i for $i \in \{16, 17, \ldots 21\}$. This results in a deterministic 22-step attack. We explain this fact below.

First of all, note that the local collision starts from Step 7. It can be seen from the structure of the local collision that $\delta W_7 = 1$ and $\delta W_9 = \delta W_{11} = \delta W_{12} = \delta W_{13} = \delta W_{14} = 0$. In addition, δW_{15} is -1. Messages outside the span of the local collision are taken to have zero differentials. Therefore $\delta W_i = 0$ for $i \in \{0, 1, 2, 3, 4, 5, 6\}$. Consider the first 6 steps of message expansion for SHA-2 next.

$$\left.\begin{array}{l} W_{16} = \sigma_1(W_{14}) + \ W_9 + \sigma_0(W_1) + W_0, \\ W_{17} = \underline{\sigma_1(W_{15}) + W_{10}} + \sigma_0(W_2) + W_1, \\ W_{18} = \sigma_1(W_{16}) + W_{11} + \sigma_0(W_3) + W_2, \\ W_{19} = \underline{\sigma_1(W_{17})} + W_{12} + \sigma_0(W_4) + W_3, \\ W_{20} = \sigma_1(W_{18}) + W_{13} + \sigma_0(W_5) + W_4, \\ W_{21} = \underline{\sigma_1(W_{19})} + W_{14} + \sigma_0(W_6) + W_5. \end{array}\right\} \quad (2)$$

Terms which *may have* non-zero differentials in the above equations are underlined. To obtain 22-step collisions in SHA-2, it is sufficient to ensure that $\delta\{\sigma_1(W_{15}) + W_{10}\} = 0$ so that $\delta W_{17} = 0$. This also ensures that next 4 steps of the message expansion do not produce any difference, and we have a 22-step collision. By using the local collision described earlier, it is possible to deterministically satisfy the condition $\delta\{\sigma_1(W_{15}) + W_{10}\} = 0$. Further details are available in [10].

5 A General Idea for Obtaining 23 and 24-Step SHA-2 Collisions

Obtaining deterministic collisions up to 22 steps did not require the (single) local collision to extend beyond step 15. For obtaining collisions for more number of steps, we will need to start the local collision at Step 8 (or farther) and hence the local collision will end at Step 16 (or farther). This will require us to analyze the message expansion more carefully.

For obtaining collisions up to 22 steps, we also needed to consider message expansion. But, following Nikolić and Biryukov, we ensured that there were

no differences in message words from Step 16 onwards. However, now that we consider the local collision to end at Step 16 (or farther), this will necessarily mean that one or more δW_i (for $i \geq 16$) will be non-zero. This will require a modification of the Nikolić-Biryukov strategy. Instead of requiring $\delta W_i = 0$ for $i \geq 16$, we will require $\delta W_i = 0$ for a few i's after the local collision ends. So, supposing that the local collision ends at Step 16 and we want a 23-step collision, then δW_{16} is necessarily -1 and we will require $\delta W_{17} = \cdots = \delta W_{22} = 0$.

5.1 Satisfying Conditions on the Differential Path

Conditions on δW_{i+2}, δW_{i+3} and δW_{i+4} shown in Table 2 give rise to the following conditions on the values of λ, γ and μ.

$$\left.\begin{aligned} \delta W_{i+2} = \delta_1 &= -1 - \Sigma_1(\mu - 1) + \Sigma_1(\mu) - f_{IF}(\mu - 1, 0, \gamma + 1) \\ &\quad + f_{IF}(\mu, -1, \gamma + 1) \\ \delta W_{i+3} = \delta_2 &= -\Sigma_1(\lambda - 1) + \Sigma_1(\lambda) - f_{IF}(\lambda - 1, \mu - 1, 0) \\ &\quad + f_{IF}(\lambda, \mu, -1) \\ 1 &= -f_{IF}(\lambda - 1, \lambda - 1, \mu - 1) + f_{IF}(\lambda - 1, \lambda, \mu). \end{aligned}\right\} \quad (3)$$

Similar equations for the Nikolić-Biryukov differential path have been reported in [3] and a method for solving them has been discussed. The method to solve these equation is different for SHA-256 and for SHA-512. We discuss the exact details about solving them later. In describing our attacks on the SHA-2 family, we assume that some solutions to these equations have been obtained. These solutions are required to obtain colliding message pairs for the hash functions.

6 23-Step SHA-2 Collisions

We show that by suitably placing a local collision of the type described in Column (I) of Table 2 and using proper values for α, γ and μ, it is possible to obtain 23-step collisions for SHA-2.

6.1 Case $i = 8$

The local collision is started at $i = 8$ and ends at $i = 16$. Setting $\beta = \overline{\alpha}$, $u = 0$ and $\delta_1 = 0$, we need to choose a suitable value for δ_2 which is the value of $\delta W_{i+3} = \delta W_{11}$. For this case, we let $\delta = \delta_2$.

Since the local collision ends at Step 16, it necessarily follows that $\delta W_{16} = -1$. Consequently, we need to consider δW_{18} to ensure that it is zero. Since the collision starts at $i = 8$, all δW_j for $0 \leq j \leq 7$ are zero. Consequently, we can write $\delta W_{18} = \delta\sigma_1(W_{16}) + \delta W_{11}$, where $\delta\sigma_1(W_{16}) = \sigma_1(W_{16} - 1) - \sigma_1(W_{16})$. So, for δW_{18} to be zero, we need $\delta W_{11} = -\delta\sigma_1(W_{16})$, so that δW_{11} should be one of the values which occur in the distribution of $\sigma_1(W) - \sigma_1(W - 1)$ for some W.

Obtaining proper values for the constants only ensures that the local collision holds from Steps i to $i + 8$ as expected. It does not, however, guarantee that the reduced round collision holds. In the present case, we need to have δW_{18} to be

Table 3. Values of a and e register for the δWs given by Column (I) of Table 2 to hold. We have $\beta = \overline{\alpha}$ and using CDE, $\lambda = \beta + \alpha - \Sigma_0(\beta) - f_{MAJ}(\beta, -1, \alpha) = -\Sigma_0(\overline{\alpha})$. The value of u is either 0 or 1. Thus, the independent quantities are α, γ and μ.

index	$i-2$	$i-1$	i	$i+1$	$i+2$	$i+3$	$i+4$	$i+5$	$i+6$
a	α	α	-1	β	β				
e	γ	$\gamma+1$	-1	μ	λ	$\lambda-1$	-1	-1	$-1-u$

zero. This will happen only if W_{16} takes a value such that $\sigma_1(W_{16}-1)-\sigma_1(W_{16})$ is equal to $-\delta$. This can be ensured probabilistically in the following manner. Let the frequency of δ used in the attack be freq_δ. This means that trying approximately freq_δ possible random choices of W_0 and W_1, we expect a proper value of W_{16} and hence, a 23-step collision for SHA-2. We discuss the cases of SHA-256 and SHA-512 separately later.

Since $i = 8$, from Table 3, we see that a_6 to a_{10} get defined and e_6 to e_{14} get defined. Using CDE, the values of e_9 down to e_6 is set by fixing values of a_5 down to a_2. In other words, the values of a_2 to a_{10} are fixed. Now, consider

$$e_{14} = \Sigma_1(e_{13}) + f_{IF}(e_{13}, e_{12}, e_{11}) + a_{10} + e_{10} + K_{14} + W_{14}.$$

Note that in this equation all values other than W_{14} have already been fixed. So, W_{14} and hence $\sigma_1(W_{14})$ is also fixed. Now, from the update function of the a register, we can write

$$W_9 = a_9 - \Sigma_0(a_8) - f_{MAJ}(a_8, a_7, a_6) - \Sigma_1(e_8) - f_{IF}(e_8, e_7, e_6) - e_5 - K_9.$$

On the right hand side, all quantities other than e_5 have fixed values. Using CDE,

$$e_5 = a_5 + a_1 - \Sigma_0(a_4) - f_{MAJ}(a_4, a_3, a_2).$$

Again in the right hand side, all quantities other than a_1 have fixed values. So, we can write $W_9 = C - a_1$, where C is a fixed value. (This relation has already been observed in [3].)

Now,

$$a_1 = \Sigma_0(a_0) + f_{MAJ}(a_0, b_0, c_0) + \Sigma_1(e_0) + f_{IF}(e_0, f_0, g_0) + h_0 + K_1 + W_1$$

where a_0 and e_0 depend on W_0 whereas b_0, c_0, f_0, g_0 and h_0 depend only on IV and hence are constants. Thus, we can write $a_1 = \Phi(W_0) + W_1$, where

$$\Phi(W_0) = \Sigma_0(a_0) + f_{MAJ}(a_0, b_0, c_0) + \Sigma_1(e_0) + f_{IF}(e_0, f_0, g_0) + h_0 + K_1.$$

We write $\Phi(W_0)$ to emphasize that this depends only on W_0. At this point, we can write

$$\begin{aligned} W_{16} &= \sigma_1(W_{14}) + W_9 + \sigma_0(W_1) + W_0 \\ &= \sigma_1(W_{14}) + C - \Phi(W_0) - W_1 + \sigma_0(W_1) + W_0 \\ &= D - \Phi(W_0) - W_1 + \sigma_0(W_1) + W_0. \end{aligned}$$

Estimate of Computation Effort. Let there be freq_δ values of W_{16} for which $\sigma(W_{16}-1)-\sigma(W_{16})$ equals δ. So, we have to solve this equation for W_0 and W_1 such that W_{16} is one of these freq_δ possible values. The simplest way to do this is to try out random choices of W_0 and W_1 until W_{16} takes one of the desired values. On an average, success is obtained after freq_δ trials. Each trial corresponds to about a single step of SHA-2 computation. So, the total cost of finding suitable W_0 and W_1 is about $\frac{\mathsf{freq}_\delta}{2^{4.5}}$ tries of 23-step SHA-2 computations.

For each such solution (W_0, W_1) and an arbitrary choice of W_{15} we obtain a 23-step collision for SHA-2. Note that after W_0 and W_1 has been obtained everything else is deterministic, i.e., no random tries are required. The task of obtaining a suitable W_0 and W_1 can be viewed as a pre-computation of the type required to find the values of α, γ and μ. Then, the actual task of finding collisions becomes deterministic.

6.2 Relation to the 23-Step Collision from [3]

The NB local collision has been used in [3]. The local collision was placed from Step 9 to Step 17. In comparison, we have shown that the SS local collision gives rise to two kinds of 23-step collision. The first one is obtained by placing the local collision from Steps 8 to 16, and the second one is obtained by placing the local collision from Steps 9 to 17.

The description of the attack in [3] is quite complicated. First they consider a 23-step pseudo-collision which is next converted into 23-step collision. This two-step procedure is unnecessary. Our analysis allows us to directly describe the attacks.

7 24-Step Collisions

The local collision described in Column (I) of Table 2 is placed from Step $i = 10$ to Step $i + 8 = 18$ with $u = 1$. The values of δ_1, δ_2 as well as suitable values of α, γ and μ need to be chosen.

Since, the collision ends at Step 18 and $u = 1$, we will have $\delta W_{17} = 1$ and $\delta W_{18} = -1$. As a result, to ensure $\delta W_{19} = \delta W_{20} = 0$, we need to have $\delta_1 = \delta W_{12} = -(\sigma_1(W_{17}+1) - \sigma_1(W_{17}))$ and $\delta_2 = \delta W_{13} = -(\sigma_1(W_{18}-1) - \sigma_1(W_{18}))$. Based on the differential behaviour of σ_1 described in [10], we should try to choose δ_1 and δ_2 such that $\mathsf{freq}_{-\delta_1}$ and freq_{δ_2} are as high as possible. (Here $-\delta_1$ denotes $-\delta_1 \bmod 2^n$, where n is the word size 32 or 64.) But, at the same time, the chosen δ_1 and δ_2 must be such that (3) are satisfied.

Now we consider Table 3. This table tells us what the values of the different a and e-registers need to be. Since messages up to W_{15} are free, we can set values for a and e registers up to Step 15. But, we see that $e_{16} = -1 - u = -2$. This can be achieved by setting W_{16} to

$$W_{16} = e_{16} - \Sigma_1(e_{15}) - f_{IF}(e_{15}, e_{14}, e_{13}) - a_{12} - e_{12} - K_{16}. \tag{4}$$

Since we want $e_{16} = -2$ and all other values on the right hand side are constants, we have that W_{16} is a constant value. On the other hand, W_{16} is defined by

message recursion. So, we have to ensure that W_{16} takes the correct value. In addition, we need to ensure that W_{17} and W_{18} take values such that $\sigma_1(W_{17}+1)-\sigma_1(W_{17})=-\delta_1$ and $\sigma_1(W_{18}-1)-\sigma_1(W_{18})=-\delta_2$.

Since $i=10$, from Table 3, we see that a_8 to a_{12} have to be set to fixed values and e_8 to e_{16} have to be set to fixed values. Using CDE, the values of e_{11} down to e_8 are determined by a_7 to a_4. So, the values of a_0 to a_3 are free and correspondingly the choices of words W_0 to W_3 are free.

We have already seen that W_{16} is a fixed value. Note that

$$\left.\begin{aligned} W_{14} &= e_{14}-\Sigma_1(e_{13})-f_{IF}(e_{13},e_{12},e_{11})-a_{10}-e_{10}-K_{14}\\ W_{15} &= e_{15}-\Sigma_1(e_{14})-f_{IF}(e_{14},e_{13},e_{12})-a_{11}-e_{11}-K_{15}. \end{aligned}\right\} \tag{5}$$

Since for both equations, all the quantities on the right hand side are fixed values, so are W_{14} and W_{15}.

Using CDE twice, we can write

$$\left.\begin{aligned} W_9 &= -W_1+C_4+f_{MAJ}(a_4,a_3,a_2)-\Phi_0\\ W_{10} &= -W_2+C_5+f_{MAJ}(a_5,a_4,a_3)-\Phi_1\\ W_{11} &= -W_3+C_6+f_{MAJ}(a_6,a_5,a_4)-\Phi_2 \end{aligned}\right\} \tag{6}$$

where

$$\left.\begin{aligned} C_i &= e_{i+5}-\Sigma_1(e_{i+4})-f_{IF}(e_{i+4},e_{i+3},e_{i+2})-2a_{i+1}-K_{i+5}\\ &\quad +\Sigma_0(a_i),\\ \Phi_i &= \Sigma_0(a_i)+f_{MAJ}(a_i,b_i,c_i)+\Sigma_1(e_i)+f_{IF}(e_i,f_i,g_i)+h_i+\\ &\quad K_{i+1}. \end{aligned}\right\} \tag{7}$$

Using the expressions for W_9, W_{10} and W_{11} we obtain the following expressions for W_{16}, W_{17} and W_{18}.

$$\left.\begin{aligned} W_{16} &= \sigma_1(W_{14})+C_4-W_1+f_{MAJ}(a_4,a_3,a_2)-\Phi_0+\sigma_0(W_1)\\ &\quad +W_0\\ W_{17} &= \sigma_1(W_{15})+C_5-W_2+f_{MAJ}(a_5,a_4,a_3)-\Phi_1+\sigma_0(W_2)\\ &\quad +W_1\\ W_{18} &= \sigma_1(W_{16})+C_6-W_3+f_{MAJ}(a_6,a_5,a_4)-\Phi_2+\sigma_0(W_3)\\ &\quad +W_2. \end{aligned}\right\} \tag{8}$$

We need to ensure that W_{16} has the desired value given by (4) and that W_{17} and W_{18} take values which lead to desired values for $\delta\sigma_1(W_{17})$ and $\delta\sigma_1(W_{18})$ as explained above.

The only free quantities are W_0 to W_3 which determine a_0 to a_3. The value of C_4 depends on e_8, e_7 and e_6, where e_8 has a fixed value and e_7 and e_6 are in turn determined using CDE by a_3 and a_2. Similarly, C_5 is determined by e_9, e_8 and e_7; where e_9, e_8 have fixed values and e_7 is determined using a_3. The value of C_6 on the other hand is fixed. Coming to the Φ values, Φ_0 is determined only by W_0; Φ_1 determined by W_0 and W_1; and Φ_2 determined by W_0, W_1 and W_2. Let

$$D = W_{16}-(\sigma_1(W_{14})+C_4+f_{MAJ}(a_4,a_3,a_2)-\Phi_0+W_0). \tag{9}$$

If we fix W_0 and a_3, a_2, then the value of D gets fixed and we need to find W_1 such that the following equation holds.

$$D = -W_1 + \sigma_0(W_1). \tag{10}$$

A guess-then-determine algorithm can be used to solve this equation. This algorithm will be different for SHA-256 and for SHA-512 since the σ_0 function is different for the two. The guess-then-determine algorithms for both SHA-256 and SHA-512 are described in [10].

Solving (10) Using Table Look-Up. An alternative approach would be to use a pre-computed table. For each of the 2^n possible W_1s (n is the word size 32 or 64), prepare a table of entries $(W_1, -W_1 + \sigma_0(W_1))$ sorted on the second column. Then all solutions (if there are any) for (10) can be found by a simple look-up into the table using D. The table would have 2^n entries and if a proper index structure is used, then the look-up can be done very fast. We have not implemented this method.

Given $a_1, b_1, \ldots, h_1$ and a_2 the value of W_2 gets uniquely defined; similarly, given $a_2, b_2, \ldots, h_2$ and a_3, the value of W_3 gets uniquely defined. The equations are the following.

$$\left.\begin{aligned} W_2 &= a_2 - (\Sigma_0(a_1) + f_{MAJ}(a_1, b_1, c_1) + h_1 + \Sigma_1(e_1) \\ &\quad + f_{IF}(e_1, f_1, g_1) + K_2) \\ W_3 &= a_3 - (\Sigma_0(a_2) + f_{MAJ}(a_2, b_2, c_2) + h_2 + \Sigma_1(e_2) \\ &\quad + f_{IF}(e_2, f_2, g_2) + K_3) \end{aligned}\right\} \tag{11}$$

The strategy for determining suitable $W_0, \ldots, W_3$ is the following.

1. Make random choices for W_0 and a_2, a_3.
2. Run SHA-2 with W_0 and determine Φ_0.
3. From a_3 and a_2 determine e_7 and e_6 using CDE.
4. Determine C_4 using (7) and then D using (9).
5. Solve (10) for W_1 using the guess-then-determine algorithm.
6. Run SHA-2 with W_1 to define $a_1, \ldots, h_1$.
7. Determine Φ_1 using (7) and then W_2 using (11).
8. Run SHA-2 with W_2 to define $a_2, \ldots, h_2$.
9. Determine Φ_2 using (7) and then W_3 using (11).
10. Compute W_{17} and W_{18} using (8).
11. If $\sigma_1(W_{17} + 1) - \sigma_1(W_{17}) = -\delta_1$ and $\sigma_1(W_{18} - 1) - \sigma_1(W_{18}) = \delta_2$, then return W_0, W_1, W_2 and W_3.

The values of W_0, W_1, W_2 and W_3 returned by this procedure ensure that the local collision ends properly at Step 18 and that $\delta W_j = 0$ for $j = 19, \ldots, 23$. This provides a 24-step collision.

Estimate of Computation Effort. Let Step 5 involve a computation of g operations, where each operation is much faster than a single step of SHA-2; by our assessment the time for each operation is around 2^{-4} times the cost of

a single step of SHA-2. Thus, the time for Step 5 is about $\frac{g}{2^4}$ single SHA-2 steps. Further, let the success probability of the guess-then-determine attack be p. Then Step 5 needs to be repeated roughly $\frac{1}{p}$ times to obtain a solution.

By the choice of δ_1, the equality $\sigma_1(W_{17}+1)-\sigma_1(W_{17}) = -\delta_1$ holds roughly with probability $\frac{\mathsf{freq}_{\delta_1}}{2^n}$ while by the choice of δ_2 the equality $\sigma_1(W_{18}-1)-\sigma_1(W_{18}) = \delta_2$ holds roughly with probability $\frac{\mathsf{freq}_{\delta_2}}{2^n}$ and we obtain success in Step 11 with roughly $\frac{\mathsf{freq}_{\delta_1}\times\mathsf{freq}_{\delta_2}}{2^{2n}}$ probability. So, the entire procedure needs to be carried out around $\frac{2^{2n}}{\mathsf{freq}_{\delta_1}\times\mathsf{freq}_{\delta_2}}$ times to obtain a collision.

The guess-then-determine step takes about $g/2^4$ single SHA-2 steps. The time for executing the entire procedure once is about $(\frac{g}{2^4}+3)$ single SHA-2 steps which is about $2^{-4.5}\times(\frac{g}{2^4}+3)$ 24-step SHA-2 computations. Since the entire process needs to be repeated many times for obtaining success, the number of 24-step SHA-2 computations till success is obtained is about $(\frac{2^{2n}}{\mathsf{freq}_{\delta_1}\times\mathsf{freq}_{\delta_2}})\times(2^{-4.5}\times(\frac{g}{2^4}+3)\times\frac{1}{p})$.

If (10) is solved using a table look-up, then the cost estimate changes quite a lot. The cost of Step 5 reduces to about a single SHA-2 step so that the overall cost reduces to about $(\frac{2^{2n}}{\mathsf{freq}_{\delta_1}\times\mathsf{freq}_{\delta_2}})\times(2^{-4.5}\times 3\times\frac{1}{p})$ 24-step SHA-2 computations. The trade-off is that we need to use a look-up table having 2^n entries.

8 Exhibiting Colliding Message Pairs

The description in the previous sections provide an outline of how to obtain colliding message pairs. To actually find collisions, a lot more details are required. Due to lack of space, we are unable to provide these details here. (The reader may refer to [10] for further details.) Here we simply provide examples of actual collisions that we have found. These are given in Tables 4 to 10.

Table 4. Colliding message pair for 22-step SHA-512 with standard IV

W_1	0-3	0000000000000000	0000000000000000	c2bc8e9a85e2eb5a	6d623c5d5a2a1442
	4-7	cd38e6dee1458de7	acb73305cddb1207	148f31a512bbade5	ecd66ba86d4ab7e9
	8-11	92aafb1e9cfa1fcb	533c19b80a7c8968	e3ce7a41b11b4d75	aef3823c2a004b20
	12-15	8d41a28b0d847692	7f214e01c4e96950	0000000000000000	0000000000000000
W_2	0-3	0000000000000000	0000000000000000	c2bc8e9a85e2eb5a	6d623c5d5a2a1442
	4-7	cd38e6dee1458de7	acb73305cddb1207	148f31a512bbade5	ecd66ba86d4ab7ea
	8-11	90668fd7ec6718ee	533c19b80a7c8968	dfce7a41b11b4d76	aef3823c2a004b20
	12-15	8d41a28b0d847692	7f214e01c4e96950	0000000000000000	ffffffffffffffff

Table 5. Colliding message pair for 22-step SHA-256 with standard IV

W_1	0-7	00000000	00000000	0be293bf	99c539c9	1c672194	99b6a58a	5bf1d0ae	0a9a18d3
	8-15	0c18cf1c	329b3e6e	dc4e7a43	ab33823f	8d41a28d	7f214e03	00000000	00000000
W_2	0-7	00000000	00000000	0be293bf	99c539c9	1c672194	99b6a58a	5bf1d0ae	0a9a18d4
	8-15	07d56809	329b3e6e	dc0e7a44	ab33823f	8d41a28d	7f214e03	00000000	ffffffff

Table 6. Colliding message pair for 23-step SHA-256 with standard IV. These messages utilize a single local collision starting at Step $i = 8$.

W_1	0-7	122060e3	000f813f	d92d3fc6	ea4a475f	fb0c6581	dc4558c4	d86428b4	6e2ca576
	8-15	c8d597bf	6372d4c2	ddbd721c	79d654c4	f0064002	a894b7b6	91b7628e	3224db20
W_2	0-7	122060e3	000f813f	d92d3fc6	ea4a475f	fb0c6581	dc4558c4	d86428b4	6e2ca576
	8-15	c8d597c0	6372d4c1	ddbd721c	78d6b4c5	f0064002	a894b7b6	91b7628e	3224db20

Table 7. Colliding message pair for 23-step SHA-256 with standard IV. These messages utilize a single local collision starting at Step $i = 9$.

W_1	0-7	c201bef2	14cc32c9	3b80da44	d8212037	8987161d	a790cb4a	53b8d726	89e9a288
	8-15	3edd76e0	05f41ddc	9ebc0fc3	e099698a	2eaec58f	e7060b78	95d7030d	6bf777c0
W_2	0-7	c201bef2	14cc32c9	3b80da44	d8212037	8987161d	a790cb4a	53b8d726	89e9a288
	8-15	3edd76e0	05f41ddd	9ebc0fc2	e099c98a	2daf2590	e7060b78	95d7030d	6bf777c0

Table 8. Colliding message pair for 24-step SHA-256 with standard IV. These messages utilize a single local collision starting at Step $i = 10$.

W_1	0-7	657adf63	06c066d7	90f0b709	95a3e1d1	c3017f24	fad6c2bf	dff43685	6abff0da
	8-15	e6cfc63f	de8fb4c1	c20ca05b	f74815cc	c2e789d9	208e7105	cc08b6cf	70171840
W_2	0-7	657adf63	06c066d7	90f0b709	95a3e1d1	c3017f24	fad6c2bf	dff43685	6abff0da
	8-15	e6cfc63f	de8fb4c1	c20ca05c	f74815cb	c2e7e9d9	1f8ed106	cc08b6cf	70171840

Table 9. Colliding message pair for 23-step SHA-512 with standard IV. These messages utilize a single local collision starting at Step $i = 8$.

W_1	0-3	b9fa6fc4729ca55c	8718310e1b3590e1	1d3d530cb075b721	99166b30ecbdd705
	4-7	27ed55b66c090b62	754b2163ff6feec5	6685f40fd8ab08f8	590c1c0522f6fdfd
	8-11	b947bb4013b688c1	d9d72ca8ab1cac04	69d0e120220d4edc	30a2e93aeef24e3f
	12-15	84e76299718478b9	f11ae711647763e5	d621d2687946e862	0ee57069123ecc8b
W_2	0-3	b9fa6fc4729ca55c	8718310e1b3590e1	1d3d530cb075b721	99166b30ecbdd705
	4-7	27ed55b66c090b62	754b2163ff6feec5	6685f40fd8ab08f8	590c1c0522f6fdfd
	8-11	b947bb4013b688c2	d9d72ca8ab1cac03	69d0e120220d4edc	30a3493aeef25076
	12-15	84e76299718478b9	f11ae711647763e5	d621d2687946e862	0ee57069123ecc8b

Table 10. Colliding message pair for 24-step SHA-512 with standard IV. These messages utilize a single local collision starting at Step $i = 10$.

W_1	0-3	dedb689cfc766965	c7b8e064ff720f7c	c136883560348c9c	3747df7d0cf47678
	4-7	855e17555cfedc5f	88566babccaa63e9	5dda9777938b73cd	b17b00574a4e4216
	8-11	86f3ff48fd12ea19	cd15c6f8d6da38ce	5e2c6b7b0411e70b	36ed67e93a794e66
	12-15	1b65e96b02767821	04d0950089db6c68	5bc9b9673e38eff3	b05d879ad024d3fa
W_2	0-3	dedb689cfc766965	c7b8e064ff720f7c	c136883560348c9c	3747df7d0cf47678
	4-7	855e17555cfedc5f	88566babccaa63e9	5dda9777938b73cd	b17b00574a4e4216
	8-11	86f3ff48fd12ea19	cd15c6f8d6da38ce	5e2c6b7b0411e70c	36ed67e93a794e65
	12-15	1b66096b02767829	04d0f50089db6e9f	5bc9b9673e38eff3	b05d879ad024d3fa

Note

The submitted version of the paper contained much more details than is provided in the current version. Due to page-limit restrictions on the published version of the paper, we are unable to provide such details, which to a certain extent may affect the readability of the paper. A longer and more detailed version is available at [10].

References

1. Secure Hash Standard. Federal Information Processing Standard Publication 180-2. U.S. Department of Commerce, National Institute of Standards and Technology (NIST) (2002), http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf
2. Gilbert, H., Handschuh, H.: Security Analysis of SHA-256 and Sisters. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 175–193. Springer, Heidelberg (2004)
3. Indesteege, S., Mendel, F., Preneel, B., Rechberger, C.: Collisions and other Non-Random Properties for Step-Reduced SHA-256. Cryptology eprint Archive (April 2008); Selected Areas in Cryptography (accepted, 2008), http://eprint.iacr.org/2008/131
4. Mendel, F., Pramstaller, N., Rechberger, C., Rijmen, V.: Analysis of Step-Reduced SHA-256. In: Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 126–143. Springer, Heidelberg (2006)
5. Mendel, F., Pramstaller, N., Rechberger, C., Rijmen, V.: Analysis of Step-Reduced SHA-256. Cryptology eprint Archive (March 2008), http://eprint.iacr.org/2008/130
6. Nikolić, I., Biryukov, A.: Collisions for Step-Reduced SHA-256. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 1–16. Springer, Heidelberg (2008)
7. Sanadhya, S.K., Sarkar, P.: New Local Collisions for the SHA-2 Hash Family. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 193–205. Springer, Heidelberg (2007)
8. Sanadhya, S.K., Sarkar, P.: Attacking Reduced Round SHA-256. In: Bellovin, S., Gennaro, R. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 130–143. Springer, Heidelberg (2008)
9. Sanadhya, S.K., Sarkar, P.: Deterministic Constructions of 21-Step Collisions for the SHA-2 Hash Family. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222. Springer, Heidelberg (2008)
10. Sanadhya, S.K., Sarkar, P.: New Collision attacks Against Up To 24-step SHA-2. Cryptology eprint Archive (September 2008), http://eprint.iacr.org/2008/270
11. Sanadhya, S.K., Sarkar, P.: Non-Linear Reduced Round Attacks Against SHA-2 Hash family. In: Mu, Y., Susilo, W. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 254–266. Springer, Heidelberg (2008)

Secure Hierarchical Identity Based Encryption Scheme in the Standard Model

Yanli Ren and Dawu Gu

Dept. of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

Abstract. An identity based cryptosystem is a public key cryptosystem where the public key can be represented as an arbitrary string. Hierarchical identity based cryptography is a generalization of identity based encryption that mirrors an organizational hierarchy. It allows a root private key generator to distribute the workload by delegating private key generation and identity authentication to lower-level private key generators. Most of hierarchical identity based encryption schemes are provably secure in the random oracles or weak models without random oracles such as selective-ID model.

Currently, there is no hierarchical identity based encryption scheme that is fully CCA2 secure in the standard model, with short public parameters and a tight reduction. In this paper, we first propose a hierarchical identity based encryption scheme that is fully secure in the standard model. And it achieves IND-ID-CCA2 security based on the decision q-TBDHE problem. The ciphertext size is independent of the level of the hierarchy. Moreover, our scheme has short public parameters, high efficiency and a tight reduction simultaneously.

1 Introduction

An identity based (ID-based) cryptosystem [1] is a public key cryptosystem where the public key can be represented as an arbitrary string such as an email address. The user's private key is generated by a trusted authority, called a Private Key Generator (PKG), which applies its master key to issue private keys to identities that request them. For an Identity Based Encryption (IBE) scheme, Alice can securely encrypt a message to Bob using an unambiguous name of him, such as email address, as the public key. ID-based cryptosystems can simplify key management procedure compared to CA-based systems, so it can be an alternative way for CA-based public key systems in some occasions, especially when efficient key management and moderate security are required.

Shamir proposed the notion of IBE in 1984 as a way to simplify public key and certificate management. Many ID-based schemes have been proposed after that, but practical ID-based encryption schemes were not found until the work of Boneh and Franklin [6] in 2001. Their IBE scheme was based on groups with efficiently computable bilinear maps, but it is only provably secure in the random oracle model. It has been shown that when random oracles are instantiated with

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 104–115, 2008.

concrete hash functions, the resulting scheme may not be secure [7][13]. Canetti, Halevi, and Katz [14] suggested a weaker security notion for IBE, known as selective identity(selective-ID) security, relative to which they were able to build an inefficient but secure IBE scheme without using random oracles. Boneh and Boyen [8] presented two very efficient IBE systems ("BB1" and "BB2") with selective-ID security proofs, also without random oracles. The same authors [7] then proposed a coding-theoretic extension to their "BB1" scheme that allowed them to prove security for the full notion of adaptive identity(adaptive-ID) security without random oracles, but the construction was impractical. Waters [2] then proposed a much simpler extension to "BB1" also with an adaptive-ID security proof without random oracles; its efficiency was further improved in two independent papers, [17] and [10]. Almost all of the IBE systems since Boneh-Franklin follow the "common strategy" for proving security; consequently, they suffer from long parameters (when security is proven in the standard model) and lossy reductions (in the standard model or the random oracle model). Until 2006, there is no IBE system that is fully secure without random oracles, yet has short public parameters, or has a tight security reduction. Given this state of affairs, several papers [5,8,2] have encouraged work on the open problem of tight security; Waters [2] posed the open problem regarding compact public parameters. So Gentry [3] proposed an IBE scheme that is fully secure in the standard model with short public parameters and a tight security reduction, where the ciphertext does not leak the identity of the recipient. His scheme is simple and efficient, and his proof technique differs substantially from the "common strategy" described above.

Although having a single PKG would completely eliminate online lookup, it is undesirable for a large network because the PKG has a burdensome job. Not only is private key generation computationally expensive, but also the PKG must verify proofs of identity and establish secure channels to transmit private keys. Hierarchical ID-based cryptography was first proposed in [4] and [11] in 2002. It allows a root PKG to distribute the workload by delegating private key generation and identity authentication to lower-level PKGs. In an HIBE scheme, a root PKG needs only generate private keys for domain-level PKGs, who in turn generate private keys for users in their domains in the next level. Authentication and private key transmission can be done locally. To encrypt a message to Bob, Alice needs only obtain the public parameters of Bob's root PKG (and Bob's identifying information); there are no "lower-level parameters". Another advantage of HIBE schemes is damage control: disclosure of a domain PKG's secret does not compromise the secrets of higher-level PKGs.

The first construction for HIBE is due to Gentry and Silverberg [4] where security is based on the Bilinear Diffie-Hellman (BDH) assumption in the random oracle model. A subsequent construction due to Boneh and Boyen gives an efficient selective-ID secure HIBE based on BDH without random oracles [7]. In both constructions, the length of ciphertexts and private keys, as well as the time needed for decryption and encryption, grows linearly in the depth of the hierarchy. Then Boneh, Boyen and Goh [9] present an HIBE system where the ciphertext size as

well as the decryption cost are independent of the hierarchy depth. And they prove that the scheme is selective-ID secure in the standard model. Though the schemes that are selective-ID secure are also fully secure as long as one hashes the identity prior to using it, the reduction is not tight. Then Chatterjee and Sarkar [16] proposed an HIBE scheme that are fully secure in the standard model, but the size of public parameters and the ciphertext are dependent of the level of the hierarchy. In 2006, Man Ho Au constructed a constant size HIBE scheme that is fully secure in the standard model [12]. However, the scheme is only valid for a user with identity $ID = (ID_1, ID_2, \ldots, ID_i), i \geq 2$ and it is not secure when $i = 2$. Moreover, the scheme is only CPA secure in the standard model though an adaptive CCA-secure l-level hierarchical identity based encryption (HIBE) scheme Π can be constructed from a CPA-secure $l + 1$-level HIBE scheme Π' and a strong one-time signature scheme Sig. In fact, most HIBE scheme achieves CCA2 security using this technique showed by Canetti et al.[15].

Our Contributions. In Gentry's IBE scheme, they proposed an open problem, that is to construct a hierarchical IBE system that has a tight reduction based on a reasonable assumption. Currently, there is no HIBE scheme that is fully CCA2 secure in the standard model, with short public parameters and a tight reduction. In this paper, we propose an HIBE scheme that is fully CCA2 secure in the standard model. The ciphertext size is independent of the level of the hierarchy. Moreover, our scheme has short public parameters, high efficiency and a tight reduction simultaneously.

2 Definitions

2.1 Bilinear Map

Let p be a large prime number, G_1, G_2 are two groups of order p, and g is a generator of G_1. $e : G_1 \times G_1 \rightarrow G_2$ is a bilinear map, which satisfies the following properties [2]:

(1)Bilinearity: For all $u, v \in G_1$ and $a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$.
(2)Non-degeneracy: $e(g, g) \neq 1$.
(3)Computability: There exists an efficient algorithm to compute $e(u, v)$, $\forall u, v \in G_1$.

2.2 Complexity Assumptions

The security of our scheme is based on a complexity assumption that we call the decisional truncated bilinear Diffie-Hellman exponent assumption(TBDHE).

First, we recall the decisional version of q-BDHE problem [9], which is as follows: Given a vector of $2q + 2$ elements

$$(g', g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^q}, g^{\alpha^{q+2}}, \ldots, g^{\alpha^{2q}}, Z) \in G_1^{2q+1} \times G_2$$

to decide whether $Z = e(g, g')^{\alpha^{q+1}}$.

Since the tuple does not have the term $g^{\alpha^{q+1}}$, the bilinear map does not seem to help decide whether $Z = e(g, g')^{\alpha^{q+1}}$. Instead, we can use a truncated version of the q-BDHE problem, in which the terms $(g^{\alpha^{q+2}}, \ldots, g^{\alpha^{2q}})$ are omitted from the input vector. We call it the q-TBDHE problem for convenience. Clearly, the decision q-TBDHE problem is hard if the decision q-BDHE problem is hard.

In the scheme of [3], the author define the q-ABDHE problem as follows: Given a vector of $q + 4$ elements

$$(g', g'^{\alpha^{q+2}}, g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^q}, Z) \in G_1^{q+3} \times G_2$$

to decide whether $Z = e(g, g')^{\alpha^{q+1}}$.
Obviously, the q-ABDHE problem can be solved once the q-TBDHE problem is solved whereas not. So we can say that the TBDHE problem is at least as difficult as the ABDHE problem.

An algorithm A that outputs $w \in \{0, 1\}$ has advantage ε in solving the decision q-TBDHE if

$$|Pr[A(g', g, g^{\alpha}, \ldots, g^{\alpha^q}, e(g', g)^{\alpha^{q+1}}) = 0] \\ -Pr[A(g', g, g^{\alpha}, \ldots, g^{\alpha^q}, Z) = 0]| \geq \varepsilon,$$

where the probability is over the random choice of generators $g, g' \in G_1, \alpha \in Z_p^*, Z \in G_2$, and the random bits consumed by A. We refer to the distribution on the left as P_{TBDHE} and the distribution on the right as R_{TBDHE}.

We say that the decision (t, ε, q)-TBDHE assumption holds in G_1, G_2 if no t-time algorithm has advantage at least ε in solving the decision q-TBDHE problem in G_1, G_2.

2.3 Secure Models

IND-ID-CCA2: HIBE security (IND-ID-CCA2) [9] is defined by the following game between an adversary A and a challenger B.

Setup. The challenger B runs the Setup algorithm and gives A the resulting system parameters *params*, keeping the master key to itself.

Phase 1. A adaptively issues queries $q_1, \ldots, q_m$ where query q_i is one of the following:

Key generation query$< ID_i >$. B responds by running algorithm KeyGen to generate the private key d_i corresponding to the public key ID_i and sends d_i to A.

Decryption query$< ID_i, c_i >$. B responds by running algorithm KeyGen to generate the private key d_i corresponding to ID_i. It then runs algorithm Decrypt to decrypt the ciphertext c_i using the private key d_i and sends the resulting plaintext to A.

Challenge. A outputs an identity ID^* and two equal length plaintexts m_0, m_1 on which it wishes to be challenged. The only restriction is that A did not

previously issue a key generation query for ID^* or a prefix of ID^*. B picks a random bit $w \in \{0,1\}$ and sends c^* to A, where $c^* = Encrypt(params, ID^*, m_w)$.

Phase 2. A issues additional queries $q_{m+1}, \ldots, q_n$, where q_i is one of:

Key generation query $< ID_i >$. where $ID_i \neq ID^*$ and ID_i is not a prefix of ID^*.

Decryption query $c_i \neq c^*$ for ID^* or any prefix of ID^*. In both cases, B responds as in Phase 1. These queries may be adaptive.

Guess. Finally, the adversary outputs a guess $w' \in \{0,1\}$ and wins if $w = w'$.

We call an adversary A in the above game an IND-ID-CCA2 adversary. The advantage of A is defined as $|Pr[w' = w] - \frac{1}{2}|$.

Definition 1. An HIBE system is $(t, \varepsilon, q_k, q_d)$ IND-ID-CCA2 secure if all t-time IND-ID-CCA2 adversaries making at most q_k key generation queries and at most q_d decryption queries have advantage at most ε in winning the above game.

3 Hierarchical Identity Based Encryption Scheme

3.1 Set Up

Let G_1, G_2 be defined as above, and g is a generator of G_1. $g_1 = g^\alpha$, where $\alpha \in Z_p^*$ is a random number. $e : G_1 \times G_1 \to G_2$ is a bilinear map, and l is the maximum number of levels in the HIBE. $h : G_1^2 \times G_2^3 \longrightarrow Z_p^*, H : G_1^2 \times G_2^5 \longrightarrow Z_p^*$ are hash functions randomly chosen from a family of universal one-way hash functions. The PKG randomly chooses $g_2, g_3, h_i \in G_1 (i = 0, 1, \ldots, l)$, and $f(x) = ax + b$, where $a, b \in Z_p^*$. If $g_2 = g_3^{-a}$ or $h_0 = g_3^{-b}$, choose another $f(x)$ again. The public parameters are $(g, g_1, g_2, g_3, f(x), h, H, h_0, \ldots, h_l)$, α is the private key of PKG.

3.2 Key Generation

To a user U with identity $ID = (ID_1, ID_2, \ldots, ID_i) \in (Z_p^*)^i$, the PKG randomly chooses $r_{-1,i}, r_{0,i} \in Z_p^*$, and computes

$$d_{0,i} = (h_0 g_2^{r_{-1,i}} g_3^{f(r_{-1,i})})^\alpha (\prod_{k=1}^{i} h_l h_k^{ID_k})^{r_{0,i}},$$
$$d_{-1,i} = r_{-1,i}, d_{1,i} = g^{r_{0,i}}, d_{i+1,i} = h_{i+1}^{r_{0,i}}, \ldots, d_{l,i} = h_l^{r_{0,i}},$$

so the private key of U is $d = (d_{0,i}, d_{-1,i}, d_{1,i}, d_{i+1,i}, \ldots, d_{l,i})$.

If $h_0 g_2^{r_{-1,i}} g_3^{f(r_{-1,i})} = 1$, randomly choose $r_{-1,i}$ again.

The private key can also be generated by its parent $(ID_1, ID_2, \ldots, ID_{i-1})$ having the secret key $(d_{0,i-1}, d_{-1,i-1}, d_{1,i-1}, d_{i,i-1}, \ldots, d_{l,i-1})$. It computes:

$$d_{0,i} = d_{0,i-1} \cdot d_{l,i-1} \cdot d_{i,i-1}^{ID_i} \cdot (\prod_{k=1}^{i} h_l h_k^{ID_k})^t, d_{-1,i} = d_{-1,i-1},$$
$$d_{1,i} = d_{1,i-1} \cdot g^t, d_{k,i} = d_{k,i-1} \cdot h_k^t (k = i+1, \ldots, l), \text{ where } r_{0,i} = r_{0,i-1} + t.$$

3.3 Encryption

To encrypt a message $m \in G_2$, randomly choose $s \in Z_p^*$, and compute

$$c_1 = (\prod_{k=1}^{i} h_l h_k^{ID_k})^s, c_2 = g^s, c_3 = e(g_1, g_2)^s, c_4 = e(g_1, g_3)^s,$$
$$c_5 = m \cdot e(g_1, h_0)^{s+\gamma}, \beta = H(c_1, c_2, c_3, c_4, c_5, m, m \cdot e(g_1, h_0)^s),$$

where $\gamma = h(c_1, c_2, c_3, c_4, e(g_1, h_0)^s)$.

The ciphertext of message m is $c = (c_1, c_2, c_3, c_4, c_5, \beta)$.

Notice that encryption does not require any pairing computations once $e(g_1, h_0)$, $e(g_1, g_2), e(g_1, g_3)$ have been pre-computed.

3.4 Decryption

The recipient first decrypts $\frac{e(c_2, d_{0,i})}{c_4^{f(d_{-1,i})} \cdot c_3^{d_{-1,i}} e(c_1, d_{1,i})} = e(g_1, h_0)^s$, and

$$\gamma = h(c_1, c_2, c_3, c_4, e(g_1, h_0)^s), c_5 / e(g_1, h_0)^\gamma = R, R / e(g_1, h_0)^s = m.$$

Then he computes $\beta' = H(c_1, c_2, c_3, c_4, c_5, m, R)$ and verifies whether $\beta' = \beta$. If the equation holds, the ciphertext is valid. Otherwise, the recipient returns an error message.

4 Analysis of the HIBE Scheme

4.1 Indistinguishability of the Ciphertext

Theorem 1. Assume that the (t', ε', q)-TBDHE assumption holds in G_1, G_2, then the HIBE scheme is $(t, \varepsilon, q_k, q_d)$ IND-ID-CCA2 secure for $t = t' - O(t_{exp} \cdot lq) - O(t_{pair} \cdot q), \varepsilon = \varepsilon' + 1/(p-1), q_k + q_d \leq q - 1$, where t_{exp}, t_{pair} are the average time required to exponentiate and pairing in G_1, G_2 respectively.

Proof. Assume A is an IND-ID-CCA2 adversary described as above. We construct an algorithm B that solves the q-TBDHE problem as follows. At the outset of the game, B is given a vector $(g', g, g^\alpha, \ldots, g^{\alpha^q}, Z) \in G_1^{q+2} \times G_2$ to decide whether $Z = e(g', g)^{\alpha^{q+1}}$.

Set Up. B randomly chooses $f_1(x), f_2(x), f_3(x) \in (Z_p^*)[x]$ of degree q, where $f_1(x) = \sum_{i=0}^{q} a_i x^i, f_2(x) = \sum_{i=0}^{q} b_i x^i, f_3(x) = \sum_{i=0}^{q} c_i x^i$.

Let $g_1 = g^\alpha, h_0 = g^{f_1(\alpha)}, g_2 = g^{f_2(\alpha)}, g_3 = g^{f_3(\alpha)}, f(x) = -\frac{b_q}{c_q} x - \frac{a_q}{c_q}, h_i = g^{u_i} (i = 1, 2, \ldots, l)$, $u_i \in Z_p^*$ is a random number. If $g_2 = g_3^{b_q/c_q}$ or $h_0 = g_3^{a_q/c_q}$, randomly choose $f_1(x), f_2(x), f_3(x)$ again. Then B sends the public parameters $(g, g_1, g_2, g_3, f(x), h_0, h_1, \ldots, h_l)$ to A. Observe that from the viewpoint of the adversary, the distribution of these public parameters is identical to the real construction since $f_1(x), f_2(x), f_3(x), u_i$ are randomly chosen.

Phase 1. Key generation query.
A sends identity $ID = (ID_1, ID_2, \ldots, ID_i)$ to B. If $ID = \alpha$, B uses α to solve the q-TBDHE problem immediately. Else, B randomly chooses $r_{-1,i}, r_{0,i} \in Z_p^*$, and computes

$$d_{0,i} = (g^{\sum_{i=0}^{q-1}(a_i + r_{-1,i}b_i + f(r_{-1,i})c_i)\alpha^{i+1}}) \cdot (\prod_{k=1}^{i} h_l h_k^{ID_k})^{r_{0,i}},$$
$$d_{-1,i} = r_{-1,i}, d_{1,i} = g^{r_{0,i}}, d_{i+1,i} = g^{u_{i+1}r_{0,i}}, \ldots, d_{l,i} = g^{u_l r_{0,i}}.$$

So $d_{ID} = (d_{0,i}, d_{-1,i}, d_{1,i}, d_{i+1,i}, \ldots, d_{l,i})$. If $h_0 g_2^{r_{-1,i}} g_3^{f(r_{-1,i})} = 1$, randomly choose $r_{-1,i}$ again.

It is a valid private key, because
$f(r_{-1,i}) = -\frac{b_q}{c_q} r_{-1,i} - \frac{a_q}{c_q}$, and $a_q + r_{-1,i}b_q + f(r_{-1,i})c_q = 0$,

$$\begin{aligned} g^{\sum_{i=0}^{q-1}(a_i + r_{-1,i}b_i + f(r_{-1,i})c_i)\alpha^{i+1}} &= g^{\sum_{i=0}^{q}(a_i + r_{-1,i}b_i + f(r_{-1,i})c_i)\alpha^{i+1}} \\ &= (g^{f_1(\alpha)} \cdot g^{r_{-1,i}f_2(\alpha)} \cdot g^{f(r_{-1,i})f_3(\alpha)})^{\alpha} \\ &= (h_0 g_2^{d_{-1,i}} g_3^{f(d_{-1,i})})^{\alpha}, \end{aligned}$$

$d_{0,i} = (h_0 g_2^{d_{-1,i}} g_3^{f(d_{-1,i})})^{\alpha} \cdot (\prod_{k=1}^{i} h_l h_k^{ID_k})^{r_{0,i}}$.

Therefore, d_{ID} is randomly distributed because of the randomness of $r_{-1,i}, r_{0,i}$.

Decryption query. A sends (ID, c) to B.

B first executes the key generation query to identity ID as above, then decrypts and verifies c with the private key of identity ID according to the decryption process. If c can pass the verification, B sends A the plaintext; otherwise, B returns an error message.

Challenge. A sends (ID^*, m_0, m_1) to B, where ID^* or its prefix have never been queried the private key in phase 1.

B randomly chooses $m_w, w \in \{0, 1\}$, and computes

$$c_1^* = \prod_{k=1}^{i} (g')^{u_l + u_k ID_k^*}, c_3^* = Z^{b_q} \cdot e(g', g)^{\sum_{i=0}^{q-1} b_i \alpha^{i+1}},$$
$$c_2^* = g', c_4^* = Z^{c_q} \cdot e(g', g)^{\sum_{i=0}^{q-1} c_i \alpha^{i+1}},$$
$$c_5^* = m_w \cdot \frac{e(c_2^*, d_{0,i^*})}{(c_4^*)^{f(d_{-1,i}^*)} (c_3^*)^{d_{-1,i}^*} e(d_{1,i^*}, c_1^*)} \cdot e(g_1, h_0)^{\gamma^*},$$
$$\beta^* = H(c_1^*, c_2^*, c_3^*, c_4^*, c_5^*, m_w, m_w \cdot \frac{e(c_2^*, d_{0,i^*})}{(c_4^*)^{f(d_{-1,i}^*)} (c_3^*)^{d_{-1,i}^*} e(d_{1,i^*}, c_1^*)},$$

where $\gamma^* = h(c_1^*, c_2^*, c_3^*, c_4^*, \frac{e(c_2^*, d_{0,i^*})}{(c_4^*)^{f(d_{-1,i}^*)} (c_3^*)^{d_{-1,i}^*} e(d_{1,i^*}, c_1^*)})$, and
$d_{0,i^*}, d_{-1,i^*}, d_{1,i^*}$ are the elements of a private key of ID^*.
(**Remark:** For any private key of ID^*,

$$\frac{e(c_2^*, d_{0,i^*})}{(c_4^*)^{f(d_{-1,i}^*)} (c_3^*)^{d_{-1,i}^*} e(d_{1,i^*}, c_1^*)} = Z^{a_q} \cdot e(g', g)^{\sum_{i=0}^{q-1} a_i \alpha^{i+1}}.$$

Therefore, B cannot decide whether $Z = e(g', g)^{\alpha^{q+1}}$ even if he can generate multiple random decryption keys for ID^*.)

Then B sends c^* to A, where $c^* = (c_1^*, c_2^*, c_3^*, c_4^*, c_5^*, \beta^*)$.

let $s^* = log_g g'$, c^* is a valid ciphertext for m_w under the randomness of s^*. Since $log_g g'$ is uniformly random, s^* is uniformly random, and so c^* is a valid, appropriately-distributed challenge to A.

Phase 2. A issues additional queries as phase 1.

Key generation query $< ID_i >$, where $ID_i \neq ID^*$ and ID_i is not a prefix of ID^*.

Decryption query $c_i \neq c^*$ for ID^* or any prefix of ID^*. In both cases, B responds as in Phase 1. These queries may be adaptive.

Guess. A submits a guess $w' \in \{0,1\}$. If $w' = w$, B outputs 0 (indicating that $Z = e(g', g)^{\alpha^{q+1}}$); otherwise, it outputs 1.

Probability Analysis

Lemma 1. When Z is sampled according to P_{TBDHE}, the joint distribution of A's view and the bit w is indistinguishable from that in the actual construction, except with probability $1/(p-1)$.

Proof. When B's input is sampled from P_{TBDHE}, B's simulation appears perfect to A if A makes only key generation queries. B's simulation still appears perfect if A makes decryption queries only on identities for which it queries the private key, since B's responses give A no additional information. Furthermore, querying well-formed ciphertexts to the decryption oracle does not help A distinguish between the simulation and the actual construction, since, by the correctness of Decrypt, well-formed ciphertexts will be accepted in either case. Finally, querying a non-well-formed ciphertext for ID does not help A distinguish, since this cihertext will fail the "decrypt" check under every valid private key for ID. Thus, the lemma follows from the following two claims.

Claim 1 Assuming the adversary does not find a collision in h, H, then the decryption oracle, in the simulation and in the actual construction, rejects all invalid ciphertexts under identities or their prefix not queried by A.

Proof. Let $log(\cdot)$ denote the logarithms to the base g, and an invalid ciphertext $c = (c_1, c_2, c_3, c_4, c_5, \beta)$ associated with an identity ID for

$$c_1 = (\prod_{k=1}^{i} h_l h_k^{ID_k})^{s_1}, c_2 = g^{s_2}, c_3 = e(g_1, g_2)^{s_3},$$
$$c_4 = e(g_1, g_3)^{s_4}, c_5 = m \cdot e(g_1, h_0)^{s_5+\gamma}, \beta,$$

where $\gamma = h(c_1, c_2, c_3, c_4, e(g_1, h_0)^{s_5})$, and $s_1 \neq s_2, s_3, s_4$ or s_5.

According to the decryption process, a ciphertext c can be accepted if

$$\frac{e(c_2, d_{0,i})}{c_4^{f(d-1,i)} \cdot c_3^{d-1,i} \cdot e(c_1, d_{1,i})} = e(g_1, h_0)^{s_5}, c_5 / e(g_1, h_0)^{\gamma} = R,$$
$$R / e(g_1, h_0)^{s_5} = m, \beta = H(c_1, c_2, c_3, c_4, c_5, m, R), \quad (1)$$

where $d_{-1,i}, d_{0,i}, d_{1,i}$ are the elements of a private key of ID.

And according to (1),

$$\frac{e(c_2, d_{0,i})}{c_4^{f(d-1,i)} \cdot c_3^{d-1,i} \cdot e(c_1, d_{1,i})} = \frac{e(c_2, (h_0 g_2^{r-1,i} g_3^{f(r-1,i)})^{\alpha} (\prod_{k=1}^{i} h_l h_k^{ID_k})^{r0,i})}{c_3^{r-1,i} \cdot c_4^{f(r-1,i)} \cdot e(c_1, g^{r0,i})} = e(g_1, h_0)^{s_5}.$$

Since A has not queried the decryption key associated with the identity or its prefix, and $r_{0,i}$ is randomly chosen from Z_p^*, we know that

$$e(g_1, h_0)^{s_5} c_4^{f(d-1,i)} c_3^{d-1,i} = e(c_2, (h_0 g_2^{d-1,i} g_3^{f(d-1,i)})^{\alpha}), \quad (2)$$
$$e(c_2, \prod_{k=1}^{i} h_l h_k^{ID_k}) = e(c_1, g). \quad (3)$$

From (3), we know $s_1 = s_2$. Since $r_{-1,i}$ is randomly chosen from Z_p^* and $f(r_{-1,i}) = -\frac{b_q}{c_q} r_{-1,i} - \frac{a_q}{c_q}$, according to (2),

$$e(g_1, h_0)^{s_5} c_4^{-\frac{a_q}{c_q}} = e(c_2, (h_0 g_3^{-\frac{a_q}{c_q}})^{\alpha}), c_3 c_4^{-\frac{b_q}{c_q}} = e(c_2, (g_2 g_3^{-\frac{b_q}{c_q}})^{\alpha}). \quad (4)$$

From (4),

$$(s_5 - s_2) log h_0 - \frac{a_q}{c_q} log g_3 (s_4 - s_2) = 0,$$
$$(s_3 - s_2) log g_2 - \frac{b_q}{c_q} log g_3 (s_4 - s_2) = 0. \quad (5)$$

Since $log h_0 = f_1(\alpha), log g_2 = f_2(\alpha), log g_3 = f_3(\alpha)$, $f_1(x), f_2(x), f_3(x)$ are randomly chosen, $log h_0, log g_2, log g_3$ are uniformly random. And because $h_0 \neq g_3^{\frac{a_q}{c_q}}, g_2 \neq g_3^{\frac{b_q}{c_q}}$, we know $s_2 = s_3 = s_4 = s_5$ from (5).

Therefore, $s_1 = s_2 = s_3 = s_4 = s_5$. A ciphertext can be accepted only if it is valid. The decryption oracle, in the simulation and in the actual construction, rejects all invalid ciphertexts under identities or their prefix not queried by A.

Claim 2 If the decryption oracle rejects all invalid ciphertexts, then A has advantage $1/(p-1)$ in guessing the bit w.

When Z is sampled from P_{TBDHE}, a challenge ciphertext c^* is a valid ciphertext for the randomness of s^*.

First, we show the adversary cannot obtain a valid ciphertext $c = (c_1, c_2, c_3, c_4, c_5, \beta)$ for m_w associated with an identity ID from c^*, where

$$c_1 = (\prod_{k=1}^{i} h_l h_k^{ID_k})^{s'}, c_2 = g^{s'}, c_3 = e(g_1, g_2)^{s'},$$
$$c_4 = e(g_1, g_3)^{s'}, c_5 = m_w \cdot e(g_1, h_0)^{s'+\gamma}, \beta,$$

where $\gamma = h(c_1, c_2, c_3, c_4, e(g_1, h_0)^{s'})$.

There are three cases to consider:

(1) $s' = s^*, ID = ID^*$: $c = c^*$, the ciphertext will certainly be rejected.
(2) $s' = s^*, ID \neq ID^*$: $(c_2, c_3, c_4) = (c_2^*, c_3^*, c_4^*)$.

Assume $ID = (ID_1, \ldots, ID_j), ID^* = (ID_1^*, \ldots, ID_i^*)$.

When $j \geq i$, $c_1 = c_1^* \cdot \prod_{k=1}^{i} (h_k^{s^*})^{ID_k - ID_k^*} \cdot \prod_{k=i+1}^{j} (h_l h_k^{ID_k})^{s^*}$.

Otherwise, $c_1 = c_1^* \cdot \prod_{k=1}^{j} (h_k^{s^*})^{ID_k - ID_k^*} \cdot \prod_{k=j+1}^{i} (h_l h_k^{ID_k^*})^{-s^*}$.

Since $s^* = log_g g'$ is uniformly random, the adversary cannot compute a valid c_1 from c^*.

(3) $s' \neq s^*$:

$$(c_1, c_2, c_3, c_4, \gamma) \neq (c_1^*, c_2^*, c_3^*, c_4^*, \gamma^*), c_5 = c_5^* \cdot e(g_1, h_0)^{s'+\gamma-s^*-\gamma^*}.$$

Since $s^* = log_g g'$ is uniformly random, γ^* is uniformly random, the adversary cannot compute a valid c_5 from c^*.

Therefore, the adversary cannot obtain a valid ciphertext c for m_w associated with identity ID from c^*.

Finally, We know

$c_5^* = m_w \cdot \frac{e(c_2^*, d_{0,i^*})}{(c_4^*)^{f(d_{-1,i}^*)} (c_3^*)^{d_{-1,i}^*} e(d_{1,i^*}, c_1^*)} \cdot e(g_1, h_0)^{\gamma^*} = m_w \cdot e(g_1, h_0)^{s^*+\gamma^*}$, where $\gamma^* = h(c_1^*, c_2^*, c_3^*, c_4^*, e(g_1, h_0)^{s^*})$.

Since $s^* = log_g g'$ is uniformly random, γ^* is uniformly random, and $s^* + \gamma^* = 0$ with probability $1/(p-1)$, c_5^*/m_w is uniformly random for the adversary except probability $1/(p-1)$. So A can guess $w' = w$ with probability $\frac{1}{2} + \frac{1}{p-1}$.

Lemma 2. When Z is sampled according to R_{TBDHE}, the joint distribution of A's view and the bit w is indistinguishable from that in the actual construction.

Proof. The lemma follows from *Claim 1* and the following claim.

Claim 3 If the decryption oracle rejects all invalid ciphertexts, then A has no advantage in guessing the bit w.

When Z is sampled from R_{TBDHE}, we know that $s_3, s_4 \neq s^*$. As above, the adversary cannot obtain a valid ciphertext c for m_w associated with identity ID from c^*. And

$$\begin{aligned} c_5^* &= m_w \cdot \frac{e(c_2^*, d_{0,i^*})}{(c_4^*)^{f(d_{-1,i}^*)}(c_3^*)^{d_{-1,i}^*} e(d_{1,i^*}, c_1^*)} \cdot e(g_1, h_0)^{\gamma^*} \\ &= m_w \cdot e(g_1^{s^*}, h_0 g^{-a_q \alpha^q}) \cdot Z^{a_q} \cdot e(g_1, h_0)^{\gamma^*}, \end{aligned}$$

where $\gamma^* = h(c_1^*, c_2^*, c_3^*, c_4^*, e(g_1^{s^*}, h_0 g^{-a_q \alpha^q}) \cdot Z^{a_q})$.

Since s^*, a_q, Z are uniformly random, γ^* is uniformly random, and c_5^*/m_w is random for the adversary. So A can only guess $w' = w$ with probability $1/2$ and has no advantage in guessing the bit w.

Time Complexity: In the simulation, B's overhead is dominated by computing private keys and decrypting the ciphertexts in response to A's queries. Each key generation computation requires $O(l)$ exponentiations in G_1, and each decryption computation requires $O(1)$ exponentiations and pairings in G_1, G_2. Since A makes at most $q-1$ such queries, $t' = t + O(t_{exp} \cdot lq) + O(t_{pair} \cdot q)$.

In the reductions , B's success probability and time complexity are the same as A's, except for additive factors depending on p and q respectively. So, one could say that our HIBE system has a tight security reduction in the standard model, addressing an open problem posed in [3].

4.2 Efficiency

In the following table, we compare the efficiency of the known HIBE schemes in the standard model.

Table 1. Comparison to other HIBE schemes in the standard model

Scheme	Security model	Public key size	Private key size	Ciphertext size	Pairing operation
BB[7]	IND-sID-CPA	$O(l)$	$O(i)$	$O(i)$	$O(i)$
BBG[9]	IND-sID-CPA	$O(l)$	$O(i)$	$O(1)$	$O(1)$
CS[16]	IND-ID-CPA	$O(l)$	$O(i)$	$O(i)$	$O(i)$
ALYW[12]	IND-ID-CPA	$O(l)$	$O(i)$	$O(1)$	$O(1)$
Ours	IND-ID-CCA2	$O(l)$	$O(i)$	$O(1)$	$O(1)$

In this table, i represents the number of levels of identity on which the operations are performed, l is the maximum number of levels in the HIBE. "sID, ID" denote selective-ID and adaptive-ID model respectively.

5 Conclusions

Currently, there is no hierarchical identity based encryption scheme that is fully CCA2 secure in the standard model, with short public parameters and a tight reduction. In this paper, we first propose an HIBE scheme that is fully secure in the standard model. And it achieves IND-ID-CCA2 security based on the decision q-TBDHE problem. The ciphertext size is independent of the level of the hierarchy. Moreover, our scheme has short public parameters, high efficiency and a tight reduction simultaneously.

Acknowledgements

We would like to thank anonymous referees for their helpful comments and suggestions. The work described in this paper was supported by the National Science Foundation of China under Grant (No.60573031), and also funded by 863 Hi-tech Research and Development Program of China (2006AA01Z405).

References

1. 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)
2. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)
3. Gentry, C.: Practical Identity-based encryption without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 445–464. Springer, Heidelberg (2006)
4. Gentry, C., Silverberg, A.: Hierarchical id-based cryptography. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 548–566. Springer, Heidelberg (2002)
5. Boneh, D., Gentry, C., Waters, B.: Collusion-resistant broadcast encryption with short ciphertexts and private keys. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 258–275. Springer, Heidelberg (2005)
6. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
7. Boneh, D., Boyen, X.: Efficient selective-ID secure identity based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004)
8. Boneh, D., Boyen, X.: Secure identity based encryption without random oracles. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 443–459. Springer, Heidelberg (2004)

9. Boneh, D., Boyen, X., Goh, E.J.: Hierarchical identity based encryption with constant size ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005)
10. Naccache, D.: Secure and practical identity-based encryption. Cryptology ePrint Archive, Report 2005/369 (2005), http://eprint.iacr.org/
11. 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)
12. Au, M.H., Liu, J.K., Yuen, T.H., Wong, D.S.: Practical hierarchical identity based encryption and signature schemes without random oracles, http://eprint.iacr.org/2006/368
13. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited (preliminary version. In: STOC 1998, pp. 209–218 (1998)
14. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 255–271. Springer, Heidelberg (2003)
15. Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 207–222. Springer, Heidelberg (2004)
16. Chatterjee, S., Sarker, P.: On Hierarchical Identity Based Encryption Protocols with Short Public Parameters, http://eprint.iacr.org/2006/279
17. Chatterjee, S., Sarkar, P.: Trading time for space: towards an efficient IBE scheme with short(er) public parameters in the standard model. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 424–440. Springer, Heidelberg (2006)

A Fuzzy ID-Based Encryption Efficient When Error Rate Is Low

Jun Furukawa[1], Nuttapong Attrapadung[2], Ryuichi Sakai[3], and Goichiro Hanaoka[4]

[1] NEC Corporation, Japan
j-furukawa@ay.jp.nec.com
[2] AIST, Japan
n.attrapadung@aist.go.jp
[3] Osaka Electro-Communication University, Japan
sakai@isc.osakac.ac.jp
[4] AIST, Japan
hanaoka-goichiro@aist.go.jp

Abstract. The fuzzy identity-based encryption schemes are attribute-based encryption schemes such that each party with the private key for an attribute set $\mathcal{S}$ is allowed to decrypt ciphertexts encrypted by an attribute set $\mathcal{S}'$, if and only if the two sets $\mathcal{S}$ and $\mathcal{S}'$ are close to each other as measured by the set-overlap-distance metric. That is, there is a threshold t and, if t out of n attributes of $\mathcal{S}$ are also included in $\mathcal{S}'$, the receivers can decrypt the ciphertexts. In previous schemes, this threshold t is fixed when private keys are generated and the length of ciphertexts are linear to n. In this paper, we propose a novel fuzzy identity-based encryption scheme where the threshold t is flexible by nature and the length of ciphertexts are linear to $n - t$. The latter property makes the scheme short if it allows receivers to decrypt ciphertexts when error rate $n - t$, i.e., distance between the two attribute sets, is low.

Keywords: Fuzzy, biometrics, identity-based, low error rate.

1 Introduction

The notion of fuzzy identity-based encryption (FIBE) schemes was introduced by Sahai et al. in [16]. These schemes are in a class of identity-based encryption schemes where each identity is viewed as a set of descriptive attributes [9,11,1,8,10,7,5] and each party with the private key of an attribute set $\mathcal{S}$ is allowed to decrypt ciphertexts encrypted by an attribute set $\mathcal{S}'$, if and only if $\mathcal{S}$ and $\mathcal{S}'$ are close to each other as measured by the set-overlap-distance metric.

Interesting applications of FIBE schemes can be found where attributes are biometrics. Since biometrics are hard to be free from noise, error-tolerance property of FIBE schemes is what solves this problem of using biometrics as identities. In such applications, for example, giving a privilege to Alice by issuing a private key is easy if she is physically present since the authentication process is direct

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 116–129, 2008.

and straight forward. Then, Alice can present her authorized public key, i.e., her biometrics, to Bob even if she carries no electronic devices. Then, if Bob wants to send some data to her which data are allowed to be revealed to only privileged people, he encrypts this data by Alice's biometric and sends it her. Now, Alice can decrypt it later from her home or office.

According to Naor's observation in [4], identity-based encryption schemes can be transformed into signature schemes. Hence, by generating a signature, Alice can also prove to Bob standing in front of her that she is privileged person.

The first two FIBE schemes were proposed in [16]. In these schemes, a receiver is able to decrypt a ciphertext that is encrypted by a set $\mathcal{S}$ of n attributes only when he has more than t keys that correspond to attributes in $\mathcal{S}$. Here t is a threshold that is fixed when those keys for attributes are generated and the length of ciphertext is linear to n. Although it is possible to make this t flexible by preparing multiple systems or by preparing attributes whose corresponding keys are distributed to every players. These modifications increase costs of the system. When FIBE schemes are applied to cases where attributes are biometrics, it is natural that one's two biometrics at different observations are close to each other. Hence, it is likely that $n - t << n$ in majority of the cases.

In this paper, we propose a novel FIBE scheme where the threshold t is flexible by nature and the length of ciphertexts is linear to $n-t$. The latter property implies that ciphertexts are much shorter than those in previous schemes in most natural cases. More precisely, users who are granted d attributes are required to keep only $d+1$ private elements in an elliptic curve with a paring. And ciphertexts, that are encrypted by a set $\mathcal{S}$ of n attributes and that can be decrypted by t keys that correspond to attributes in $\mathcal{S}$, are composed of $2(n-t+1)$ elements in the elliptic curve or in its target field of the pairing. Our scheme also has a nice property that attributes can be dynamically granted to receivers while they are granted to receiver as whole in the schemes in [16].

Our scheme is based on the identity-based broadcast encryption scheme proposed by Sakai et al. in [15]. Later, the security of this scheme is proved in [12] in the generic group model and an essentially the same scheme is independently proposed in [13] with a proof in the random oracle model. In our scheme, the keys for attributes are similar to those of receivers in [15]. A ciphertext in our scheme for an attribute set $\mathcal{S}$ is similar to that for the corresponding receiver set with an exception that the public key in our scheme is reencrypted. Hence, a receiver, in the sense of [12], needs this new public key as well as his private key for the decryption. The number of elements required for the decryption among this new public key is the number of receivers in the set for which the ciphertext is encrypted. Moreover, every time the receiver obtained a private key of another receiver, the number of the elements of the new public key required for the decryption decreases by one. Hence, if a ciphertext includes $n-t$ elements of the new public key, t private keys of receivers, who are included in the $\mathcal{S}$ for which the ciphertext is encrypted, are enough for decrypting it in our scheme.

We prove that our scheme is key indistinguishable under fuzzy selective ID-and-attribute and chosen plaintext attacks in the random oracle model if the

general bilinear decision assumption holds. This security model is a variant of that introduced in [16] along the line of selective security introduced in [6]. The general bilinear decision assumption is proved to hold in the generic bilinear group model.

The paper is organized as follows. Section 2 describes the model of fuzzy selective ID-and-attribute security. Section 3 proposes our FIBE scheme. Section 4.2 proves our scheme is key indistinguishable under fuzzy selective ID-and-attribute and chosen plaintext attacks. Section 5 compares our scheme with previous schemes.

2 Model

2.1 System and Algorithms

Players in FIBE schemes are private key generator (PKG), receivers, and senders. Each receiver is labeled by an index and is given some attributes. They use four algorithms Setup, AttGrant, Encrypt, and Decrypt. Let R_i denote a receiver with an index i. Let $\mathcal{N}$ be the set of indices of the all receivers and $\mathcal{M}$ be the set of indices of the all attributes. Receivers should be distinguished by indices so that the PKG can decides whether or not it may give a key of new attribute to the relevant receiver. Note that the PKG can give keys for multiple of attributes and they are not required to be given at the same time. Let $\mathcal{K}$ denote key space.

Setup: A probabilistic algorithm for the PKG that, given some security parameters, outputs the public parameter *params* and the master key *mkey*. The descriptions of an attribute index space $\mathcal{M}$, receiver identity space $\mathcal{N}$, and a key space $\mathcal{K}$ are included in *params*. *params* is given to all interested players while *mkey* is kept secret.

AttGrant: A (possibly) probabilistic algorithm for the PKG to give receivers attributes. Given *params*, an index $i \in \mathcal{N}$ of receiver, an index $j \in \mathcal{M}$ of attribute, and *mkey*, it outputs an attribute key $skey_{i,j}$ for R_i' s j-attribute. $skey_{i,j}$ is given to R_i.

Encrypt: A probabilistic algorithm for senders that, given a set of attribute indices $\mathcal{S} \subset \mathcal{M}$ and an error tolerance parameter t, outputs a key $key \in \mathcal{K}$ and a header *hdr*. $\mathcal{S}$ and t are included in *hdr*.
Receivers are expected to have at least t out of $|\mathcal{S}|$ attribute keys among those whose indices are in $\mathcal{S}$ to compute *key* from this *hdr*.

Decrypt: A deterministic algorithm for receiver that, given $hdr, params$, and $\{skey_{i,j}\}_{j \in T}$ such that $T \subset \mathcal{S} \in hdr$ and $|T| = t \in hdr$ for some $i \in \mathcal{N}$, outputs a key *key*.

We require well constructed FIBE schemes to be complete. That is, it holds that for every security parameters, for every random tapes input to algorithms,

$$(mkey, params) \leftarrow \mathsf{Setup}$$
$$\forall \mathcal{S}, t \text{ s.t. } \mathcal{S} \subset \mathcal{M} \in params, 0 < t \leq |\mathcal{S}|$$

$$
\begin{aligned}
(key, hdr) &\leftarrow \mathsf{Encrypt}(\mathcal{S}, t,) \\
\forall i, T \text{ s.t. } T \subset \mathcal{S}, |T| &= t \\
(skey_{i,j})_{j \in T} &\leftarrow (\mathsf{AttGrant}(params, mkey, i, j))_{j \in T} \\
key' &\leftarrow \mathsf{Decrypt}(hdr, (skey_{i,j})_{j \in T}) \\
key &= key'
\end{aligned}
$$

Our set of algorithms has a number of advantages over that in [16]. Our Setup does not include the error tolerance parameter in *params*. This is because the error tolerance parameter can vary among different ciphertexts in our scheme. Our AttGrant, which corresponds to Extract in [16], is able to dynamically grant receivers new attributes while Extract should give a set of attributes to each receiver as whole.

2.2 Security Requirements

We first consider the following game.

Definition 1. Key-distinguishing game under fuzzy selective ID-and-attribute and chosen plaintext attacks *proceeds between an adversary* $\mathcal{A}$ *and a challenger* $\mathcal{C}$ *as in the following:*

1. $\mathcal{A}$ *outputs a receiver identity* i^**, an attribute set* $\mathcal{S}^*$*, and an attribute set* T^* *of size* $t^* - 1$.
2. $\mathcal{C}$*, given some security parameters, runs* Setup *and obtains* params *and* mkey. params *is given to* $\mathcal{A}$.
3. $\mathcal{A}$ *sends the following queries for polynomial times to* $\mathcal{C}$.
   - $\mathcal{A}$ *sends to* $\mathcal{C}$ *a pair* $(i, j) \in \mathcal{N} \times \mathcal{M}$ *such that* $(i, j) \notin i^* \times (\mathcal{S}^* \setminus T^*)$. *Then,* $\mathcal{C}$*, by running* AttGrant *with* mkey*, generates a private key* $skey_{i,j}$ *and sends it to* $\mathcal{A}$

   During the above queries $\mathcal{A}$ *asks* $\mathcal{C}$ *for the test ciphertext once.*
   - *Then,* $\mathcal{C}$ *first randomly choose* $b \in \{0, 1\}$ *and* $key_{1-b} \in \mathcal{K}$*. Next,* $\mathcal{C}$ *generates* (key_b^*, hdr^*) *by running* Encrypt *with the input of* $\mathcal{S}^*$ *and* t^**. Finally,* $\mathcal{C}$ *sends* $(key_0^*, key_1^*, hdr^*)$ *to* $\mathcal{A}$.
4. $\mathcal{A}$ *outputs a guess* $b' \in \{0, 1\}$.

Let $\mathsf{Adv}_{\mathcal{A}}(\mathsf{b})$ *be the probability that* $\mathcal{C}$ *outputs* b *in the above game.*

Definition 2. *A fuzzy identity-based encryption scheme is key-indistinguishable under fuzzy selective ID-and-attribute and chosen plaintext attacks if for all polynomial-time adversary* $\mathcal{A}$*,* $|\mathsf{Adv}_{\mathcal{A}}(0) - \mathsf{Adv}_{\mathcal{A}}(1)|$ *is negligible.*

If we consider a game where $\mathcal{S}^*, t^*, i^*$ are adaptively chosen and where $\mathcal{A}$ may ask decryption queries, then full and stronger notion of security is introduced. As usual, such a weaker notion is introduced since it is hard to prove that the proposed scheme satisfies the stronger notion. Except for artificial schemes, no concrete adaptive attacks are found for schemes that are only selectively secure.

3 Proposed Scheme

Our scheme uses cyclic groups with bilinear pairing.

Definition 3. *Let $\mathcal{G}$ and $\tilde{\mathcal{G}}$ denote two cyclic groups of prime order q. A* bilinear pairing *is an efficient mapping $e : \mathcal{G} \times \mathcal{G} \to \tilde{\mathcal{G}}$ such that $e(u^\alpha, v^\beta) = e(u, v)^{\alpha\beta}$ for all $u, v \in \mathcal{G}$ and $\alpha, \beta \in \mathbb{Z}/q\mathbb{Z}$ and there exists a generator g of $\mathcal{G}$ such that $e(g, g)$ generates $\tilde{\mathcal{G}}$. We will refer to such a tuple $(\mathcal{G}, \tilde{\mathcal{G}}, e, g)$ as a* bilinear pairing quadruple.

Let $E^j_\chi[f]$ denote the coefficient of χ^j-term in a polynomial f of χ. That is, $E^j_\chi[f] = \frac{1}{j!}\frac{\partial^j}{\partial \chi^j} f\Big|_{\chi=0}$ where $\frac{\partial}{\partial x}$ denote formal derivative[1].

Setup: Suppose $m, n, \kappa \in \mathbb{N}$ and a random tape are given as a security parameters. It first chooses a hash function $\mathcal{H}$, $params = (\mathcal{H}, m, n, (\mathcal{G}, \tilde{\mathcal{G}}, e, q))$ for $|q| = \kappa$. Then, it randomly chooses $\omega, \chi, \delta \in \mathbb{Z}/q\mathbb{Z}$. Let each of $(\theta_j)_{j\in\mathcal{M}} \in_R \mathbb{Z}/q\mathbb{Z}^m$ be the hash ($\mathcal{H}$) value of a string that describe the corresponding attribute, i.e., $\theta_j = \mathcal{H}(params, \text{description of } j\text{-th attribute})$. Next, it generates

$$G = e(g, g^\delta)\ , \quad y = g^\omega\ ,$$
$$(g_i)_{i\in\mathcal{M}} = (g^{\chi^i})_{i\in\mathcal{M}}\ , \quad (y_i)_{i\in\mathcal{M}} = (y^{\chi^i})_{i\in\mathcal{M}}\ , \quad (G_i)_{i\in\mathcal{M}} = (G^{\chi^i})_{i\in\mathcal{M}}$$

The public key and the master key are

$$pkey = (params, g, y, G, (g_i)_{i\in\mathcal{M}}, (y_i)_{i\in\mathcal{M}}, (\theta_j)_{j\in\mathcal{M}}, (G_i)_{i\in\mathcal{M}})$$
$$mkey = (\omega, \chi, \delta).$$

AttGrant: Suppose $pkey, mkey$, an identity $i \in \mathcal{N}$, an attribute index $j \in \mathcal{M}$, and a random tape are given. If no attribute keys for i is generated before, AttGrant randomly chooses $\phi_i \in \mathbb{Z}/q\mathbb{Z}$. If any of them is generated before, AttGrant uses the same ϕ_i as before. Then, AttGrant generates a private key $skey_{i,j}$ for an identity i related to the attribute index j as

$$skey_{i,j} := (d_i, d_{i,j}) = \left(g^{\phi_i}, g^{\frac{\omega\phi_i - \delta}{(\chi+\theta_j)}}\right)$$

and outputs it.

Encrypt: Suppose $pkey$, an attribute set $\mathcal{S} \subset \mathcal{M}$, a threshold t, and a random tape are given. It first randomly chooses $\rho \in \mathbb{Z}/q\mathbb{Z}$. We let $g_0 = g$ and i_j be j-th index in $\mathcal{S}$. Then, it generates key and hdr for $\mathcal{S}$ as

$$key := C = G^\rho$$
$$hdr := (\mathcal{S}, c_0, c_{1,0}, \ldots, c_{1,n-t}, C_{2,1}, \ldots, C_{2,n-t})$$
$$= \left(\mathcal{S}, \prod_{j=0,\ldots,n} g_j^{\rho E^j_\chi\left[\prod_{j|i_j\in\mathcal{S}}(\chi+\theta_{i_j})\right]}, (y_i{}^\rho)_{i=0,\ldots,n-t}, (G_i{}^\rho)_{i=1,\ldots,n-t}\right)$$

[1] For example, if $f(\chi) = \prod_{i=1}^3(\chi - \alpha_i) = \chi^3 - (\alpha_1+\alpha_2+\alpha_3)\chi^2 + (\alpha_1\alpha_2 + \alpha_2\alpha_3 + \alpha_3\alpha_1)\chi - \alpha_1\alpha_2\alpha_3$, $E^2_\chi[f] = -(\alpha_1+\alpha_2+\alpha_3)$.

Note that $c_0 = g^{\rho \prod_{j \in \mathcal{S}} (\chi + \theta_j)}$. Finally, it outputs hdr.
$(c_{1,0}, \ldots, c_{1,n-t}, C_{2,1}, \ldots, C_{2,n-t})$ in hdr corresponds to "reencrypted public key portion" of the original scheme [15].

Decrypt: Let $\mathcal{S}_i$ be set of all $j \in \mathcal{S}$ such that user i has $skey_{i,j}$. We assume without loosing generality $|\mathcal{S}_i| = t$. We also let $\bar{\mathcal{S}}_i = \mathcal{S} \setminus \mathcal{S}_i$.
Decrypt computes

$$d_{\mathcal{S}_i} = \prod_{j \in \mathcal{S}_i} \frac{d_{i,j}}{\prod_{k \in \mathcal{S}_i, k \neq j} (\theta_k - \theta_j)} \qquad \left(= g^{\frac{\omega \phi_i - \delta}{\prod_{j \in \mathcal{S}_i} (\chi + \theta_j)}} \right)$$

$$c_{\mathcal{S}_i} = \prod_{j=0}^{t} y_j^{E_\chi^j \left[\prod_{j \in \bar{\mathcal{S}}_i} (\chi + \theta_j) \right]} \qquad (= y^{\rho \prod_{j \in \bar{\mathcal{S}}_i} (\chi + \theta_j)})$$

$$G_{\mathcal{S}_i} = \prod_{j=1}^{n-t} C_j^{E_\chi^j \left[\prod_{j \in \bar{\mathcal{S}}_i} (\chi + \theta_j) \right]} \qquad (= G^{\rho (\prod_{j \in \bar{\mathcal{S}}_i} (\chi + \theta_j) - \prod_{j \in \bar{\mathcal{S}}} \theta_j)})$$

and computes

$$G' = \frac{e(d_i, c_{\mathcal{S}_i})}{e(c_0, d_{\mathcal{S}_i})} = \frac{e(g^{\phi_i}, g^{\omega \rho \prod_{j \in \bar{\mathcal{S}}_i} (\chi + \theta_j)})}{e(g^{\rho \prod_{j \in \mathcal{S}} (\chi + \theta_j)}, g^{\frac{\omega \phi_i - \delta}{\prod_{j \in \mathcal{S}_i} (\chi + \theta_j)}})}$$
$$= \frac{e(g^{\rho}, g^{\omega \phi_i \prod_{j \in \bar{\mathcal{S}}_i} (\chi + \theta_j)})}{e(g^{\rho \prod_{j \in \bar{\mathcal{S}}_i} (\chi + \theta_j)}, g^{\omega \phi_i - \delta})} = e(g, g^{\delta})^{\rho \prod_{j \in \bar{\mathcal{S}}_i} (\chi + \theta_j)}.$$

Then, it computes

$$key' = (G' / G_{\mathcal{S}_i})^{\frac{1}{\prod_{j \in \bar{\mathcal{S}}_i} \theta_j}} = G^{\rho}$$

It outputs key'.

The completeness of the scheme is clear from the above description.

4 Security Analysis

4.1 Preliminaries

We define a "General Bilinear Decision Problem" in bilinear groups, which is a straight forward generalization of "General Diffie-Hellman Exponent problem" of [3]. Then introduce the "General Bilinear Decision Assumption" which is proved to hold in the generic bilinear group model.

Definition 4. General Bilinear Decision Problem: *Let p be a prime and let s and m be two positive integer constants. Let $\mathcal{G}$ and $\tilde{\mathcal{G}}$ be order q cyclic groups with an efficient bilinear mapping $e : \mathcal{G} \times \mathcal{G} \to \tilde{\mathcal{G}}$ and g is a generator of $\mathcal{G}$ and $G = e(g, g)$. Let $\mathbf{P}, \mathbf{P}', \mathbf{Q}, \mathbf{Q}'$ be tuples of polynomials, where $\mathbf{P} = (p_1, \ldots, p_s), \mathbf{Q} = (q_1, \ldots, q_s), \mathbf{P}' = (p'_1, \ldots, p'_s), \mathbf{Q}' = (q'_1, \ldots, q'_s) \in \mathbb{F}_p[X_1, \ldots,$*

$X_m]^s$ *and* $p_1 = q_1 = p'_1 = q'_1 = 1$. *Let* $\mathbf{P}(x_1, \ldots, x_n)$ *denote* $(p_1(x_1, \ldots, x_m), \ldots, p_s(x_1, \ldots, x_m))$ *and* $g^{\mathbf{P}(x_1,\ldots,x_m)} = (g^{p_1(x_1,\ldots,x_m)}, \ldots, g^{p_s(x_1,\ldots,x_m)})$. *We use similar notation for* $\mathbf{Q}, \mathbf{P}', \mathbf{Q}'$.

We say an algorithm $\mathcal{B}$ has an advantage ϵ in solving general bilinear decision problem with respect to $(\mathbf{P}, \mathbf{Q})$ and $(\mathbf{P}', \mathbf{Q}')$ *if*

$$|\Pr[\mathcal{B}(g^{\mathbf{P}(x_1,\ldots,x_m)}, G^{\mathbf{Q}(x_1,\ldots,x_m)}) = 1] - \Pr[\mathcal{B}(g^{\mathbf{P}'(x_1,\ldots,x_m)}, G^{\mathbf{Q}'(x_1,\ldots,x_m)}) = 1]| > \epsilon$$

where the probability is taken over random choice of $x_0, \ldots, x_m \in_R \mathbb{Z}/q\mathbb{Z}$ *and random tapes of* $\mathcal{B}$.

Definition 5. Dependent and Independent Polynomials: *We say a pair of tuples* $(\mathbf{P}, \mathbf{Q})$ *and a pair of tuples* $(\mathbf{P}', \mathbf{Q}')$ *are* dependent *if there exists tuple of* $s^2 + s$ *constants*

$\{a_{ij}\}_{i=1,\ldots,s,j=1,\ldots,s}$, $\{b_i\}_{i=1,\ldots,s}$ *such that either*

$$0 \equiv \sum_{i,j=1}^{s} a_{ij} p_i p_j + \sum_{i=1}^{s} b_i q_i \wedge 0 \not\equiv \sum_{i,j=1}^{s} a_{ij} p'_i p'_j + \sum_{i=1}^{s} b_i q'_i$$

or

$$0 \not\equiv \sum_{i,j=1}^{s} a_{ij} p_i p_j + \sum_{i=1}^{s} b_i q_i \wedge 0 \equiv \sum_{i,j=1}^{s} a_{ij} p'_i p'_j + \sum_{i=1}^{s} b_i q'_i$$

holds. We let $(\mathbf{P}, \mathbf{Q}) \not\sim (\mathbf{P}', \mathbf{Q}')$ *denote this.*

We say that a general bilinear decision problem with respect to $(\mathbf{P}, \mathbf{Q})$ *and* $(\mathbf{P}', \mathbf{Q}')$ *is* independent *if* $(\mathbf{P}, \mathbf{Q})$ *and* $(\mathbf{P}', \mathbf{Q}')$ *are not dependent. We let* $(\mathbf{P}, \mathbf{Q}) \sim (\mathbf{P}', \mathbf{Q}')$ *denote this.*

The general Diffie-Hellman Exponent problem in [3] is a special case of the above problem when each of $\{p_i = p'_i\}_{i=1,\ldots,s}$ is a polynomial of $x_1, \ldots, x_{m-1}$, $\mathbf{Q} = (1, f(x_1, \ldots, x_{m-1}))$, and $\mathbf{Q}' = (1, x_m)$.

Definition 6. The General Bilinear Decision Assumption: *We say that the* general bilinear decision assumption *holds, if for every poly-time adversary, the advantage* ϵ *in solving every general bilinear decision problem with respect to every sets of* $(\mathbf{P}, \mathbf{Q})$ *and* $(\mathbf{P}', \mathbf{Q}')$ *that are independent is negligible.*

The general bilinear decision assumption is justified by the following Theorem 1. That is, its difficulty is proved in the the generic bilinear group model introduced below. This is an extension of the generic group model [14] for ordinary bilinear groups of prime order defined in [2].

Definition 7. The generic bilinear group model: *Let us consider the case the bilinear groups of prime order* q *are* $\mathcal{G}$ *and* $\tilde{\mathcal{G}}$ *and* g *is a generator of* $\mathcal{G}$. *In this model, elements of* $\mathcal{G}$ *and* $\tilde{\mathcal{G}}$ *appear to be encoded as unique random strings, so*

that no property other than equality can be directly tested by the adversary. There exist two oracles in this model. Those are, oracles that perform group operations in each $\mathcal{G}$ and $\tilde{\mathcal{G}}$ and an oracle that performs paring e. The opaque encoding of an element in $\mathcal{G}$ is modeled as an injective random function $\psi : \mathbb{Z}/q\mathbb{Z} \to \Sigma \subset \{0,1\}^*$, which maps all $\alpha \in \mathbb{Z}/q\mathbb{Z}$ to the string representation $\psi(\alpha)$ of $g^\alpha \in \mathcal{G}$. We similarly define $\tilde{\psi} : \mathbb{Z}/q\mathbb{Z} \to \tilde{\Sigma}$ for $\tilde{\mathcal{G}}$. The attacker communicates with the oracles using the $(\psi, \tilde{\psi})$-representations of the group elements only.

Theorem 1. *Let $d_P, d_{P'}, d_Q$, and $d_{Q'}$ be, respectively, the maximum degree of polynomials in $\mathbf{P}, \mathbf{P}', \mathbf{Q}$, and $\mathbf{Q}'$ and let*

$d = \max(2d_P, 2d_{P'}, d_Q, d_{Q'})$. *In the generic bilinear group model, no algorithm $\mathcal{A}$ that makes a total of at most q_g queries to the oracles computing group operations in $\mathcal{G}$, $\tilde{\mathcal{G}}$, and $e : \mathcal{G} \times \mathcal{G} \to \tilde{\mathcal{G}}$ has an advantage ϵ in solving any of general bilinear decision problem with respect to $(\mathbf{P}, \mathbf{Q})$ and $(\mathbf{P}', \mathbf{Q}')$ which are independent. Where,*

$$\epsilon = \frac{(q_g + 2s)^2 d}{p}.$$

The proof of the theorem is given in Appendix A. The above Theorem 1 implies that the general bilinear decision assumption holds in the generic bilinear group model.

4.2 Security of the Proposed Scheme

Theorem 2. *Suppose that the general bilinear decision assumption holds. Then, the proposed fuzzy identity-based encryption scheme is key-indistinguishable under fuzzy selective ID-and-attribute and chosen plaintext attacks in the random oracle model.*

Proof. We first describe two distributions $(\mathbf{P}, \mathbf{Q})$ and $(\mathbf{P}', \mathbf{Q}')$. One is the same as the distribution of values that $\mathcal{A}$ receives from $\mathcal{C}$ in the key-distinguishing game when $b = 0$ and the other is the same as the one that $\mathcal{A}$ receives from $\mathcal{C}$ in the key-distinguishing game when $b = 1$.

Let Ω be the set of all pair of $(i, j) \in \mathcal{N} \times \mathcal{M}$ that $\mathcal{A}$ asks the challenger for $skey_{i,j}$. Suppose that $\tilde{g}$ is a generator of $\mathcal{G}$. Let ω, χ, δ be random variables and $(\theta_j)_{j \in \mathcal{M}}$ be distinct hash values. Let

$$C = e(\tilde{g}, \tilde{g})^{\rho\delta \prod_{j \in \mathcal{M}} (\chi + \theta_j)^2} \ , \ G = e(\tilde{g}, \tilde{g})^{\delta \prod_{j \in \mathcal{M}} (\chi + \theta_j)^2}$$

$$(C_{2,i})_{i=1,\ldots,n-t} = (e(\tilde{g}, \tilde{g})^{\chi^i \rho\delta \prod_{j \in \mathcal{M}} (\chi + \theta_j)^2})_{i=1,\ldots,n-t}$$

$$(G_i)_{i \in \mathcal{M}} = (e(\tilde{g}, \tilde{g})^{\delta \chi^i \prod_{j \in \mathcal{M}} (\chi + \theta_j)^2})_{i \in \mathcal{M}}$$

$$g = \tilde{g}^{\prod_{j \in \mathcal{M}} (\chi + \theta_j)} \ , \ y = \tilde{g}^{\omega \prod_{j \in \mathcal{M}} (\chi + \theta_j)} \ , \ c_0 = \tilde{g}^{\rho \prod_{j \in \mathcal{S}} (\chi + \theta_j) \prod_{j \in \mathcal{M}} (\chi + \theta_j)}$$

$$(g_i)_{i \in \mathcal{M}} = (\tilde{g}^{\chi^i \prod_{j \in \mathcal{M}} (\chi + \theta_j)})_{i \in \mathcal{M}} \ , \ (y_i)_{i \in \mathcal{M}} = (\tilde{g}^{\omega \chi^i \prod_{j \in \mathcal{M}} (\chi + \theta_j)})_{i \in \mathcal{M}}$$

$$(d_i, d_{i,j})_{(i,j) \in \Omega} = \ ((\tilde{g}^{\phi_i \prod_{j \in \mathcal{M}} (\chi + \theta_j)}, \tilde{g}^{(\omega\phi_i - \delta) \prod_{k \in \mathcal{M}, k \neq j} (\chi + \theta_k)}))_{(i,j) \in \Omega}$$

$$(c_{1,i})_{i=0,\ldots,n-t} = (\tilde{g}^{\chi^i \rho\omega \prod_{j \in \mathcal{M}} (\chi + \theta_j)})_{i=0,\ldots,n-t}.$$

Let $\mathbf{P} = \left(g, y, c_0, (g_i)_{i\in\mathcal{M}}, (y_i)_{i\in\mathcal{M}}, (d_i, d_{i,j})_{(i,j)\in\Omega}, (c_{1,i})_{i=0,\ldots,n-t}\right)$ and $\mathbf{Q} = (C, G, (C_{2,i})_{i=1,\ldots,n-t}, (G_i)_{i\in\mathcal{M}})$. Let $(\mathbf{P}', \mathbf{Q}')$ be the same as $(\mathbf{P}, \mathbf{Q})$ except C is replaced with random elements in $\tilde{\mathcal{G}}$. Then, if we consider (ω, χ, δ) are random variables, the distribution of $(\mathbf{P}, \mathbf{Q})$ is exactly the same as that $\mathcal{A}$ receives from $\mathcal{C}$ in the key-distinguishing game when $b = 0$ and the distribution of $(\mathbf{P}', \mathbf{Q}')$ is exactly the same as that $\mathcal{A}$ receives from $\mathcal{C}$ in the key-distinguishing game when $b = 1$.

One can construct an algorithm B that can break the general bilinear decision assumption as follows. First, B randomly chooses size of $\mathcal{S}^*$ and t^*. Then, B get a polynomial that is either $(\mathbf{P}, \mathbf{Q})$ or $(\mathbf{P}', \mathbf{Q}')$ that is consistent with the $|\mathcal{S}^*|$ and t^* as a general bilinear decision problem. We note that this polynomial, an instance of the problem, specifies all ϕ_i, θ_j, and Ω.

Next, B play the role of $\mathcal{C}$ and let $\mathcal{A}$ choose $\mathcal{S}^*, T^*$, and i^*. If the size of $\mathcal{S}^*$ and t^* are different from what B had expected, it aborts. If they are as expected, B chooses random oracles (i.e., determines the correspondence between $i \in \mathcal{M}$ and θ_i and the correspondence between $i^* \in \mathcal{N}$ and ϕ_i) so that Ω fits to $\mathcal{S}^*$ and T^* that $\mathcal{A}$ has chosen. Then, $\mathcal{B}$ can play the role of $\mathcal{C}$ by using the given polynomial.

Now, if $\mathcal{A}$ is able to distinguish between the distribution for $b = 0$ and that for $b = 1$, B can distinguish whether the given polynomial is $(\mathbf{P}, \mathbf{Q})$ or $(\mathbf{P}', \mathbf{Q}')$.

Therefore, what was left to prove is that $(\mathbf{P}, \mathbf{Q})$ and $(\mathbf{P}', \mathbf{Q}')$ are independent. This is proven in Appendix B.

5 Comparison

Let n be the number of attributes by which messages are encrypted. Let t be an error tolerance parameter such that at least t out of n attributes need to coincide for the receiver to decrypt ciphertexts. Let d be the number of attributes the receiver is granted.

In our scheme, each ciphertext consists of $2(n - t + 1)$ elements and each private key consists of $d+1$ elements. In the first scheme of [16], each ciphertext consists of $n+2$ elements and each private consists of d elements. In the second scheme of [16], each ciphertext consists of $n + 3$ elements and each private key consists of $2d$ elements. In our scheme, t can be chosen for each ciphertext. But in both schemes in [16], t is a fixed parameter. The comparison is given in the Table. 1. This shows that ciphertexts of our scheme is shorter than that of

Table 1. Comparison with the previous works

	ciphertext length	private key length	threshold
Our Scheme	$2(n - t + 1)$	$d + 1$	flexible
1st scheme in [16]	$n + 2$	d	fixed
2nd scheme in [16]	$n + 3$	$2d$	fixed

previous scheme when $t < n/2$, which we believe is the most of the natural cases. Besides these efficiency, our scheme provides flexible tolerance parameter and dynamical grant of attributes.

References

1. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-Policy Attribute-Based Encryption. In: IEEE Symposium on Security and Privacy 2007, pp. 321–334 (2007)
2. Boneh, D., Boyen, X.: Short Signatures Without Random Oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004)
3. Boneh, D., Boyen, X., Goh, E.-J.: Hierarchical Identity Based Encryption with Constant Size Ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005)
4. Boneh, D., Franklin, M.K.: Identity-Based Encryption from the Weil Pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
5. Cheung, L., Newport, C.C.: Provably secure ciphertext policy ABE. In: ACM Conference on Computer and Communications Security 2007, pp. 456–465 (2007)
6. Canetti, R., Halevi, S., Katz, J.: A Forward-Secure Public-Key Encryption Scheme. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 255–271. Springer, Heidelberg (2003)
7. Chase, M.: Multi-authority Attribute Based Encryption. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 515–534. Springer, Heidelberg (2007)
8. Goyal, V., Jain, A., Pandey, O., Sahai, A.: Bounded Ciphertext Policy Attribute Based Encryption. ICALP (2), 579–591 (2008)
9. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: ACM Conference on Computer and Communications Security, pp. 89–98 (2006)
10. Ostrovsky, R., Sahai, A., Waters, B.: Attribute-based encryption with non-monotonic access structures. In: ACM Conference on Computer and Communications Security 2007, pp. 195–203 (2007)
11. Pirretti, M., Traynor, P., McDaniel, P., Waters, B.: Secure attribute-based systems. In: ACM Conference on Computer and Communications Security 2006, pp. 99–112 (2006)
12. Sakai, R., Furukawa, J.: Identity-Based Broadcast Encryption. Cryptographic eprint archive 2007/217
13. Delerablée, C.: Identity-Based Broadcast Encryption with Constant Size Ciphertexts and Private Keys. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833. Springer, Heidelberg (2007)
14. Shoup, V.: Lower Bounds for Discrete Logarithms and Related Problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997)
15. Sakai, R., Kasahara, M.: New Efficient Broadcast Encryption. In: SCIS 2007, 3C3-1 (2007)
16. Sahai, A., Waters, B.: Fuzzy Identity-Based Encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005)

A Proof of Theorem 1

Proof. The proof is similar to that of Theorem A.2 in [3] and no novel technique is introduced here.

Consider an algorithm $\mathcal{B}$ that plays the following game with $\mathcal{A}$. Algorithm $\mathcal{B}$ maintains two lists of pairs, $L_P = \{(p_i, \psi_{P,i}) : i = 1, \ldots, \tau_P\}, L_Q = \{(q_i, \psi_{Q,i}) : i = 1, \ldots, \tau_Q\}$ under the invariant that at step τ in the game, $\tau_P + \tau_Q = \tau + 2s$. Here, $p_i \in \mathbb{F}_p[X_1, \ldots, X_m]$ and $q_i \in \mathbb{F}_p[X_1, \ldots, X_m]$ are multi-variate polynomials. The $\psi_{P,i}$ and $\psi_{Q,i}$ are strings in $\{0,1\}^k$. The lists are initialized at step $\tau = 0$ by initializing $\tau_P = \tau_Q = s$. $\mathcal{B}$ chooses $b \in \{0,1\}$ at the beginning of the game. We set $p_1, \ldots, p_s$ in L_P and $q_1, \ldots, q_s$ in L_Q to be, respectively, the polynomials in $\mathbf{P}$ and $\mathbf{Q}$ if $b = 0$. We set them to be polynomials in $\mathbf{P}'$ and $\mathbf{Q}'$ if $b = 1$. Algorithm $\mathcal{B}$ completes the preparation of the lists L_P and L_Q by setting the ψ-strings associated with distinct polynomials to random strings in $\{0,1\}^k$.

We can assume that $\mathcal{A}$ makes oracle queries only on strings obtained from $\mathcal{B}$, since $\mathcal{B}$ can make the strings in $\mathcal{G}$ and $\mathcal{G}_T$ arbitrarily hard to guess by increasing k.

We note that $\mathcal{B}$ can easily determine the index i of any given string $\psi_{P,i} \in L_P$ and $\psi_{Q,i} \in L_Q$. $\mathcal{B}$ starts the game by providing $\mathcal{A}$ with the value of p and a tuple of strings $\{\psi_{P,i}\}_{i=1,\ldots,s}, \{\psi_{Q,i}\}_{i=1,\ldots,s}$ meant to encode some tuple in $\mathcal{G}^s \times \mathcal{G}_T^s$. Algorithm $\mathcal{B}$ responds to $\mathcal{A}$'s oracle queries as follows.

Group Operation in $\mathcal{G}, \mathcal{G}_T$. A query in $\mathcal{G}$ consists of two operands $\psi_{P,i}$ and $\psi_{P,j}$ with $1 \le i, j \le \tau_P$ and a selection bit indicating whether to multiply or divide the group elements. To answer, let $\tau'_P \leftarrow \tau_P + 1$. Perform the polynomial addition or subtraction $p_{\tau'_P} = p_i \pm p_j$ depending on whether multiplication or division is requested. If the result $p_{\tau'_P} = p_l$ for some $l \le \tau_P$, then set $\psi_{P,\tau'_P} = \psi_{P,l}$; otherwise, set ψ_{P,τ'_P} to a new random string in $\{0,1\}^k \setminus \{\psi_{P,1}, \ldots, \psi_{P,\tau_P}\}$. Insert the pair $(p_{\tau'_P}, \psi_{P,\tau'_P})$ into the list L_P and update the counter $\tau_P \leftarrow \tau'_P$. Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with the string ψ_{P,τ_P}. $\tilde{\mathcal{G}}$ queries are handled analogously, this time by working with the list L_Q and the counter τ_Q.

Bilinear Pairing. A query of this type consists of two operands $\psi_{P,i}$ and $\psi_{P,j}$ with $1 \le i, j \le \tau_P$. To answer, let $\tau'_Q \leftarrow \tau_Q + 1$. Perform the polynomial multiplication $q_{\tau'_Q} = p_i \cdot p_j$. If the result $q_{\tau'_Q} = q_l$ for some $l \le \tau_Q$, then set $\psi_{Q,\tau'_Q} = \psi_{Q,l}$; otherwise, set ψ_{Q,τ'_Q} to a new random string in $\{0,1\}^k \setminus \{\psi_{Q,1}, \ldots, \psi_{Q,\tau_Q}\}$. Insert the pair $(q_{\tau'_Q}, \psi_{Q,\tau'_Q})$ into the list L_Q and update the counter $\tau_Q \leftarrow \tau'_Q$. Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with the string ψ_{Q,τ_Q}.

After at most q_g queries, $\mathcal{A}$ terminates and returns a guess $b' \in \{0,1\}$. At this point $\mathcal{B}$ chooses random $\{x_1, \ldots, x_m\}$. For $i = 1, \ldots, m$, we set $X_i = x_i$. It follows that the simulation provided by $\mathcal{B}$ is perfect unless the chosen random values for the variables $X_1, \ldots, X_m$ result in an equality relation between intermediate values that is not an equality of polynomials. In other words, the simulation is perfect unless for some i, j one of the following holds:

1. $p_i(x_1, \ldots, x_m) - p_j(x_1, \ldots, x_m) = 0$, yet the polynomials p_i and p_j are not equal.
2. $q_i(x_1, \ldots, x_m) - q_j(x_1, \ldots, x_m) = 0$, yet the polynomials q_i and q_j are not equal.

Let fail be the event that one of these two conditions holds. When event fail occurs, then $\mathcal{B}$'s responses to $\mathcal{A}$'s queries deviate from the real oracle's responses when the input tuple is derived from the vector $(x_1, \ldots, x_m) \in \mathbb{F}_p^m$.

We first bound the probability that event fail occurs. We need to bound the probability that for some i, j we get $(p_i - p_j)(x_1, \ldots, x_m) = 0$ even though $p_i - p_j \neq 0$ or that $(q_i - q_j)(x_1, \ldots, x_m) = 0$ even though $q_i - q_j \neq 0$. By construction, the maximum total degree of these polynomials is at most $d = \max(2d_P, 2d_{P'}, d_Q, d_{Q'})$. Therefore, for a given i, j the probability that a random assignment to $X_1, \ldots, X_n$ is a root of $q_i - q_j$ is at most d/p. The same holds for $p_i - p_j$. Since there are no more than $2(\frac{q+2s}{2})$ such pairs (p_i, p_j) and (q_i, q_j) in total, we have that

$$Pr[\mathsf{fail}] \leq \binom{q+2s}{2} \frac{2d}{p} \leq (q+2s)^2 d/p.$$

If event fail does not occur, then $\mathcal{B}$'s simulation is perfect.

Since $(\mathbf{P}, \mathbf{Q})$ and $(\mathbf{P}', \mathbf{Q}')$ are independent, the distributions $\mathcal{A}$ is given are exactly the same unless fail happens. Therefore, the theorem is proved.

B Proof of Independence (Sketch)

Lemma 1. $\mathbf{P}, \mathbf{Q}, \mathbf{P}', \mathbf{Q}'$ *defined in the proof of Theorem 2, are independent.*

Proof. We let Ω' be $\{i | \exists j, (i,j) \in \Omega\}$. If $(\mathbf{P}, \mathbf{Q})$ and $(\mathbf{P}', \mathbf{Q}')$ are dependent, then there exist

$$\begin{gathered}
a_1, a_{2,i}, a_{3,i}, \\
b_{1,1}, b_{1,2}, b_{1,3}, b_{1,4,i}, b_{1,5,i}, b_{1,6,i}, b_{1,7,i,j}, b_{1,8,i}\ ,\ b_{2,2}, b_{2,3}, b_{2,4,i}, b_{2,5,i}, b_{2,6,i}, b_{2,7,i,j}, b_{2,8,i}, \\
b_{3,3}, b_{3,4,i}, b_{3,5,i}, b_{3,6,i}, b_{3,7,i,j}, b_{3,8,i}\ ,\ b_{4,4,i}, b_{4,5,i}, b_{4,6,i}, b_{4,7,i,j}, b_{4,8,i} \\
b_{5,5,i}, b_{5,6,i}, b_{5,7,i,j}, b_{5,8,i}\ ,\ b_{6,6,i}, b_{6,7,i,j}, b_{6,8,i}, \\
b_{7,7,i,j}, b_{7,8,i}\ ,\ b_{8,8,i}
\end{gathered}$$

for appropriate i, j such that the following equation (*abbreviated*) holds.

$$\begin{aligned}
&\rho\delta \prod_{k \in \mathcal{M}} (\chi + \theta_k)^2 + a_1 \delta \prod_{k \in \mathcal{M}} (\chi + \theta_k)^2 + \sum_{i=1}^{n-t^*} a_{2,i} \chi^i \rho\delta \prod_{k \in \mathcal{M}} (\chi + \theta_k)^2 + \ldots \\
&= b_{1,1} \prod_{k \in \mathcal{M}} (\chi + \theta_k) \cdot \prod_{k \in \mathcal{M}} (\chi + \theta_k) + b_{1,2} \prod_{k \in \mathcal{M}} (\chi + \theta_k) \cdot \omega \prod_{k \in \mathcal{M}} (\chi + \theta_k) + \ldots
\end{aligned}$$

Note that the coefficient of the first term is fixed to 1. We show that these however do not exist. Instead of checking total of this lengthy equation, we see only terms that contain ρ. So, we do not see the detail of the above equation.

For the terms that contain ρ, the following equation (*abbreviated*) should holds. Here we divided all terms by $\prod_{k\in\mathcal{M}}(\chi+\theta_k)$.

$$\rho\delta\prod_{k\in\mathcal{M}}(\chi+\theta_k)^+\sum_{i=1}^{n-t^*}a_{2,i}\chi^i\rho\delta\prod_{k\in\mathcal{M}}(\chi+\theta_k)$$
$$=b_{1,3}\cdot\rho\prod_{k\in\mathcal{S}^*}(\chi+\theta_k)\prod_{k\in\mathcal{M}}(\chi+\theta_k)+\sum_{i\in\mathcal{M}}b_{1,4,i}\cdot\chi^i\prod_{k\in\mathcal{M}}(\chi+\theta_k)+\ldots$$

We now see terms that contain only $\rho\omega\chi^j\phi_i$ for some i,j. That is,

$$0=\sum_{(i,j)\in\Omega}b_{3,7,i,j}\rho\prod_{k\in\mathcal{S}^*}(\chi+\theta_k)\cdot\omega\phi_i\prod_{k\in\mathcal{M},k\neq j}(\chi+\theta_k)$$
$$+\sum_{i\in\Omega'}\sum_{j=0}^{n-t^*}b_{6,8,i,j}\phi_i\cdot\chi^j\rho\omega\prod_{k\in\mathcal{M}}(\chi+\theta_k)$$

Inserting $-\theta_m\in\mathcal{M}$ to χ, we get

$$0=\sum_{(i,m)\in\Omega}b_{3,7,i,m}\rho\prod_{k\in\mathcal{S}^*}(\theta_k-\theta_m)\cdot\omega\phi_i\prod_{k\in\mathcal{M},k\neq m}(\theta_k-\theta_m)$$

Hence,

Fact 1: $b_{3,7,i,j}=0$ for (i,j) such that $j\notin\mathcal{S}^*$.

Thus, for terms that contain only $\rho\omega\chi^j\phi_i$, we have the following for some i,j:

$$0=\rho\omega\sum_{i\in\Omega'}\phi_i\left(\sum_{j\in\Omega(i)\cap\mathcal{S}^*}b_{3,7,i,j}\prod_{k\in\mathcal{S}^*,k\neq j}(\chi+\theta_k)+\sum_{j=0}^{n-t^*}b_{6,8,i,j}\cdot\chi^j\right)$$

Here $\Omega(i)=\{j|(i,j)\in\Omega\}$. Therefore, for every $i\in\Omega'$

$$0=\sum_{j\in\Omega(i)\cap\mathcal{S}^*}b_{3,7,i,j}\prod_{k\in\mathcal{S}^*,k\neq j}(\chi+\theta_k)+\sum_{j=0}^{n-t^*}b_{6,8,i,j}\cdot\chi^j \tag{1}$$

should hold.

We will see the condition for Eq. (1) to hold. Since $|\Omega(i)\cap\mathcal{S}^*=T^*|=t-1$ and consider a matrix $(t-1)\times(t-1)$ matrix $A_{\ell,k}$ such that

$$\sum_{j\in\Omega(i)\cap\mathcal{S}^*}b_{3,7,i,j}+\sum_{j=0}^{n-t^*}b_{6,8,i,j}\chi^j\prod_{k\in\mathcal{S}^*,k\neq j}(\chi+\theta_k)\equiv\sum_{j\in\Omega(i)\cap\mathcal{S}^*}\sum_{k=1}^{t-1}b_{3,7,i,j}A_{\ell_j,k}\chi^{n-k}.$$

Here, ℓ_j's are indices such that $\{\ell_j\}_j=\{1,\ldots,t-1\}$. Since the determinant of $(A_{\ell_j,k})$ is $\prod_{i,j\in\Omega(i)\cap\mathcal{S}^*,i<j}(\theta_i-\theta_j)$ and $\theta_i\neq\theta_j$ if $i\neq j$, the matrix $(A_{\ell_j,k})$ is regular. Hence, Eq. (1) holds only when the following fact 2 holds:

Fact 2: $b_{3,7,i,j} = 0$ for all (i, j) such that $i \in \Omega'$ and $j \in \Omega(i) \cap \mathcal{S}^*$.

We now see terms that contain only $\rho\delta$. Then, the following equation should holds:

$$\rho\delta \prod_{k \in \mathcal{M}} \theta_k = -\rho\delta \sum_{i \in \Omega'} \left(\sum_{j \in \Omega(i) \cap \mathcal{S}^*} b_{3,7,i,j} + \sum_{j \in \Omega(i), j \notin \mathcal{S}^*} b_{3,7,i,j} \right) \prod_{k \in \mathcal{S}^*} \theta_k \prod_{k \in \mathcal{M}, k \neq j} \theta_k$$

From Fact 1 and Fact 2, the terms in right hand side are 0. This contradicts to non-zero of the left hand side.

Type-Based Proxy Re-encryption and Its Construction

Qiang Tang

Faculty of EWI, University of Twente, the Netherlands
q.tang@utwente.nl

Abstract. Recently, the concept of proxy re-encryption has been shown very useful in a number of applications, especially in enforcing access control policies. In existing proxy re-encryption schemes, the delegatee can decrypt all ciphertexts for the delegator after re-encryption by the proxy. Consequently, in order to implement fine-grained access control policies, the delegator needs to either use multiple key pairs or trust the proxy to behave honestly. In this paper, we extend this concept and propose type-based proxy re-encryption, which enables the delegator to selectively delegate his decryption right to the delegatee while only needs one key pair. As a result, type-based proxy re-encryption enables the delegator to implement fine-grained policies with one key pair without any additional trust on the proxy. We provide a security model for our concept and provide formal definitions for semantic security and ciphertext privacy which is a valuable attribute in privacy-sensitive contexts. We propose two type-based proxy re-encryption schemes: one is CPA secure with ciphertext privacy while the other is CCA secure without ciphertext privacy.

1 Introduction

In a proxy re-encryption scheme [5,15], a delegator (say, Alice) and a delegatee (say, Bob) generate a proxy key that allows a semi-trusted third party (say, the proxy) to convert ciphertexts encrypted under Alice's public key into ciphertexts which can be decrypted by Bob. Recently, proxy re-encryption has been shown very useful in a number of applications such as access control in file storage [1], email forwarding [19], and law enforcement [13].

Motivation. We have the following observations on the existing proxy re-encryption schemes and their applications. One is that, with respect to one key pair of the delegator, the proxy is able to re-encrypt all the ciphertexts so that the delegatee can obtain all the plaintexts. The other is that, in many applications, it is likely that the delegator only wishes a specific delegatee to see a subset of his messages. In order to implement fine-grained access control policies, the delegator can choose a different key pair for each possible subset of his messages and choose a proxy to delegate his decryption right. However, this approach is infeasible in practice. Alternatively, the delegator can choose to

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 130–144, 2008.

trust the proxy to enforce his policies by re-encrypting the pre-defined subset of his ciphertexts to the specific delegatee. This approach is also infeasible because of the strong trust requirement on the proxy. For example, if the proxy colludes with a malicious delegatee (or, the proxy is compromised), then all messages of the delegator will be compromised.

In fact, the delegator might have more concerns about the delegation in practical applications. An observation is that, with a public key encryption scheme, if Alice encrypts a message for Bob using his public key, then Alice can stay anonymous. However, with a proxy re-encryption scheme, this kind of anonymity might not be trivially achieved. For example, in the schemes in [11], Bob can (at least) tell whether messages are from the same user or not.

Contribution. In this paper, we propose the concept of type-based proxy re-encryption which enables the delegator to selectively delegate his decryption right to delegatees. In a type-based proxy re-encryption scheme, the delegator categorizes his messages (ciphertexts) into different subsets and is capable of delegating the decryption right of each subset to a specific delegatee. The ciphertexts for the delegator are generated based on the delegator's public key and the message type which is used to identify the message subset. This new primitive has (at least) the following promising properties. One is that the delegator only needs one key pair so that the key management problem is simplified. The other is that the delegator can choose a particular proxy for a specific delegatee, which might be based on the sensitiveness of the delegation. Compromise of one proxy key will only affect one subset of messages.

We provide a security model for our concept and provide formal definitions for semantic security and ciphertext privacy which is a valuable attribute in privacy-sensitive contexts. If a type-based proxy re-encryption scheme achieves ciphertext privacy property then all re-encrypted ciphertexts are indistinguishable from normal ciphertexts originally generated for the delegatee, therefore, message senders (the delegators) remain anonymous. We propose two type-based proxy re-encryption schemes. The first scheme achieves ciphertext privacy and is IND-PR-CPA secure based on the XDH and the Co-BDH assumptions in the random oracle model. The second scheme is IND-PR-CCA secure based on the BDH and the KE assumptions in the random oracle model, but it does not achieve ciphertext privacy.

Related Work. Mambo and Okamoto [15] firstly propose the concept of delegation of decryption right in the context of speeding up decryption operations. Blaze, Bleumer and Strauss [5] introduce the concept of atomic proxy cryptography which is proxy re-encryption. They present an Elgamal [10] based proxy re-encryption scheme, in which the proxy is also capable of converting ciphertexts encrypted for Bob into ciphertexts which can be decrypted by Alice. Jakobsson [14] and Zhou *et al.* [20] simultaneously propose quorum-based protocols, which divide the proxy into many components. Dodis and Ivan [13] propose generic constructions of proxy re-encryption schemes by using double-encryption. Ateniese *et al.* [1] propose an Elgamal based scheme and show its application

in securing file systems. In addition, Ateniese *et al.* also point out a number of desirable properties for proxy re-encryption schemes. Note that these papers are mainly focused on the traditional public-key encryption schemes. Apart from the generic construction of Dodis and Ivan [13], there are two identity-based proxy re-encryption schemes: one is proposed by Green and Ateniese [11] and the other is proposed by Matsuo [16]. In both schemes, the delegator and the delegatee are assumed to be registered at the same domain (or, the same key generation center).

Organization. The rest of the paper is organized as follows. In Section 2 we provide some preliminary knowledge. In Section 3 we present the security model for type-based proxy re-encryption. In Section 4 we present the IND-PR-CPA secure type-based proxy re-encryption scheme with ciphertext privacy and prove its security. In Section 5 we present the IND-PR-CCA secure type-based proxy re-encryption scheme without ciphertext privacy and prove its security. In Section 6 we conclude the paper.

2 Preliminaries

If x is chosen uniformly at random from the set Y, then we write $x \in_R Y$. The symbol $\perp$ denotes an error message. Our security analysis is done in the random oracle model [4]. Next, we review the necessary knowledge about pairing and the related assumptions, and then review the Public Key Encryption (PKE).

2.1 Review of Pairing

More detailed information can be found in [6,12]. Generally, a pairing (or, bilinear map) satisfies the following properties:

1. $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are three multiplicative groups of prime order p;
2. g_1 is a generator of $\mathbb{G}_1$ and g_2 is a generator of $\mathbb{G}_2$;
3. $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is an efficiently-computable bilinear map with the following properties:
   - Bilinear: for all $a, b \in \mathbb{Z}_p$, we have $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab}$.
   - Non-degenerate: $\hat{e}(g_1, g_2) \neq 1$.

The Co-Bilinear Diffie-Hellman (Co-BDH) problem is as follows: given $g_1, g_1^a, g_1^b \in \mathbb{G}_1, g_2, g_2^c \in \mathbb{G}_2$ as input, output $\hat{e}(g_1, g_2)^{abc} \in \mathbb{G}_T$. An algorithm $\mathcal{A}$ has advantage ϵ in solving Co-BDH in $\mathbb{G}$ if

$$\Pr[\mathcal{A}(g_1, g_2, g_1^a, g_1^b, g_2^c) = \hat{e}(g_1, g_2)^{abc}] \geq \epsilon,$$

where the probability is over the random choice of $a, b, c \in \mathbb{Z}_p$, and the random bits of $\mathcal{A}$.

Definition 1. *We say that the Co-BDH assumption holds if any polynomial-time adversary has only a negligible advantage ϵ in solving the Co-BDH problem.*

Note that the Co-BDH assumption is closely related to the Asymmetric Bilinear Diffie-Hellman (ABDH) assumption defined in [7].

The eXternal Diffie-Hellman (XDH) assumption is also widely used in the literature (e.g. [8]). We say that an algorithm $\mathcal{A}$ has advantage ϵ in solving the XDH problem if

$$|\Pr[\mathcal{A}(g_2, g_2^a, g_2^b, g_2^{a \cdot b}) = 0] - \Pr[\mathcal{A}(g_2, g_2^a, g_2^b, g_2^c) = 0]| \geq \epsilon,$$

where the probability is over the random choice of $a, b, c \in \mathbb{Z}_p$, and the random bits of $\mathcal{A}$.

Definition 2. *We say that the XDH assumption holds if any polynomial-time adversary has only a negligible advantage ϵ in solving the XDH problem.*

Instead of the above general setting, the simplified setting is also widely used (e.g. [6]). Here, a pairing (or, bilinear map) satisfies the following properties:

1. $\mathbb{G}$ and $\mathbb{G}_T$ are two multiplicative groups of prime order p;
2. g is a generator of $\mathbb{G}$;
3. $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is an efficiently-computable bilinear map with the following properties:
   - Bilinear: for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, we have $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$.
   - Non-degenerate: $\hat{e}(g, g) \neq 1$.

The Bilinear Diffie-Hellman (BDH) problem is as follows: given a tuple $g, g^a, g^b, g^c \in \mathbb{G}$ as input, output $\hat{e}(g, g)^{abc} \in \mathbb{G}$. An algorithm $\mathcal{A}$ has advantage ϵ in solving BDH if

$$\Pr[\mathcal{A}(g, g^a, g^b, g^c) = \hat{e}(g, g)^{abc}] \geq \epsilon,$$

where the probability is over the random choice of $a, b, c \in \mathbb{Z}_p$, and the random bits of $\mathcal{A}$.

Definition 3. *We say that the BDH assumption holds if no polynomial-time algorithm has only a negligible advantage ϵ in solving the BDH problem.*

Besides these computational/decisional assumptions, the Knowledge of Exponent (KE) assumption is also used in a number of papers (e.g. [3,9]). The KE assumption is defined as follows.

Definition 4. *For any adversary $\mathcal{A}$, which takes a KE challenge (g, g^a) as input and returns (C, Y) where $Y = C^a$, there exists an extractor $\mathcal{A}'$, which takes the same input as $\mathcal{A}$ returns c such that $g^c = C$.*

2.2 Review of Public Key Encryption

A Public Key Encryption (PKE) scheme [17] involves a Trusted Third Party (TTP) and users, and consists of four algorithms (Setup, KeyGen, Encrypt, Decrypt) which are defined as follows. As a standard practice, the security of

a PKE scheme is evaluated by an attack game played between a challenger and an adversary, where the challenger simulates the protocol execution and answers the oracle queries from the adversary. Corresponding to the PKE algorithms, we also introduce the oracles for the adversary.

- $\mathsf{Setup}(k)$: Run by the TTP, this algorithm takes a security parameter k as input and generates the public parameter $params$, which is an implicit input for other algorithms and we omit it in the description for simplicity.
- $\mathsf{KeyGen}(k)$: Run by a user, this algorithm generates a key pair (pk, sk). An KeyGen oracle can be queried with a public key pk; the challenger returns the corresponding private key sk.
- $\mathsf{Encrypt}(m, pk)$: Run by the message sender, this algorithm takes a message m and a public key pk as input, and outputs a ciphertext c encrypted under the public key pk.
- $\mathsf{Decrypt}(c, sk)$: Run by the message receiver, this algorithm takes a ciphertext c and the private key sk as input, and outputs the message m. A $\mathsf{Decrypt}$ oracle can be queried with a pair (c, pk) as input; the challenger returns $\mathsf{Decrypt}(c, sk)$.

We extend the concept of PKE to type-based PKE which enables a message sender to explicitly include some type information in the encryption process. A type-based PKE consists of four algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$, where Setup and KeyGen are defined as above, and

- $\mathsf{Encrypt}(m, t, pk)$: Run by the message sender, this algorithm takes a message m, a type string t, and a public key pk as input, and outputs a ciphertext c encrypted under the public key pk. Note that both c and t should be sent to the message receiver.
- $\mathsf{Decrypt}(c, t, sk)$: Run by the message receiver, this algorithm takes a ciphertext c, a message type t, and the private key sk as input, and outputs the message m. A $\mathsf{Decrypt}$ oracle can be queried with a pair (c, t, pk) as input; the challenger returns $\mathsf{Decrypt}(c, t, sk)$.

In the above descriptions, the type information t can also be included as a part of the ciphertext c, but we have explicitly used type information t as a label of the ciphertext c. This is only a description preference.

3 The Concept of Type-Based Proxy Re-encryption

In a type-based proxy re-encryption scheme, the delegator possesses a key pair (pk, sk) with a type-based PKE scheme $(\mathsf{Setup}_1, \mathsf{KeyGen}_1, \mathsf{Encrypt}_1, \mathsf{Decrypt}_1)$, as defined in Section 2.2. We assume that the delegatees use a PKE scheme $(\mathsf{Setup}_2, \mathsf{KeyGen}_2, \mathsf{Encrypt}_2, \mathsf{Decrypt}_2)$.

Suppose the delegator wants to delegate his decryption right for messages with type t to a delegatee with key pair (pk', sk'), he runs the $\mathsf{Pextract}$ algorithm to generate the proxy key.

– $\mathsf{Pextract}(pk, pk', t, sk)$: This algorithm takes the delegator's public key pk, the delegatee's public key pk', a message type t, the delegator's private key sk as input and outputs the delegation key $rk_{pk \xrightarrow{t} pk'}$. A $\mathsf{Pextract}$ oracle can be queried with a tuple (pk, pk', t) as input; the challenger returns the proxy key $rk_{pk \xrightarrow{t} pk'}$.

Note that all proxy keys, $rk_{pk \xrightarrow{t} pk'}$ for any t and pk', are generated based on the delegator's single key pair. To delegate his right, the delegator assigns $rk_{pk \xrightarrow{t} pk'}$ to an appropriate proxy, which will preform the re-encryption for the delegator's ciphertexts.

– $\mathsf{Preenc}(c, t, rk_{pk \xrightarrow{t} pk'})$: Run by the proxy, this algorithm takes a ciphertext c (for the delegator), a message type t, and the proxy key $rk_{pk \xrightarrow{t} pk'}$ as input, and outputs a new ciphertext c' (for the delegatee with (pk', sk')). A Preenc oracle can be queried with (c, t, pk, pk') as input; the challenger returns $\mathsf{Preenc}(c, t, rk_{pk \xrightarrow{t} pk'})$.

In contrast to the assumption of multi-level delegation (e.g. in [11]), we assume that there is only one level delegation, namely the delegatees will not further delegate their decryption rights to other users.

3.1 Threat Model for Type-Based Proxy Re-encryption

In practice, there might be multiple different parties acting as proxies. For example, Alice may choose Proxy 1 to delegate her decryption right to Bob and choose a proxy 2 to delegate his decryption right to Charlie, where these two proxies have no relationship. Every involved proxy is assumed to be semi-honest in the following sense: it will honestly convert the delegator's ciphertexts using the proxy key; however, it might act maliciously to obtain some information about the plaintexts of the delegator and the delegatee.

We identify the following security requirements with respect to the semantic security for plaintexts.

1. Firstly, the proxy should not obtain any information about the plaintexts of either the delegator or the delegatee.
2. Secondly, the delegatee should be able to decrypt all the appropriate type of plaintexts of the delegator after the re-encryption by the proxy. However, the delegatee alone should not obtain any information about the plaintexts before the re-encrypted by the proxy. This is essential when we want the proxy to be a policy enforcer.

In our formal definitions, these requirements lead to the IND-PR-CCA/IND-PR-CPA security of type-based proxy encryption schemes for the delegator.

With a public key encryption scheme, if Alice encrypts a message to Bob using his public key, then Alice can stay anonymous. However, with a proxy re-encryption scheme, this kind of anonymity might not be trivially achieved.

1. $params_1 \xleftarrow{\$} \mathsf{Setup}_1(k); params_2 \xleftarrow{\$} \mathsf{Setup}_2(k); (pk, sk) \xleftarrow{\$} \mathsf{KeyGen}_1(k)$
2. $(m_0, m_1, t^*) \xleftarrow{\$} \mathcal{A}^{(\mathsf{KeyGen}_2, \mathsf{Pextract}, \mathsf{Preenc}, \mathsf{Decrypt}_1, \mathsf{Decrypt}_2)}(params_1, params_2, pk)$
3. $b \xleftarrow{\$} \{0,1\}; c_b \xleftarrow{\$} \mathsf{Encrypt}_1(m_b, t^*, pk)$
4. $b' \xleftarrow{\$} \mathcal{A}^{(\mathsf{KeyGen}_2, \mathsf{Pextract}, \mathsf{Preenc}, \mathsf{Decrypt}_1, \mathsf{Decrypt}_2)}(params_1, params_2, pk, c_b)$

Fig. 1. IND-PR-CCA Security

For example, in the schemes in [11], Bob can tell whether messages are from the same user or not. In many potential applications of proxy re-encryption, this might become a privacy concern if the delegator does not want the delegatee to know whether a certain message is from him or not. Therefore, a re-encrypted ciphertext should be indistinguishable from a normal ciphertext generated under the delegatee's public key. This requirement leads to the definition of ciphertext privacy.

3.2 Formal Security Definitions

Before the formal definitions, we first introduce the idea behind our IND-PR-CCA definition (the idea of IND-PR-CPA security follows immediately). The definition covers two types of adaptive adversaries: a malicious delegatee and a malicious proxy. In the case of a malicious delegatee with key pair (pk', sk'), the adversary is allowed to know all proxy keys for message types $t \neq t^*$, either the private key sk'' or the proxy key $rk_{pk \xrightarrow{t^*} pk''}$ for any other delegatee with (pk'', sk''), but not $rk_{pk \xrightarrow{t^*} pk'}$. In the case of a malicious proxy with $rk_{pk \xrightarrow{t^*} pk'}$, the adversary is allowed to know all proxy keys for message types $t \neq t^*$, either the private key sk'' or the proxy key $rk_{pk \xrightarrow{t^*} pk''}$ for any other delegatee with (pk'', sk''), but not sk'. In addition, the adversary is capable of issuing decryption queries (see our remarks below). We believe the adversary has been granted the most privileges possible.

IND-PR-CCA Security. We first define the semantic security against a chosen ciphertext attack for the delegator.

Definition 5. *A type-based proxy re-encryption scheme is said to be IND-PR-CCA secure for the delegator if any polynomial time adversary has only a negligible advantage in the IND-PR-CCA game, where the adversary's advantage is defined to be* $|\Pr[b' = b] - \frac{1}{2}|$.

Analogous to the IND-CCA definition [2], as depicted in Figure 1, the IND-PR-CCA game is as follows.

1. Game setup: The challenger takes a security parameter k as input, runs Setup_1 to generate the public system parameter $params_1$ and runs Setup_2 to generate the public system parameter $params_2$. The challenger runs KeyGen_1 to generate a key pair (pk, sk).

2. Phase 1: The adversary takes $params_1$, $params_2$, and pk as input, and has access to the following types of oracles: KeyGen_2, $\mathsf{Pextract}$, Preenc, $\mathsf{Decrypt}_1$, and $\mathsf{Decrypt}_2$. Once the adversary decides that Phase 1 is over, it outputs two equal length plaintexts m_0, m_1, and a message type t^*. At the end of Phase 1, the following constraint should be satisfied: For any pk', if (pk, pk', t^*) has been queried to the $\mathsf{Pextract}$ oracle, then pk' should not have been queried to the KeyGen_2 oracle.
3. Challenge: The challenger picks a random bit $b \in \{0, 1\}$ and returns $c_b = \mathsf{Encrypt}_1(m_b, t^*, pk)$ as the challenge to the adversary.
4. Phase 2: The adversary is allowed to continue querying the same types of oracles as in Phase 1. At the end of Phase 2, we have the following constraints.
   (a) (c_b, t^*, pk) has not been queried to the $\mathsf{Decrypt}_1$ oracle.
   (b) For any pk', if (pk, pk', t^*) has been queried to the $\mathsf{Pextract}$ oracle, then pk' should not have been queried to the KeyGen_2 oracle.
   (c) If pk' has been queried to the KeyGen_2 oracle, (c_b, t^*, pk, pk') should not have been queried to the Preenc oracle.
   (d) If (pk, pk', t^*) has been queried to the $\mathsf{Pextract}$ oracle, then (c'_b, pk') should not have been queried to the $\mathsf{Decrypt}_2$ oracle where c'_b is a valid output of $\mathsf{Preenc}(c_b, t^*, pk, pk')$.
5. Guess (game ending): The adversary outputs a guess $b' \in \{0, 1\}$.

We remark on the constraints (c) and (d) in the above game. In the case of type-based proxy re-encryption, if the adversary obtains sk', then a Preenc query with the input (c_b, t^*, pk, pk') is equivalent to a $\mathsf{Decrypt}_1$ query with the input (c_b, t^*, pk). If the adversary obtains $rk_{pk \xrightarrow{t^*} pk'}$, then a $\mathsf{Decrypt}_2$ query with the input (c'_b, pk'), where c'_b is a valid output of a Preenc query with the input (c_b, t^*, pk, pk'), is equivalent to a $\mathsf{Decrypt}_1$ query with the input (c_b, t^*, pk). These constraints are necessary to prevent the adversary from winning the game trivially.

IND-PR-CPA Security. We define the semantic security against a chosen plaintext attack for the delegator. Analogous to the IND-CPA definitions for traditional PKE schemes [2], we can just remove the decryption privileges of the adversary to define the IND-PR-CPA game, i.e. the oracle accesses to Preenc, $\mathsf{Decrypt}_1$, and $\mathsf{Decrypt}_2$ are removed. However, we need to provide the adversary a $\mathsf{Preenc}^\dagger$ oracle to cover the case that a malicious delegatee with (pk', sk') can always decrypt the re-encrypted ciphertexts for him.

- $\mathsf{Preenc}^\dagger(m, t, pk, pk')$: the challenger returns $\mathsf{Preenc}(\mathsf{Encrypt}_1(m, t, pk), t, rk_{pk \xrightarrow{t} pk'})$.

Note that the re-encrypted ciphertext might leak some information about the delegator's private key and hence help the adversary to obtain some information of the delegator's ciphertexts. This has been ignored in [11].

The IND-PR-CPA game is depicted in Figure 2, and a detailed description can be obtained by forbidding the Preenc, $\mathsf{Decrypt}_1$, and $\mathsf{Decrypt}_2$ oracle access and providing $\mathsf{Preenc}^\dagger$ oracle access in the IND-PR-CCA game.

1. $params_1 \xleftarrow{\$} \mathsf{Setup}_1(k); params_2 \xleftarrow{\$} \mathsf{Setup}_2(k); (pk, sk) \xleftarrow{\$} \mathsf{KeyGen}_1(k)$
2. $(m_0, m_1, t^*) \xleftarrow{\$} \mathcal{A}^{(\mathsf{KeyGen}_2, \mathsf{Pextract}, \mathsf{Preenc}^\dagger)}(params_1, params_2, pk)$
3. $b \xleftarrow{\$} \{0,1\}; c_b \xleftarrow{\$} \mathsf{Encrypt}_1(m_b, t^*, pk)$
4. $b' \xleftarrow{\$} \mathcal{A}^{(\mathsf{KeyGen}_2, \mathsf{Pextract}, \mathsf{Preenc}^\dagger)}(params_1, params_2, pk, c_b)$

Fig. 2. IND-PR-CPA Security

Definition 6. *A type-based proxy re-encryption scheme is said to be IND-PR-CPA secure for the delegator if any polynomial time adversary has only a negligible advantage in the IND-PR-CPA game (depicted in Figure 2), where the adversary's advantage is defined to be* $|\Pr[b' = b] - \frac{1}{2}|$.

Ciphertext privacy. The ciphertext privacy attribute means that, for any delegatee, a re-encrypted ciphertext is indistinguishable from a normal ciphertext generated under the delegatee's public key. The attack game for ciphertext privacy is as follows.

1. Game setup: The challenger takes a security parameter k as input, runs Setup_1 to generate the public system parameter $params_1$ and runs Setup_2 to generate the public system parameter $params_2$. The challenger runs KeyGen_1 to generate a key pair (pk, sk).
2. Phase 1: The adversary takes $params_1$, $params_2$, and pk as input, and has access to the following types of oracles: KeyGen_2 and $\mathsf{Pextract}$. Once the adversary decides that Phase 1 is over, it outputs a message m, a message type t, and pk'.
3. Challenge: The challenger picks a random bit $b \in \{0,1\}$ and returns a challenge c_b as follows.
   - If $b = 0$, then $c_b = \mathsf{Encrypt}_2(m, pk')$.
   - If $b = 1$, then $c_b = \mathsf{Preenc}(\mathsf{Encrypt}_1(m, t, pk), t, rk_{pk \xrightarrow{t} pk'})$.
4. Phase 2: The adversary has access to the same types of oracles as in Phase 1.
5. Guess (game ending): The adversary outputs a guess $b' \in \{0,1\}$.

Definition 7. *A type-based proxy re-encryption scheme achieves ciphertext privacy if any polynomial time adversary has only a negligible advantage in the above game, where the adversary's advantage is defined to be* $|\Pr[b' = b] - \frac{1}{2}|$.

4 CPA-Secure Scheme with Ciphertext Privacy

In this section we propose a new type-based proxy re-encryption scheme which achieves ciphertext privacy. The scheme is IND-PR-CPA secure based on the Co-BDH and the XDH assumptions in the random oracle model. We note that the ciphertext privacy is achieved through the re-randomization by the proxy.

4.1 Description of the Scheme

The Delegator's Type-Based PRE Scheme. The delegator uses the following type-based PKE scheme $(\mathsf{Setup}_1, \mathsf{KeyGen}_1, \mathsf{Encrypt}_1, \mathsf{Decrypt}_1)$.

1. $\mathsf{Setup}_1(k)$: This algorithm generates three multiplicative cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ of prime order p, a random generator g_1 of $\mathbb{G}_1$, a random generator g_2 of $\mathbb{G}_2$, a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, and two hash functions
$$\mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}_2, \ \mathsf{H}_2 : \{0,1\}^* \to \{0,1\}^\ell,$$
where ℓ is a polynomial in k and $\{0,1\}^\ell$ is the plaintext space. The public parameter is denoted as $params_1 = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, \mathsf{H}_1, \mathsf{H}_2, \hat{e}, \ell)$, and we further assume the type information is $t \in \{0,1\}^*$.
2. $\mathsf{KeyGen}_1(k)$: This algorithm outputs a key pair (pk, sk) where $u \in_R \mathbb{Z}_p$, $pk = g_1^u$, and $sk = u$.
3. $\mathsf{Encrypt}_1(m, t, pk)$: This algorithm outputs a ciphertext $c = (c_1, c_2, c_3)$, where
$$r \in_R \mathbb{Z}_p, \ c_1 = g_1^r, \ h \in_R \mathbb{G}_T, \ c_2 = m \oplus \mathsf{H}_2(h).$$
$$\begin{aligned} c_3 &= h \cdot \hat{e}(pk, \mathsf{H}_1(0||t))^r \\ &= h \cdot \hat{e}(g_1, \mathsf{H}_1(0||t))^{u \cdot r} \end{aligned}$$
4. $\mathsf{Decrypt}_1(c, t, sk)$: This algorithm recovers m as follows:
$$\begin{aligned} m' &= c_2 \oplus \mathsf{H}_2\left(\frac{c_3}{\hat{e}(c_1, \mathsf{H}_1(0||t))^{sk}}\right) \\ &= m \oplus \mathsf{H}_2(h) \oplus \mathsf{H}_2\left(\frac{h \cdot \hat{e}(g_1, \mathsf{H}_1(0||t))^{u \cdot r}}{\hat{e}(g_1, \mathsf{H}_1(0||t))^{u \cdot r}}\right) \\ &= m \end{aligned}$$

The delegatee's PRE scheme. The delegatees use the following PKE scheme $(\mathsf{Setup}_2, \mathsf{KeyGen}_2, \mathsf{Encrypt}_2, \mathsf{Decrypt}_2)$.

1. $\mathsf{Setup}_2(k)$: Set $params_2 = params_1 = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, \mathsf{H}_1, \mathsf{H}_2, \hat{e}, \ell)$.
2. $\mathsf{KeyGen}_2(k)$: This algorithm outputs a key pair (pk, sk), where $v \in_R \mathbb{Z}_p$, $pk = g_2^v$, and $sk = v$.
3. $\mathsf{Encrypt}_2(m, pk)$: This algorithm outputs a ciphertext $c = (c_0, c_1, c_2, c_3)$, where
$$x, y \in_R \mathbb{Z}_p, \ c_0 = g_1^x, \ c_1 = g_2^{v \cdot y}, \ w \in_R \mathbb{G}_T, \ c_2 = m \oplus \mathsf{H}_2(w),$$
$$\begin{aligned} c_3 &= w \cdot \hat{e}(g_1^x, g_2^y) \\ &= w \cdot \hat{e}(g_1, g_2)^{x \cdot y} \end{aligned}$$
4. $\mathsf{Decrypt}_2(c, sk)$: This algorithm recovers m as follows:
$$\begin{aligned} m' &= c_2 \oplus \mathsf{H}_2\left(\frac{c_3}{\hat{e}(c_0, c_1)^{v^{-1}}}\right) \\ &= m \oplus \mathsf{H}_2(w) \oplus \mathsf{H}_2\left(\frac{w \cdot \hat{e}(g_1, g_2)^{x \cdot y}}{\hat{e}(g_1^x, g_2^{v \cdot y})^{v^{-1}}}\right) \\ &= m \end{aligned}$$

The delegation algorithms. Suppose the delegator has a key pair (pk, sk), where $pk = g_1^u$ and $sk = u$. Suppose a delegatee has a key pair (pk', sk'), where $pk' = g_2^v$ and $sk' = v$. The algorithms $\mathsf{Pextract}$ and Preenc are defined as follows.

– $\mathsf{Pextract}(pk, pk', t, sk)$: The proxy key is $rk_{pk \xrightarrow{t} pk'}$, where

$$s \in_R \mathbb{Z}_p,\ \ rk_{pk \xrightarrow{t} pk'} = (g_2^{v \cdot s}, g_2^s \cdot \mathsf{H}_1(0||t)^{-sk}).$$

– $\mathsf{Preenc}(c, t, rk_{pk \xrightarrow{t} pk'})$: Given a ciphertext $c = (c_1, c_2, c_3)$, where

$$c_1 = g_1^r,\ \ c_2 = m \oplus \mathsf{H}_2(h),\ \ c_3 = h \cdot \hat{e}(g_1, \mathsf{H}_1(0||t))^{u \cdot r},$$

this algorithm computes a new ciphertext $c' = (c'_0, c'_1, c'_2, c'_3)$, where

$$z \in_R \mathbb{Z}_p,\ \ c'_0 = c_1,\ \ c'_1 = g_2^{v \cdot (s+z)},\ \ c'_2 = c_2,$$

$$\begin{aligned} c'_3 &= c_3 \cdot \hat{e}(c'_0, g_2^{(s+z)} \cdot \mathsf{H}_1(0||t)^{-sk}) \\ &= h \cdot \hat{e}(g_1^r, \mathsf{H}_1(0||t)^u) \cdot \hat{e}(g_1^r, g_2^{(s+z)} \cdot \mathsf{H}_1(0||t)^{-u}) \\ &= h \cdot \hat{e}(g_1, g_2)^{r \cdot (s+z)}. \end{aligned}$$

The re-encrypted ciphertext is also a valid ciphertext for the delegatee so that the delegatee can obtain the plaintext m by running $\mathsf{Decrypt}_2$.

4.2 Security Analysis

The proposed scheme is proven to be IND-PR-CPA secure from Lemma 1 and achieve ciphertext privacy from Lemma 2. The proofs appear in the full paper [18].

Lemma 1. *The proposed type-based proxy re-encryption scheme is IND-PR-CPA secure based on the Co-BDH and the XDH assumptions in the random oracle model.*

Lemma 2. *The proposed type-based proxy re-encryption scheme achieves ciphertext privacy unconditionally.*

5 CCA-Secure Scheme without Ciphertext Privacy

In this section we propose a new type-based proxy re-encryption scheme which is IND-PR-CCA secure based on the BDH and the KE assumptions in the random oracle model. However, the scheme does not achieve ciphertext privacy.

5.1 Description of the Scheme

The delegator's type-based PKE scheme. The delegator uses the following type-based PKE scheme $(\mathsf{Setup}_1, \mathsf{KeyGen}_1, \mathsf{Encrypt}_1, \mathsf{Decrypt}_1)$.

1. $\mathsf{Setup}_1(k)$: This algorithm generates two cyclic groups $\mathbb{G}$ and $\mathbb{G}_T$ of prime order p, a generator g of $\mathbb{G}$, a bilinear map $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, and three hash functions
$$\mathsf{H}_1 : \{0,1\}^* \to \mathbb{G},\ \mathsf{H}_2 : \{0,1\}^* \to \mathbb{Z}_p,\ \mathsf{H}_3 : \{0,1\}^* \to \{0,1\}^\ell,$$
where ℓ is a polynomial of k and $\{0,1\}^\ell$ is the plaintext space. The public parameter is denoted as $params_1 = (\mathbb{G}, \mathbb{G}_T, p, g, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3, \hat{e}, \ell)$, and we further assume the type information is $t \in \{0,1\}^*$.
2. $\mathsf{KeyGen}_1(k)$: This algorithm outputs a key pair (pk, sk) where $u \in_R \mathbb{Z}_p$, $pk = g^u$, and $sk = u$.
3. $\mathsf{Encrypt}_1(m, t, pk)$: This algorithm outputs the ciphertext $c = (c_1, c_2, c_3, c_4)$, where
$$h \in \mathbb{G}_T,\ c_1 = g^{\mathsf{H}_2(m||h)},\ c_2 = h \cdot \hat{e}(pk, \mathsf{H}_1(0||t))^{\mathsf{H}_2(m||h)},$$
$$c_3 = m \oplus \mathsf{H}_3(h),\ c_4 = \mathsf{H}_1(1||c_1||c_2||c_3)^{\mathsf{H}_2(m||h)}.$$
4. $\mathsf{Decrypt}_1(c, t, sk)$: This algorithm recovers the plaintext as follows:
   (a) Verify $\hat{e}(c_1, \mathsf{H}_1(1||c_1||c_2||c_3)) = \hat{e}(g, c_4)$.
   (b) Compute $h = \frac{c_2}{\hat{e}(\mathsf{H}_1(0||t), c_1)^{sk}}$ and $m = c_3 \oplus \mathsf{H}_3(h)$.
   (c) Verify $c_1 = g^{\mathsf{H}_2(m||h)}$.
   (d) Return m.

   During decryption, if any of the verifications fails, the algorithm returns an error symbol $\perp$.

The delegatee's PKE scheme. Suppose $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is a PKE scheme which has the plaintext space $\mathbb{Z}_p$. The delegatees use a PKE scheme $(\mathsf{Setup}_2, \mathsf{KeyGen}_2, \mathsf{Encrypt}_2, \mathsf{Decrypt}_2)$.

1. $\mathsf{Setup}_2(k)$: Output $params_2 = (params, \mathbb{G}, \mathbb{G}_T, p, g, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3, \hat{e}, \ell)$ where $params$ is the public parameter generated by $\mathsf{Setup}(k)$.
2. $\mathsf{KeyGen}_2(k)$: Output a key pair (pk, sk) which is the output of $\mathsf{KeyGen}(k)$.
3. $\mathsf{Encrypt}_2(m, pk)$: This algorithm outputs the ciphertext $c = (c_{-1}, c_0, c_1, c_2, c_3)$, where
$$w \in \mathbb{G}_T,\ x, y \in_R \mathbb{Z}_p,\ c_{-1} = \mathsf{Encrypt}(x, pk),\ c_0 = \mathsf{Encrypt}(y, pk),\ c_1 = g^{\mathsf{H}_2(m||w)},$$
$$c_2 = w \cdot \hat{e}(g^{\mathsf{H}_2(m||w)}, \mathsf{H}_1(2||x||c_{-1}||pk) \cdot \mathsf{H}_1(2||y||c_0||pk)),\ c_3 = m \oplus \mathsf{H}_3(w).$$
4. $\mathsf{Decrypt}_2(c, sk)$: This algorithm recovers the plaintext as follows:
   (a) Compute $w = \frac{c_2}{\hat{e}(c_1, \mathsf{H}_1(2||\mathsf{Decrypt}(c_{-1}, sk)||c_{-1}||pk) \cdot \mathsf{H}_1(2||\mathsf{Decrypt}(c_0, sk)||c_0||pk))}$ and $m = c_3 \oplus \mathsf{H}_3(w)$.
   (b) Verify $c_1 = g^{\mathsf{H}_2(m||w)}$.
   (c) Return m.

   During decryption, if any of the verifications fails, the algorithm returns an error symbol $\perp$.

The delegation algorithms. Suppose the delegator has a key pair (pk, sk). Suppose a delegatee has the key pair (pk', sk'). The algorithms $\mathsf{Pextract}$ and Preenc are defined as follows.

– $\mathsf{Pextract}(pk, pk', t, sk)$: The proxy key is $rk_{pk \xrightarrow{t} pk'}$, where

$$s_1 \in_R \mathbb{Z}_p, \;\; s_2 = \mathsf{Encrypt}(s_1, pk'),$$

$$rk_{pk \xrightarrow{t} pk'} = (s_2, \mathsf{H}_1(2||s_1||s_2||pk') \cdot \mathsf{H}_1(0||t)^{-sk}).$$

– $\mathsf{Preenc}(c, t, rk_{pk \xrightarrow{t} pk'})$: Given a ciphertext $c = (c_1, c_2, c_3, c_4)$, where

$$c_1 = g^{\mathsf{H}_2(m||h)}, \;\; c_2 = h \cdot \hat{e}(pk, \mathsf{H}_1(0||t))^{\mathsf{H}_2(m||h)},$$

$$c_3 = m \oplus \mathsf{H}_3(h), \;\; c_4 = \mathsf{H}_1(1||c_1||c_2||c_3)^{\mathsf{H}_2(m||h)}.$$

this algorithm first verifies $\hat{e}(c_1, \mathsf{H}_1(1||c_1||c_2||c_3)) = \hat{e}(g, c_4)$. If the verification passes, it computes a new ciphertext $c' = (c'_{-1}, c'_0, c'_1, c'_2, c'_3)$, where

$$r \in_R \mathbb{Z}_p, \;\; c'_{-1} = \mathsf{Encrypt}(r, pk'), \;\; c'_0 = s_2, \;\; c'_1 = c_1, \;\; c'_3 = c_3,$$

$$\begin{aligned} c'_2 &= c_2 \cdot \hat{e}(c'_1, \mathsf{H}_1(2||r||c'_{-1}||pk') \cdot \mathsf{H}_1(2||s_1||s_2||pk') \cdot \mathsf{H}_1(0||t)^{-sk}) \\ &= h \cdot \hat{e}(pk, \mathsf{H}_1(0||t))^{\mathsf{H}_2(m||h)} \cdot \hat{e}(g^{\mathsf{H}_2(m||h)}, \mathsf{H}_1(2||r||c'_{-1}||pk') \\ &\quad \cdot \mathsf{H}_1(2||s_1||s_2||pk') \cdot \mathsf{H}_1(0||t)^{-sk}) \\ &= h \cdot \hat{e}(g^{\mathsf{H}_2(m||h)}, \mathsf{H}_1(2||r||c'_{-1}||pk') \cdot \mathsf{H}_1(2||s_1||s_2||pk')). \end{aligned}$$

Otherwise, it return an error symbol $\perp$.

It is clear that the re-encrypted ciphertext is also a valid ciphertext for the delegatee so that the delegatee can obtain the plaintext m by running $\mathsf{Decrypt}_2$.

5.2 Security Analysis

The proposed scheme is proven to be IND-PR-CCA secure from Lemma 3. The proof appears in the full paper [18].

Lemma 3. *The proposed type-based proxy re-encryption scheme is IND-PR-CCA secure based on the BDH assumption and the KE assumptions in the random oracle model, given that* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *is deterministic and one-way.*

6 Conclusion

In this paper we have introduced the concept of type-based proxy re-encryption to address the inefficiency issues of traditional proxy re-encryption schemes in

practical applications. We have also proposed two schemes which are IND-PR-CPA secure and IND-PR-CCA secure respectively. The IND-PR-CPA secure scheme also achieves ciphertext privacy (which means that a re-encrypted ciphertext is indistinguishable from a normal ciphertext for the delegatee), but the IND-PR-CCA secure scheme does not achieve this attribute. Designing an IND-PR-CCA scheme with ciphertext privacy is left as an open problem. The security model proposed in this paper is particularly designed for traditional PKE schemes, hence, it is interesting to extend it to the ID-based setting.

References

1. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Trans. Inf. Syst. Secur. 9(1), 1–30 (2006)
2. 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)
3. Bellare, M., Palacio, A.: The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 273–289. Springer, Heidelberg (2004)
4. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Proceedings of the 1st ACM conference on Computer and communications security, pp. 62–73. ACM Press, New York (1993)
5. Blaze, M., Bleumer, G., Strauss, M.: Divertible protocols and atomic proxy cryptography. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 127–144. Springer, Heidelberg (1998)
6. Boneh, D., Franklin, M.K.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
7. Borisov, N., Mitra, S.: Restricted queries over an encrypted index with applications to regulatory compliance. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 373–391. Springer, Heidelberg (2008)
8. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact E-Cash. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 302–321. Springer, Heidelberg (2005)
9. Damgård, I.: Towards practical public key systems secure against chosen ciphertext attacks. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 445–456. Springer, Heidelberg (1992)
10. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985)
11. Green, M., Ateniese, G.: Identity-based proxy re-encryption. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 288–306. Springer, Heidelberg (2007)
12. Seroussi, G., Blake, I.F., Smart, N.P.: Elliptic Curves in Cryptography. Cambridge University Press, Cambridge (1999)
13. Ivan, A., Dodis, Y.: Proxy cryptography revisited. In: Proceedings of the Network and Distributed System Security Symposium. The Internet Society (2003)

14. Jakobsson, M.: On quorum controlled asymmetric proxy re-encryption. In: Imai, H., Zheng, Y. (eds.) PKC 1999. LNCS, vol. 1560, pp. 112–121. Springer, Heidelberg (1999)
15. Mambo, M., Okamoto, E.: Proxy cryptosystems: Delegation of the power to decrypt ciphertexts. IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences E80-A(1), 54–63 (1997)
16. Matsuo, T.: Proxy re-encryption systems for identity-based encryption. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 247–267. Springer, Heidelberg (2007)
17. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1997)
18. Tang, Q.: Type-based proxy re-encryption and its construction. Technical Report TR-CTIT-08-47, Centre for Telematics and Information Technology, University of Twente (2008)
19. Wang, L., Cao, Z., Okamoto, T., Miao, Y., Okamoto, E.: Authorization-Limited Transformation-Free Proxy Cryptosystems and Their Security Analyses*. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences (1), 106–114 (2006)
20. Zhou, L., Marsh, M.A., Schneider, F.B., Redz, A.: Distributed blinding for distributed elgamal re-encryption. In: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, pp. 824–824. IEEE Computer Society, Los Alamitos (2005)

Toward a Generic Construction of Universally Convertible Undeniable Signatures from Pairing-Based Signatures

Laila El Aimani

b-it (Bonn-Aachen International Center for Information Technology),
Dahlmannstr. 2, D-53113 Bonn, Germany
elaimani@bit.uni-bonn.de

Abstract. Undeniable signatures were proposed to limit the verification property of ordinary digital signatures. In fact, the verification of such signatures cannot be attained without the help of the signer, via the confirmation/denial protocols. Later, the concept was refined to give the possibility of converting the issued undeniable signatures into ordinary ones by publishing a *universal* receipt that turns them publicly verifiable.

In this paper, we present the first generic construction for universally convertible undeniable signatures from certain weakly secure cryptosystems and any secure digital signature scheme. Next, we give two specific approaches for building universally convertible undeniable signatures from a large class of pairing-based signatures. These methods find a nice and practical instantiation with known encryption and signature schemes. For instance, we achieve the most efficient undeniable signatures with regard to the signature length and cost, the underlying assumption and the security model. We believe these constructions could be an interesting starting point to develop more efficient schemes or give better security analyses of the existing ones.

Keywords: Undeniable signatures, Pairing-based signatures, Generic construction.

1 Introduction

Undeniable signatures were originally introduced in 1990 by Chaum and van Antwerpen [8] to limit the self-authenticating property of digital signatures. In fact, the verification algorithm in these signatures is replaced by a confirmation (denial) protocol between the verifier and the signer, in which the verifier learns the validity (invalidity) of the issued signature without being able to transfer his conviction to a third person. This cryptographic primitive proved valuable in many applications where privacy is a big concern, e.g., licensing software.

In 1991, the notion of undeniable signature was boosted by Boyar et al. [3] to allow the conversion of a selected undeniable signature into an ordinary one by releasing a piece of information at a later time. The model supported also the universal conversion achieved by publishing a universal receipt (by the signer) that transforms all undeniable signatures into publicly verifiable ones.

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 145–157, 2008.

1.1 Related Work

Since the introduction of undeniable signatures, a series of proposals sprang up, covering a variety of different aspects. Pairing-based signatures [1] have received a lot of attention in these settings. Actually, most such signatures include in the verification equation a pairing computation between a part of the signature and some other parameters. Therefore, if we implement the same signature in a non bilinear group, namely a group where the Decisional Diffie-Hellman problem (DDH) is intractable, the resulting signature cannot be publicly verifiable. Hence, the signer must perform a proof of equality/inequality of two discrete logarithms with the verifier. Such a duality between pairing-based signatures and undeniable signatures has been illustrated in the literature by some proposals, e.g., the BLS signatures [2] whose undeniable variant are the early Chaum and van Antwerpen [8] signatures or Boneh and Boyen's signatures [1] which resulted in Laguillaumie and Vergnaud's undeniable signatures [12]. All these signatures inherit the security properties of their underlying digital signatures and have their invisibility based on a variant of the DDH problem.

Unfortunately, this approach does not give the possibility of converting the resulting signatures. A tantalizing challenge is to propose a general approach that constructs undeniable signatures from (a large category of) pairing-based signatures with the possibility of converting them to ordinary ones.

1.2 Our Contributions

We propose the first generic construction of universally convertible undeniable signatures from secure digital signatures and some weakly secure cryptosystems. Our design uses the "encryption of a signature" method [2] and relaxes the security requirement on the underlying cryptosystem, without compromising the overall security. As a consequence, we allow malleable cryptosystems in our design which impacts positively the efficiency of the confirmation/denial protocols.

Next, we give an efficient generic construction of universally convertible undeniable signatures. In fact, following the same principle, we shrink the set of signatures, upon which we build the undeniable signatures, down to a certain class of pairing-based signatures and we use an appropriate Key Encapsulation Mechanism. This construction finds a very efficient instantiation and results in the most efficient universally convertible undeniable signature scheme without random oracles and whose security rests on standard assumptions.

Finally, we enlarge the set of pairing-based signatures to include most proposals that appeared in the literature so far. In this way, the resulting undeniable signatures inherit the same virtues of the underlying digital signatures and acquire other interesting properties concerning their invisibility.

[1] See Section 2 for definitions of pairings, bilinear groups, etc...

[2] This method has been successfully used in a number of primitives such as designated confirmer signatures [5]. It consists in generating a signature on the message to be signed, then encrypting it. The validity or invalidity of the resulting signature are checked via concurrent proofs of knowledge.

2 Preliminaries

2.1 Bilinear Maps

Definition 1. *Let* $(\mathbb{G}, +)$ *and* $(\mathbb{H}, \times)$ [3] *be groups of prime order* d*. Let* P *be a generator of* $\mathbb{G}$*.* $\mathbb{G}$ *is called a* bilinear group *if there exists a map* $e : \mathbb{G} \times \mathbb{G} \to \mathbb{H}$*, with the following properties:*

1. *bilinearity: for all* $(P, Q) \in \mathbb{G}^2$ *and* $a, b \in \mathbb{Z}_d$*,* $e(aP, bQ) = e(P, Q)^{ab}$*,*
2. *efficient computability for any input pair, and*
3. *non-degeneracy:* $e(P, P) \neq 1_{\mathbb{H}}$*.*

2.2 Digital Signatures

A signature scheme Σ comprises three algorithms, keygen, sign, and verify:

- keygen is a probabilistic key generation algorithm which returns pairs of private and public keys $(\mathsf{sk}, \mathsf{pk})$ depending on the security parameter k,
- sign is a signing algorithm which takes on input a private key sk and a plaintext m and returns a signature σ, and
- verify is a deterministic algorithm which takes on input a public key pk, a signature σ and outputs 1 if the signature is valid and 0 otherwise.

Definition 2. *A signature scheme is said to be* (t, ϵ, q_s)*-EUF-CMA secure if no adversary* $\mathcal{A}$*, operating in time* t *and issuing at most* q_s *queries, wins the following game with probability greater than* ϵ*, where the probability is taken over all the random choices:*
Setup. *$\mathcal{A}$ is given the public parameters of the given signature scheme.*
Queries. *$\mathcal{A}$ queries the challenger for signatures on at most q_s messages.*
Output. *$\mathcal{A}$ outputs a pair (m, σ) and wins the game if m has not been queried before and* $\mathsf{verify}_{\mathsf{pk}}(m, \sigma) = 1$.

2.3 Public-Key Encryption Schemes

An asymmetric encryption scheme comprises the following algorithms:

- keygen is a probabilistic key generation algorithm which returns pairs of private and public keys $(\mathsf{sk}, \mathsf{pk})$ depending on the security parameter k,
- $\mathsf{encrypt}$ is a probabilistic encryption algorithm which takes on input a public key pk and a plaintext m, and returns a ciphertext c, and
- $\mathsf{decrypt}$ is a deterministic decryption algorithm which takes on input a secret key sk and a ciphertext c, and returns the corresponding plaintext m or $\perp$.

A cryptosystem provides indistinguishability (IND) if it is difficult to distinguish pairs of ciphertexts based on the messages they encrypt. In case the adversary against the scheme has access to a decryption oracle, the scheme is said to be indistinguishable under chosen ciphertext attacks (IND-CCA), otherwise it is indistinguishable under chosen plaintext attacks (IND-CPA). Formal definitions can be found in [4].

[3] In the rest of the document, the group $\mathbb{G}$ is denoted additively whereas the group $\mathbb{H}$ is denoted multiplicatively.

2.4 Key Encapsulation Mechanisms (KEM)

A KEM is a tuple of algorithms $\mathcal{K} = (\mathsf{keygen}, \mathsf{encap}, \mathsf{decap})$ where

- keygen probabilistically generates a key pair $(\mathsf{sk}, \mathsf{pk})$,
- encap, or the *encapsulation* algorithm which, on input a random nonce r and the public key pk, generates a *session key* denoted k and its *encapsulation* c, and
- decap, or the *decapsulation* algorithm. Given the private key sk and the element c, this algorithm computes the decapsulation k of c, or returns $\perp$ if c is invalid.

Definition 3. *A KEM is said to be (t, ϵ)-IND-CPA secure if no adversary $\mathcal{A}$, operating in time t, wins the following game with probability greater than ϵ:*

- ***Phase 1.*** *$\mathcal{A}$ gets the parameters of the KEM from his challenger.*
- ***Challenge.*** *The challenger computes a given encapsulation $c^\star$, then picks uniformly at random a bit b from $\{0, 1\}$. If $b = 1$, then he sets $k^\star$ to k_1 where $k_1 = \mathsf{decap}(c^\star)$. Otherwise, he sets $k^\star$ to a uniformly chosen string from the session keys space. The challenge is $(c^\star, k^\star)$.*
- ***Phase 2.*** *$\mathcal{A}$ outputs a bit b' (representing his guess of $k^\star$ being the decapsulation of $c^\star$) and wins the game if $b = b'$. We define $\mathcal{A}$'s advantage as $\mathsf{Adv}(\mathcal{A}) = |\Pr[b = b'] - \frac{1}{2}|$, where the probability is taken over the random choices of the adversary $\mathcal{A}$ and the challenger.*

The Hybrid Encryption Paradigm. It consists in combining KEMs with secure secret key encryption algorithms or Data Encapsulation Mechanisms (DEMs) to build encryption schemes. In fact, one can fix a session key k using the KEM, then uses it to encrypt a message using an efficient DEM. Decryption is achieved by first recovering the key from the encapsulation (part of the ciphertext) then applying the DEM decryption algorithm. It can be shown that one can obtain an IND-CPA cryptosystem from an IND-CPA KEM combined with a DEM indistinguishable under a one time attack (IND-OT). We refer to [11] for the necessary and sufficient conditions on KEMs and DEMs in order to obtain a certain level of security for the resulting hybrid encryption scheme.

3 Universally Convertible Undeniable Signatures (UCUS)

3.1 Definition

Setup. On input the security parameter k, outputs the public parameters.
Key Generation. Generates probabilistically a key pair $(\mathsf{sk}, \mathsf{pk})$.
Signature. On input the public parameters, the private key sk and a message m, outputs an undeniable signature μ.
Verification. This is an algorithm run by the signer to check the validity of an undeniable signature μ issued on m, using his private key sk.

Confirmation/Denial Protocol. These are interactive protocols between a prover and a verifier. Their common input consists of the public parameters of the scheme, the signature μ and the message m in question. The prover, that is the signer, uses his private key sk to convince the verifier of the validity (invalidity) of the signature μ on m.

Universal Conversion. Releases a universal receipt, using sk, that makes all undeniable signatures universally verifiable.

Universal Verification. On input a signature, a message, a receipt and the public key pk, outputs 1 if the signature is valid and 0 otherwise.

3.2 Security Model

In addition to the completeness, soundness and non-transferability of the proofs inherent to the confirmation/denial protocols, a convertible undeniable signature scheme requires two further properties, that are unforgeability and invisibility.

Unforgeability. The natural security requirement that a universally convertible signature scheme should fulfill is the existential unforgeability against a chosen message attack (EUF-CMA). It is defined through the following game.

- **Setup.** The adversary $\mathcal{A}$ is given the public parameters of the scheme in addition to the universal receipt.
- **Queries.** $\mathcal{A}$ queries the signing oracle adaptively on at most q_s messages. Note that there will be no need to query the confirmation/denial oracles since $\mathcal{A}$ has the universal receipt at his disposal.
- **Output.** At the end, $\mathcal{A}$ outputs a pair consisting of a message m, that has not been queried before, and a string μ. $\mathcal{A}$ wins the game if μ is a valid undeniable signature on m.

We say that a universally convertible undeniable signature scheme is (t, ϵ, q_s)-EUF-CMA secure if there is no adversary, operating in time t, that wins the above game with probability greater than ϵ.

Invisibility. Invisibility against a chosen message attack (INV-CMA) is defined through the following game between an attacker $\mathcal{A}$ and his challenger $\mathcal{R}$.

- $\mathcal{A}$ gets the parameters of the scheme from $\mathcal{R}$.
- **Phase 1.** $\mathcal{A}$ adaptively query the signing and confirmation/denial oracles.
- **Challenge.** Eventually, $\mathcal{A}$ outputs a message $m^\star$ that has not been queried before to the signing oracle and requests a challenge signature $\mu^\star$. $\mathcal{R}$ picks a bit $b \in_R \{0, 1\}$. If $b = 1$, then $\mu^\star$ is generated as usual using the signing oracle, otherwise it is chosen uniformly at random from the signatures space.
- **Phase 2.** $\mathcal{A}$ can adaptively query the previous oracles with the exception of not querying $m^\star$ to the signing oracle or $(m^\star, \mu^\star)$ to the verification oracles.
- **Output.** $\mathcal{A}$ outputs a bit b' representing his guess on $\mu^\star$ being a valid signature on $m^\star$. He wins the game if $b = b'$. We define $\mathcal{A}$'s advantage as $\mathsf{Adv}(\mathcal{A}) = |\Pr[b = b'] - \frac{1}{2}|$.

We say that a convertible undeniable signature scheme is (t, ϵ, q_s, q_v)-INV-CMA secure if no adversary operating in time t, issuing q_s queries to the signing oracle and q_v queries to the confirmation/denial oracles wins the above game with advantage greater than ϵ.

4 A Systematic Approach for UCUS from Some Cryptosystems and Digital Signatures

4.1 Design Principle

We use the "encryption of a signature" method. Thus, we first generate a digital signature on the message to be signed, then encrypt the resulting signature using a suitable cryptosystem obtained from the hybrid encryption paradigm. Confirmation or denial of the resulting signatures exist by virtue of Goldreich et al.'s result [10]. In fact, the verification and decryption algorithms in a signature scheme and a cryptosystem respectively define an NP (co-NP) language for which there exists a zero knowledge proof system.

This method has been in use for some time ago. For instance, Camenisch and Michels [5] used it for designated confirmer signatures. One of the main differences between the two proposals dwells in the security assumption on the cryptosystem. We actually require only IND-CPA secure KEMs (thus IND-CPA cryptosystems), as we do not allow individual conversions of the undeniable signatures, versus IND-CCA cryptosystems. The consequences of this are twofold. First, we require a weak security notion on the cryptosystem without compromising the overall security. This gives many and simpler choices for the cryptosystem to be used. Second, we allow malleable cryptosystems in our construction, which impacts positively the confirmation/denial protocols efficiency. In fact, cryptosystems with homomorphic properties possess efficient decryption proofs of knowledge, i.e, one can prove efficiently the knowledge of the plaintext corresponding to a given ciphertext. Such schemes are not ruled out from our design.

4.2 Proposed Construction

Let Σ be a digital signature scheme given by $\Sigma.\mathsf{keygen}$ which generates a key pair (private key = $\Sigma.\mathsf{sk}$, public key= $\Sigma.\mathsf{pk}$), $\Sigma.\mathsf{sign}$ and $\Sigma.\mathsf{verify}$.

Let furthermore Γ be a cryptosystem obtained using the hybrid encryption paradigm and described by $\Gamma.\mathsf{keygen}$ (that generates the pair (private key = $\Gamma.\mathsf{sk}$, public key= $\Gamma.\mathsf{pk}$)), $\Gamma.\mathsf{encrypt}$ and $\Gamma.\mathsf{decrypt}$. Note that the encapsulation of the key used to encrypt a given string is always contained in the ciphertext.

We assume for simplicity that the space of signatures produced by Σ is the same as the space of messages encrypted by Γ.

Let $m \in \{0,1\}^\star$ be a message, we propose the following scheme:

Setup. Invoke $\Gamma.\mathsf{setup}$ and $\Sigma.\mathsf{setup}$.

Key Generation. Invoke $\Sigma.\mathsf{keygen}$ and $\Gamma.\mathsf{keygen}$ to generate $\Sigma.\mathsf{sk}$, $\Sigma.\mathsf{pk}$ $\Gamma.\mathsf{sk}$ and $\Gamma.\mathsf{pk}$. Set the public key to $(\Sigma.\mathsf{pk}, \Gamma.\mathsf{pk})$ and the private key to ($\Sigma.\mathsf{sk}$, $\Gamma.\mathsf{sk}$).

Signature. First compute an encapsulation c together with its decapsulation k using $\Gamma.\mathsf{pk}$. Then compute a (digital) signature $\sigma = \Sigma.\mathsf{sign}_{\Sigma.\mathsf{sk}}(m\|c)$ on $m\|c$. Finally encrypt the resulting signature under $\Gamma.\mathsf{pk}$ (using k). Output $\mu = \Gamma.\mathsf{encrypt}_{\Gamma.\mathsf{pk}}(\sigma)$. Note that c is part of μ.

Verification (By the Signer.) To check the validity of an undeniable signature μ (that comprises the encapsulation c), issued on a certain message m, the signer first computes $\sigma = \Gamma.\mathsf{decrypt}_{\Gamma.\mathsf{sk}}(\mu)$, then calls $\Sigma.\mathsf{verify}$ on σ and $m\|c$ using $\Sigma.\mathsf{pk}$. μ is valid if and only if the output of the latter item is 1.

Confirmation/Denial Protocol. To confirm (deny) a purported signature μ (containing the encapsulation c) on a certain message m, the signer first computes $\sigma = \Gamma.\mathsf{decrypt}_{\Gamma.\mathsf{sk}}(\mu)$, then invokes the algorithm $\Sigma.\mathsf{verify}$ on σ and $m\|c$. According to the result, the signer issues a proof of knowledge of the decryption of μ that passes (does not pass) the verification algorithm $\Sigma.\mathsf{verify}$.

Universal Conversion. Release $\Gamma.\mathsf{sk}$.

4.3 Security Analysis and Efficiency Considerations

We first note that the properties of completeness, soundness and non-transferability of the confirmation/denial protocols are met by our construction as a direct consequence of the zero-knowledge proofs of knowledge. In the sequel, we prove that the construction resists existential forgeries and that signatures are invisible.

Theorem 1. *Our generic construction is (t, ϵ, q_s)-EUF-CMA secure if the underlying digital signature scheme is (t, ϵ, q_s)-EUF-CMA secure.*

Proof. Let $\mathcal{A}$ be an attacker that (t, ϵ, q_s)-EUF-CMA breaks the existential unforgeability of our construction. We will construct an adversary $\mathcal{R}$ that (t, ϵ, q_s)-EUF-CMA breaks the underlying digital signature scheme:

Key generation. $\mathcal{R}$ gets the parameters of the signature scheme in question from his challenger. Then he chooses an appropriate cryptosystem Γ (obtained from the encryption of a signature paradigm) with parameters $\Gamma.\mathsf{pk}$, $\Gamma.\mathsf{sk}$, $\Gamma.\mathsf{encrypt}$ and $\Gamma.\mathsf{decrypt}$. $\mathcal{R}$ fixes the above parameters as a setting for the undeniable signatures $\mathcal{A}$ is trying to attack.

Signature queries. For a signature query on a message m, $\mathcal{R}$ will first compute an encapsulation c together with its decapsulation k (using $\Gamma.\mathsf{pk}$). Then he will request his challenger for a digital signature σ on $m\|c$. Finally, he will encrypt σ under $\Gamma.\mathsf{pk}$ (using k) and output the result to $\mathcal{A}$.

Final Output. Once $\mathcal{A}$ outputs his forgery $\mu^\star$ on $m^\star$. $\mathcal{R}$ will decrypt the signature to obtain $\sigma^\star$. If $\mu^\star$ is valid then by definition $\sigma^\star$ is valid too. $\mathcal{R}$ will output $\sigma^\star$ as a forgery on the message $(m^\star\|c^\star)$ where $c^\star$ is the encapsulation of the key that was used to encrypt $\sigma^\star$. In fact the probability that $m^\star\|c^\star$ has been queried by $\mathcal{R}$ on a query $m_i\|c_i$ ($m_i \neq m^\star$) is negligeable since c_i is obtained by $\mathcal{R}$ from a random process (the encapsulation algorithm).

Note that there will be no need to simulate the confirmation/denial oracles since $\mathcal{A}$ has the universal receipt $\Gamma.\mathsf{sk}$ allowing the verification of the signatures. □

Theorem 2. *Our proposed construction is (t, ϵ, q_s, q_v)-INV-CMA secure if it is (t, ϵ', q_s)-EUF-CMA secure and the KEM used in the underlying cryptosystem is $(t + q_s q_v, \epsilon \cdot (1 - \epsilon')^{q_v})$-IND-CPA secure.*

Proof. Let $\mathcal{A}$ be an attacker that (t, ϵ, q_s, q_v)-INV-CMA breaks our undeniable signatures, assumed to be (t, ϵ', q_s)-EUF-CMA secure. We will construct an algorithm $\mathcal{R}$ that $(t + q_s q_v, \epsilon \cdot (1 - \epsilon')^{q_v})$-IND-CPA breaks the underlying KEM:

Phase 1

Key Generation. $\mathcal{R}$ gets the parameters of the KEM $\mathcal{K}$ from his challenger. Then he chooses an appropriate IND-OT secure DEM together with a signature scheme Σ.

Signature Queries. For a signature query on m. $\mathcal{R}$ first fixes a session key k together with its decapsulation c using $\mathcal{K}.pk$. Then he computes a (digital) signature σ on $m\|c$ using $\Sigma.\mathsf{sk}$. Finally, he encrypts the produced signature (using k) and outputs the result to $\mathcal{A}$. $\mathcal{R}$ will maintain a list $\mathcal{L}$ of the queries he got (messages), the corresponding digital signatures and finally the signatures he issued.

Verification (Confirmation/Denial) Queries. For a signature μ on m, $\mathcal{R}$ will look up the list $\mathcal{L}$. If a record having as first component the message m and third component μ appears in the list, then $\mathcal{R}$ will execute the confirmation protocol, otherwise, he will run the denial protocol. This simulation differs from the real one when the signature μ is valid and has not been obtained from a signature query. Thus, μ will correspond to a valid existential forgery of the undeniable signature scheme in question[4]. Hence, the probability that this scenario does not happen is at least $(1 - \epsilon')^{q_v}$ because the undeniable signature scheme is (t, ϵ', q_s)-EUF-CMA secure by assumption. Finally, $\mathcal{R}$ can issue such proofs of knowledge, without knowing the private key of $\mathcal{K}$, using the rewinding technique because the protocols are zero knowledge, thus simulatable.

Challenge. Eventually, $\mathcal{A}$ outputs a challenging message $m^\star$. $\mathcal{R}$ will use his challenge $(c^\star, k^\star)$ to compute a digital signature using $\Sigma.\mathsf{sk}$ on $m^\star\|c^\star$. Then he encrypts the resulting signature using $k^\star$ and outputs the result $\mu^\star$ to $\mathcal{A}$. Therefore $\mu^\star$ is either a valid signature on $m^\star$ or a random element from the (undeniable) signatures space ($k^\star$ is random according to 2.4 and the DEM is IND-OT), which conforms to the game rules defined in 3.2.

Phase 2 $\mathcal{A}$ will continue issuing queries to the signing, confirmation and denial oracles and $\mathcal{R}$ can answer as previously.

[4] This is the reason for generating a signature on the message in question concatenated with the encapsulation. In fact, valid signatures can only be obtained from the signing oracle (under the assumption that the scheme is EUF-CMA secure) even if the underlying cryptosystem offers the possibility of generating a different ciphertext for the same message (e.g., ElGamal [9]).

Final Output
When $\mathcal{A}$ outputs his answer $b \in \{0,1\}$, $\mathcal{R}$ will forward this answer to his own challenger. Therefore $\mathcal{R}$ will $(t + q_s q_v, \epsilon \cdot (1-\epsilon')^{q_v})$-IND-CPA break Γ. □

5 Construction of UCUS from Certain Pairing-Based Signatures Using KEMs

In the generic construction proposed in [4], the confirmation/denial protocols involve proofs of knowledge of the decryption of the undeniable signature and that this decryption is a digital signature on some known data. Therefore, one needs to consider a set of cryptosystems and signatures for which such proofs could be performed efficiently. One solution to achieve this is to consider the following class of signatures (KEMs).

5.1 Defining the Class $\mathbb{C}_1$ of Signatures and $\mathbb{K}$ of KEMs

Definition 4. $\mathbb{C}_1$ *is the set of pairing-based signatures such that:*

1. *The considered pairing e is from $\mathbb{G} \times \mathbb{G}$ to $\mathbb{H}$.*
2. *The signature σ on a message m is written as $\sigma = (S, \bar{\sigma})$ such that*
   (a) *$\bar{\sigma} = \sigma \backslash S$ reveals no information about m nor about $(\mathsf{sk}, \mathsf{pk})$ the key pair related to the given signature scheme.*
   (b) *$S \in \mathbb{G}$ and the verification equation of the signature is of the form: $e(S, P) = f(\bar{\sigma}, m, PP)$.*

   where P is a known generator of the group $\mathbb{G}$ (set as a public parameter of the scheme), f is a public function, m is the message in question and PP are the known public parameters of the signature scheme

The definition above may seem too restrictive but it already captures two very important pairing-based signatures, namely BLS [2] (where the message-key-independent part is the empty string) and Waters' [14] signatures.

Definition 5. $\mathbb{K}$ *is the set of KEMs such that:*

1. *The KEM is implemented in a bilinear group $\mathbb{G}$ where the considered pairing e is from $\mathbb{G} \times \mathbb{G}$ to a group $\mathbb{H}$.*
2. *P is a known generator of the group $\mathbb{G}$.*
3. *The session keys space K is the same as the group $\mathbb{G}$.*
4. *Let $k \in \mathbb{G}$ be an element and c a given encapsulation. On common input $e(k, P)$ and c:*
   - *If k is the decapsulation of c, then there exists an efficient zero-knowledge proof $\mathcal{C}$ of this assertion, using the private key of the KEM,*
   - *otherwise, there exists an efficient zero-knowledge proof $\mathcal{D}$ of k not being the decapsulation of c (using also the private key of the KEM).*

A KEM in the Class $\mathbb{K}$:

- setup. Consider a bilinear group $\mathbb{G}$, with prime order d, generated by P.
- keygen. Generate two values $x_1, x_2 \in \mathbb{Z}_d^\times$ and compute $X_1 = x_1 P$ and $X_2 = x_2 P$. Set the private key to $\mathsf{sk} = (x_1, x_2)$ and the public key to $\mathsf{pk} = (X_1, X_2)$.
- encap. On input a nonce $(a, b) \in_R \mathbb{Z}_d^2$ and pk, generate the *session key* $k = (a + b)P$ and its *encapsulation* $c = (aX_1, bX_2)$.
- decap. Given sk and $c = (aX_1, bX_2)$, compute k as $k = x_1^{-1} aX_1 + x_2^{-1} bX_2$.

This KEM is IND-CPA secure assuming the intractability of the *Decision Linear Problem*.

Definition 6. ***Decision Linear Problem (DLP).*** *Given U, V, H, aU, bV, $cH \in \mathbb{G}$, output 1 if $a + b = c \bmod (\#\mathbb{G})$ and 0 otherwise.*

The traditional DDH problem (corresponding to $b = 0$) can be reduced to DLP. In fact, DLP is believed to be hard even in bilinear groups where DDH is easy.

Fact 1 *The KEM described above is in the class $\mathbb{K}$.*

Proof. – X_1 is a generator of $\mathbb{G}$.
– The proof $\mathcal{C}$ ($\mathcal{D}$) consists of the proof of equality (inequality) of the discrete logarithm of X_2 in base P and of $e(bX_2, X_1)$ in base $e(k, X_1)e(aX_1, P)^{-1}$. We refer to [7] ([6]) for the proof of equality (inequality) of two discrete logarithms. □

5.2 Construction

Following the notations in 5.1 we consider an EUF-CMA digital signature scheme $\Sigma \in \mathbb{C}_1$ and an IND-CPA secure KEM $\mathcal{K} \in \mathbb{K}$, where the considered groups $\mathbb{G}$ and H, and the generator P are the same for both Σ and $\mathcal{K}$. We assume that the proofs $\mathcal{C}$ and $\mathcal{D}$ are known to the signer. A universally convertible undeniable signature, on a given message m, can be obtained by first invoking $\mathcal{K}$ to fix a key k and its encapsulation c, then generating a digital signature $\sigma = (S, \bar{\sigma})$ on $m\|c$. The result is $\mu = (\mu_1, \mu_2, \mu_3) = (c, S + k, \bar{\sigma})$ [5]. Confirmation or denial of such a signature are achieved via the proofs $\mathcal{C}$ or $\mathcal{D}$ respectively, on the common input m, μ_1 and $e(\mu_2, P)f(\mu_3, m\|c, PP)^{-1}$. In fact, if $k = \mathcal{K}.\mathsf{decap}(\mu_1)$ and $e(k, P) = e(\mu_2, P)f(\mu_3, m\|c, PP)^{-1}$, then the signer issues $\mathcal{C}$ (using the private key of the KEM). Otherwise, if $k = \mathcal{K}.\mathsf{decap}(\mu_1)$ and $e(k, P) \neq e(\mu_2, P)f(\mu_3, m\|c, PP)^{-1}$, he issues the proof $\mathcal{D}$. Finally, the universal conversion is done by releasing $\mathcal{K}.\mathsf{sk}$.

Unforgeability of such a construction is easily guaranteed by virtue of Theorem 1. As far as invisibility is concerned, we can base it directly on the underlying KEM. In fact, since $\bar{\sigma}$ does not reveal any information about the signing/verifying key (of the digital signature scheme) nor about the message in question, an attacker $\mathcal{A}$ capable of deciding on the validity of a given undeniable signature must definitely use information leaked by the encryption of the remaining part of the signature, that is $(c, k + S)$. Due to page limitation, the complete proofs will be given in the full version of the paper.

[5] The DEM encryption algorithm consists in adding the key to the message, whereas the decryption is the addition of the key inverse (in $\mathbb{G}$) to the ciphertext.

Theorem 3. *Let $\mathcal{A}$ be a (t, ϵ, q_s)-EUF-CMA adversary against the above construction. Then, there exists a (t, ϵ, q_s)-EUF-CMA adversary against the underlying digital signature scheme.* □

Theorem 4. *Our proposed construction is (t, ϵ, q_s, q_v)-INV-CMA secure if it is (t, ϵ', q_s)-EUF-CMA secure and the underlying KEM is $(t + q_s q_v, \epsilon.(1 - \epsilon')^{q_v})$-IND-CPA secure.* □

Instantiation of our framework with Waters' signatures [14] and the KEM described above results in a very efficient universally convertible undeniable signature scheme. In fact, the best scheme that was proposed so far [15] achieves the same security features (standard model and the same underlying standard assumptions), and thought it presents the additional quality of selective conversion, it has a longer signature and a higher signature generation and verification cost (approximately a multiplicative parameter k) and a higher key generation and universal conversion cost (a multiplicative parameter $2^{n/k}$), where k is a public parameter to be optimized and n is the length of the message to be signed.

6 Toward a Generic Construction of UCUS from Pairing-Based Signatures

In this section, we give the first generic construction of universally convertible undeniable signatures from a large class of pairing-based signatures, denoted $\mathbb{C}_2$, and from any IND-CPA cryptosystem whose decryption is efficiently verifiable.

6.1 Generic Construction

Definition 7. *$\mathbb{C}_2$ is the same set of signatures defined in Definition 4 with the exception of the verification equation being of the form $e(S, E) = f(\bar{\sigma}, m, PP)$, where $E \in \mathbb{G}$ is not necessarily a fixed generator of $\mathbb{G}$.*

It is clear that this class of signatures captures a large category of pairing-based signatures. In fact, almost all (pairing-based) signatures [2,1,16,14], that have been proposed so far, involve a pairing computation in the verification equation, between the key-message-dependent part of the signature and other entities. Note that the key-message-independent part in [2,16] is the empty string.

Proposed Construction. Let $\Sigma \in \mathbb{C}_2$ be an EUF-CMA signature from $\mathbb{C}_2$ and Γ be an efficient decryption verifiable IND-CPA cryptosystem. Let further d denote the group order of $\mathbb{G}$ and p a suitable integer such that Γ is IND-CPA secure in $\mathbb{Z}_p$ (the message space of Γ is included in $\mathbb{Z}_p$). Note that $p > d$ due the contrast of key sizes between finite-field (or ring) and elliptic-curve cryptography.

We devise a universally convertible undeniable signature scheme as follows. First, we choose $r \in_R \mathbb{Z}_p$ then encrypt it under Γ to result in $s = \Gamma.\mathsf{encrypt}_{\Gamma.\mathsf{pk}}(r)$. Then, generate a digital signature $(S, \bar{\sigma})$ on the message to be signed m concatenated with s. The signature consists of the triple $\mu = (s, rS = (r \bmod d)S, \bar{\sigma})$.

To confirm (deny) a signature $\mu = (s, rS, \bar{\sigma})$, the signer decrypts s then proves the equality (inequality) of the decryption of s and the discrete logarithm of $e(rS, E)$ in base $f(\bar{\sigma}, m \| s, PP)$. Finally, the universal conversion is achieved by releasing $\Gamma.\mathsf{sk}$.

Theorem 5. *Let $\mathcal{A}$ be a (t, ϵ, q_s)-EUF-CMA adversary against the above construction. Then, there exists a (t, ϵ, q_s)-EUF-CMA adversary against the underlying digital signature scheme.* □

Theorem 6. *Our proposed construction is (t, ϵ, q_s, q_v)-INV-CMA secure if it is (t, ϵ', q_s)-EUF-CMA secure and the underlying cryptosystem is $(t + q_s q_v, \epsilon.(1 - \epsilon')^{q_v})$-IND-CPA secure.* □

Efficient realizations using this technique could be obtained by combining Waters' signatures [14] with an IND-CPA cryptosystem such as ElGamal [9] or Paillier [13].

7 Conclusion

In this paper, we proposed a construction for universally convertible undeniable signatures from secure digital signatures and some weakly secure cryptosystems. Next, we designed two efficient generic constructions for undeniable signatures from a large class of pairing-based signatures. These constructions found practical instantiations with some known signatures and cryptosystems. It might be good to analyze the security of the existing undeniable signature schemes or propose efficient ones using this technique. Finally, one is tempted to extend this approach to other "opaque" signatures such as directed signatures, or combine it with the techniques using commitment schemes in order to get better constructions.

Acknowledgments

I would like to thank the anonymous reviewers for their helpful comments. Thanks go also to Joachim von zur Gathen for suggestions that improved the quality of the paper. This work was supported by the B-IT foundation.

References

1. Boneh, D., Boyen, X.: Short Signatures Without Random Oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004)
2. Boneh, D., Lynn, B., Shacham, H.: Short Signatures from the Weil Pairing. J. Cryptology 17(4), 297–319 (2004)
3. Boyar, J., Chaum, D., Damgård, I.B., Pedersen, T.B.: Convertible undeniable signatures. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 189–205. Springer, Heidelberg (1991)

4. Rackoff, C., Simon, D.R.: Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 433–444. Springer, Heidelberg (1992)
5. Camenisch, J., Michels, M.: Confirmer Signature Schemes Secure against Adaptative Adversaries. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 243–258. Springer, Heidelberg (2000)
6. Camenisch, J., Shoup, V.: Practical Verifiable Encryption and Decryption of Discrete Logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (2003)
7. Chaum, D., Pedersen, T.P.: Wallet Databases with Observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993)
8. Chaum, D., van Antwerpen, H.: Undeniable Signatures. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 212–216. Springer, Heidelberg (1990)
9. El Gamal, T.: A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms. IEEE Trans. Inf. Theory 31, 469–472 (1985)
10. Goldreich, O., Micali, S., Wigderson, A.: How to Prove all NP-Statements in Zero-Knowledge, and a Methodology of Cryptographic Protocol Design. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 171–185. Springer, Heidelberg (1987)
11. Herranz, J., Hofheinz, D., Kiltz, E.: KEM/DEM: Necessary and Sufficient Conditions for secure Hybrid Encryption (August 2006), http://eprint.iacr.org/2006/265.pdf
12. Laguillaumie, F., Vergnaud, D.: Short Undeniable Signatures Without Random Oracles: the Missing Link. In: Maitra, S., Veni Madhavan, C.E., Venkatesan, R. (eds.) INDOCRYPT 2005. LNCS, vol. 3797, pp. 283–296. Springer, Heidelberg (2005)
13. 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)
14. Waters, B.: Efficient Identity-Based Encryption Without Random Oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)
15. Yuen, T., Au, M.H., Liu, J.K., Susilo, W. (Convertible) Undeniable Signatures Without Random Oracles. In: Qing, S., Imai, H., Wang, G. (eds.) ICICS 2007. LNCS, vol. 4861, pp. 83–97. Springer, Heidelberg (2007)
16. Zhang, F., Safavi-Naini, R., Susilo, W.: An Efficient Signature Scheme from Bilinear Pairings and Its Applications. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 277–290. Springer, Heidelberg (2004)

Concrete Security for Entity Recognition: The Jane Doe Protocol

Stefan Lucks[1], Erik Zenner[2], André Weimerskirch[3], and Dirk Westhoff[4]

[1] Bauhaus-Universität Weimar, Germany
http://medsec.medien.uni-weimar.de/
[2] Technical University of Denmark
http://www.erikzenner.name/
[3] escrypt Inc., USA
http://weimerskirch.org/
[4] NEC Europe Ltd
Dirk.Westhoff@nw.neclab.eu

Abstract. Entity recognition does not ask whether the message is from some entity X, just whether a message is from the same entity as a previous message. This turns turns out to be very useful for low-end devices. The current paper proposes a new protocol – the "Jane Doe Protocol" –, and provides a formal proof of its concrete security. The protocol neither employs asymmetric cryptography, nor a trusted third party, nor any key pre-distribution. It is suitable for light-weight cryptographic devices such as sensor network motes and RFID tags.

1 Introduction

Consider the following story: Two strangers meet at a party and make a bet. They introduce themselves as Jane and John Doe, which may or may not be their real names. Some days later, however, it turns out that Jane is the winner, and John receives a message: *"John, please transfer the prize to bank account [...] Thank you. Jane."*. How does John know that this message actually has been sent from that person, who had called herself "Jane" at that party? In other words, how does John recognise Jane – or a message from her?

Below, we will use the names Alice and Bob instead of Jane and John Doe for sender and receiver. As the protocol goal is about entity recognition, "real" names are unimportant. Alice and Bob are technical devices communicating in a hostile environment. Recognising each other would be easy if they could use unique identities and digital signatures: Initially, Alice would send Bob her public key. Later, Alice would sign all the messages she sends to Bob, and Bob would verify these signatures. But digital signatures are computationally expensive, and may seem an "overkill" to the problem at hand.

In this paper, we present the **Jane Doe protocol**, a light-weight solution to entity recognition using only symmetric primitives (namely, message authentication codes). Even low-end devices, which are too slow for digital signatures or the like, can run our protocol. The protocol does not depend on any trusted

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 158–171, 2008.

third party. Neither does it require a pre-established common secret key. It runs efficiently enough for real-time applications. In addition, it is interactive and provides information about the freshness and timeliness of messages.

Our research is motivated by the emergence of extremely low-power and low-cost devices such as sensor network motes and RFID tags. The continued desire to make these devices smaller at an attractive price offsets the technological advancements of increasing computational power. While implementing digital signatures and public-key techniques on such devices is technologically feasible, it is a hard burden from an economic viewpoint. Also, such devices are often used in networks where one can neither assume availability of a trusted third party, nor availability of pre-deployed secret or authentic information, and with a dynamic network topology. Another motivation is the question to what degree one can imitate the functionality of public-key cryptography and digital signatures by just using some simple primitives from symmetric cryptography. The Jane Doe protocol turns out to be as powerful as the common two step protocol for authenticating messages, consisting of a non-authenticated Diffie-Hellman key agreement at initialisation time followed by MAC authenticated messages.

Previous Work: The security goal of entity recognition has independently been proposed by a couple of different authors under different names [2,18,16,10,8].

An early protocol to actually address entity recognition was the **Resurrecting Duckling protocol** [17]. As it requires the exchange of a secret key in the initialisation phase, it does meet our security requirements. The **Guy Fawkes protocol** by [1] is more suitable for entity recognition, but it implicitly assumes Alice to know when Bob has seen her commitment a_i. While this may be the case in the original use case (Guy Fawkes would publish his commitments in a newspaper), an explicit confirmation of receipt may be desirable in most application contexts. The **Remote User Authentication protocol** [14] uses a message authentication code (MAC) and a cut-and-choose approach, which is much more demanding than our protocol. In [15], messages are authenticated using MACs, with a symmetric key being exchanged **using Diffie-Hellman key exchange** at protocol start. The problem here is that the key exchange requires public-key operations, which are too onerous for low-end systems. In the full paper [13], we provide a rough comparison of this approach with our proposal. The **zero-common-knowledge protocol** [18] from SAC 2003 uses hash chains, like our protocol, but turned out to be flawed [12,13].

2 Scenario Description

Sending messages: Alice is the sender of messages, Bob the receiver. All protocols start with an **initialisation phase**, where Alice and Bob for the first time contact each other and exchange some initial material. Later, messages are sent from Alice to Bob in distinct time frames, which we denote as **epochs**. There can be at most n such epochs. Each such epoch i consists of four basic steps:

1. Alice receives some external data x_i, the origin of which lies outside the scope of the protocol (e.g. a measurement from a sensor).

2. Alice authenticates and sends the message to Bob. Formally, we write *CommitMessage*(x_i,i).
3. Bob sends a confirmation that he received some data, supposedly from Alice.
4. Alice opens the commitment and proves that it was really her who send the message. We write *AcceptMessage*(x_i, i) if Bob believes the message x_i to be authentic and fresh in epoch i.

Adversary capabilities: The well-known Dolev-Yao model [7] assumes that Eve is in full control over the connection between Alice and Bob, i.e. she is an **active adversary**. In particular, she can

- read all messages sent from Alice or from Bob,
- modify messages, delay them or send them multiple times to Alice, Bob, or to both of them,
- and send messages generated by herself to Alice or Bob or both.

This is considered as reasonable pessimism: Over-estimating the adversary is not as bad as under-estimating her capabilities. However, e.g. Gollmann [9] argues that novel applications may need more specific models. In our case, we make the special assumption that during the initialisation phase, Eve behaves like a **passive adversary**. She can read the messages between Alice and Bob (which precludes any kind of secret key exchange), but she relays them faithfully. Note that this is a weakening of the usual assumption that Alice and Bob can use a protected communication channel for initialisation, i.e. our scenario requires less external protection than most other proposals.

In typical application scenarios, Eve may even be able to extract secret data inside the devices by tampering, in addition to controlling the network. Our protocol does not protect against this kind of threat. If this threat is relevant for the application at hand, and if it can not be mitigated by using tamper-resistant hardware, then additional protection measures (like introducing redundancy and using secure multi-party computation algorithms) have to be introduced.

Adversary goal: Driven by reasonable pessimism as before, we assume that Eve aims for an *existential forgery* in a *chosen message* scenario:

- Eve may have some influence on x_i. Thus, for purposes of security analysis, we allow her to choose messages x_i which Alice will authenticate and send, i.e. *CommitMessage*(x_i,i).
- She succeeds if Bob accepts any message $x' \neq x_i$ as authentic, i.e. *AcceptMessage*(x', i).

At the beginning of the protocol, Alice and Bob choose initial random values a_0 resp. b_0. From then on, Alice and Bob act as strictly deterministic machines. When receiving a message, Alice and Bob update their internal state and send a response, if necessary. Eve is a probabilistic machine with independent connections to Alice and to Bob. In the context of this paper, the actual choice of a machine model is not important – any reasonable machine model will do.

We require the initial random values (=keys) a_0 and b_0 to be chosen independently from the keys for other sessions. To this regard, our setting is much simpler than any communication scenario where *the same* key material can be used in more than one session (see e.g. [4,3]).

Limitation: We assume that the number of messages to be authenticated is known in advance, or a reasonable upper bound is known. During the initialisation phase, both Alice and Bob commit to the endpoint of a hash chain. The length of this hash chain bounds the number of messages to be authenticated. This limits of our approach, compared to other solutions employing public-key cryptography. Those, however, may be less efficient than our scheme, [13].

Reliability: Since Eve has full control over the connection between Alice and Bob, *denial of service* attacks are trivial for Eve. In addition, if the communication channel itself is unreliable, messages may be lost or faulty messages may be received even without the active involvement of a malicious adversary. Such problems can not be solved at cryptographical level, but have to be managed outside of the protocol. But the following reliability properties can be guaranteed:

Soundness: If the network is reliable and Eve behaves like a passive wire, the protocol works well: Bob accepts each message x_i Alice has committed to.

Recoverability: If Eve suppresses or modifies some messages, or creates some messages of her own, Bob may refuse to accept a message x_i Alice has committed to. However, once Eve begins again to honestly transmit all messages, like a passive wire, the soundness with respect to new messages is regained.

3 The Jane Doe Protocol

In this section, we describe the Jane Doe protocol to solve the entity recognition problem without using public-key cryptography. We write s for the size of a symmetric key. A second security parameter is the tag size $c \leq s$ for message authentication. (Typically: $s \geq 80$ and $c \geq 32$.) We use two functions, a MAC $m : \{0,1\}^s \times \{0,1\}^* \to \{0,1\}^c$ and a one-way function $h : \{0,1\}^s \to \{0,1\}^s$. (In Section 4, we will describe how to derive both m and h from a single MAC.) We write $x \in_{\mathrm{R}} \{0,1\}^s$ to indicate a random s-bit string x, uniformly distributed.

Initialisation phase: For initialisation, Alice chooses $a_0 \in_{\mathrm{R}} \{0,1\}^s$ and generates a hash chain $a_1 := h(a_0), \ldots, a_n := h(a_{n-1})$. Similarly, Bob chooses $b_0 \in_{\mathrm{R}} \{0,1\}^s$ and generates $b_1 := h(b_0), \ldots, b_n := h(b_{n-1})$. When running the protocol, both Alice and Bob learn some values b_i resp. a_i from the other's hash chain. If Alice accepts b_i as authentic, we write $\mathit{AcceptKey}(b_i)$. Similarly for Bob and $\mathit{AcceptKey}(a_i)$. The initialisation phase, where Eve can read the messages but relays them faithfully, consists of two messages:

1. Alice → Bob: a_n. (Thus: $\mathit{AcceptKey}(a_n)$.)
2. Bob → Alice: b_n. (Thus: $\mathit{AcceptKey}(b_n)$.)

Message authentication: We split the protocol up into n epochs, plus the initialisation phase. The epochs are denoted by $n-1, \ldots, 0$ (in that order). Each epoch allows Alice to send one authenticated message[1], and Bob to receive and verify it. The internal state of each Alice and Bob consists of

- an epoch counter i,
- the most recent value from the other's hash chain, i.e., b_{i+1} for Alice, and a_{i+1} for Bob (we write $\mathit{AcceptKey}(b_{i+1})$ and $\mathit{AcceptKey}(a_{i+1})$), and
- a one-bit flag, to select between program states A0 and A1 for Alice resp. B0 and B1 for Bob.

Also, both Alice and Bob store the root a_0 resp. b_0 of their own hash chain.[2] This value does not change during the execution of the protocol. Note that after the initial phase, and before the first epoch $n-1$, Alice's state is $i = n-1$, $\mathit{AcceptKey}(b_n)$, and A0, and Bob's is $i = n-1$, $\mathit{AcceptKey}(a_n)$, and B0. One epoch i can be described as follows:

A0 (Alice's initial program state)
Wait for x_i (from the outside), then $\mathit{CommitMessage}(x_i,i)$:

1. compute $d_i = m(a_i, x_i)$ (using a_i as the key to authenticate x_i);
2. send (d_i, x_i); **goto** A1.

A1 Wait for a message b' (supposedly from Bob), then

1. **if** $h(b') = b_{i+1}$
   then $b_i := b'$; $\mathit{AcceptKey}(b_i)$; send a_i; set $i := i-1$; **goto** A0
   else goto A1.

B0 (Bob's initial program state)
Wait for a message (d', x') (supposedly from Alice), then

1. send b_i and **goto** B1.

B1 Wait for a message a' (supposedly from Alice), then

1. **if** $h(a') = a_{i+1}$ **then**
   (a) $a_i := a'$; $\mathit{AcceptKey}(a_i)$;
   (b) **if** $m(a_i, x') = d'$
   then $x_i := x'$; $\mathit{AcceptMessage}(x_i,i)$
   (else do not accept any message in epoch i);
   (c) set $i := i-1$; **goto** B0
   else goto B1

Figure 1 gives a simplified view on the protocol.

[1] Several messages can be sent per epoch. For ease of presentation, we combine them.
[2] Alice can either store a_0 and compute the a_i on demand by making i calls to h, or store all the a_i using n units of memory. Her third option is to implement a *time-storage trade-off*, requiring only about $\log_2 n$ units of memory and $\log_2 \sqrt{n}$ calls to h [6]. Similarly for Bob and the b_i.

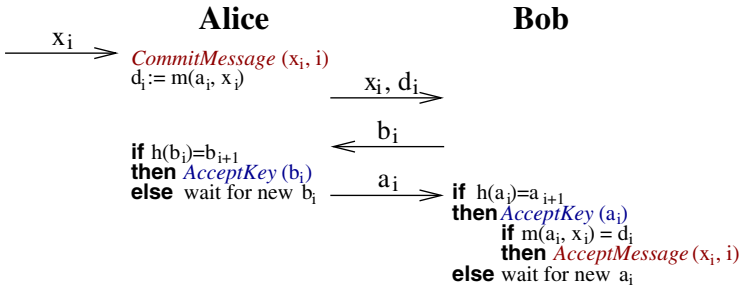


Fig. 1. Simplified description of one epoch of the protocol

Reliability: The following reliability properties are met:

Soundness: The protocol is **sound:** If all messages are faithfully relayed, Alice commits to the message x_i in the beginning of epoch i and Bob accepts x_i at the end of the same epoch.

Recoverability: Repeating old messages cannot harm security – Eve may know them anyway. We thus allow Alice to re-send a_{i+1} and (x_i, d_i) if she is in state A1 and has been waiting too long for the value b_i from Bob. Similarly, if Bob is in state B1 and has been waiting too long for a_i, Bob sends the value b_i again. This allows our protocol to recover. On the other hand, if Bob receives a faulty $(x', d') \neq (x_i, d_i)$, he will refuse to accept *any* message in epoch i. Recovering means that soundness can be restored in epoch $i-1$.

4 Security

4.1 Building Blocks and Assumptions

The main cryptographic building block in this paper is a MAC

$$m^* : \{0,1\}^s \times \{0,1\}^* \to \{0,1\}^s$$

We fix some constant message const and define the two functions m and h we actually use in the protocol

$$h : \{0,1\}^s \to \{0,1\}^s, \quad h(k) = m^*(k, \text{const}), \quad \text{and}$$

$$m : \{0,1\}^s \times \{0,1\}^* \to \{0,1\}^c, \quad m(k,x) = \text{truncate-to-}c\text{-bit}(m^*(k,x)).$$

In the case of m, a restriction is $x \neq$ const. If neccessary, we we can, e.g., define const as a single zero-bit, and prepend a single one-bit to every message x.

Security against adaptive chosen message attacks has been established as a standard requirement for MACs:

Assumption 1. *It is infeasible for the adversary to provide an* ***existential forgery*** *in an* ***adaptive chosen message attack*** *scenario against* m^**. I.e., the adversary is given access to an authentication oracle, computing* $t_i = m(y, x_i)$ *for the adversary, where* $y \in_{\mathrm{R}} \{0,1\}^s$ *is secret and the adversary is allowed to choose arbitrary messages* x_i*. "Adaptive" means that the adversary is allowed to choose* x_i *after having seen* t_{i-1}*. The adversary wins if she can produce a pair* (x', t') *with* $m(y, x') = t'$*, without previously asking the oracle for* $m(y, x')$*.*

Unfortunately, this standard assumption is not quite sufficient for our purposes. Below, we will not make use of assumption 1 at all, but instead, define two similar assumptions. Firstly, we use m instead of m^* as a MAC, i.e., the truncation of m^* to $c \leq s$ bit. The security of m does not follow from the security of m^*. So we need to make the same assumption for m instead of m^*:

Assumption 2. *It is infeasible for the adversary to provide an* ***existential forgery*** *in an* ***adaptive chosen message attack*** *scenario against* m*.*

Furthermore, we use h to build a hash chain, which implies that h must be one-way. It may be surprising, but m^* being secure against existential forgery is not sufficient for the one-wayness of $h = m^*(\cdot, \mathrm{const})$. If, given $k^* = h(k) = m^*(k, \mathrm{const})$, the adversary can find the secret k, then she can forge messages. But the adversary could just as well find some value $k' \neq k$ with $k^* = h(k') = m^*(k', \mathrm{const})$ without necessarily being able to to generate existential forgeries. We thus need to exclude this case:

Assumption 3. *The function* m^* *is* ***one-way****. I.e., given a random* $k \in \{0,1\}^s$*, and a message const, it is infeasible to find any* $k' \in \{0,1\}^s$ *with* $m^*(k, const) = m^*(k', const)$*.*

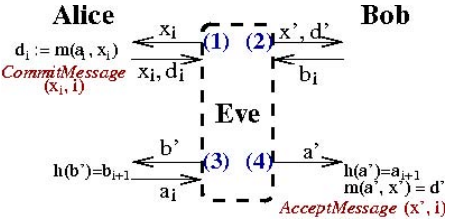
Note that inverting m^* (i.e., breaking the one-wayness of h) would either allow us to find a secret key and thus to forge messages, or provide a 2nd preimage, i.e., a value $k' \neq k$ with $h(k) = h(k')$. Indeed, for our formal proof of security we could replace assumption 3 by assuming 2nd preimage resistance. The proof would be slightly more complicated, though.

4.2 Proving Security for Epoch 0

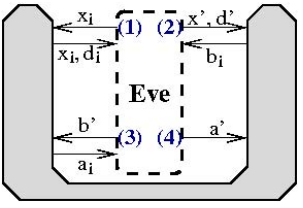
Theorem 1. *If the adversary can efficiently break epoch* 0 *of the protocol, she can efficiently break either assumption 2 or assumption 3.*
Concrete security. If she can break the protocol in time t *with probability* p*, she can either invert* h *or forge a message for* m *in time* $\leq t + 2t^*$ *with probability* $p/2$*. Here,* t^* *is the time to evaluate either* h *or* m*, which ultimately boils down to the time for evaluating* m^**.*

Proof. Eve can send the following messages (see also left side of Figure 2):

(1) If Alice's program state is A0: x_0 to Alice.
Alice responds $d_0 := m(a_0, x_0)$ (and x_0, but x_0 is known to Eve, anyway).



Left: The four types of messages



Right: Eve, connected to some game

Fig. 2. Eve in epoch i

(2) If Bob's program state is B0: (x', d') to Bob – with $x' \neq x_0$.
(3) If Alice's program state is A1: b' to Alice – with $h(b') = b_1$.
(4) If Bob's program state is B1: a' to Bob – with $h(a') = a_1$.

Remember that she is successful if she gets Bob to *AcceptMessage*(x',i) for a message x' that Alice has not send in epoch i.

Note that (3)-like messages b' with $h(b') \neq b_1$ to Alice do not affect Alice's state; Alice ignores them. Since Eve can check $h(b') = b_1$ on her own, we assume w.l.o.g. Eve not to send any message b' with $h(b') \neq b_1$ to Alice. Similarly, for (4)-like messages, we assume, Eve not to send any a' with $h(a') \neq a_1$ to Bob.

In order to successfully attack, Eve *must* send *exactly* one message (1) to Alice (to ensure *CommitMessage*$(x_0, 0)$) and both messages (2) and (4) to Bob (for *AcceptMessage*$(x', 0)$). Eve may send *at most* one message (3) to Alice. W.l.o.g., we assume Eve to send *exactly* one message (3). (If she wins her attack game without sending message (3), she has sent message (2) and did learn b_0 from Bob. She can always send a final message (3) with $b' = b_0$.)

While (1,2,3,4) is the protocol-defined "natural" order for sending the messages, Eve is not bound to this order. There are some restrictions though:

- Message (1) must be sent before message (3). Until she knows and has committed to x_0, Alice wouldn't even listen to message (3).
- Also, Bob wouldn't listen to (4) before having received (2).

In the context of this proof, we just need to distinguish between two cases, which we represent by two games: Either message (2) is sent before message (3), or the other way. Consider disconnecting Eve from Alice and Bob, and connecting her with either of two games (cf. right side of Figure 2). If we win such a game, we can either invert h or forge messages. We will show that Eve cannot distinguish her participation in such a game from the "real" attack against the protocol and show that a successful attack by Eve is essentially the same as us winning one of our games. So at the end, if Eve can feasibly attack the protocol, we can feasibly invert $h(\cdot) = m(\cdot, \text{const})$ or forge messages for m^*. The games are the following:

1st game (inverting h): Given $k^* = h(k) = m^*(k, 0)$, for a uniformly distributed random k, find some k' with $m^*(k', 0) = k^*$.

- Randomly choose a_0, compute $a_1 := h(a_0)$.
- If Eve sends message
  - (1), the value x_0: compute and respond $d_0 := m(a_0, x_0)$.
  - (2): abort the game.
  - (4): Report an error! (Message (2) must be sent before message (4), and this algorithm aborts after message (2).)
- When Eve sends (3), the value b': print $k' := b'$ and stop.

The values provided to Eve during the 1st game are distributed exactly as in the case of the real attack game. Namely, a_0 and b_0 are independent uniformly distributed random values, and all the other values are derived from a_0 and b_0. Note that if Eve sends message (3) before message (2), the game succeeds; else it doesn't. To compute a_1, we call h. To compute d_0, we call m. Thus, we need two function calls. As Eve herself runs in time t, the game takes time $t + 2t^*$.

2nd game (existential forgery for m): Consider an unknown random y, known $y^* = h(y)$, and the ability to ask an oracle for $m(y, \cdot)$. Proceed as follows.

- Set $a_1 := y^*$; randomly choose b_0; compute $b_1 := h(b_0)$.
- If Eve sends message
  - (1), the value x_0: ask the oracle for the response $d_0 = m(y, x_0)$.
  - (3): abort the game.
  - (4): Report an error!
- When Eve sends (2), the pair (x', d'): print (x', d') and stop.

Eve's attack succeeds if and only if (x', d') is an existential forgery.

Similarly to above, the distribution of values provided during the game is identical to the real attack game. The only computation during the game is the one for $b_1 := h(b_0)$, so the game needs time $t + t^* \leq t + 2t^*$.

Completing the proof: The 1st game is the counterpart of the second game: one succeeds if message (2) is sent before message (3), the other one, if message (3) is sent before message (2). Eve doesn't know which game we play – or rather, that we are playing games with her at all, instead of mounting the "real" attack. So Eve still succeeds with probability p. If we randomly choose the game we play, we succeed with $p/2$. Neither game takes more than time $t + 2t^*$. □

4.3 Security in Any Epoch i

At a first look, it may seem that the security proof for epoch 0 is also valid for epochs $i > 0$. But in epoch 0, the keys for the MAC m^* are uniformly distributed random values a_0 and b_0 in $\{0,1\}^s$, while later, we use a_i and b_i:

- Our security assumptions for m^* require *uniformly distributed* random keys.
- Our security assumptions for m^* do not ensure the *uniform distribution* of the output values $a_i = h(a_{i-1}) = m^*(a_{i-1}, 0)$ and $b_i = \ldots$

Now m^* could be defined such that the one-way function $h(x) = m^*(x, 0)$ permutes over $\{0,1\}^s$. This would solve our problem, but restrict our choices m^* too much. In practice, however, most cryptographic MACs can reasonably be assumed to behave *pseudorandomly*. Thus, we make an additional assumption.

Let $u \in_R \{0,1\}^s$ be a random variable chosen according to the uniform distribution. Let w be a random variable chosen by applying the function h to a uniformly distributed input, i.e., $v \in_R \{0,1\}^s$, and $w := h(v)$. Let A be a distinguishing adversary for u and w. The advantage Adv_A of A in distinguishing u from w is defined in the usual way:

$$\mathrm{Adv}_A = \big|\Pr[A(u) = 1] - \Pr[A(w) = 1]\big|$$

Assumption 4. *No efficient adversary A can feasibly distinguish the distribution of the random variable $w = h(v)$, $v \in_R \{0,1\}^s$, from the distribution of $u \in_R \{0,1\}^s$. I.e., for all efficient A the advantage Adv_A is negligible.*

Recall that h is defined by $h(\cdot) = m^*(\cdot, \mathrm{const})$. For typical MACs m^*, this assumption is highly plausible.

We use assumption 4 to prove the pseudorandomness of values $a_1 := h(a_0)$, $\ldots$, $a_n := h(a_{n-1})$ for a random a_0, along an entire hash chain.

Lemma 1. *If, for any $i \in \{1, 2, \ldots, n-1\}$, the adversary can efficiently distinguish a_i from a_{i-1}, she can also distinguish a_1 from a_0, thus breaking Assumption 4.*
Concrete security. Let $i \in \{1, 2, \ldots, n-1\}$ be given. If the adversary can distinguish a_i from a_{i-1} in time t with an advantage α, she can distinguish a_1 from a_0 im time at most $t + (i-1) * t^*$ with the same advantage α. Here, t^* is the time for evaluating h.

Proof. Let a value r_0 be given, either distributed like a_0 or like a_1. Compute $r_1 := h(r_0) \ldots, r_{i-1} := h(r_{i-2})$. Now, r_{i-1} is either distributed like a_{i-1}, or like a_i, and we can distinguish between both options for r_{i-1} in the same time and with the same advantage as for a_{i-1} and a_i. Computing r_{i-1} takes at most $i-1$ calls to h. □

One more issue has to be taken into account. In the single-epoch case, we argued that finding 2nd preimages, i.e., values $a' \neq a_i$ with $h(a') = h(a_i) = a_{i+1}$ when given a_i, is infeasible under our assumptions. But when dealing with more than one epoch, Eve might possibly trick Alice into committing to some new message x_{i-1} and sending $d_i := m(a_{i-1}, x_{i-1})$ – even before Bob has seen a_i (see below). In contrast to an ordinary 2nd preimage attack, Eve now does not just know a_i, but she also has some additional information about a_{i-1}. Driven by the usual reasonable pessimism, we even assume Eve to know a_{i-1} itself. We consider finding an $a' \neq a_i$ with $h(a') = h(a_i) = a_{i+1}$ as a *guided 2nd preimage*. Theoretically, such guided 2nd preimages might be possible, even under all the assumptions we made so far. Thus, we make one additional assumption.

Assumption 5. *It is infeasible to find* guided 2nd preimages *for h. I.e., given $a_0 \in_R \{0,1\}^s$, $a_1 = h(a_0)$, and $a_2 = h(a_1)$, it is infeasible to find any $a' \neq a_1$ with $h(a') = a_2$.*

Recall that the adversary wins in epoch i if she can make Alice to *CommitMessage*(x_i,i) and Bob to *AcceptMessage*(x', i) for any $x' \neq x_i$.

Theorem 2. *If there is any epoch $i \in \{0,\ldots,n-1\}$ in which the adversary can feasibly win with significant probability, at least one of the assumptions 2, 3, 4, or 5 is false.*
Concrete security. If she can win in epoch i, in time t with probability p, she can either invert h, forge a message for m, or generate a guided 2nd preimage for h in time $\leq t + 2t^*$ with probability $p/4$. Or she can distinguish (a_i, b_i) from (a_{i-1}, b_{i-1}) with advantage $p/4$. Here, t^* is the time for calling either h or m, which ultimately boils down to calling m^*.

Proof. We say, the protocol in a "synchronised state", if there is an $i \in \{0,\ldots,n\}$ such that Bob knows a_i but not a_{i-1}, while Alice knows b_i but not b_{i-1}. I.e., the protocol is in a synchronised state if both Alice and Bob are in the same epoch $i-1$. After the initialisation, both are in epoch $n-1$, hence the protocol is in a synchronised state.

For the proof, we need to analyse independently how Eve can benefit from *non-synchronised states*, and how she can benefit from *synchronised states*.

Non-synchronised states: Consider Alice and Bob to be in epoch i, thus the protocol state is synchronised. Alice will not move forward into epoch $i-1$ without having seen b_i with $h(b_i) = b_{i+1}$. If Eve could provide such a b_i without obtaining it from Bob, she could win in epoch i anyway. Thus we can safely assume that Alice does not move forward before Bob sends b_i. For the same reason, we may assume Bob not moving forward to epoch $i-1$ without having seen a_i from Alice. Bob only sends b_i *after* having seen a_i from Alice. Thus, Bob can never be ahead of Alice. Temporarily, Alice can be ahead of Bob – especially if Eve does not forward a_i to Bob. This would give a protocol state with Alice living in epoch $i-1$ while Bob still lives in epoch i. But without having seen b_{i-1}, Alice cannot move ahead into epoch $i-2$, and Bob does not send this while he is still in epoch i.

At this point, Eve has but two options to proceed. One is to forward a_i to Bob, thus creating a new synchronised state. The second is to choose a message x_{i-1} and send it to Alice, who responds with the authentication tag $d_{i-1} = m(a_{i-2}, x_{i-1})$. If, after sending x_{i-1} to Alice, Eve sends the value a_i to Bob which she has seen before, there is no gain for Eve. The order of messages has changed, but the messages are the same, anyway. To benefit from the second option, Even has to send a value $a' \neq a_i$ with $h(a') = h(a_i) = a_{i+1}$ to Bob. If Eve could find such a value a', she could find guided 2nd preimages, thus breaking assumption 5.

Synchronised states: Now consider both Alice and Bob being in some epoch i, and Eve trying to win in this epoch. This part of the proof is done by induction.

We start with epoch 0. Recall that if both assumption 2 and assumption 3 hold, the adversary cannot feasibly win in epoch 0.

Now assume that no efficient adversary can win in epoch epoch (i-1), but there is an efficient algorithm to win epoch i with significant probability. Clearly, we can use this algorithm to distinguish (a_{i-1}, b_{i-1}) from (a_{i-2}, b_{i-1}), thus breaking assumption 4.

Concrete security (sketch): This part is quite similar to the proof of theorem 1, the single-epoch case. Instead of two different games, we need to define four:

1. One game to invert h (like the 1st game in the proof of theorem 1).
2. One game to forge messages for m (like the 2nd game above).
3. One game to generate guided 2nd preimages for h.
4. One game to distinguish (a_{i-1}, b_{i-1}) from (a_{i-2}, b_{i-1}).

If Eve wins, we succeed in at least one of the games. Which game we succeed in depends on Eve's behaviour. As we must commit to one game in advance (i.e. before we know how Eve behaves), the probability of success decreases from p (for Eve) to $p/4$ (for us). □

5 Final Remarks and Conclusion

The Jane Doe protocol does not provide security against *denial of service attacks*. I.e., if Eve sends a fake d_i in epoch i, Bob will send b_i and then not accept the "real" d_i Alice may later send.

Freshness means that a message has been committed to recently. In our case, when Bob accepts message x_i in epoch i, he can be sure that Alice (following the protocol rules) did not commit to that message before she had seen and verified Bob's response b_{i+1} from the previous epoch. In this sense, our protocol ensures the freshness of the messages authenticated.

The messages are "fresh" by belonging to the current epoch. But Eve is able to stretch any epoch at her will. Assume, e.g., that Alice commits to a message $m_i =$"I am well", but Eve delays forwarding $d_i = m(a_i,$ "all is well") to Bob. Later, Alice would need to raise an alarm, but instead Eve forwards d_i to Bob who sends b_i, which Eve immediately forwards to Alice. The protocol logic would require Alice to reply a_i, thus confirming that she is well. Instead of confirming such an outdated message, Alice could simply terminate communication with Bob. Eve has the power to cut the communication between Alice and Bob, anyway, and Bob will eventually notice that Alice doesn't respond any more.

Assuming some underlying primitive (from which we derive m^*) *to behave like a random oracle* is theoretically sound and would allow us to greatly simplify our security proofs. But in practice, cryptographic primitives never behave like random oracles. Results in the random oracle model hardly provide any guideline for the choice of a good primitive. Our very specific standard model assumptions on m^* are meant to serve as such a guideline.

Note that we have two functions, a message authentication code (MAC) m and a hash function h, both of which are derived from another MAC m^*.

In principle, one could choose m and h independently from each other, without deriving them from the same underlying primitive, as has been suggested in [12]. Under appropriate assumptions, one can still prove the security of the Jane Doe protocol. This requires more complex and less natural assumtions than those made here. Even if m is a secure MAC and h is modelled as a random oracle, the protocol may actually be insecure[13]. Deriving both m and h from one single primitive m^* thus saves us from some difficult technical issues.

Furthermore, we believe that deriving both h and m from the same underlying primitive is natural and meets practical necessities very well.

Conclusions: Entity recognition is an adopted security goal especially useful for constrained pervasive applications. The Jane Doe protocol provides entity recognition. The protocol is efficient, runs on on very low-end devices, and is provably secure. We believe this to be a significant step into the direction of provably secure protocols for low-end devices.

References

1. Anderson, R., Bergadano, F., Crispo, B., Lee, J.-H., Manifavas, C., Needham, R.: A New Family of Authentication Protocols. ACM Operating Systems Review 32 (1998)
2. Arkko, J., Nikander, P.: Weak Authentication: How to Authenticate Unknown Principals without Trusted Parties. In: Proc. Security Protocols Workshop 2002 (2002)
3. Bellare, M., Rogaway, P.: Entity Authentication and Key Distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773. Springer, Heidelberg (1994)
4. Bird, R., Gopal, I., Herzberg, A., Janson, P., Kutten, S., Molva, R., Yung, M.: Systematic design of two-party authentication protocols. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576. Springer, Heidelberg (1992)
5. Buonadonna, P., Hill, J., Culler, D.: Active Message Communication for Tiny Networked Sensors. In: Proc. 20th Joint Conference of the IEEE Computer and Communications Societies. IEEE, Los Alamitos (2001)
6. Coppersmith, D., Jakobsson, M.: Almost Optimal Hash Sequence Traversal. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357. Springer, Heidelberg (2003)
7. Dolev, D., Yao, A.: On the Security of Public Key Protocols. IEEE Trans. Information Theory 29(2), 198–208 (1983)
8. Dielsma, P., Mödersheim, S., Vigano, L., Basin, D.: Formalizing and Analyzing Sender Invariance. In: Dimitrakos, T., Martinelli, F., Ryan, P.Y.A., Schneider, S. (eds.) FAST 2006. LNCS, vol. 4691. Springer, Heidelberg (2007)
9. Gollmann, D.: Protocol Design: Coming Down from the Cloud (Invited Talk). In: Workshop on RFID and Lightweight Crypto 2005 (2005), http://www.iaik.tugraz.at/research/krypto/events/
10. Hammell, J., Weimerskirch, A., Girao, J., Westhoff, D.: Recognition in a Low-Power Environment. In: Proc. ICDCSW 2005. IEEE, Los Alamitos (2005)
11. Hodjat, A., Verbauwhede, I.: The Energy Cost of Secrets in Ad-hoc Networks. In: IEEE Circuits and Systems workshop on wireless communications and networking. IEEE, Los Alamitos (2002)

12. Lucks, S., Zenner, E., Weimerskirch, A., Westhoff, D.: Entity Recognition for Sensor Network Motes. In: Proc. INFORMATIK 2005, vol. 2, pp. 145–149 (2005); LNI Vol. P-68, ISBN 3-88579-379-0 (an early 5-page abstract of the current research)
13. Lucks, S., Zenner, E., Weimerskirch, A., Westhoff, D.: Concrete Security for Entity Recognition: The Jane Doe Protocol (Full Paper), eprint, full version of the current paper
14. Mitchell, C.: Remote User Authentication Using Public Information. In: Paterson, K.G. (ed.) Cryptography and Coding 2003. LNCS, vol. 2898. Springer, Heidelberg (2003)
15. Russell, S.: Fast Checking of Individual Certificate Revocation on Small Systems. In: Proc. 15th Annual Computer Security Application Conference. IEEE, Los Alamitos (1999)
16. Seigneur, J.-M., Farrell, S., Jensen, C., Gray, E., Chen, Y.: End-to-end trust in pervasive computing starts with recognition. In: Hutter, D., Müller, G., Stephan, W., Ullmann, M. (eds.) Security in Pervasive Computing. LNCS, vol. 2802. Springer, Heidelberg (2004)
17. Stajano, F., Anderson, R.: The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In: Malcolm, J.A., Christianson, B., Crispo, B., Roe, M. (eds.) Security Protocols 1999. LNCS, vol. 1796. Springer, Heidelberg (2000)
18. Weimerskirch, A., Westhoff, D.: Zero Common-Knowledge Authentication for Pervasive Networks. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006. Springer, Heidelberg (2004)
19. Weimerskirch, A., Westhoff, D., Lucks, S., Zenner, E.: Efficient Pairwise Authentication Protocols for Sensor and Ad-hoc Networks. In: Sensor Network Operations. IEEE Press, Los Alamitos (2004)

Efficient and Strongly Secure Password-Based Server Aided Key Exchange
(Extended Abstract)

Kazuki Yoneyama*

The University of Electro-Communications
yoneyama@ice.uec.ac.jp

Abstract. In ACNS'06, Cliff et al. proposed the password-based server aided key exchange (PSAKE) as one of password-based authenticated key exchanges in the three-party setting (3-party PAKE) in which two clients with different passwords exchange a session key by the help of their corresponding server. Though they also studied a strong security definition of 3-party PAKE, their security model is not strong enough because there are desirable security properties which cannot be captured. In this paper, we define a new formal security model of 3-party PAKE which is stronger than the previous model. Our model captures all known desirable security requirements of 3-party PAKE, like resistance to key-compromise impersonation, to leakage of ephemeral private keys of servers and to undetectable on-line dictionary attack. Also, we propose a new scheme as an improvement of PSAKE with the optimal number of rounds for a client, which is secure in the sense of our model.

Keywords: password-based key exchange, password-based server aided key exchange, leakage of internal states, undetectable on-line dictionary attack.

1 Introduction

Recently, password-based authenticated key exchange (PAKE) protocols are received much attention as practical schemes in order to share a mutual session key secretly and reliably. Basic PAKE schemes enable two entities to authenticate each other and agree on a large session key from a human memorable password. Thus, PAKE schemes are regarded as practical key exchange schemes because entities do not have any pre-shared cryptographic symmetric key, certificate or support from a trusted third party. Such basic schemes which two entities pre-share a *common* password are classified into a model called same password-authentication (SPA) model. The SPA model is most cultivated PAKE model in previous studies and is usually used for *client-to-server* key exchanges. The concept of PAKE was first introduced by Bellovin and Merritt [1] in 1992 known as encrypted key exchange (EKE). First construction of password-only PAKE

* Supported by JSPS Research Fellowships for Young Scientists.

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 172–184, 2008.

in SPA model was proposed by Jablon [2] in 1996 known as simple password exponential key exchange (SPEKE). Formal definitions for this setting were first given by Bellare et al. [3] and Boyko et al. [4], and a concrete construction was also given in the random oracle (RO) model. And, various protocols have been proposed to achieve secure PAKE scheme in SPA model.

1.1 Password-Based Key Exchange in the 3-Party Setting

With a variety of communication environments such as mobile network, it is considered as one of main concerns to establish a secure channel between clients with *different* passwords. Several schemes have been presented to provide PAKE between two entities with their different passwords, called different password-authentication (DPA) model. Practically, clients prefer to remember very few passwords but not many. Consequently, PAKE in DPA model is useful to solve this problem. In DPA model, entities carry out key exchange with the assistance of intermediate server because entities have no secret common information. So, it is usually called password-based authenticated key exchanges in the three-party setting (3-party PAKE) and is usually used for *client-to-client* key exchanges.

Basic security requirements of 3-party PAKE are known-key security (KS) (i.e., the session key is not compromised in the face of adversaries who have learned some other session keys), basic impersonation (BI) (i.e., the adversary cannot impersonate any honest client to the other client of the session without the client's password), and resistance to off-line dictionary attacks (offDA). The resistance to offDA means that there is no successful adversary as follows: The adversary guesses a password and verifies his guess off-line. No participation of the server is required, so the server do not notice the attack. If his guess fails the adversary tries again with another password, until he finds the proper one.

Though 3-party PAKE has been considered in early papers [5,6], these schemes assume trusted intermediate server perfectly because the server can know the session key of clients. Several works [7,8,9] considered *key privacy* against passive server (KP) (i.e., a semi-honest server cannot know information of the session key of clients). However, none of their schemes enjoys provable security. Indeed, a scheme [7] is known to be vulnerable to an undetectable on-line dictionary attack (UDonDA) [10]. The central idea of UDonDA is that an attacker guesses a password of a client, completes some computations with it and sends the server the result as a part of his request for a session key. Then, if the server cannot tell this request from the request from honest clients, the server performs some further computations on the result using the correct password of the client and responses. This response helps the attacker to verify his guess. So, the server is used as an oracle without taking notice of the attack.

First formal security definition of 3-party PAKE (AFP model) was proposed by Abdalla et al. [11]. They also provided a generic method to construct provably secure 3-party PAKE protocol from 2-party PAKE. To reduce the complexity of generic construction, the first concrete protocol of provably secure 3-party PAKE protocol in the random oracle model is proposed in [12]. Wang and Hu pointed out that schemes in [11,12] are vulnerable to UDonDA, and provided a

stronger definition of 3-party PAKE (WH model) which captures resistance to UDonDA.

Cliff et al. [13] proposed another security definition of 3-party PAKE (CTB model) which is an extension of the Canetti-Krawczyk model [14] for 2-party AKE. Also, they proposed a variant of 3-party PAKE, called the password-based server aided key exchange (PSAKE), which has the similar setting of 3-party PAKE except the server uses password and encryption based authenticators. The encryption based authenticator in PSAKE means that a client has the server's public-key as well as the password between with the server, and the server has his private-key as well as clients' passwords. They also prove security of PSAKE in the standard model, i.e., without random oracle. By helping of the encryption based authenticator, PSAKE has strong security which cannot be prevent in password-only setting 3-party PAKE schemes. Indeed, PSAKE seems to be secure against leakage of ephemeral private keys of servers (LEP) (i.e., even if all the session specific ephemeral private key of *the server* in a session is compromised, then secrecy of the session key is not compromised)[1] and against key-compromise impersonation (KCI) (i.e., when a client's password is compromised, this event does not enable an outside adversary to impersonate other entities to the client).

1.2 Need for New Security Models

AFP model, CTB model and WH model formalize indistinguishability of session keys against outside adversaries. However, each model has some uncaptured security requirement, respectively. For example, AFP model and WH model cannot grasp the notion of forward secrecy (FS) (i.e., secrecy of the past session keys after leakage of passwords). Also, CTB model and WH model cannot grasp KP. Furthermore, in AFP model and CTB model, resistance to UDonDA is out of scope. In addition, there are some security requirements which is not captured in these models (see Section 2.2).[2] Indeed, schemes in [11,15] are insecure against LEP because these schemes include 2-party PAKE between a client and a server. In 2-party PAKE, if an ephemeral private key of either party is leaked, then the password of the party is easily derived by offDA because the session key deterministically depends on the client's ephemeral key, static password, and communication received from the other party. Thus, the secrecy of the session key is not guaranteed. Therefore, by LEP the temporary session key is revealed and schemes in [11,15] are clearly insecure against BI. Similarly, the scheme in

[1] This property is not guaranteed when the ephemeral private key of a client of the session is leaked. In this case, password of the client is easily derived by off-line dictionary attacks because the session key deterministically depends on the client's ephemeral key, static password, and communication received from other parties. Thus, secrecy of the session key is not guaranteed. So, we only consider leakage with respect to the server.

[2] Indeed, the scheme in [13] may be secure against UDonDA and satisfies other desirable security requirements. However, CTB model itself does not support these requirements.

Table 1. Comparison between previous schemes and our scheme

	setting of setup	# of rounds for a client	UDonDA	LEP
[11]	password-only	2 + P	insecure	insecure
[12]	password-only	2	insecure	insecure
[15]	password-only	2 + P	secure	insecure
[13]	password and public-key crypto	3	unproven	unproven
Our scheme	password and public-key crypto	2	secure	secure

Where P denote the number of moves of a secure 2-party PAKE.

[12] is also insecure against LEP because from the ephemeral private key of the server passwords can be revealed by offDAs.

1.3 Our Contribution

We define a new stronger security model of 3-party PAKE than previous models. Our model is based on the recent formal model of authenticated key exchange by LaMacchia et al. [16]. The major difference between our model and previous models consists in adversary's available oracle queries, specifically, revealing of static secret or ephemeral secret separately, and in adversary's capability in the target session, i.e., the adversary can obtain static secrets of all entities and ephemeral secrets of the server in the target session. Therefore, our model can represent resistance to complicated attacks which cannot be captured in previous models.

Also, we construct a new 3-party PAKE scheme based on Abdalla-Pointcheval scheme in [12]. Our scheme is the same setting as PSAKE (i.e., use of public-key crypto). Also, our scheme only needs the optimal number of rounds, i.e., 2-rounds between a client and the server, as Abdalla-Pointcheval scheme. Thus, our scheme is more efficient than general constructions in [11,15] and PSAKE. Furthermore, we show that our scheme is secure in the sense of our model in the random oracle model. While public-key encryption schemes are time-consuming, as same as PSAKE, by helping of the server's public-key crypto, our scheme can satisfy strong security like resistance to LEP and to KCI. Based on our knowledge, our scheme is the first 3-party PAKE scheme which resistance to LEP is proved.

The comparison between previous schemes and ours is shown in Table 1.

2 Preliminaries

2.1 3-Party PAKE

3-party PAKE schemes contain three parties (two clients and a server) who will engage in the protocol. We denote the set of clients by $\mathcal{U}$ and the server by S. Let each password be pre-shared between a client and the server and be uniformly and independently chosen from fixed low-entropy dictionary $\mathcal{D}$ of the size $|\mathcal{D}|$.

Note that clients do not need to share passwords with other clients. In addition, in PSAKE and our scheme, the server pre-establishes his public-key and private-key pair and goes public the public-key. We denote with U^l the l^{th} instance which clients $U \in \mathcal{U}$ runs. Also, we denote with S^l the l^{th} instance which the server S runs. All instances finally output *accept* symbol and halt if their specified execution is correctly finished. The session identifier $\mathsf{sid}_P^{l_i}$ of an instance P^{l_i} is represented via matching conversations, i.e., concatenations of messages which are sent and received between clients in the session, along with their identity strings, (initialized as *null*). Note that, we say that two instances $P_i^{l_i}$ and $P_j^{l_j}$ are partnered if both $P_i^{l_i}$ and $P_j^{l_j}$ output *accept*, both $P_i^{l_i}$ and $P_j^{l_j}$ share the same sid but not *null*, and the partner identification set for $P_i^{l_i}$ coincides with the one for $P_j^{l_j}$.

2.2 Problems of Previous Models

In AFP model, FS, and resistance to KCI, LEP and UDonDA cannot be captured. First, resistance to KCI and LEP, and FS cannot be represented because adversary capabilities do not include any query for corruption of parties in the test session. Therefore, conditions of KCI, LEP and FS cannot be represented. Also, resistance to UDonDA is out of scope in AFP model. They count UDonDA in the number of queries for message modifications which are limited to certain numbers. Hence, in AFP model, UDonDA is not discriminated from detectable on-line dictionary attacks.

Since CTB model is the extension for 3-party PAKE from the Canetti-Krawczyk model [14] for 2-party AKE, CTB model inherits uncaptured security properties from the Canetti-Krawczyk model. More specifically, in CTB model, KP, and resistance to KCI, LEP and UDonDA cannot be captured. First, resistance to KCI cannot be represented because adversary capabilities do not include any query for corruption of parties in the test session before completing the session. And, resistance to LEP cannot be represented because adversary capabilities do not include any query for reveal of ephemeral keys of parties in the test session. Therefore, conditions of KCI and LEP cannot be represented. Also, KP and resistance to UDonDA are out of scope in CTB model. Though KP requires that a passive server (i.e., passwords of clients can be known), cannot distinguish the real session key in a session and a random key, there is no definition which captures such a situation. Thus, KP is not guaranteed even if the security in the CTB model is satisfied. As AFP model, they count UDonDA in the number of queries for message modifications which are limited to certain numbers. Hence, in CTB model, UDonDA is not discriminated from detectable on-line dictionary attacks.

WH model can be regarded as AFP model plus resistance to UDonDA. Thus, FS, and resistance to KCI and LEP cannot be captured from the same reason as AFP model.

3 New Model: Strong 3-Party PAKE Security

3.1 Adversary Capabilities

An outside adversary or a malicious insider can obtain and modify messages on unauthenticated-links channels. Furthermore, the adversary is given oracle access to client and server instances. We remark that unlike the standard notion of an "oracle", in this model instances maintain state which is updated as the protocol progresses.

- $\mathsf{Execute}(U_1^{l_1}, U_2^{l_2}, S^{l_3})$: This query models passive attacks. The output of this query consists of the messages that were exchanged during the honest execution of the protocol among $U_1^{l_1}$, $U_2^{l_2}$ and S^{l_3}.
- $\mathsf{SendClient}(U^l, m)$: This query models active attacks against a client. The output of this query consists of the message that the client instance U^l would generate on receipt of message m.
- $\mathsf{SendServer}(S^l, m)$: This query models active attacks against the server. The output of this query consists of the message that the server instance S^l would generate on receipt of message m.
- $\mathsf{SessionKeyReveal}(U^l)$: This query models misuses of session keys. The output of this query consists of the session key held by the client instance U^l if the session is completed for U^l. Otherwise, return $\perp$.
- $\mathsf{StaticKeyReveal}(P)$: This query models leakage of the static secret of P (i.e., the password between the client and the server, or the private information for the server). The output of this query consists of the static secret of P. Note that, there is no giving the adversary full control of P or revealing any ephemeral secret information.
- $\mathsf{EphemeralKeyReveal}(P^l)$: This query models leakage of all session-specific information (ephemeral key) used by instance P^l. The output of this query consists of the ephemeral key of the instance P^l.
- $\mathsf{EstablishParty}(U, S, pw_U)$: This query models the adversary to register a static secret pw_U on behalf of a client. In this way the adversary totally controls that client. Clients against whom the adversary did not issue this query are called *honest*.
- $\mathsf{Test}(U^l)$: This query doesn't model the adversarial ability, but indistinguishability of the session key. At the beginning a hidden bit b is chosen. If no session key for the client instance U^l is defined, then return the undefined symbol $\perp$. Otherwise, return the session key for the client instance U^l if $b = 1$ or a random key from the same space if $b = 0$. Note that, the adversary can make an only Test query at any time during the experiment. The target session is called the test session.
- $\mathsf{TestPassword}(U, pw')$: This query doesn't model the adversarial ability, but no leakage of the password. If the guess password pw' is just the same as the client U's password pw, then return 1. Otherwise, return 0. Note that, the adversary can an only $\mathsf{TestPassword}$ query at any time during the experiment.

3.2 Definition of Indistinguishability

Firstly, we consider the notion of indistinguishability. This notion provides security properties with respect to session keys, i.e., KS, FS, KP, resistance to BI, resistance to KCI and resistance to LEP. Note that, to capture notions of FS and resistance to KCI, an adversary can obtain static keys in the test session.

The adversary is considered successful if it guesses whether the challenge is the true session key or a random key. The adversary is allowed to make Execute, SendClient, SendServer, SessionKeyReveal, StaticKeyReveal, EphemeralKeyReveal, EstablishParty and Test queries, and outputs a guess bit b'. Let Succ^{ind} denote the event that $b' = b$ where b is the random bit chosen in the $\mathsf{Test}(U^l)$ query. Note that, we restrict the adversary such that U^l and the partnered client $\bar{U}^{l'}$ of the session are honest, and none of the following conditions hold:

1. The adversary reveals the session key of sid_U^l or of $\mathsf{sid}_{\bar{U}}^{l'}$.
2. The adversary asks no $\mathsf{SendClient}(U^l, m)$ or $\mathsf{SendClient}(\bar{U}^{l'}, m')$ query. Then the adversary either makes queries:
   - $\mathsf{EphemeralKeyReveal}(U^l)$ or
   - $\mathsf{EphemeralKeyReveal}(\bar{U}^{l'})$.
3. The adversary asks $\mathsf{SendClient}(\bar{U}^{l'}, m)$ query. Then the adversary either makes queries:
   - $\mathsf{StaticKeyReveal}(U)$,
   - $\mathsf{StaticKeyReveal}(S)$,
   - $\mathsf{EphemeralKeyReveal}(U^i)$ for any session i or
   - $\mathsf{EphemeralKeyReveal}(\bar{U}^{l'})$.
4. The adversary asks $\mathsf{SendClient}(U^l, m)$ query. Then the adversary either makes queries:
   - $\mathsf{StaticKeyReveal}(\bar{U})$,
   - $\mathsf{StaticKeyReveal}(S)$,
   - $\mathsf{EphemeralKeyReveal}(U^l)$ or
   - $\mathsf{EphemeralKeyReveal}(\bar{U}^i)$ for any session i.

Now, the adversary $\mathcal{A}$'s advantage is formally defined by:

$$\mathsf{Adv}^{ind}(\mathcal{A}) = |2 \cdot \Pr[\mathsf{Succ}^{ind}] - 1| \quad \text{and} \quad \mathsf{Adv}^{ind}(t, R) = \max_{\mathcal{A}}\{\mathsf{Adv}^{ind}(\mathcal{A})\},$$

where the maximum is over all $\mathcal{A}$ with time-complexity at most t and using the number of queries to its oracle at most R.

We say that a 3-party PAKE satisfies indistinguishability of the session key if the advantage Adv^{ind} is only negligibly larger than $n \cdot q_{send}/|\mathcal{D}|$, where n is a constant and q_{send} is the number of send queries, and parties who complete matching sessions compute the same session key.

Capturing Security Properties. The condition of KS is represented as the adversary can obtain session keys except one of the test session by SessionKeyReveal query. The condition of KP against passive server is represented as the freshness condition 2, that is, the adversary can obtain static and ephemeral private key of the server by StaticKeyReveal and EphemeralKeyReveal query but no SendClient query for the test session. BI is represented as the freshness condition 3, that is, the adversary can freely eavesdrop messages, obtain ephemeral private key of the server, and send any message to honest clients except the target client by Execute and SendClient queries but no StaticKeyReveal query to the target client and the server. KCI and the condition of FS are also represented as the freshness condition 4, that is, the adversary can obtain static secret of the target client by StaticKeyReveal query but cannot ask StaticKeyReveal query to the partnered client and the server. LEP is represented as the adversary can obtain ephemeral key of the server on the test session by EphemeralKeyReveal query. Also, our model captures resistance to unknown-key share (UKS) (i.e., any client C including a malicious client insider cannot interfere with the session establishment between two honest clients A and B such that at the end of the attack both parties compute the same session key which C may not learn it, yet while A is convinced that the key is shared with B, B believes that the peer to the session has been C). UKS is represented as the adversary can establish a malicious insider by EstablishParty query and try to make a honest client which thinks that he shares the session key with the insider share the session key with an another honest client by choosing these two honest clients for the test session.

By the definition of indistinguishability, we can guarantee to prevent these attacks in our model.

3.3 Definition of Password Protection

Next, we consider the notion of password protection. This notion provides security properties with respect to passwords, i.e., resistance to UDonDA and to offDA. Beyond the notion of indistinguishability, the notion of password protection is needed because we have to consider security for passwords against attacks by malicious client insiders which can trivially know the session key. Thus, just the notion of indistinguishability cannot capture insider attacks. Also, we cannot allow the adversary to reveal ephemeral private keys of the target client. Given the ephemeral key, the target password is easily derived by offDA because the session key in a session deterministically depends on the client's ephemeral key, the password, and communication received from the other party.

The adversary is considered successful if it guesses a password of a client. The adversary is allowed to make Execute, SendClient, SendServer, SessionKeyReveal, StaticKeyReveal, EphemeralKeyReveal and TestPassword queries. Let Succ^{pw} denote the event that TestPassword(U) outputs 1. Note that, we restrict the adversary such that U and the server of the session are honest, and none of the following conditions hold:

– We suppose that S is the corresponding server of U. Then the adversary either makes queries:
  - $\mathsf{StaticKeyReveal}(U)$,
  - $\mathsf{StaticKeyReveal}(S)$ or
  - $\mathsf{EphemeralKeyReveal}(U^i)$ for any session i.

Now, the adversary $\mathcal{A}$'s advantage is formally defined by:

$$\mathsf{Adv}^{pw}(\mathcal{A}) = \Pr[\mathsf{Succ}^{pw}] \quad \text{and} \quad \mathsf{Adv}^{pw}(t, R) = \max_{\mathcal{A}}\{\mathsf{Adv}^{pw}(\mathcal{A})\},$$

where the maximum is over all $\mathcal{A}$ with time-complexity at most t and using the number of queries to its oracle at most R.

We say that a 3-party PAKE satisfies password protection if the advantage Adv^{pw} is only negligibly larger than $n \cdot q_{send}/|\mathcal{D}|$, where n is a constant and q_{send} is the number of send queries which messages are found as "invalid" by the party. "Invalid" message means the message which is not derived according to the protocol description.

Capturing Security Properties. UDonDA is represented as the adversary can unlimitedly use $\mathsf{SendClient}$ and $\mathsf{SendServer}$ queries as far as the party does not find that the query is "invalid". offDA is represented as the adversary can be the insider by $\mathsf{SessionKeyReveal}$, $\mathsf{StaticKeyReveal}$ and $\mathsf{EphemeralKeyReveal}$ queries except the target client and her corresponding server.

By the definition of password protection, we can guarantee to prevent these attacks in our model.

4 Proposed Scheme

In this section, we show our 3-party PAKE scheme in the same setting as PSAKE.

4.1 Notation

Let p be a prime and let g be a generator of a finite cyclic group G of order p. $A, B \in \mathcal{U}$ are identities of two clients, and S is identity of their corresponding server. $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a public-key encryption scheme, where $\mathsf{Gen}(1^k)$ is key generation algorithm, $\mathsf{Enc}_{pk}(m; \omega)$ is encryption algorithm of a message m using a public key pk and randomness ω, and $\mathsf{Dec}_{sk}(c)$ is decryption algorithm of a cipher-text c using a private key sk. A and S (resp. B and S) have shared common secret password pw_A (resp. pw_B), and S has pre-established his private key sk_S with his public key pk_S. $H_1 : \mathcal{D} \times \mathcal{U}^2 \to G$, $H_2 : \mathcal{D} \times \{0,1\}^k \times G \to G$ and $H_3 : \mathcal{U}^2 \times \mathcal{S}^2 \times Cspace^2 \times G^3 \to \{0,1\}^k$ are hash functions modeled as random oracles, where $Cspace$ is the space of a cipher-text for $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and k is a sufficiently large security parameter.

For simplicity, we omit "(mod p)" in this paper when computing the modular exponentiation. "$v \xleftarrow{R} V$" means randomly choosing an element v of a set V.

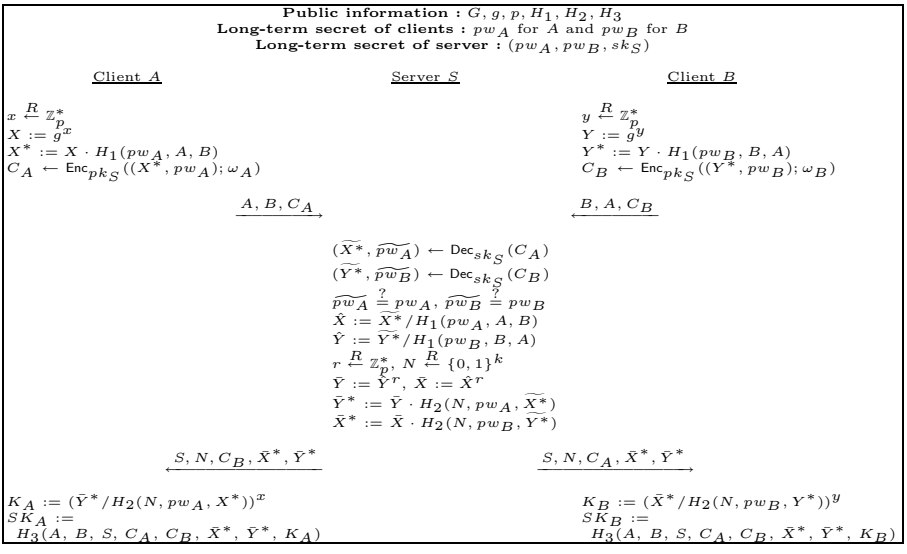


Fig. 1. A high-level overview of our protocol

4.2 Protocol Description

Here, we show the construction of our scheme. To guarantee resistance to UDonDA, we apply public-key encryption for servers like PSAKE and the 3-party PAKE scheme in [8]. A high-level overview of our protocol appears in Figure 1.

Then, our protocol is described as follows:

Step 1. Clients A and B choose $x, y \in \mathbb{Z}_p^*$ randomly, compute $X = g^x$ and $Y = g^y$, and blind them as $X^* = X \cdot H_1(pw_A, A, B)$ and $Y^* = Y \cdot H_1(pw_B, B, A)$ respectively. Next, they generate $C_A \leftarrow \mathsf{Enc}_{pk_S}((X^*,\ pw_A\ ); \omega_A)$ and $C_B \leftarrow \mathsf{Enc}_{pk_S}((Y^*,\ pw_B\ ); \omega_B)$ by using their corresponding server's public-key pk_S with randomness ω_A and ω_B respectively. Finally, A sends (A, B, C_A) to the server S and B sends (B, A, C_B) to the server S. So, ephemeral private-keys of A and B are (x, X, X^*, ω_A) and (y, Y, Y^*, ω_B) respectively.

Step 2. The server S decrypts $(\widetilde{X}^*, \widetilde{pw_A}) \leftarrow \mathsf{Dec}_{sk_S}(C_A)$ and $(\widetilde{Y}^*, \widetilde{pw_B}) \leftarrow \mathsf{Dec}_{sk_S}(C_B)$ by using sk_S respectively. If $\widetilde{pw_A} \neq pw_A$ or $\widetilde{pw_B} \neq pw_B$, then S aborts the session. It is also crucial that the server rejects any value $\widetilde{X}^*$ or $\widetilde{Y}^*$ whose underlying value X or Y is equal to 1. Otherwise, S computes $\hat{X} = \widetilde{X}^*/H_1(pw_A, A, B)$, blinds it as $\bar{X} := \hat{X}^r$ where r is S's first random value from $\mathbb{Z}_p^*$. S also computes $\hat{Y}$ and $\bar{Y}$ similarly. Next, S computes $\bar{Y}^* = \bar{Y} \cdot H_2(N, pw_A, \widetilde{X}^*)$ where N is S's second random value from $\{0,1\}^k$. S performs similar operations and obtains $\bar{X}^*$. Finally, S sends $(S, N, C_B, \bar{X}^*, \bar{Y}^*)$ to A, sends $(S, N, C_A, \bar{X}^*, \bar{Y}^*)$ to B, and deletes session-specific information ($\widetilde{X}^*$, $\widetilde{Y}^*$, $\widetilde{pw_A}$, $\widetilde{pw_B}$, r, N, $\hat{X}$, $\hat{Y}$, $\bar{X}$, $\bar{Y}$). So, ephemeral private-keys of S is empty.

Step 3. A and B compute their Diffie-Hellman keys $K_A = (\bar{Y}^* \,/\, H_2(N, pw_A, X^*))^x$ and $K_B = (\bar{X}^* \,/\, H_2(N, pw_B, Y^*))^y$ respectively. Session keys are generated from the Diffie-Hellman key and transcripts, $SK_A = H_3(A, B, S, C_A, C_B, \bar{X}^*, \bar{Y}^*, K_A)$ and $SK_B = H_3(A, B, S, C_A, C_B, \bar{X}^*, \bar{Y}^*, K_B)$. When session keys are honestly generated, $SK_A = SK_B$ because $K_A = (g^{yr})^x$ and $K_B = (g^{xr})^y$.

4.3 Design Principles

Our protocol can be viewed as an extension of Abdalla-Pointcheval scheme [12]. The main deference consists in use of public-key encryption.

First, upon receiving an input from a client, the corresponding server verifies the validity of encrypted password of the client and him. This procedure prevents UDonDA as the technique of Lin et al. [8]. Applying the server's public-key may put a burden on clients because they have to verify the server's public-key in advance, and the certificate infrastructure is needed. However, we can easily resolve this problem by applying ID-based encryption for the server instead of standard public-key encryption for the server. Since clients can encrypt messages by using only corresponding the server's ID in ID-based encryption, clients need no keeping nor verifying the server's public-key. If we replace use of public-key encryption to use of ID-based encryption, security of our scheme is not changed.

Next, elimination of ephemeral states except necessary states is needed for resistance to LEP as the technique of [16]. Even if $\mathsf{EphemeralKeyReveal}$ query is asked, information of passwords and the session key do not leak from leakage information because all critical states are deleted immediately when these states are used.

Finally, when a client blinds X or Y with his password, we make the client include the identities of both clients into the computation of the password-based blinding factors. This procedure prevents KCI and UKS by a malicious client insider as the technique of Choo et al. [17].

5 Security of Our Scheme

In this section, we show security properties of our scheme.

5.1 Building Blocks

We recall the definition of the decisional Diffie-Hellman assumptions which we use in the security proof of our scheme. Let p be a prime and let g be a generator of a finite cyclic group G of order p.

Decisional Diffie-Hellman Assumption (DDH). We can define the DDH assumption by defining two experiments, $\mathsf{Exp}_{g,p}^{ddh-real}(\mathcal{I})$ and $\mathsf{Exp}_{g,p}^{ddh-rand}(\mathcal{I})$. For a solver $\mathcal{I}$, inputs (g^u, g^v, Z) is provided, where u, v are drawn at random from $\mathbb{Z}_p^*$. $Z = g^{uv}$ in $\mathsf{Exp}_{g,p}^{ddh-real}(\mathcal{I})$ and $Z = g^w$ in $Exp_{g,p}^{ddh-rand}(\mathcal{I})$, where w is drawn at random from $\mathbb{Z}_p^*$. We define the advantage of $\mathcal{I}$ in violating the DDH assumption, $\mathsf{Adv}_{g,p}^{ddh}(\mathcal{I})$, as $|\Pr[\mathsf{Exp}_{g,p}^{ddh-real}(\mathcal{I}) = 1] - \Pr[\mathsf{Exp}_{g,p}^{ddh-rand}(\mathcal{I}) = 1]|$. The advantage function of the group, $\mathsf{Adv}_{g,p}^{ddh}(t)$, is defined as the maximum value of $\mathsf{Adv}_{g,p}^{ddh}(\mathcal{I})$ over all $\mathcal{I}$ with time-complexity at most t.

5.2 Main Theorems

Theorem 1. *Assuming* (Gen, Enc, Dec) *is a semantically secure public-key encryption scheme and DDH problem is hard, then our scheme satisfies indistinguishability in Sec. 3.2.*

Theorem 2. *Assuming* (Gen, Enc, Dec) *is a semantically secure public-key encryption scheme, then our scheme satisfies password protection in Sec. 3.3.*

Owing to lack of space, we cannot give proofs of Theorem 1 and 2. Please refer to [18] for these proofs.

6 Conclusion

Firstly, we pointed out that previous security definitions of 3-party PAKE cannot capture all desirable security requirements. Next, we proposed a new stronger definition of 3-party PAKE which captures all desirable security requirements. Finally, we introduced a 3-party PAKE protocol in the same setting as PSAKE with optimal rounds for client and proved its security in the sense of our stronger definition.

Our scheme use public-key encryption as a building block in order to guarantee resistance to UDonDA. However, public-key encryption schemes are time-consuming. Thus, a remaining problem of further researches is efficient construction which satisfies stronger security.

References

1. Bellovin, S.M., Merritt, M.: Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In: IEEE S&P 1992, pp. 72–84 (1992)
2. Jablon, D.P.: Strong Password-Only Authenticated Key Exchange. Computer Communication Review, ACM SIGCOMM 26(5), 5–26 (1996)
3. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated Key Exchange Secure against Dictionary Attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
4. Boyko, V., MacKenzie, P.D., Patel, S.: Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer, Heidelberg (2000)
5. Steiner, J.G., Neuman, B.C., Schiller, J.I.: Kerberos: An Authentication Service for Open Network Systems. In: USENIX Winter 1988, pp. 191–202 (1988)
6. Lomas, T.M.A., Gong, L., Saltzer, J.H., Needham, R.M.: Reducing Risks from Poorly Chosen Keys. In: SOSP 1989, pp. 14–18 (1989)
7. Steiner, M., Tsudik, G., Waidner, M.: Refinement and Extension of Encrypted Key Exchange. ACM Operating Systems Review 29(3), 22–30 (1995)
8. Lin, C.L., Sun, H.M., Hwang, T.: Three-party Encrypted Key Exchange: Attacks and A Solution. ACM Operating Systems Review 34(4), 12–20 (2000)
9. Chang, Y.F., Chang, C.C.: Password-authenticated 3PEKE with Round Efficiency without Server's Public Key. In: CW 2005, pp. 340–344 (2005)

10. Ding, Y., Horster, P.: Undetectable On-line Password Guessing Attacks. Operating Systems Review 29(4), 77–86 (1995)
11. Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-Based Authenticated Key Exchange in the Three-Party Setting. In: Public Key Cryptography 2005, pp. 65–84 (2005)
12. Abdalla, M., Pointcheval, D.: Interactive Diffie-Hellman Assumptions with Applications to Password-Based Authentication. In: Financial Cryptography 2005, pp. 341–356 (2005)
13. Cliff, Y., Tin, Y.S.T., Boyd, C.: Password Based Server Aided Key Exchange. In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 2006. LNCS, vol. 3989, pp. 146–161. Springer, Heidelberg (2006)
14. Canetti, R., Krawczyk, H.: Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001)
15. Wang, W., Hu, L.: Efficient and Provably Secure Generic Construction of Three-Party Password-Based Authenticated Key Exchange Protocols. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 118–132. Springer, Heidelberg (2006)
16. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger Security of Authenticated Key Exchange. In: Provsec 2007 (2007)
17. Choo, K.K.R., Boyd, C., Hitchcock, Y.: Examining Indistinguishability-Based Proof Models for Key Establishment Protocols. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 585–604. Springer, Heidelberg (2005)
18. Yoneyama, K.: Efficient and Strongly Secure Password-based Server Aided Key Exchange (2008), http://www.oslab.ice.uec.ac.jp/member/yoneyama/IC08.pdf

Round Efficient Unconditionally Secure Multiparty Computation Protocol

Arpita Patra*, Ashish Choudhary**, and C. Pandu Rangan***

Department of Computer Science and Engineering
IIT Madras, Chennai 600036, India
{arpita,ashishc}@cse.iitm.ernet.in, rangan@iitm.ernet.in

Abstract. In this paper, we propose a round efficient *unconditionally secure multiparty computation* (UMPC) protocol in *information theoretic* model with $n > 2t$ players, in the absence of any physical broadcast channel. Our protocol communicates $\mathcal{O}(n^4)$ field elements per multiplication and requires $\mathcal{O}(n \log(n) + \mathcal{D})$ rounds, even if up to t players are under the control of an active adversary having *unbounded computing power*, where $\mathcal{D}$ denotes the multiplicative depth of the circuit representing the function to be computed securely. In the absence of a physical broadcast channel and with $n > 2t$ players, the best known UMPC protocol with minimum number of rounds, requires $\mathcal{O}(n^2\mathcal{D})$ rounds and communicates $\mathcal{O}(n^6)$ field elements per multiplication. On the other hand, the best known UMPC protocol with minimum communication complexity requires communication overhead of $\mathcal{O}(n^2)$ field elements per multiplication, but has a round complexity of $\mathcal{O}(n^3 + \mathcal{D})$ rounds. Hence our UMPC protocol is the most round efficient protocol so far and ranks second according to communication complexity.

Keywords: Multiparty Computation, Information Theoretic Security.

1 Introduction

Secure Multiparty Computation (MPC): Secure multiparty computation (MPC) allows a set of n players to securely compute an agreed function, even if up to t players are under the control of a centralized adversary. More specifically, assume that the desired functionality can be specified by a function $f : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$ and player P_i has input $x_i \in \{0,1\}^*$. At the end of the computation of f, P_i gets $y_i \in \{0,1\}^*$, where $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$. The function f has to be computed securely using a protocol where at the end of the protocol all (honest) players receive correct outputs and the messages seen by the adversary during the protocol contain no *additional* information about

* Financial Support from Microsoft Research India Acknowledged.
** Financial Support from Infosys Technology India Acknowledged.
*** Work Supported by Project No. CSE/05-06/076/DITX/CPAN on Protocols for Secure Communication and Computation Sponsored by Department of Information Technology, Govt. of India.

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 185–199, 2008.

the inputs and outputs of the honest players, other than what can be computed from the inputs and outputs of the corrupted players. In the *information theoretic model*, the adversary who *actively* controls at most t players, is adaptive, rushing [10] and has *unbounded computing power*. The function to be computed is represented as an arithmetic circuit over a finite field $\mathbb{F}$ consisting of five type of gates, namely addition, multiplication, random, input and output. A MPC protocol securely evaluates the circuit gate-by-gate [5].

The MPC problem was first defined and solved by Yao [16]. The research on MPC in information theoretic model was initiated by Ben-Or et. al. [5] and Chaum et. al. [8] in two different independent work and carried forward by the works of [15,2]. Information theoretic security can be achieved by MPC protocols in two flavors –(a) Perfect: The outcome of the protocol is **perfect** in the sense that no probability of error is involved in the computation of the function (b) Unconditional: The outcome of the protocol is correct except with negligible error probability. While Perfect MPC can be achieved iff $t < n/3$ [5], unconditional MPC (UMPC) requires only honest majority i.e $t < n/2$ [15]. In the recent years, lot of research concentrated on designing communication efficient protocols for both perfect and unconditional MPC (see [4,11,10,3]).

Broadcast: Broadcast is an important primitive used in all MPC and UMPC protocols and allows a sender to distribute a value x, identically among all the players. If a physical broadcast channel is available in the network, then achieving broadcast is very trivial. But if the broadcast channel is not physically available in the network, then broadcasting an ℓ bit(s) message can be simulated by executing some protocol. In particular, for unconditionally (with negligible error probability) broadcasting ℓ bits, the protocol of [14] communicates $\Omega(n^2\ell + n^6\kappa)$ bits and requires $\Omega(n)$ rounds with $t < n/2$ on the availability of information theoretic PKI setup,where κ is the error parameter. From the recent results of Fitzi et. al. [12], broadcast can be achieved with communication complexity of $O(n\ell + n^7\kappa)$ bits and round complexity of $O(n)$.

Our Motivation: Two important parameters of MPC protocols are *communication* and *round* complexity which have been the subject of intense study over the past two decades. Reducing the communication and round complexity of MPC protocols is crucial, if we ever hope to use these protocols in practice. But looking at the most recent advancements in the arena of MPC, we find that round complexity of MPC protocols has been increased to an unacceptable level in order to reduce communication complexity. In the sequel, we present a table which gives an overview of the communication complexities and round complexities of perfect and unconditional MPC protocols. The communication complexities are given in terms of bits where κ represents the bit length of a field element in the case of perfect MPC and error parameter in the case of UMPC, respectively. c_M and $\mathcal{D}$ denote the number of multiplication gates and multiplicative depth of the circuit, respectively.

Reference	Type?	Resilience	Broadcast Protocol	Communication Complexity	Round Complexity
[10]	Unconditional	$t < n/2$	[12]	$\mathcal{O}((c_M n^6 + n^7)\kappa)$	$\mathcal{O}(n^2\mathcal{D})$
[3]	Unconditional	$t < n/2$	[12]	$\mathcal{O}((c_M n^2 + n^7)\kappa)$	$\mathcal{O}(n^3 + \mathcal{D})$
[4]	Perfect	$t < n/3$	[6,7]	$\mathcal{O}((c_M n + \mathcal{D}n^2 + n^3)\kappa)$	$\mathcal{O}(n^2 + \mathcal{D})$

If the practical applicability of multiparty protocols are of primary focus, then it is always desirable not to sacrifice one parameter for the other. So it is very essential to design protocol which balances both the parameters appropriately, which is the motivation of this paper.

Our Network Model: We denote the set of $n = 2t + 1$ players involved in the secure computation by $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ where player P_i possesses c_i input values. We assume that all the n players are connected with each other by pairwise secure channels. Moreover, the system is synchronous and the protocols proceed in rounds, where in each round a player performs some computations, sends (broadcasts) values to its neighbors (everybody), receives values from neighbors and may again perform some more computation, in that order. The function to be computed is specified as an arithmetic circuit over a finite field $\mathbb{F}$ with input, addition, multiplication, random and output gates. We denote the number of gates of each type by c_I, c_A, c_M, c_R and c_O, respectively. We model the distrust in the system by a centralized adversary $\mathcal{A}_t$, who has *unbounded computing* power and can actively control at most t players during protocol execution. To actively control a player means to take full control over it and make it behave arbitrarily. Moreover $\mathcal{A}_t$ is adaptive and rushing [10].

Our protocol provides information theoretic security with a negligible error probability of $2^{-\mathcal{O}(\kappa)}$ for some security parameter κ. To achieve this error probability, all our computation are done over a finite field $\mathbb{F} = GF(2^\kappa)$. Thus each field element can be represented by κ bits. We assume that $n = poly(\kappa)$. We also assume that the messages sent through the channels are from the specified domain. Thus if a player receives a message which is not from the specified domain (or no message at all), he replaces it with some pre-defined default message.

Our Contribution: We propose a new UMPC protocol which communicates $\mathcal{O}(n^4)$ field elements per multiplication and requires $\mathcal{O}(n \log(n) + \mathcal{D})$ rounds over a point-to-point network (in the absence of physical broadcast channel) with $n = 2t+1$ players. We introduce a new technique called *Rapid Player Elimination* (RPE) which is used in the preprocessing stage of our UMPC protocol. Loosely speaking, RPE works as follows: The preprocessing stage of our UMPC protocol may fail several times due to the (mis)behavior of certain number of corrupted players whose corruptions are identified. RPE creates a win-win situation, where the adversary must reveal the identities of 2^i new corrupted players at the i^{th} step. Otherwise, the preprocessing stage will not fail. Thus RPE ensures that preprocessing stage may fail at most $\lceil \log(t) \rceil$ times.

2 Unconditionally Secure MPC Protocol with $n = 2t + 1$

We now present an UMPC protocol with $n = 2t + 1$. Prior to that we present a number of sub-protocols each solving a specific task. Some of the sub-protocols are based on few existing techniques while some are proposed by us for the first time. We describe all our sub-protocols in the following settings: We define two sets $\mathcal{C}$ and $\mathcal{P}'$ where at any point of time $\mathcal{C}$ denotes the set of corrupted players identified so far and $\mathcal{P}' = \mathcal{P} \setminus \mathcal{C}$. So, $\mathcal{P}'$ denotes the set of players involved in the execution of the sub-protocols. Initially, $\mathcal{P}' = \mathcal{P}$ and $\mathcal{C} = \emptyset$. As the protocol proceeds, some players will be detected as corrupted and will be added to $\mathcal{C}$ and removed from $\mathcal{P}'$. We denote the number of players in $\mathcal{P}'$ by n' which is initially equal to n. The number of players which can be still corrupted in $\mathcal{P}'$ is denoted by t' where $t' = t - |\mathcal{C}|$. Note that n' will always maintain the following: $n' \geq t + 1 + t' \geq 2t' + 1$ since $t \geq t'$ and at any stage $\mathcal{P}'$ will always contain all the $t + 1$ honest players. Also at any point of time $\mathcal{P} = \mathcal{P}' \cup \mathcal{C}$.

2.1 Information Checking (IC [10,15])

It is an information theoretically secure method for authenticating data and is used to generate IC signatures. When a player $INT \in \mathcal{P}'$ receives an IC signature from a dealer $\mathbf{D} \in \mathcal{P}'$ on some secret value(s) S, then INT can later produce the signature and have the players in $\mathcal{P}'$ verify that it is in fact a valid signature of $\mathbf{D}$ on S. The complete definition of IC scheme, its outcome and the properties that should be satisfied by its outcome is provided in [13]. We now present an IC protocol, called **EfficientIC**, given in Table 1, which allows $\mathbf{D}$ to sign on an ℓ length secret $S \in \mathbb{F}^\ell$. Let $S = (s^{(1)}, \ldots, s^{(\ell)}) \in \mathbb{F}^\ell$.

Lemma 1. *Protocol* **EfficientIC** *correctly generates IC signature on ℓ field elements (each of size κ bits) at once by communicating and broadcasting $\mathcal{O}((\ell + n)\kappa)$ bits. The protocol satisfies the properties of IC signature with probability at least $1 - 2^{-\mathcal{O}(\kappa)}$.*

We also use another IC protocol which is a slight modification of the IC protocol, called *IC* described in [10]. The protocol allows $\mathbf{D}$ to sign on a single field element $s \in \mathbb{F}$ (i.e. $\ell = 1$; $S = s$). The protocol is given in [13]. Like *EfficientIC*, protocol *IC* has three sub-protocols: **Distr**$(\mathbf{D}, INT, \mathcal{P}', s)$, **AuthVal**$(\mathbf{D}, INT, \mathcal{P}', s)$, **RevealVal**$(\mathbf{D}, INT, \mathcal{P}', s)$.

Linearity of Protocol IC and EfficientIC: Protocol **IC** and **EfficientIC** satisfies the *linearity* property as specified by the following lemma:

Lemma 2 (Linearity of Protocol EfficientIC[10]). *The IC signature generated by* **EfficientIC** *satisfies* **linearity** *property. In particular, INT can compute $ICSig_{((r_1 s^{(1,1)} + r_2 s^{(2,1)}), \ldots, (r_1 s^{(1,\ell)} + r_2 s^{(2,\ell)}))}(\mathbf{D}, INT)$ from $ICSig_{(s^{(1,1)}, s^{(1,2)} \ldots, s^{(1,\ell)})}(\mathbf{D}, INT)$ and $ICSig_{(s^{(2,1)}, s^{(2,2)} \ldots, s^{(2,\ell)})}(\mathbf{D}, INT)$ and receivers can compute verification information corresponding to $ICSig_{((r_1 s^{(1,1)} + r_2 s^{(2,1)}), \ldots, (r_1 s^{(1,\ell)} + r_2 s^{(2,\ell)}))}(\mathbf{D}, INT)$, without doing any further computation.*

Table 1. EfficientIC($\mathbf{D}, INT, \mathcal{P}', \ell, s^{(1)}, \ldots, s^{(\ell)}$)

EfficientDistr($\mathbf{D}, INT, \mathcal{P}', \ell, s^{(1)}, \ldots, s^{(\ell)}$): Round 1: $\mathbf{D}$ selects a random $\ell+t'-1$ degree polynomial $F(x)$ over $\mathbb{F}$, whose lower order ℓ coefficients are $s^{(1)}, \ldots, s^{(\ell)}$. In addition, $\mathbf{D}$ selects another random $\ell + t' - 1$ degree polynomial $R(x)$, over $\mathbb{F}$, which is independent of $F(x)$. $\mathbf{D}$ selects n' distinct random elements $\alpha_1, \alpha_2, \ldots, \alpha_{n'}$ from $\mathbb{F}$ such that each $\alpha_i \in \mathbb{F} - \{0, 1, \ldots, n'-1\}$. $\mathbf{D}$ privately gives $F(x)$ and $R(x)$ to INT. To receiver $P_i \in \mathcal{P}'$, $\mathbf{D}$ privately gives α_i, v_i and r_i, where $v_i = F(\alpha_i)$ and $r_i = R(\alpha_i)$. The polynomial $R(x)$ is called **authentication information**, while for $1 \leq i \leq n'$, the values α_i, v_i and r_i are called **verification information**.

EfficientAuthVal($\mathbf{D}, INT, \mathcal{P}', \ell, s^{(1)}, \ldots, s^{(\ell)}$): Round 2: INT chooses a random $d \in \mathbb{F} \setminus \{0\}$ and broadcasts $d, B(x) = dF(x) + R(x)$.

Round 3: For $1 \leq j \leq n'$, $\mathbf{D}$ checks $dv_j + r_j \stackrel{?}{=} B(\alpha_j)$. If $\mathbf{D}$ finds any inconsistency, he broadcasts $F(x)$. Parallely, receiver P_i broadcasts "Accept" or "Reject", depending upon whether $dv_i + r_i = B(\alpha_i)$ or not.

Local Computation (by each player): IF $F(x)$ is broadcasted in **Round 3** then accept the lower order ℓ coefficients of $F(x)$ as $\mathbf{D}$'s secret and terminate. ELSE construct an n' length bit vector V^{Sh}, where the $j^{th}, 1 \leq j \leq n'$ bit is 1(0), if $P_j \in \mathcal{P}'$ has broadcasted "Accept" ("Reject") during **Round 3**. The vector V^{Sh} is public, as it is constructed using broadcasted information. If V^{Sh} does not contain $n' - t'$ 1's, then $\mathbf{D}$ fails to give any signature to INT and IC protocol terminates here.

If $F(x)$ is not broadcasted during **Round 3**, then $(F(x), R(x))$ is called $\mathbf{D}$'s IC signature on $S = (s^{(1)}, \ldots, s^{(\ell)})$ denoted by $ICSig_{(s^{(1)}, \ldots, s^{(\ell)})}(\mathbf{D}, INT)$.

EfficientRevealVal($\mathbf{D}, INT, \mathcal{P}', \ell, s^{(1)}, \ldots, s^{(\ell)}$): (a) **Round 1**: INT broadcasts $F(x), R(x)$; (b) **Round 2**: P_i broadcasts α_i, v_i and r_i.

Local Computation (by each player): For the polynomial $F(x)$ broadcasted by INT, construct an n' length vector $V^{Rec}_{F(x)}$ whose j^{th} bit contains 1 if $v_j = F(\alpha_j)$, else 0. Similarly, construct the vector $V^{Rec}_{R(x)}$ corresponding to $R(x)$. Finally compute $V^{Rec}_{FR} = V^{Rec}_{F(x)} \otimes V^{Rec}_{R(x)}$, where $\otimes$ denotes bit wise AND. If V^{Rec}_{FR} and V^{Sh} matches at least at $t+1$ locations (irrespective of bit value at these locations), then accept the lower order ℓ coefficients of $F(x)$ as $S = (s^{(1)}, \ldots, s^{(\ell)})$. In this case, INT is able to prove $\mathbf{D}$'s signature on S. Otherwise he fails to prove.

2.2 Unconditional Verifiable Secret Sharing and Reconstruction

Definition 1. *t'-1D-Sharing: A value s is correctly t'-1D-shared among the players in $\mathcal{P}'$ if every honest $P_i \in \mathcal{P}'$ is holding a share s_i of s, such that there exists a degree t' polynomial $f(x)$ over $\mathbb{F}$ with $f(0) = s$ and $f(j) = s_j$ for every $P_j \in \mathcal{P}'$. The vector $(s_1, s_2, \ldots, s_{n'})$ of shares is called a t'-sharing of s and is denoted by $[s]_{t'}$. A set of shares (possibly incomplete) is t'-consistent if they lie on a t' degree polynomial.*

Definition 2. *t'-2D-sharing [3]: A value s is correctly t'-2D-shared among the players in $\mathcal{P}'$ if there exists t' degree polynomials $f, f^1, f^2 \ldots, f^{n'}$ with $f(0) = s$ and for $i = 1, \ldots, n'$, $f^i(0) = f(i)$. Moreover, every player $P_i \in \mathcal{P}'$ holds a share $s_i = f(i)$ of s, the polynomial $f^i(x)$ for sharing s_i and a share-share $s_{ji} = f^j(i)$ of the share s_j of every player $P_j \in \mathcal{P}'$. We denote t'-2D-sharing of s as $[[s]]_{t'}$.*

Definition 3. *t'-$2D^{(+)}$-sharing: A value s is correctly t'-$2D^{(+)}$-shared among the players in $\mathcal{P}'$ if there exists t' degree polynomials $f, f^1, f^2 \ldots, f^{n'}$ with $f(0) = s$ and for $i = 1, \ldots, n'$, $f^i(0) = f(i)$. Moreover, every $P_i \in \mathcal{P}'$ holds a share $s_i = f(i)$ of s, the polynomial $f^i(x)$ for sharing s_i and P_j's IC Signature on share-share $s_{ji} = f^j(i)$ of P_j's share s_j, i.e. $ICSig_{s_{ji}}(P_j, P_i)$ for every $P_j \in \mathcal{P}'$.*

We denote the t'-$2D^{(+)}$-sharing of s as $\langle\langle s\rangle\rangle_{t'}$. Note that in [3], the authors called this sharing as $2D^*$-sharing.

Definition 4. **t'-$2D^{(+,\ell)}$-sharing:** *A set of values $s^{(1)},\ldots,s^{(\ell)}$ are correctly t'-$2D^{(+,\ell)}$-shared among the players in $\mathcal{P}'$ if every secret $s^{(l)}$ is individually t'-$2D^{(+)}$-shared. But now instead of P_i holding separate IC-signatures for each of the share-shares $s_{ji}^{(l)}$ for $l=1\ldots,\ell$ from P_j, a single IC-signature on $s_{ji}^{(1)},\ldots,s_{ji}^{(\ell)}$ is given by P_j to P_i (i.e. $ICSig_{(s_{ji}^{(1)},\ldots,s_{ji}^{(\ell)})}(P_j,P_i)$). The t'-$2D^{(+,\ell)}$-sharing is denoted as $\langle\langle s^1,\ldots,s^\ell\rangle\rangle_{t'}$.*

If a secret s is t'-1D-shared/t'-2D-shared/t'-$2D^{(+)}$-shared by a dealer $\mathbf{D}\in\mathcal{P}'$ (any player from $\mathcal{P}'$ may perform the role of a dealer), then we denote the sharing by $[s]^{\mathbf{D}}_{t'}/[[s]]^{\mathbf{D}}_{t'}/\langle\langle s\rangle\rangle^{\mathbf{D}}_{t'}$. Similarly if a set of ℓ secrets $s^{(1)},\ldots,s^{(\ell)}$ are t'-$2D^{(+,\ell)}$-shared by player $\mathbf{D}$, we denote it by $\langle\langle s^1,\ldots,s^\ell\rangle\rangle^{\mathbf{D}}_{t'}$. Notice that when a secret s is t'-$2D^{(+)}$-shared, then s is also t'-1D-Shared and t'-2D-shared by default. Hence t'-$2D^{(+)}$-sharing is the strongest sharing among t'-1D-sharing, t'-2D-sharing and t'-$2D^{(+)}$-sharing. In some sense, t'-$2D^{(+,\ell)}$-sharing is the extension of t'-$2D^{(+)}$-sharing for ℓ secrets. If a dealer $\mathbf{D}\in\mathcal{P}'$ is honest, then he will always correctly t'-1D-share/t'-2D-share/t'-$2D^{(+)}$-share a secret s. Among these three types of sharing, t'-$2D^{(+)}$-sharing of a secret s allows efficient reconstruction of the secret with n' players. However, a corrupted $\mathbf{D}$ may perform sharing in an incorrect way. To achieve parallelism, in the sequel, we describe a protocol called $2D^{(+,\ell)}$**Share** which allows a dealer $\mathbf{D}\in\mathcal{P}'$ to *verifiably* t'-$2D^{(+,\ell)}$-share $\ell\geq 1$ length secret $[s^{(1)},s^{(2)},\ldots,s^{(\ell)}]$. The protocol ensures correct t'-$2D^{(+,\ell)}$-sharing even for a corrupted $\mathbf{D}$ and is given in Table 2.

The goal of the protocol is as follows: (a) If $\mathbf{D}$ is honest then he correctly generates t'-$2D^{(+,\ell)}$-sharing of the secret $[s^{(1)},s^{(2)},\ldots,s^{(\ell)}]$, such that all the honest players publicly verify that $\mathbf{D}$ has correctly generated the sharing. Also when $\mathbf{D}$ is honest, then the secret will be information theoretically secure from the adversary $\mathcal{A}_t$. (b) If $\mathbf{D}$ is corrupted and has not generated correct t'-$2D^{(+,\ell)}$-sharing, then with very high probability, everybody will detect it.

Lemma 3. *Protocol $2D^{(+,\ell)}$**Share** correctly generates t'-$2D^{(+,\ell)}$-sharing of ℓ field elements, with overwhelming probability. The protocol takes ten rounds communicates, communicates $\mathcal{O}((\ell n^2+n^3)\kappa)$ bits and broadcasts $\mathcal{O}((\ell n^2+n^3)\kappa)$ bits.*

Proof: See full version of the paper [13]. □

Remark: When an ℓ length secret $[s^{(1)},\ldots,s^{(\ell)}]$ is t'-$2D^{(+,\ell)}$-shared, then implicitly the individual secrets are t'-1D-shared by polynomials $g_0^{(1)}(y),\ldots,g_0^{(\ell)}(y)$. Also note that given $\langle\langle a^{(1)},\ldots,a^{(\ell)}\rangle\rangle_{t'}$ and $\langle\langle b^{(1)},\ldots,b^{(\ell)}\rangle\rangle_{t'}$, the players in $\mathcal{P}'$ can compute $\langle\langle c^{(1)},\ldots,c^{(\ell)}\rangle\rangle_{t'}$ where for $l=1,\ldots,\ell$, $c^{(l)}=\mathcal{F}(a^{(l)},b^{(l)})$ and $\mathcal{F}$ denotes any linear combination.

Conversion From a t'-$2D^{(+,\ell)}$-sharing to ℓ t'-$2D^{+}$-sharing: Given $\langle\langle s^{(1)},s^{(2)},\ldots,s^{(\ell)}\rangle\rangle_{t'}$, we present a protocol **Convert**$2D^{(+,\ell)}$**to**$2D^{+}$ which produces the ℓ t'-$2D^{+}$-sharing of the individual ℓ secrets, namely $\langle\langle s^{(l)}\rangle\rangle_{t'}$ for $l=1,\ldots,\ell$.

$(\langle\langle s^{(1)}\rangle\rangle_{t'}, \ldots, \langle\langle s^{(\ell)}\rangle\rangle_{t'}) = \mathbf{Convert}2D^{(+,\ell)}\mathbf{to}2D^{+}(\mathcal{P}', t', \ell, \langle\langle s^{(1)}, s^{(2)}, \ldots, s^{(\ell)}\rangle\rangle_{t'})$

Let $f_i^{(l)}(x), 1 \leq i \leq n', 1 \leq l \leq \ell$ be the polynomials used for generating $\langle\langle s^{(1)} \ldots, s^{(\ell)}\rangle\rangle_{t'}$. For every pair of players P_i and P_j from $\mathcal{P}'$, the following is done:

1. Player P_i as a dealer executes **Distr** and **AuthVal** of $\mathbf{IC}(P_i, P_j, \mathcal{P}, f_i^{(l)}(j))$ for all $l \in \{1, \ldots, \ell\}$ to give $ICSig_{f_i^{(l)}(j)}(P_i, P_j)$ to P_j. Since $\langle\langle s^{(1)}, \ldots, s^{(\ell)}\rangle\rangle_{t'}$ is generated using $2D^{(+,\ell)}$**Share**, it implies that either P_j already holds $ICSig_{(f_i^{(1)}(j), f_i^{(2)}(j), \ldots, f_i^{(\ell)}(j))}(P_i, P_j)$ from P_i or every player from $\mathcal{P}'$ has ignored P_i's signature. In the later case, players in $\mathcal{P}'$ will again ignore P_i's signature. Otherwise P_j can now check if P_i has given signature on the same individual values.

2. Upon receiving the signatures on say $\bar{f}_i^{(1)}(j), \ldots, \bar{f}_i^{(\ell)}(j)$ (i.e $ICSig_{\bar{f}_i^{(l)}(j)}(P_i, P_j)$ for $l = 1, \ldots, \ell$) , P_j checks $f_i^{(l)}(j) \stackrel{?}{=} \bar{f}_i^{(l)}(i)$. If there is inconsistency for some $l \in \{1, \ldots, \ell\}$ then P_j along with all players in $\mathcal{P}'$ invoke $\mathbf{EfficientRevealVal}(P_i, P_j, \mathcal{P}', \ell, f_i^{(1)}(j), f_i^{(2)}(j), \ldots, f_i^{(\ell)}(j))$ and $\mathbf{RevealVal}(P_i, P_j, \mathcal{P}', \bar{f}_i^{(l)}(j))$ for all $l \in \{1, \ldots, \ell\}$.

3. In the previous step, if P_j is not able to produce the signature that he received from P_i, then all the players from $\mathcal{P}'$ ignore the IC signatures received from P_j during step 1. Otherwise if the signatures are valid then $f_i^{(1)}(j), \ldots, f_i^{(\ell)}(j)$ and $\bar{f}_i^{(1)}(j), \ldots, \bar{f}_i^{(\ell)}(j)$ are public. All players in $\mathcal{P}'$ check $f_i^{(l)}(j) \stackrel{?}{=} \bar{f}_i^{(l)}(j)$ for $l = 1, \ldots, \ell$. If the test fails for some l, then all the players in $\mathcal{P}'$ ignore the values received from P_i during first step. Otherwise the signature produced by P_j will be ignored.

Lemma 4. *Protocol* $\mathbf{Convert}2D^{(+,\ell)}\mathbf{to}2D^{+}$ *takes five rounds and communicates* $\mathcal{O}((\ell n^3 + n^4)\kappa)$ *bits and broadcasts* $\mathcal{O}((\ell n^3 + n^4)\kappa)$ *bits.*

Reconstruction of a t'-$2D^{+}$-shared secret: Let $\langle\langle s\rangle\rangle_{t'}$ be a t'-$2D^{+}$-sharing, shared using the polynomials $H(x, y), f_i(x), g_i(y), 1 \leq i \leq n'$ among the players in $\mathcal{P}'$. We now present a protocol $2D^{+}$**Recons** which allows the (honest) players to correctly recover s with very high probability.

$s = 2D^{+}\mathbf{Recons}(\mathcal{P}', t', \langle\langle s\rangle\rangle_{t'})$

For all $P_j \in \mathcal{P}'$ such that P_j's IC signatures are not ignored by the players in $\mathcal{P}'$, player P_i sends $ICSig_{f_j(i)}(P_j, P_i)$ to every player P_k in $\mathcal{P}'$. Player $P_k \in \mathcal{P}'$ checks the validity of $ICSig_{f_j(i)}(P_j, P_i)$ with respect to his own *verification information*. If the verification passes then P_k accepts $ICSig_{f_j(i)}(P_j, P_i)$. Now if for all $P_j \in \mathcal{P}'$, P_k accepts $ICSig_{f_j(i)}(P_j, P_i)$ (which he receives from P_i) then P_k checks whether $f_j(i)$s are t'-consistent (ideally $f_j(i) = g_i(j)$ for all $P_j \in \mathcal{P}'$; so $f_j(i)$s will lie on t' degree polynomial $g_i(y)$). If yes then P_k adds P_i to his $CORE$ set and let the t' degree polynomial (on which $f_j(i)$s lie on) be $g_i(y)$. Player P_k takes all the $g_i(y)$ polynomials corresponding to the players in his $CORE$ and interpolates the bivariate polynomial $H(x, y)$ and finally sets the secret $s = H(0, 0)$. It is easy to check that all honest players from $\mathcal{P}'$ recovers the same secret s.

Lemma 5. *Protocol* $2D^{+}$**Recons** *takes one round and privately communicates* $\mathcal{O}(n^3\kappa)$ *bits.*

2.3 Generating Random t'-$2D^{(+,\ell)}$-Sharing

We now present protocol $\mathbf{Random}(\mathcal{P}', t', \ell)$ in Table 3, which allows the players in $\mathcal{P}'$ to jointly generate a random t'-$2D^{(+,\ell)}$-sharing, $\langle\langle r^{(1)}, \ldots, r^{(\ell)}\rangle\rangle_{t'}$.

Table 2. $\langle\langle s^{(1)},\ldots,s^{(\ell)}\rangle\rangle_{t'}^{\mathbf{D}} = 2D^{(+,\ell)}\mathbf{Share}(\mathbf{D},\mathcal{P}',t',\ell,s^{(1)},\ldots,s^{(\ell)})$

1. For every $l = 1,\ldots,\ell$, $\mathbf{D}$ picks a random bivariate polynomial $H^{(l)}(x,y)$ of degree t' in both the variables, with $H^{(l)}(0,0) = s^{(l)}$. Let $f_i^{(l)}(x) = H^{(l)}(x,i)$ and $g_i^{(l)}(y) = H^{(l)}(i,y)$. Now $\mathbf{D}$ wants to hand over n' values on $f_i^{(l)}(x)$ and $g_i^{(l)}(y)$ for $l = 1,\ldots,\ell$ to P_i with his IC signature on them. For that $\mathbf{D}$ executes **EfficientDistr** and **EfficientAuthVal** of $\mathbf{EfficientIC}(\mathbf{D},P_i,\mathcal{P}',\ell,f_i^{(1)}(j),f_i^{(2)}(j),\ldots,f_i^{(\ell)}(j))$ for for all $j \in \{1,\ldots,n'\}$. Similarly $\mathbf{D}$ executes **EfficientDistr** and **EfficientAuthVal** of $\mathbf{EfficientIC}(\mathbf{D},P_i,\mathcal{P}',\ell,g_i^{(1)}(j),g_i^{(2)}(j),\ldots,g_i^{(\ell)}(j))$.

2. For $l = 1,\ldots,\ell$, P_i checks whether the sets $f_i^{(l)}(1),\ldots,f_i^{(l)}(n')$ and $g_i^{(l)}(1),\ldots,g_i^{(l)}(n')$ are t'-consistent. If the values are not t'-consistent, for some $l \in \{1,\ldots,\ell\}$ then P_i along with all players in $\mathcal{P}'$ invoke $\mathbf{EfficientRevealVal}(\mathbf{D},P_i,\mathcal{P}',\ell,f_i^{(1)}(j),f_i^{(2)}(j),\ldots,f_i^{(\ell)}(j))$ and $\mathbf{EfficientRevealVal}(\mathbf{D},P_i,\mathcal{P}',\ell,g_i^{(1)}(j),g_i^{(2)}(j),\ldots,g_i^{(\ell)}(j))$ for all $j \in \{1,\ldots,n'\}$. If the signatures produced by P_i are valid and for some $l \in \{1,\ldots,\ell\}$, either $f_i^{(l)}(1),\ldots,f_i^{(l)}(n')$ or $g_i^{(l)}(1),\ldots,g_i^{(l)}(n')$ is not t'-consistent, then the protocol terminates without generating desired output.

3. For every pair of players P_i and P_j from $\mathcal{P}'$ the following will be executed:

(a) P_i as a dealer executes **EfficientDistr** and **EfficientAuthVal** of $\mathbf{EfficientIC}(P_i,P_j,\mathcal{P}',\ell,f_i^{(1)}(j),\ldots,f_i^{(\ell)}(j))$ to give his IC signature on $f_i^{(1)}(j),\ldots,f_i^{(\ell)}(j)$ to P_j. Upon receiving the signature, P_j checks whether $f_i^{(l)}(j) \stackrel{?}{=} g_j^{(l)}(i)$ for $l = 1,\ldots,\ell$. If there is an inconsistency then P_j along with all players in $\mathcal{P}'$ invoke $\mathbf{EfficientRevealVal}(\mathbf{D},P_j,\mathcal{P}',\ell,g_j^{(1)}(i),g_j^{(2)}(i),\ldots,g_j^{(\ell)}(i))$.

(b) If P_j fails to produce valid signature in the previous step, then all the players from $\mathcal{P}'$ ignore the IC signatures received from P_j in previous step. Otherwise, if P_j is able to produce valid signature then $g_j^{(1)}(i),g_j^{(2)}(i),\ldots,g_j^{(\ell)}(i)$ become public. Using the public values P_i checks whether $f_i^{(l)}(j) \stackrel{?}{=} g_j^{(l)}(i)$ for $l = 1,\ldots,\ell$. If he finds any inconsistency, then P_i along with all players in $\mathcal{P}'$ invoke $\mathbf{EfficientRevealVal}(\mathbf{D},P_i,\mathcal{P}',\ell,f_i^{(1)}(j),f_i^{(2)}(j),\ldots,f_i^{(\ell)}(j))$.

(c) If P_i fails to produce valid signature in the previous step, then all the players from $\mathcal{P}'$ ignore the IC signatures received from P_i in step 3(a). Else if P_i is able to produce valid signature, then all the values $f_i^{(1)}(j),f_i^{(2)}(j),\ldots,f_i^{(\ell)}(j)$ become public. Every player then verifies $f_i^{(l)}(j) \stackrel{?}{=} g_j^{(l)}(i)$ for $l = 1,\ldots,\ell$. If $f_i^{(l)}(j) \neq g_j^{(l)}(i)$ for some $l \in \{1,\ldots,\ell\}$ then the protocol terminates without generating the desired output.

Lemma 6. *With overwhelming probability,* **Random** *generates random* $\langle\langle r^{(1)},\ldots,r^{(\ell)}\rangle\rangle_{t'}$ *in eight rounds and privately communicates and broadcasts* $\mathcal{O}((\ell n^3+n^4)\kappa)$ *bits.*

2.4 Proving $c = ab$

Definition 5. t'-$1D^{(+)}$**-sharing:** *A value s is correctly t'-$1D^{(+)}$-shared among the players in $\mathcal{P}'$, denoted by $\langle s\rangle_{t'}$, if there exists t' degree polynomial $f(x)$ held*

Table 3. Protocol for Generating ℓ Random t'-$2D^{(+,\ell)}$-Sharing

$\langle\langle r^{(1)},\ldots,r^{(\ell)}\rangle\rangle_{t'} = \mathbf{Random}(\mathcal{P}',t',\ell)$

Every player $P_i \in \mathcal{P}'$ invokes $2D^{(+,\ell)}\mathbf{Share}(P_i,\mathcal{P}',t',\ell,r^{(1,P_i)},\ldots,r^{(\ell,P_i)})$ to generate $\langle\langle r^{(1,P_i)},\ldots,r^{(\ell,P_i)}\rangle\rangle_{t'}^{P_i}$, where $r^{(1,P_i)},\ldots,r^{(\ell,P_i)}$ are randomly selected from $\mathbb{F}$. Let $Pass$ denotes the set of players P_i in $\mathcal{P}'$ such that t'-$2D^{(+,\ell)}\mathbf{Share}(P_i,\mathcal{P}',t',\ell,r^{(1,P_i)},\ldots,r^{(\ell,P_i)})$ is executed successfully. Now all the players in $\mathcal{P}'$ jointly computes $\langle\langle r^{(1)},\ldots,r^{(\ell)}\rangle\rangle_{t'} = \sum_{P_i\in Pass}\langle\langle r^{(1,P_i)},\ldots,r^{(\ell,P_i)}\rangle\rangle_{t'}^{P_i}$.

by $\mathbf{D}$ *with* $f(0) = s$. *Every player* $P_i \in \mathcal{P}'$ *holds a share* $s_i = f(i)$ *of* s *with an IC signature on it from the dealer* $\mathbf{D}$ *(i.e.* $ICSig_{s_i}(\mathbf{D}, P_i)$*).*

Definition 6. t'-$1D^{(+,\ell)}$**-sharing:** *We say that a set of secrets* $s^{(1)}, \ldots, s^{(\ell)}$ *are correctly* t'-$1D^{(+,\ell)}$*-shared among the players in* $\mathcal{P}'$ *if there exists* t' *degree polynomials* $f^{(1)}, \ldots, f^{(\ell)}$ *held by* $\mathbf{D}$*, with* $f^{(l)}(0) = s^{(l)}$ *for* $l = 1, \ldots, \ell$. *Every player* $P_i \in \mathcal{P}'$ *holds shares* $s_i^{(1)} = f^{(1)}(i), \ldots, s_i^{(\ell)} = f^{(\ell)}(i)$ *of* $s^{(1)}, \ldots, s^{(\ell)}$ *along with a single IC signature on them from the dealer* $\mathbf{D}$ *(i.e.* $ICSig_{(s_i^{(1)}, \ldots, s_i^{(\ell)})}(\mathbf{D}, P_i)$*)). We denote the* t'-$1D^{(+,\ell)}$*-sharing of* ℓ *length secret by* $\langle s^1, \ldots, s^\ell \rangle_{t'}$.

If ℓ secrets $s^{(1)}, \ldots, s^{(\ell)}$ are t'-$1D^{(+,\ell)}$-shared by player $\mathbf{D}$, we denote it by $\langle s^1, \ldots, s^\ell \rangle_{t'}^{\mathbf{D}}$. Now let $\mathbf{D} \in \mathcal{P}'$ has already correctly t'-$1D^{(+,\ell)}$-shared $a^{(1)}, \ldots, a^{(\ell)}$ and $b^{(1)}, \ldots, b^{(\ell)}$ among the players in $\mathcal{P}'$. Now $\mathbf{D}$ wants to correctly t'-$2D^{(+,\ell)}$-share $c^{(1)}, \ldots, c^{(\ell)}$ without leaking any *additional* information about $a^{(l)}$, $b^{(l)}$ and $c^{(l)}$, such that every (honest) player in $\mathcal{P}'$ knows that $c^{(l)} = a^{(l)} b^{(l)}$ for $l = 1, \ldots, \ell$. We propose a protocol **ProveCeqAB**, given in Table 4 to achieve this task. The idea of the protocol is inspired from [10] with the following modification: we make use of our protocol $2D^{(+,\ell)}$-**Share**, which provides us with high efficiency. For a detailed explanation, see the full version of the paper [13].

Lemma 7. *In* **ProveCeqAB***, if* $\mathbf{D}$ *does not fail, then with overwhelming probability, every* $(a^{(l)}, b^{(l)})$, $c^{(l)}$ *satisfies* $c^{(l)} = a^{(l)} b^{(l)}$. **ProveCeqAB** *takes twenty five rounds and communicates* $\mathcal{O}((\ell n^2 + n^4)\kappa)$ *bits and broadcasts* $\mathcal{O}((\ell n^2 + n^4)\kappa)$ *bits. Moreover, if* $\mathbf{D}$ *is honest then* $a^{(l)}, b^{(l)}$ *and* $c^{(l)}$ *are information theoretically secure.*

2.5 Multiplication

Let two sets of ℓ values $a^{(1)}, \ldots, a^{(\ell)}$ and $b^{(1)}, \ldots, b^{(\ell)}$ are correctly t'-$2D^{(+,\ell)}$-shared among the players in $\mathcal{P}'$, i.e. $\langle\langle a^{(1)}, \ldots, a^{(\ell)} \rangle\rangle_{t'}$ and $\langle\langle b^{(1)}, \ldots, b^{(\ell)} \rangle\rangle_{t'}$. We now present a protocol called **Mult**, given in Table 5, which allows the players to compute t'-$2D^{(+,\ell)}$-sharing $\langle\langle c^{(1)}, \ldots, c^{(\ell)} \rangle\rangle_{t'}$ such that $c^{(l)} = a^{(l)} b^{(l)}$ for $l = 1, \ldots, \ell$. Our protocol is based on the technique used in [10] with the following difference: we make use of our protocol **ProveCeqAB**, which provides us with high efficiency. For full details, see the full version of the paper [13].

Lemma 8. *With overwhelming probability, protocol* **Mult** *produces* $\langle\langle c^{(1)}, \ldots, c^{(\ell)} \rangle\rangle_{t'}$ *from* $\langle\langle a^{(1)}, \ldots, a^{(\ell)} \rangle\rangle_{t'}$ *and* $\langle\langle b^{(1)}, \ldots, b^{(\ell)} \rangle\rangle_{t'}$ *such that* $c^{(l)} = a^{(l)} b^{(l)}$ *if less then* $n' - 2t'$ *players are added to* $\mathcal{C}$. **Mult** *takes twenty five rounds and communicates* $\mathcal{O}((\ell n^3 + n^5)\kappa)$ *bits and broadcasts* $\mathcal{O}((\ell n^3 + n^5)\kappa)$ *bits. Moreover,* $c^{(l)}, a^{(l)}$ *and* $b^{(l)}$ *remains secure.*

2.6 Proving a=b

Consider the following scenario: Let $\mathbf{D} \in \mathcal{P}'$ has t'-$1D^{(+,\ell)}$-shared ℓ values $a^{(1)}, \ldots, a^{(\ell)}$ among the players in $\mathcal{P}'$. Now some more computation has been carried out after the sharing done by $\mathbf{D}$ and during the computation some players have

Table 4. $\langle\langle c^{(1)},\ldots,c^{(\ell)}\rangle\rangle_{t'}^{\mathbf{D}}$ =**ProveCeqAB(D,**$\mathcal{P}', t', \ell, \langle a^{(1)},\ldots,a^{(\ell)}\rangle_{t'}^{\mathbf{D}}, \langle b^{(1)},\ldots,b^{(\ell)}\rangle_{t'}^{\mathbf{D}}$**)**

1. **D** randomly generates $(\beta^{(1)},\ldots,\beta^{(\ell)}) \in \mathbb{F}^{\ell}$. and invokes $2D^{(+,\ell)}$**Share(D,** $\mathcal{P}', t', \ell, c^{(1)},\ldots,c^{(\ell)}$**)**, $2D^{(+,\ell)}$**Share(D,** $\mathcal{P}', t', \ell, \beta^{(1)},\ldots,\beta^{(\ell)}$**)** and $2D^{(+,\ell)}$**Share(D,** $\mathcal{P}', t', \ell, b^{(1)}\beta^{(1)},\ldots,b^{(\ell)}\beta^{(\ell)}$**)**. If any of the **Share** protocol fails, then **D** fails and the protocol terminates. For $l = 1,\ldots,\ell$, let $a^{(l)}$, $b^{(l)}$, $c^{(l)}$, $\beta^{(l)}$ and $\beta^{(l)}b^{(l)}$ are implicitly shared using polynomials $f^{a^{(l)}}(x)$, $f^{b^{(l)}}(x)$, $f^{c^{(l)}}(x)$, $f^{\beta^{(l)}}(x)$ and $f^{\beta^{(l)}b^{(l)}}(x)$ respectively.

2. Players in $\mathcal{P}'$ jointly generate a random number r. This is done as follows: first the players in $\mathcal{P}$ execute the protocol **Random**$(\mathcal{P}', t', 1)$ to generate $\langle\langle r\rangle\rangle_{t'}$. Then the players compute $r = 2D^{+}$**Recons**$(\mathcal{P}', t', \langle\langle r\rangle\rangle_{t'})$.

3. **D** broadcasts $F^{(l)}(x) = rf^{a^{(l)}}(x) + f^{\beta^{(l)}}(x)$ for $l = 1\ldots,\ell$.

4. Player $P_i \in \mathcal{P}'$ checks $F^{(l)}(i) \stackrel{?}{=} rf^{a^{(l)}}(i) + f^{\beta^{(l)}}(i)$ for $l = 1,\ldots,\ell$. If the test fails for at least one l, then P_i and all players invoke **EfficientRevealVal(D,** $P_i, \mathcal{P}', \ell, f^{a^{(1)}}(i),\ldots,f^{a^{(\ell)}}(i)$**)** and **EfficientRevealVal(D,** $P_i, \mathcal{P}', \ell, f^{\beta^{(1)}}(i),\ldots,f^{\beta^{(\ell)}}(i)$**)**. If the signature is invalid, ignore P_i's complaints. Otherwise all the values $(f^{a^{(1)}}(i),\ldots,f^{a^{(\ell)}}(i))$ and $(f^{\beta^{(1)}}(i),\ldots,f^{\beta^{(\ell)}}(i))$ become public. Using these values all players publicly checks $F^{(l)}(i) \stackrel{?}{=} rf^{a^{(l)}}(i) + f^{\beta^{(l)}}(i)$ for $l = 1,\ldots,\ell$. If the test fails for at least one l, then **D** fails and the protocol terminates here.

5. **D** broadcasts $G^{(l)}(x) = F^{(l)}(0)f^{b^{(l)}}(x) - f^{\beta^{(l)}b^{(l)}}(x) - rf^{c^{(l)}}(x)$ for $l = 1\ldots,\ell$.

6. Player $P_i \in \mathcal{P}'$ checks $G^{(l)}(i) \stackrel{?}{=} F^{(l)}(0)f^{b^{(l)}}(i) - f^{\beta^{(l)}b^{(l)}}(i) - rf^{c^{(l)}}(i)$. If the test fails for at least one l, then P_i and all players in $\mathcal{P}'$ invoke **EfficientRevealVal(D,** $P_i, \mathcal{P}', \ell, f^{b^{(1)}}(i),\ldots,f^{b^{(\ell)}}(i)$**)**, **EfficientRevealVal(D,** $P_i, \mathcal{P}', \ell, f^{\beta^{(1)}b^{(1)}}(i),\ldots,f^{\beta^{(\ell)}b^{(\ell)}}(i)$**)** and **EfficientRevealVal(D,** $P_i, \mathcal{P}', \ell, f^{c^{(1)}}(i),\ldots,f^{c^{(\ell)}}(i)$**)**.

7. If the signature is invalid, ignore P_i's complaint. Otherwise all the values $(f^{b^{(1)}}(i),\ldots,f^{b^{(\ell)}}(i))$, $(f^{\beta^{(1)}b^{(1)}}(i),\ldots,f^{\beta^{(\ell)}b^{(\ell)}}(i))$ and $(f^{c^{(1)}}(i),\ldots,f^{c^{(\ell)}}(i))$ become public. Using these values all players publicly checks $G^{(l)}(i) \stackrel{?}{=} F^{(l)}(0)f^{b^{(l)}}(i) - f^{\beta^{(l)}b^{(l)}}(i) - rf^{c^{(l)}}(i)$ for $l = 1,\ldots,\ell$. If the test fails for at least one l, then **D** fails and protocol terminates here.

8. Every player checks whether $G^{(l)} \stackrel{?}{=} 0$ for $l = 1,\ldots,\ell$. If the test fails for at least one l, then **D** fails and protocol terminates here. Otherwise **D** has proved that $c^{(l)} = a^{(l)}b^{(l)}$ for $l = 1,\ldots,\ell$.

been detected as faulty and removed from $\mathcal{P}'$. Let us denote the snapshot of $\mathcal{P}'$ before and after the computation by $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively. Also assume $|\mathcal{P}_1| = n_1$ and the number of corrupted players in $\mathcal{P}_1$ is t_1 with $n_1 \geq t + 1 + t_1$. Similarly $|\mathcal{P}_2| = n_2$ and the number of corrupted players in $\mathcal{P}_2$ is t_2 with $n_2 \geq t+1+t_2$, $t_1 > t_2$. Now **D** wants to correctly t_2-$2D^{(+,\ell)}$-share $b^{(1)},\ldots,b^{(\ell)}$ among the players of $\mathcal{P}_2$ such that $b^{(l)} = a^{(l)}$, without leaking any *additional* information about $a^{(l)}$. We propose a protocol **ProveAeqB** to achieve this task.

Lemma 9. *In protocol* **ProveAeqB**, *if* **D** *does not fail, then with overwhelming probability, every* $(a^{(l)}, b^{(l)})$ *satisfies* $a^{(l)} = b^{(l)}$. **ProveAeqB** *takes thirteen rounds and communicates and broadcasts* $\mathcal{O}((\ell n^2 + n^3)\kappa)$ *bits. Moreover, if* **D** *is honest then* $\mathcal{A}_t$ *learns no information about* $a^{(l)}$, *for* $1 \leq l \leq \ell$.

2.7 Resharing

As described in previous section, consider the time-stamps before and after some computation where before and after the computation, $\mathcal{P}'$ is denoted by $\mathcal{P}_1$ and

Table 5. $\langle\langle c^{(1)},\ldots,c^{(\ell)}\rangle\rangle_{t'} = \mathbf{Mult}(\mathcal{P}',\ell,\langle\langle a^{(1)},\ldots,a^{(\ell)}\rangle\rangle_{t'},\langle\langle b^{(1)},\ldots,b^{(\ell)}\rangle\rangle_{t'})$

1. Given $\langle\langle a^{(1)},\ldots,a^{(\ell)}\rangle\rangle_{t'}$, it implies that i^{th} shares of $a^{(1)},\ldots,a^{(\ell)}$ are t'-$1D^{(+)}$-shared by P_i i.e. $\langle a_i^{(1)},\ldots,a_i^{(\ell)}\rangle_{t'}^{P_i}$ for $P_i \in \mathcal{P}'$. Similarly we have $\langle b_i^{(1)},\ldots,b_i^{(\ell)}\rangle_{t'}^{P_i}$.Player P_i invokes $\mathbf{ProveCeqAB}(P_i,\mathcal{P}',t',\ell,\langle a_i^{(1)},\ldots,a_i^{(\ell)}\rangle_{t'}^{P_i},\langle b_i^{(1)},\ldots,b_i^{(\ell)}\rangle_{t'}^{P_i})$ to generate $\langle\langle c_i^{(1)},\ldots,c_i^{(\ell)}\rangle\rangle_{t'}^{P_i}$.
2. If at least $n'-2t'$ players fails in executing **ProveCeqAB**, then remove them from $\mathcal{P}'$, adjust t' and terminate the protocol without generating the expected result. Otherwise for simplicity assume that the first $2t'+1$ players are successful in executing **ProveCeqAB**.
3. All the players compute: $\langle\langle c^{(1)},\ldots,c^{(\ell)}\rangle\rangle_{t'} = \sum_{i=1}^{2t'+1} r_i\langle\langle c_i^{(1)},\ldots,c_i^{(\ell)}\rangle\rangle_{t'}^{P_i}$, where $(r_1,\ldots,r_{2t'+1})$ represents the recombination vector [9].

$\langle\langle a^{(1)},\ldots,a^{(\ell)}\rangle\rangle_{t_2}^{\mathbf{D}} = \mathbf{ProveAeqB}(\mathbf{D},\mathcal{P}_2,t_1,t_2,\ell,\langle a^{(1)},\ldots,a^{(\ell)}\rangle_{t_1}^{\mathbf{D}})$

1. $\mathbf{D}$ invokes $2D^{(+,\ell)}\mathbf{Share}(\mathbf{D},\mathcal{P}_2,t_2,\ell,a^{(1)},\ldots,a^{(\ell)})$ to verifiably t_2-$2D^{(+,\ell)}$-share $(a^{(1)},\ldots,a^{(\ell)})$. $\mathbf{D}$ selects a random ℓ length tuple $(c^{(1)},\ldots,c^{(\ell)}) \in \mathbb{F}^\ell$ and invokes $2D^{(+,\ell)}\mathbf{Share}(\mathbf{D},\mathcal{P}_2,t_1-1,\ell,c^{(1)},\ldots,c^{(\ell)})$. If protocol $2D^{(+,\ell)}\mathbf{Share}$ fails then $\mathbf{D}$ fails here and the protocol terminates here. Otherwise for convenience we say that $\mathbf{D}$ has t_2-$2D^{(+,\ell)}$-shared $(b^{(1)},\ldots,b^{(\ell)})$. For an honest $\mathbf{D}$, $a^{(l)} = b^{(l)}$ for $l = 1,\ldots,\ell$.
2. For $l = 1,\ldots,\ell$, let $a^{(l)}$, $b^{(l)}$ and $c^{(l)}$ are implicitly shared using polynomials $f^{a^{(l)}}(x)$ (degree t_1), $f^{b^{(l)}}(x)$ (degree t_2) and $f^{c^{(l)}}(x)$ (degree t_1-1) respectively. Now $\mathbf{D}$ broadcasts the polynomials $F^{(l)}(x) = f^{a^{(l)}}(x) + xf^{c^{(l)}}(x) - f^{b^{(l)}}(x)$.
3. Player $P_i \in \mathcal{P}_2$ checks whether $F^{(l)}(i) \stackrel{?}{=} f^{a^{(l)}}(i) + if^{c^{(l)}}(i) - f^{b^{(l)}}(i)$. If the test fails for at least one l, $\mathbf{EfficientRevealVal}(\mathbf{D},P_i,\mathcal{P}_1,\ell,f^{a^{(1)}}(i),\ldots,f^{a^{(\ell)}}(i))$, $\mathbf{EfficientRevealVal}(\mathbf{D},P_i,\mathcal{P}_2,\ell,f^{b^{(1)}}(i),\ldots,f^{b^{(\ell)}}(i))$ and $\mathbf{EfficientRevealVal}(\mathbf{D},P_i,\mathcal{P}_2,\ell,f^{c^{(1)}}(i),\ldots,f^{c^{(\ell)}}(i))$ are invoked.
4. If the signatures are invalid, then ignore P_i's complaints. Otherwise all the values $(f^{a^{(1)}}(i),\ldots,f^{a^{(\ell)}}(i))$, $(f^{b^{(1)}}(i),\ldots,f^{b^{(\ell)}}(i))$ and $(f^{c^{(1)}}(i),\ldots,f^{c^{(\ell)}}(i))$ become public. Using these values all players publicly checks whether $F^{(l)}(i) \stackrel{?}{=} f^{a^{(l)}} + if^{c^{(l)}}(i) - f^{b^{(l)}}(i)$ for $l = 1,\ldots,\ell$. If the test fails for at least one l, then $\mathbf{D}$ fails and protocol terminates here.
5. Every player checks $F^{(l)}(0) \stackrel{?}{=} 0$ for $l = 1,\ldots,\ell$. If the test fails for at least one l, then $\mathbf{D}$ fails and protocol terminates here. Else $\mathbf{D}$ has proved that $a^{(l)} = b^{(l)}$.

$\mathcal{P}_2$ respectively. Let the players in $\mathcal{P}_1$ holds a t_1-$2D^{(+,\ell)}$-sharing of ℓ values $s^{(1)},\ldots,s^{(\ell)}$ i.e. $\langle\langle s^{(1)},\ldots,s^{(\ell)}\rangle\rangle_{t_1}$. Now the players want to jointly generate t_2-$2D^{(+,\ell)}$-sharing of same values i.e $\langle\langle s^{(1)},\ldots,s^{(\ell)}\rangle\rangle_{t_2}$ among the players in $\mathcal{P}_2$ where $t_2 < t_1$. This is done by protocol **Reshare**.

$\langle\langle a^{(1)},\ldots,a^{(\ell)}\rangle\rangle_{t_2} = \mathbf{Reshare}(\mathcal{P}',t_1,t_2,\ell,\langle\langle a^{(1)},\ldots,a^{(\ell)}\rangle\rangle_{t_1})$

1. Given $\langle\langle a^{(1)},\ldots,a^{(\ell)}\rangle\rangle_{t_1}$, it implies that the i^{th} shares of $a^{(1)},\ldots,a^{(\ell)}$ are already t_1-$1D^{(+,\ell)}$-shared by P_i, i.e. $\langle a_i^{(1)},\ldots,a_i^{(\ell)}\rangle_{t_1}^{P_i}$ for $P_i \in \mathcal{P}_1$. Player P_i in $\mathcal{P}_2$ invokes $\mathbf{ProveAeqB}(P_i,\mathcal{P}_2,t_1,t_2,\ell,\langle a_i^{(1)},\ldots,a_i^{(\ell)}\rangle_{t_1}^{P_i})$ to generate $\langle\langle a_i^{(1)},\ldots,a_i^{(\ell)}\rangle\rangle_{t_2}^{P_i}$. For simplicity assume that first t_1+1 players are successful in **ProveAeqB**. Since $n_2 = t+1+t_2$ and $t > t_1$, at least $t+1 \geq t_1+1$ honest players will always be successful.
2. All the players compute: $\langle\langle a^{(1)},\ldots,a^{(\ell)}\rangle\rangle_{t_2} = \sum_{i=1}^{t_1+1} r_i\langle\langle a_i^{(1)},\ldots,a_i^{(\ell)}\rangle\rangle_{t_2}^{P_i}$, where $(r_1,\ldots,r_{t_1+1})$ represents the recombination vector.

Lemma 10. *With overwhelming probability, protocol* **Reshare** *produces* t_2-$2D^{(+,\ell)}$-*sharing* $\langle\langle a^{(1)},\ldots,a^{(\ell)}\rangle\rangle_{t_2}$ *from* $\langle\langle a^{(1)},\ldots,a^{(\ell)}\rangle\rangle_{t_1}$ *with* $t_2 < t_1$. **Reshare** *takes thirteen rounds and communicates* $\mathcal{O}((\ell n^3 + n^4)\kappa)$ *bits and broadcasts* $\mathcal{O}((\ell n^3 + n^4)\kappa)$ *bits. Moreover,* $\mathcal{A}_t$ *learns nothing about* $a^{(l)}$, *for* $1 \leq l \leq \ell$.

2.8 Preparation Phase

We call a triple (a, b, c) as a multiplication triple if $c = ab$. The goal of the preparation phase is to generate correct d-$2D^+$-sharing of $(c_M + c_R)$ secret multiplication triples where d denotes the number of corrupted players still present in $\mathcal{P}'$ at the end of preparation phase. So in total there will be $c_M + c_R$ multiplication triples $(a^{(i)}, b^{(i)}, c^{(i)})$ such that $c^{(i)} = a^{(i)}b^{(i)}$ for $i = 1,\ldots,(c_M + c_R)$. The generation of $c_M + c_R$ multiplication triples is divided into $\lceil\log(t)\rceil$ segments, wherein each segment is responsible for generating d-$2D^{(+,\ell)}$-sharing of $\ell = \lceil\frac{c_M+c_R}{\log t}\rceil$ triples. Here we use a *novel* technique called *rapid player elimination* (RPE) which works in the following way: The computation of a segment is non-robust i.e. a segment may fail due to the (mis)behavior of certain number of corrupted players who reveal their identity during the execution of the segment. At the beginning of preparation phase we set the counter for keeping track the number of (segment) failures to zero i.e $f = 0$. We create a win-win situation, where if a segment fails, then it must be due to the revelation/detection of at least 2^f new corrupted players. After removing the corrupted players from $\mathcal{P}'$, a fresh execution of the same segment will be started with f incremented by 1. This ensures that total $\lceil\log(t)\rceil - 1$ failures can occur (i.e $f \leq \lceil\log(t)\rceil - 1$) since $\lceil\log(t)\rceil - 1$ failures are enough to reveal all the t corrupted players. Once all the t corrupted players are revealed, rest of the computation can run without occurrence of any failure. Specifically, in every segment one of the following occurs: (a) t_1-$2D^{(+,\ell)}$-sharing of $\ell = \lceil\frac{c_M+c_R}{\log t}\rceil$ secret multiplication triples will be generated, where t_1 denotes the number of corrupted players present in $\mathcal{P}'$ ($t_1 = t - |\mathcal{C}|$) at the beginning of the segment's execution; (b) the segment fails with at least 2^f corrupted players being eliminated from $\mathcal{P}'$ (and added to $\mathcal{C}$), where f denotes the number of failures occurred so far. But there are two problems here. We want all the triples to be d-$2D^+$-shared. But since the number of corrupted players in $\mathcal{P}'$ may change dynamically after every failure, the sharing produced in different segment may be of different degree. Also the sharing produced by segments are t_1-$2D^{(+,\ell)}$-sharing where t_1 may vary segment to segment. So, to achieve our goal, we first use protocol **Reshare** to obtain uniform d-$2D^{(+,\ell)}$-sharing for all the segments. Then, we use Protocol **Convert**$2D^{(+,\ell)}$**to**$2D^+$ to produce d-$2D^+$-sharing from d-$2D^{(+,\ell)}$-sharing. By using this approach, we can efficiently generate the triples with less communication overhead. For full details of the protocol, see [13].

Lemma 11. *With overwhelming probability, protocol* **Preparation Phase** *produces correct* d-$2D^+$-*sharing of* (c_M+c_R) *secret multiplication triples in* $\mathcal{O}(\log(t))$ *rounds, communicates* $\mathcal{O}((c_M+c_R)n^3+n^4)\kappa$ *bits and broadcasts* $\mathcal{O}((c_M+c_R)n^3+n^4)\kappa$ *bits. Moreover,* $\mathcal{A}_t$ *learns nothing about* $(a^{(i)}, b^{(i)}, c^{(i)})$ *for* $1 \leq i \leq \ell$.

Preparation Phase

1. Let $\mathcal{P}' = \mathcal{P}$, $n' = n$, $\ell = \lceil \frac{c_M + c_R}{\log t} \rceil$ and $t' = t$. Let $f = 0$.
2. For every segment $k = 1, \ldots, \lceil \log(t) \rceil$, do the following:
   (a) Set $\mathcal{P}_1 = \mathcal{P}'$, $n_1 = n'$ and $t_1 = t'$. Invoke **Random**$(\mathcal{P}_1, t_1, \ell)$ twice in parallel to generate $\langle\langle a_k^{(1)}, \ldots, a_k^{(\ell)} \rangle\rangle_{t_1}$ and $\langle\langle b_k^{(1)}, \ldots, b_k^{(\ell)} \rangle\rangle_{t_1}$. Then Invoke **Mult**$(\mathcal{P}_1, \ell, \langle\langle a_k^{(1)}, \ldots, a_k^{(\ell)} \rangle\rangle_{t_1}, \langle\langle b_k^{(1)}, \ldots, b_k^{(\ell)} \rangle\rangle_{t_1})$ to generate $\langle\langle c_k^{(1)}, \ldots, c_k^{(\ell)} \rangle\rangle_{t_1}$.
   (b) If **Mult** fails to produce $\langle\langle c_k^{(1)}, \ldots, c_k^{(\ell)} \rangle\rangle_{t_1}$, then it must have removed 2^f new corrupted players from $\mathcal{P}'$ (i.e $|\mathcal{P}_1| - |\mathcal{P}'| = n_1 - n' \geq 2^f$). Thus repeat this segment with $f = f + 1$. Else increment k.
3. Now let $\mathcal{P}_2 = \mathcal{P}'$, $n_2 = n'$ and $d = t'$. For every segment $k = 1, \ldots, \lceil \log(t) \rceil$, check whether $c_k^{(1)}, \ldots, c_k^{(\ell)}$ are d-$2D^{(+,\ell)}$-shared. If $c_k^{(1)}, \ldots, c_k^{(\ell)}$ are α-$2D^{(+,\ell)}$-shared with $\alpha > d$ then invoke **Reshare**$(\mathcal{P}_2, \alpha, d, \ell, \langle\langle a_k^{(1)}, \ldots, a_k^{(\ell)} \rangle\rangle_\alpha)$, **Reshare**$(\mathcal{P}_2, \alpha, d, \ell, \langle\langle b_k^{(1)}, \ldots, b_k^{(\ell)} \rangle\rangle_\alpha)$ and **Reshare**$(\mathcal{P}_2, \alpha, d, \ell, \langle\langle c_k^{(1)}, \ldots, c_k^{(\ell)} \rangle\rangle_\alpha)$ to generate $\langle\langle a_k^{(1)}, \ldots, a_k^{(\ell)} \rangle\rangle_d$, $\langle\langle b_k^{(1)}, \ldots, b_k^{(\ell)} \rangle\rangle_d$ and $\langle\langle c_k^{(1)}, \ldots, c_k^{(\ell)} \rangle\rangle_d$.
4. For every segment $k = 1, \ldots, \lceil \log(t) \rceil$, invoke **Convert**$2D^{(+,\ell)}$**to**$2D^+(\mathcal{P}_2, d, \ell, \langle\langle a_k^{(1)}, \ldots, a_k^{(\ell)} \rangle\rangle_d)$, **Convert**$2D^{(+,\ell)}$**to**$2D^+(\mathcal{P}_2, d, \ell, \langle\langle b_k^{(1)}, \ldots, b_k^{(\ell)} \rangle\rangle_d)$ and **Convert**$2D^{(+,\ell)}$**to**$2D^+(\mathcal{P}_2, d, \ell, \langle\langle c_k^{(1)}, \ldots, c_k^{(\ell)} \rangle\rangle_d)$ to obtain $\langle\langle a_k^{(1)} \rangle\rangle_d, \ldots, \langle\langle a_k^{(\ell)} \rangle\rangle_d$, $\langle\langle b_k^{(1)} \rangle\rangle_d, \ldots, \langle\langle b_k^{(\ell)} \rangle\rangle_d$ and $\langle\langle c_k^{(1)} \rangle\rangle_d, \ldots, \langle\langle c_k^{(\ell)} \rangle\rangle_d$.

2.9 Input Phase

Once **Preparation Phase** is complete, the execution of **Input Phase** begins. The goal of the input phase is to generate d-$2D^+$-sharing of c_I inputs. Assume that player $P_i \in \mathcal{P}$ has c_i inputs. Thus $c_I = \sum_{i=1}^{n} c_i$. We stress that though some players from $\mathcal{P}$ might have failed and removed during preparation phase, we still allow them to feed their input. Recall that at the end of preparation phase, $\mathcal{P}_2 = \mathcal{P}'$. So once all the players in $\mathcal{P}$ feed their input, the rest of the computation will be performed among the players in $\mathcal{P}_2$. For this, each input sharing is then reshared among the players in $\mathcal{P}_2$. Also if some player $\mathcal{P}_i$ fails to correctly share their input, then everybody accepts a default $d-2D^{(+,c_i)}$ sharing on behalf of that player.

Lemma 12. *With overwhelming probability, protocol* **Input Phase** *produces correct d-$2D^+$-sharing of c_I inputs.* **Input Phase** *takes twenty eight rounds and communicates $\mathcal{O}(c_I n^3 + n^5)\kappa$ bits and broadcasts $\mathcal{O}(c_I n^3 + n^5)\kappa$ bits.*

Input Phase

1. Every player $P_i \in \mathcal{P}$ with c_i inputs $s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(c_i)}$, invokes $2D^{(+,c_i)}$**Share**$(P_i, \mathcal{P}, t, c_i, s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(c_i)})$ to generate $\langle\langle s_i^{(1)}, \ldots, s_i^{(c_i)} \rangle\rangle_t$.
2. For every P_i, invoke **Reshare**$(\mathcal{P}_2, t, d, c_i, \langle\langle s_i^{(1)}, \ldots, s_i^{(c_i)} \rangle\rangle_t)$ to generate $\langle\langle s_i^{(1)}, \ldots, s_i^{(c_i)} \rangle\rangle_d$ provided $d < t$. For every player P_i, invoke **Convert**$2D^{(+,c_i)}$**to**$2D^+(\mathcal{P}_2, d, c_i, \langle\langle s_i^{(1)}, \ldots, s_i^{(c_i)} \rangle\rangle_d)$ to obtain $\langle\langle s_i^{(1)} \rangle\rangle_d, \ldots, \langle\langle s_i^{(c_i)} \rangle\rangle_d$.

2.10 Computation Phase

Once **Preparation Phase** and **Input Phase** are over, the computation of the circuit (of the agreed upon function f) proceeds gate-by-gate. First, to every

random and every multiplication gate, a prepared d-$2D^+$-shared random triple is assigned. And a d-$2D^+$-shared input is assigned to the corresponding input gates. A gate (except output gate) g is said to be *computed* if a d-$2D^+$-sharing $\langle\langle x_g \rangle\rangle_d$ is computed for the gate. Note that all the random and input gates will be *computed* as soon as we assign d-$2D^+$-shared random triples (generated in **Preparation Phase**) and d-$2D^+$-shared inputs (generated in **Input Phase**) to them respectively. A gate is said to be in *ready state*, when all its fanin gates have been *computed*. In the **Computation Phase**, the circuit evaluation proceeds in rounds wherein each round all the ready gates will be computed parallely. Evaluation of input, random and addition gates do not require any communication. Evaluation of multiplication and output gate requires 2 and 1 call to Protocol $2D^+$**Recons** respectively. So the individual gates in the circuit are evaluated as shown in the above table. The correctness of the steps described for multiplication gate follows from [1] which introduced the technique called *Circuit Randomization*.

Lemma 13. *With overwhelming probability, protocol* **Computation Phase** *evaluates the circuit gate-by-gate in a shared fashion and outputs the desired outputs.* **Computation Phase** *takes $\mathcal{D}$ rounds and communicates $\mathcal{O}((c_M + c_O)n^3\kappa)$ bits.*

Computation Phase
Input Gate: $\langle\langle s \rangle\rangle_d = \text{IGate}(\langle\langle s \rangle\rangle_d)$
Assign a d-$2D^+$-sharing of an input, say $\langle\langle s \rangle\rangle_d$.
Random Gate: $\langle\langle a \rangle\rangle_d = \text{RGate}(\langle\langle a \rangle\rangle_d, \langle\langle b \rangle\rangle_d, \langle\langle c \rangle\rangle_d)$
Assign a d-$2D^+$-sharing of a multiplication triple, say $(\langle\langle a \rangle\rangle_d, \langle\langle b \rangle\rangle_d, \langle\langle c \rangle\rangle_d)$, where only the first component is used.
Addition Gate: $\langle\langle z \rangle\rangle_d = \text{AGate}(\langle\langle x \rangle\rangle_d, \langle\langle y \rangle\rangle_d)$
If $\langle\langle x \rangle\rangle_d$ and $\langle\langle y \rangle\rangle_d$ are the inputs to the addition gate, all players in $\mathcal{P}_2$ compute $\langle\langle z \rangle\rangle_d = \langle\langle x \rangle\rangle_d + \langle\langle y \rangle\rangle_d$ with $\langle\langle z \rangle\rangle_d$ as the output of the gate.
Multiplication Gate: $\langle\langle z \rangle\rangle_d = \text{MGate}(\langle\langle x \rangle\rangle_d, \langle\langle y \rangle\rangle_d, (\langle\langle a \rangle\rangle_d, \langle\langle b \rangle\rangle_d, \langle\langle c \rangle\rangle_d))$
1. Let $\langle\langle x \rangle\rangle_d$ and $\langle\langle y \rangle\rangle_d$ are the inputs to the multiplication gate and $(\langle\langle a \rangle\rangle_d, \langle\langle b \rangle\rangle_d, \langle\langle c \rangle\rangle_d)$ is the random multiplication triple assigned to it. Then all players in $\mathcal{P}_2$ compute the output $\langle\langle z \rangle\rangle_d$ in the following way.
2. All players in $\mathcal{P}_2$ compute $\langle\langle \alpha \rangle\rangle_d = \langle\langle x \rangle\rangle_d - \langle\langle a \rangle\rangle_d$ and $\langle\langle \beta \rangle\rangle_d = \langle\langle y \rangle\rangle_d - \langle\langle b \rangle\rangle_d$.
3. All players in $\mathcal{P}_2$ invoke $2D^+$**Recons**$(\mathcal{P}_2, d, \langle\langle \alpha \rangle\rangle_d)$ and $2D^+$**Recons**$(\mathcal{P}_2, d, \langle\langle \beta \rangle\rangle_d)$ to reconstruct α and β.
4. All players in $\mathcal{P}_2$ compute $\langle\langle z \rangle\rangle_d = \alpha\beta + \alpha\langle\langle b \rangle\rangle_d + \beta\langle\langle a \rangle\rangle_d + \langle\langle c \rangle\rangle_d$.
Output Gate: $x = \text{OGate}(\langle\langle x \rangle\rangle_d)$
If $\langle\langle x \rangle\rangle_d$ is the input to the output gate, all players in $\mathcal{P}_2$ compute $x = 2D^+$**Recons**$(\mathcal{P}_2, d, \langle\langle x \rangle\rangle_d)$.

Now our new UMPC protocol for evaluating function f is: (1). Invoke **Preparation Phase** (2). Invoke **Input Phase** (3). Invoke **Computation Phase**.

Theorem 1. *With overwhelming probability, our new UMPC protocol can evaluate an agreed upon function securely against an active adaptive rushing adversary $\mathcal{A}_t$ with $t < n/2$ and requires $\mathcal{O}(\log(t) + \mathcal{D})$ rounds and communicates $\mathcal{O}(((c_I + c_R + c_M + c_O)n^3)\kappa)$ bits and broadcasts $\mathcal{O}((c_I + c_M + c_R)n^3 + n^5)\kappa$ bits.*

Using the protocol of [12] to simulate the broadcasts, the communication complexity and round complexity of our UMPC protocol is as follows:

Theorem 2. *With overwhelming probability, our new UMPC protocol requires* $\mathcal{O}(n\log(t)+\mathcal{D})$ *rounds and communicates* $\mathcal{O}(((c_I+c_M+c_R+c_O)n^4+n^7)\kappa)$ *bits.*

References

1. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992)
2. Beaver, D.: Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. Journal of Cryptology 4(4), 75–122 (1991)
3. Beerliová-Trubíniová, Z., Hirt, M.: Efficient multi-party computation with dispute control. In: Proc. of TCC, pp. 305–328 (2006)
4. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-secure MPC with linear communication complexity. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 213–230. Springer, Heidelberg (2008)
5. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC, pp. 1–10 (1988)
6. Berman, P., Garay, J.A., Perry, K.J.: Bit optimal distributed consensus. Computer Science Research, 313–322 (1992)
7. Carter, L., Wegman, M.N.: Universal classes of hash functions. Journal of Computer and System Sciences (JCSS) 18(4), 143–154 (1979)
8. Chaum, D., Crpeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: Proc. of FOCS 1988, pp. 11–19 (1988)
9. Cramer, R., Damgård, I.: Multiparty Computation, an Introduction. Contemporary Cryptography. Birkháuser, Basel (2005)
10. Cramer, R., Damgård, I., Dziembowski, S., Hirt, M., Rabin, T.: Efficient multiparty computations secure against an adaptive adversary. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 311–326. Springer, Heidelberg (1999)
11. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 572–590. Springer, Heidelberg (2007)
12. Fitzi, M., Hirt, M.: Optimally Efficient Multi-Valued Byzantine Agreement. In: Proc. of PODC 2006, pp. 163–168. ACM, New York (2006)
13. Patra, A., Choudhary, A., Pandu Rangan, C.: Round Efficient Unconditionally Seecure Multiparty Computation. Cryptology ePrint Archive, Report 2008/399
14. Pfitzmann, B., Waidner, M.: Information-theoretic pseudosignatures and byzantine agreement for t $\geq$ n/3. Technical report, IBM Research (1996)
15. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: STOC, pp. 73–85 (1989)
16. Yao, A.C.: Protocols for secure computations. In: Proc. of 23rd IEEE FOCS, pp. 160–164 (1982)

A New Anonymous Password-Based Authenticated Key Exchange Protocol*

Jing Yang and Zhenfeng Zhang

State Key Laboratory of Information Security,
Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
National Engineering Research Center of Information Security, Beijing 100190, China
{yangjing,zfzhang}@is.iscas.ac.cn

Abstract. In Indocrypt 2005 Viet et al. first proposed an anonymous password-based key exchange protocol: APAKE and its extension: k-out-of-n APAKE. Then Shin et al. presented an improved protocol TAP. In this paper, we first show that the TAP protocol is vulnerable to two attacks. One is an impersonating attack and the other is an off-line dictionary attack, which is also applied to k-out-of-n APAKE. Furthermore, we propose a novel anonymous password-based key exchange protocol, and prove its security in the random oracle model under the square computational Diffie-Hellman assumption and decision inverted-additive Diffie-Hellman assumption. We also extend our protocol to the distributed setting, which is secure against the proposed attacks.

Keywords: Password-based AKE, Anonymous authentication.

1 Introduction

Password-based authenticated key exchange (PAKE) has received growing attention in recent years. The two communication entities in PAKE can establish a fresh authenticated session key by only using a pre-shared human-memorable password, without the heavy-weight PKI. However PAKE also suffers from so-called exhaustive guessing or dictionary attacks due to the small space of the password. By now many password-based key establishment schemes have been proposed, including EKE [7], AuthA [5,6], SPEKE [9], SRP [7], password-based TLS [3] et al. Besides such two-party protocols, some studies focus on the group setting. A password-based group key establishment protocol enable a group of users to authenticate each other and establish a fresh session key for further secure communication. Each user either shares a common password with all others, or just shares a password with a trusted server. For the latter case, the establishment of a session key needs server's help, who may know the key or not.

Anonymity is an increasing hot issue nowadays. An anonymous authentication scheme [12] is a protocol that allows a member called a prover of a group to

* The work is supported by National Natural Science Foundation of China (90604018, 60873261), National Basic Research Program (973) of China (2007CB311202), National High-Tech R&D Program (863) of China (2006AA01Z454), and National Key Technologies R&D Program of China (2006BAH02A02).

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 200–212, 2008.

convince a verifier that she is a member of the group without revealing any information about her. Anonymity in the public key cryptosystem has been discussed for a long time. The group signature , ring signature and anonymous credential system are obvious solutions to it. However due to the low entropy of a password, anonymous authentication in a password-based scheme is a challenging thing. By now there have been only a few research results on this issue.

Viet et al. [12] first proposed, as we know, an anonymous PAKE protocol, called APAKE, which can make a user establish a session key with a server anonymously with only sharing a short password. And they extended it to the k-out-of-n APAKE protocol between a group of members and a server. They combined a password-based AKE protocol [2] for generating secure channels with an Oblivious Transfer protocol [11] for client's anonymity. Later Chai et al.[8] proposed a similar smart card-based PAKE protocol preserving user's privacy. And recently Shin et al. [10] pointed out that Viet et al.'s k-out-of-n APAKE protocol is insecure against an off-line dictionary attack, and proposed a so-called threshold anonymous PAKE (TAP) protocol. The TAP protocol is claimed to provide semantic security of session keys, unilateral authentication of server S and clients' anonymity against a passive server. With the threshold $t = 1$, the TAP protocol is more efficient than Viet et al.'s APAKE protocol. Both the k-out-of-n APAKE protocol and the TAP protocol works in distributed settings, in order to enable a group of member to establish a session key with a server anonymously. Such kind of protocols is called as distributed anonymous PAKE in this paper.

1.1 Our Contribution

Our contribution in this paper is twofold. Firstly, we point out the vulnerabilities of both Viet et al. and Shin et al. distributed anonymous PAKE protocols. Shin et al.'s TAP ($t \geq 2$) protocol is insecure against two attacks. One is an impersonating attack breaking the unilateral authentication of the scheme, and the other one is an off-line dictionary attack, which can make a malicious client get all other clients' passwords. And the latter is also applied to Viet et al.'s k-out-of-n APAKE protocol. Secondly, we propose a new anonymous password-based authenticated key exchange protocol, named NAPAKE. In the random oracle model, we prove its security under the square computational Diffie-Hellman assumption and decision inverted-additive Diffie-Hellman assumption. Furthermore, we give an extension of our protocol to the distributed setting, which is secure against the proposed two attacks.

1.2 Organization

The paper is organized as following: Section 2 presents two attacks against the afore-mentioned anonymous PAKE protocols. In section 3, we introduce the formal model and security assumptions for the two-party anonymous PAKE. Our new protocol NAPAKE is shown in section 4, along with its efficiency and security analysis. And the extension of NAPAKE to the distributed setting is given also in the section 4. Finally section 5 concludes the whole paper.

2 Security Flaws of Two Distributed Anonymous PAKE Protocols

In this section, we present two attacks against Shin et al.'s TAP ($t \geq 2$) [10] protocol. One is an impersonating attack breaking the unilateral authentication of the scheme, and the other one is an insider's off-line dictionary attack. Moreover the latter is also applied to Viet et al.'s k-out-of-n APAKE [12] protocol.

Note that, both the k-out-of-n APAKE and TAP protocol have been claimed to be secure for the AKE security and unilateral authentication. In fact, the security model they adopted is for the two-party setting, and the subgroup is treated as a whole entity in their model. Therefore, the insiders' attack has not been covered sufficiently. Although a kind of insider adversary was also considered in the k-out-of-n APAKE, it is used only for the user's anonymity.

The attacks show that insider's attack is practical and seems easier to mount in distributed (multiparty) anonymous PAKE protocols than two-party settings.

2.1 Brief Review of the TAP Protocol

Shin et al.'s threshold anonymous PAKE protocol is illustrated in Fig.1. The goal of the protocol is to make the t clients in SG establish a session key with S anonymously. It works in the group $\mathbb{G}$ with the prime order q and two generators g and h. Let $\mathcal{F} : \{0,1\}^* \rightarrow \mathbb{Z}_q$ and $\mathcal{G} : \{0,1\}^* \rightarrow \mathbb{G}$ be two full-domain hash functions, and $\mathcal{H}_1$ and $\mathcal{H}_2$ be secure cryptographic hash functions from $\{0,1\}^*$ to $\{0,1\}^l$, where l is a security parameter.

2.2 The Impersonating Attack against TAP

Suppose SG initiates the TAP protocol. Let AG be another subgroup of Γ, which consists of t' malicious clients not included in SG. AG wants to impersonate S to establish a session key with SG. The detail is shown as follows.

1′. AG intercepts the message $\{\Gamma, t, \{X_i{}^*\}_{1\leq i\leq n}\}$. Each client C_k in AG, chooses a random numbers e_k from $\mathbb{Z}_q$, computes $E_k = g^{e_k}$, $F_k = h^{\mathcal{F}(k,pw_k)}$ and $E_k{}^* = E_k \cdot F_k$, and constructs a new set $\{X_i^{*'}\}_{1\leq i\leq n}$, in which $X_i{}^{*'} = E_k^*$ if C_i is in AG, or otherwise $X_i^{*'} = X_i^*$. After that, AG initiates another run of the TAP protocol with S by sending $(\Gamma, t', \{X_i^{*'}\}_{1\leq i\leq n})$.

2′. Upon receipt of the message $(\Gamma, t', \{X_i^{*'}\}_{1\leq i\leq n})$, the server S executes the protocol as normal.

3′.
- Each C_k in AG, looks for Z_k and extracts S_k, where $\mathsf{S}_k = Z_k \oplus \mathcal{G}(k, K_k)$ and $K_k = Y^{e_k}$.
- With all S_k, AG reconstructs the polynomial $f(x) = \sum_{l=0}^{t'-1} u_l x^l$ by Lagrange interpolation, and gets $\mathsf{S}_j = f(j)$, $\mathcal{G}(j, K_j) = \mathsf{S}_j \oplus Z_j$ for all n clients.
- AG chooses a new shared secret $\tilde{\mathsf{S}} \in_R \mathbb{G}$ and new coefficients $\{\tilde{u}_l\}_{1\leq l\leq t-1}$ to construct a new polynomial $\tilde{f}(x) = \sum_{l=0}^{t-1} \tilde{u}_l x^l$ with $u_0 = \tilde{\mathsf{S}}$.

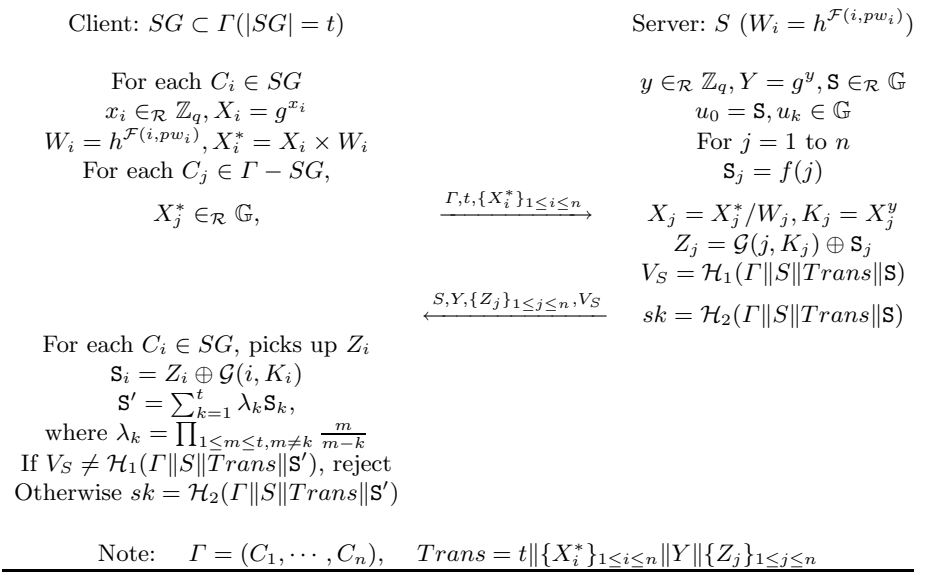


Fig. 1. Shin et al.'s threshold anonymous PAKE (TAP) protocol

- AG computes n shares $\tilde{\mathsf{S}_j} = \tilde{f}(j)$, and generates new $\{\tilde{Z}_j\}_{1 \le j \le n}$, where $\tilde{Z}_j = \mathcal{G}(j, K_j) \oplus \tilde{\mathsf{S}}_j$. Then AG computes $\tilde{V}_S = \mathcal{H}_1(\Gamma \| S \| Trans \| \tilde{\mathsf{S}})$ and $\tilde{sk} = \mathcal{H}_2(\Gamma \| S \| Trans \| \tilde{\mathsf{S}})$ where $Trans = t \| \{X_i{}^*\}_{1 \le i \le n} \| Y \| \{\tilde{Z}_j\}_{1 \le j \le n}$. Finally AG sends the message $(S, Y, \{\tilde{Z}_j\}_{1 \le j \le n}, \tilde{V}_S)$ to SG.

3. – Each client C_i in SG, looks for $\tilde{Z}_i$ and extracts $\tilde{\mathsf{S}}_i$ from $\tilde{Z}_i$, where $\tilde{\mathsf{S}}_i = \tilde{Z}_i \oplus \mathcal{G}(i, K_i)$ and $K_i = Y^{x_i}$.
   – By collaborating with one another, SG reconstructs $\tilde{\mathsf{S}'}$ from $\{\tilde{\mathsf{S}_i}\}_{1 \le i \le t}$ by Lagrange interpolation. To check $\tilde{V}_S$, SG computes $\tilde{V}_S' = \mathcal{H}_1(\Gamma || S || Trans || \tilde{\mathsf{S}'})$ which obviously equals to $\tilde{V}_S$. After that, SG generates its session key $\tilde{sk} = \mathcal{H}_2(\Gamma || S || Trans || \tilde{\mathsf{S}'})$. In the end SG believes it establishes a session key with the server S, but it is actually established with AG.

The key to the attack is that anyone who can get the shared secret S chosen by S can get all clients' hiding codes $\{\mathcal{G}(j, K_j)\}_{1 \le j \le n}$ via the corresponding S_i. With those hiding codes, the adversaries is able to generate the correct message for SG. Actually a similar attack can be mounted by malicious clients in SG to cheat other ones in SG.

2.3 The Off-Line Dictionary Attack against TAP and k-Out-of-n APAKE

The off-line dictionary attack shown in this section can be mounted to both TAP ($t \ge 2$) and k-out-of-n APAKE. We only give the detail of the implementation for TAP, and the other one is similar.

To get the shared secret chosen by the server S, the clients in SG have to collect all secret shares and compute S. Suppose C_m is the one who does the very thing. All other clients send their own S_i to C_m, and he computes S from $\{\mathsf{S}_i\}_{1\le i\le n}$, and sends it to all other clients.

However if C_m wants to know other clients' password, he can execute the following off-line dictionary attack, by which he could get all other clients' passwords without being detected. The attack flows are shown below.

1. The subgroup $SG = \{C_1, \ldots, C_m, \ldots, C_t\}$ initiates the protocol as usual, and C_m records $\{X_i^*\}_{1\le i\le n}$.

2′.
- C_m impersonates the server to send the second message. He chooses $y \in_R \mathbb{Z}_q$ to compute $Y = g^y$. Also he chooses $n+1$ random values: S and $\{Z_j\}_{1\le j\le n}$ from $\mathbb{G}$.
- C_m generates an authenticator $V_S = \mathcal{H}_1(\Gamma||S||Trans||\mathsf{S})$ and a session key $sk = \mathcal{H}_2(\Gamma||S||Trans||\mathsf{S})$, where $Trans = t\|\{X_i^*\}_{1\le i\le n}\|Y\|\{Z_j\}_{1\le j\le n}$. Then he sends $(S, Y, \{Z_j\}_{1\le j\le n}, V_S)$ to SG.

3.
- Each client C_i in SG, except C_m, looks for Z_i and computes $\mathsf{S}_i = Z_i \oplus \mathcal{G}(i, K_i)$ with $K_i = Y^{x_i}$. Then they send their own S_i to C_m. (No matter how S_i is sent, C_m must be able to recover it.)
- C_m sends S to all other clients, and they can verify its correctness by checking whether $V_S' = \mathcal{H}_1(\Gamma||S||Trans||\mathsf{S})$ is equal to V_S. Obviously they are same, so all other clients accept S and believe it indeed comes from the server.
- After collecting all S_i, C_m guesses a password pw_i' for any client C_i in SG, and get the corresponding $\mathcal{G}(i, K_i')$, where $K_i' = (X_i^*/h^{\mathcal{F}(i,pw_i')})^y$. Then C_m computes $Z_i' = \mathsf{S}_i \oplus \mathcal{G}(i, K_i')$. By checking whether Z_i' equals to Z_i, C_m can find out the correct password pw_i' for the client C_i. Similarly, C_m is able to get all other clients' passwords.

If SG doesn't specify a particular client to recover S, it may alternatively let all clients broadcast their secret shares. Then any adversary, no matter whether in the subgroup or not, can mount the above attack. And the result is the clients of SG would find that the verification is not satisfied and they terminate the run of the protocol. Yet with their broadcasted secret shares, the adversary has already gotten all clients' passwords.

TAP didn't give the specification of how to recover S for the subgroup. But no matter how S_i being sent, the protocol can hardly resist the off-line dictionary attack. In fact, members in SG have to recover S from $\{\mathsf{S}_i\}$, so the only thing unknown to an insider adversary after getting S_i is the password, if he can control the message from the server to SG. The key to the attack is that the members in SG cannot verify the second message whether from the server or not, namely they cannot distinguish Z_i from a random value. Moreover using the hiding code to protect secret shares makes the off-line dictionary attack feasible.

3 The Model and Security Notions

By now, anonymous PAKE protocols can be classified into two types: the two-party setting and the distributed setting. Defining the security model for the distributed anonymous PAKE will be our next work in the future. In this paper, we focus on the security specification in the two-party setting, and adopt the model introduced in [5] to describe an adversary's capability and define the security targets, which provides a neat treatment to dictionary attacks. Due to the anonymity property, we modify the definition of user's authentication and introduce a new definition of user's anonymity, which is different from that in [12].

3.1 Formal Model

Participants. There is a fixed set of n client users $\Gamma = \{U_1, \ldots, U_n\}$. S is a trusted server. Each user U_i in Γ shares a low-entropy secret pw_i with S, which is drawn from a small dictionary Password, according to a distribution $\mathcal{D}$. S has a list of passwords as $PW_S = \{pw_i\}_{1 \leq i \leq n}$.

The participants in the two-party anonymous PAKE setting are a single user U and the server S, with U belonging to Γ. The U and S may have several instances called oracles involved in distinct, possibly concurrent, executions of P. Generally, we denote the instance ρ (resp., δ) of participant U (resp., S) by $\prod_U^\rho$ (resp., $\prod_S^\delta$).

Adversarial Model. An adversary $\mathcal{A}$ is a probabilistic algorithm with a distinguished query tap. $\mathcal{A}$ can take the entire control of the network during the protocol execution. The capability of $\mathcal{A}$ is modelled by the following oracles:

- Execute(U, ρ, S, δ): This query models passive attacks, where the adversary gets access to honest executions of P between the instances $\prod_U^\rho$ and $\prod_S^\delta$ by eavesdropping.
- Send(I, m): This query enables to consider active attacks by having $\mathcal{A}$ sending a message to instance I (U, or S). The adversary $\mathcal{A}$ gets back the response I generates in processing the message m according to the protocol P.
- Reveal(I): This query models the misuse of the session key by instance I. The query is only available to $\mathcal{A}$ if the attacked instance actually "holds" a session key and it releases the latter to $\mathcal{A}$.

3.2 Security Notions

AKE Security. An adversary $\mathcal{A}$ is allowed to call oracles Execute and Send during an execution of an anonymous PAKE protocol P. Eventually, $\mathcal{A}$ calls Test(I)-query only one time, for some instance I (U, or S), which is defined like below:

-Test(I): This query tries to capture the adversary's ability to distinguish real keys from random ones. The Test oracle tosses a coin and obtains a bit $b \in \{0, 1\}$. If $b = 0$, then Test gives a random bit sequence, and if $b = 1$, then Test gives a session key (the output of Reveal(I)).

The $\mathsf{Test}(I)$-query is only available to $\mathcal{A}$ if attacked instance I is Fresh (which roughly means that the session key that I hold is not "obviously" known to the adversary). Receiving the output of Test, the adversary $\mathcal{A}$ outputs a bit b', which represents $\mathcal{A}$'s guess of b. An AKE advantage is

$$Adv_{P,\mathcal{D}}^{ake}(\mathcal{A}) = 2\Pr[b = b'] - 1$$

where the probability is taken over all the random coins of the adversary and all the oracles and the passwords are picked from a dictionary $\mathcal{D}$. The protocol P is said to be (t, ϵ)-AKE-secure if $\mathcal{A}$'s advantage is smaller than ϵ for any adversary $\mathcal{A}$ running in time t.

Authentication. The server's authentication is defined by the probability that $\mathcal{A}$ successfully impersonates a sever instance in an execution of P by $Succ_P^{S-auth}(\mathcal{A})$. Impersonation succeeds when a user accepts a session key which is shared with no instance of the server, that is,

$$Succ_{P,\mathcal{D}}^{S-auth}(\mathcal{A}) = \Pr\big[U \text{ accept a key with no instance of } S\big]$$

The definition of user's authentication is a little unusual. In an anonymous PAKE protocol, the server only knows that the user is in a group Γ without identifying the actual identity. So it is defined by the probability that $\mathcal{A}$ successfully impersonates an instance of a user in Γ in an execution of P by $Succ_P^{U-auth}(\mathcal{A})$. Impersonation succeeds when the server accepts a session key which is shared with no instance of any user in Γ, that is,

$$Succ_{P,\mathcal{D}}^{U-auth}(\mathcal{A}) = \Pr\big[S \text{ accept a key with no instance of } U \in \Gamma\big]$$

A protocol P is said to be (t, ϵ)-S-auth-secure (resp. (t, ϵ)-U-auth-secure) if $\mathcal{A}$'s success probability for breaking S-auth (resp. U-auth) is smaller than ϵ for any adversary $\mathcal{A}$ running in time t.

Anonymity. In terms of the user's anonymity, the server is considered as an adversary. Furthermore, it is restricted that Γ only has two users U_0 and U_1. In each execution of protocol P, a user instance U_b is randomly chosen with $b \in_R \{0, 1\}$. At the end of the execution S outputs $b' \in \{0, 1\}$, which means the server's guess on the user's identity. S's success in breaking the user's anonymity is defined by the happening of the event $\mathsf{S\_Guess\_Auth}$, that is S authenticates itself to the user and guesses the correct b. S's advantage on breaking the user's anonymity is defined as:

$$Adv_{P,\mathcal{D}}^{U\text{-}anoy}(S) = 2\Pr[\mathsf{S\_Guess\_Auth}] - 1$$

The protocol P is said to be (t, ϵ)-U-anoy-secure if S's advantage on breaking the user's anonymity is smaller than ϵ in time t.

3.3 Cryptographic Assumption

The security of NAPAKE is based on the Square Computational Diffie-Hellman assumption (SCDH) and the Decisional Inverted-Additive Diffie-Hellman

assumption (DIADH). SCDH is a variation of the standard computational Diffie-Hellman assumption (CDH), and is proven to be equivalent to the CDH assumption [4]. The DIADH assumption was introduced by Mackenzie in [9]. He proves a lower bound for solving the problem in the generic model, which asymptotically matches the lower bound for solving DDH (Decision Diffie-Hellman) in the generic model. In other words, the two assumptions are equivalent.

Square Computational Diffie-Hellman assumption (SCDH). The SCDH assumption in a represented group $\mathbb{G} = \langle g \rangle$ is that given g^x, with x randomly chosen in $\mathbb{Z}_q$, it is hard to compute g^{x^2}.

Decisional Inverted-Additive Diffie-Hellman assumption (DIADH). The DIDH assumption states that the distributions $(g^x, g^y, g^{xy/(x+y)})$ and (g^x, g^y, g^r) in a represented group $\mathbb{G} = \langle g \rangle$ are computationally indistinguishable when x, y, r are drawn at random from $\mathbb{Z}_q$ and $x + y \neq 0$.

4 The New Anonymous PAKE Protocol

In this section, we proposed a new anonymous password-based authenticated key exchange protocol, NAPAKE, as mentioned previously. The AKE security and server's authentication are proved under the SCDH and DIADH assumptions, and the user's authentication and anonymity are analyze additionally. The efficiency comparison between TAP ($t = 1$) and NAPAKE is given subsequently. Finally, a new distributed anonymous PAKE protocol based on NAPAKE is presented, which is secure against the two above attacks.

4.1 Protocol Description

Let $\mathbb{G} = \langle g \rangle$ be a finite, cyclic group of prime order q. Assume $\mathcal{G} : \{0,1\}^* \to \mathbb{G}$ is a full-domain hash function, and $\mathcal{H}_0$, $\mathcal{H}_1 : \{0,1\}^* \to \{0,1\}^l$ are two random hash functions, with l denoted as a security parameter. Let pw_i be a password shared between the client C_i and the server S, and $PW_i = \mathcal{G}(i, pw_i)$. The high-level description is shown as follows, and it is illustrated in Fig.2. We assume the two entities have already agreed on the client group Γ before a protocol run.

1. The server S chooses $r_S \in_R \mathbb{Z}_q$, and for all n clients in Γ generates $A_j = PW_j^{r_S}$ with $1 \leq j \leq n$. Then S sends $(S, \{A_j\}_{1 \leq j \leq n})$ to client C_i.
2. C_i checks all the values in $\{A_j\}$ are different from each other. If not, C_i aborts the protocol. Otherwise, C_i picks A_i from $\{A_j\}$, and draws two random values r_c and x from $\mathbb{Z}_q$. Then C_i computes $X = g^x$, $Z = A_i^{r_c}$, and generates $X^* = Z \cdot X$ and $B = PW_i^{r_c}$. After that, C_i sends (X^*, B) to S.
3. S computes $Z' = B^{r_S}$ with the random value r_S and recovers $X' = X^*/Z'$. Then he chooses y randomly from $\mathbb{Z}_q$, and computes $Y = g^y$ and $K' = X'^y$. And he generates the authenticator $Auth_S = \mathcal{H}_1(Trans||Z'||K')$ and the session key $sk = \mathcal{H}_0(Trans||Z'||K')$, where $Trans = \Gamma||S||\{A_j\}||X^*||B||Y$. Finally he sends $(Y, Auth_S)$ to C_i.

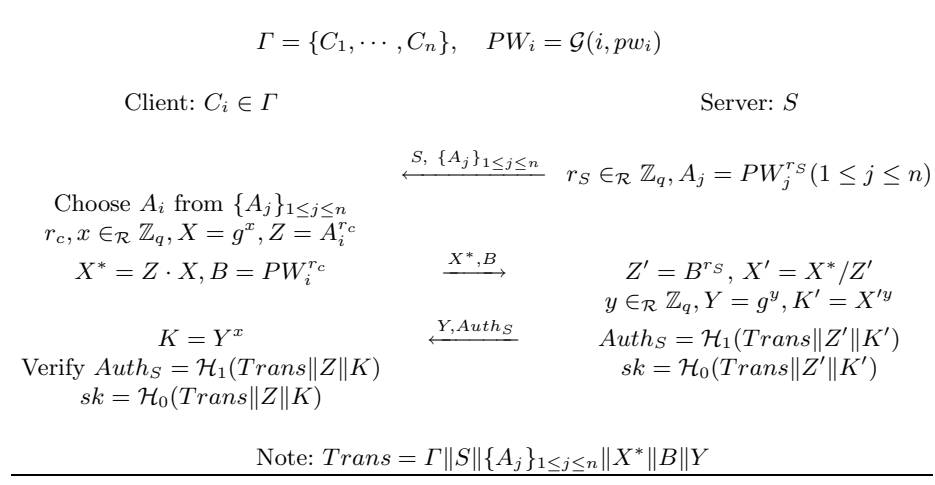


Fig. 2. The NAPAKE protocol

4. C_i computes the Diffie-Hellman value $K = Y^x$, and then check that whether $Auth_S$ equals to $\mathcal{H}_1(Trans||Z||K)$. If not, U aborts the protocol. Otherwise, he computes the session key $sk = \mathcal{H}_0(Trans||Z||K)$ and accepts it.

4.2 Security

The NAPAKE can achieve semantic security of the session key and explicit server's authentication under the DIADH and SCDH assumptions, as shown in Theorem 1. It also indicates that NAPAKE is secure against dictionary attacks since the advantage of the adversary essentially grows with the ratio of interactions (number of Send-queries) to the number of passwords. The user's authentication is preserved in an implicit way, because Theorem 1 guarantees that no adversary can impersonate a user in Γ to establish a session key with the server. Theorem 2 states that the user is anonymous against the server in NAPAKE .

Theorem 1 (AKE/S-auth Security). *Consider the protocol NAPAKE, running on a group of a prime order q and a dictionary Password equipped with the distribution $\mathcal{D}$. For any adversary $\mathcal{A}$ with a time bound t, with less than q_s Send-queries, q_e Execute-queries, and q_g and q_h hash queries to $\mathcal{G}, \mathcal{H}_0, \mathcal{H}_1$, respectively, we have*

$$Adv^{ake}_{\mathrm{NAPAKE}}(\mathcal{A}) \le 4\epsilon, \quad Succ^{S-auth}_{\mathrm{NAPAKE}}(\mathcal{A}) \le \epsilon,$$

where

$$\epsilon = \frac{q_s}{2^l} + \frac{q_h^2}{2} \times Succ^{\mathrm{DIADH}}_{g,\mathbb{G}}(t + 2\tau_e) + q_h \times Succ^{\mathrm{SCDH}}_{g,\mathbb{G}}(t + 2\tau_e) + 3\mathcal{D}(q_s) + \frac{3T^2}{2q},$$

and $T = q_s + q_e + q_g$, τ_e is the computational time for an exponentiation in $\mathbb{G}$.

Proof. (Sketch) For the AKE security and the server's authentication, the index of U is not essential, so we assume the client U is C_1 in Γ without loss of generality. Let (I_1, I_2, I_3) be an instance of DIADH, where $I_1 = g^{w_1}$, $I_2 = g^{w_2}$, $I_3 = g^{w_3}$, and w_1, w_2, w_3 are drawn randomly from $\mathbb{Z}_q$. Note that the SCDH problem is intractable in $\mathbb{G}$. We give a sequence of games, starting from the real game G0 and ending with G5, to prove the security of NAPAKE.

In G1, we simulate as usual all the hash oracles $\mathcal{G}$, $\mathcal{H}_0$, $\mathcal{H}_1$ by maintaining hash lists $\Lambda_{\mathcal{G}}$, $\Lambda_{\mathcal{H}}$, and the Send, Execute, Reveal and Test oracles. Also two private hash oracles $\mathcal{H}'_0$, $\mathcal{H}'_1$ are imported and simulated. In G2, we exclude the collision in the output of the $\mathcal{G}$ oracle or in the partial transcript $(\{X^*, B\}, \{Y\})$. In G3, sk and $Auth_S$ are computed with the private oracles $\mathcal{H}'_0$ and $\mathcal{H}'_1$ respectively. It is safe to do that because the collisions of partial transcripts $\{X^*, B\}, \{Y\})$ have been excluded. Furthermore we change all values about PW_i with random values. In such game the adversary has no advantage on the Test-query. G3 is indistinguishable from G2, only if the event AskH = AskH1$\vee$ AskH0 does not happen.

AskH1: $\mathcal{A}$ queries $\mathcal{H}_1(Trans||Z'||K')$ or $\mathcal{H}_1(Trans||Z||K)$ for the transcript $(\{S, \{A_j\}\}, \{X^*, B\}, \{Y\})$, with $Z = \mathrm{DH}_{PW}(A_1, B)$ and $K = \mathrm{DH}_g((X^*/Z), Y)$. AskH0: $\mathcal{A}$ queries $\mathcal{H}_0(Trans||Z'||K')$ or $\mathcal{H}_0(Trans||Z||K)$ for the transcript $(\{S, \{A_j\}\}, \{X^*, B\}, \{Y\})$, where some party has accepted, but event AskH1 did not happen. Furthermore the event AskH1 can be split in 3 disjoint subclass: AskH1-passive, the event that all data in a transcript come from an execution between instance of U and S (which means that A_1, X^*, B, and Y have been simulated). AskH1-withU, the event that the execution involved only an instance of U. AskH1-withS, the event that the execution involved only an instance of S.

In G4, the instance (I_1, I_2, I_3) is used to modify the simulation of some oracles as follows. Rule S_1: Choose r_S randomly from $\mathbb{Z}_q$, and compute $A_1 = I_3^{r_S}$. Rule U_1: Choose r_c randomly from $\mathbb{Z}_q$, and compute $B = I_3^{r_c}$. Rule S_2: Choose y randomly from $\mathbb{Z}_q$, and compute $Y = I_1^y$. Rule $\mathcal{G}$: Choose k randomly from $\mathbb{Z}_q$, and flip a coin d, with $d \in \{1, 2\}$. Compute $r = I_d^k$, output k and d. The record (q, k, d, r) is added to $\Lambda_{\mathcal{G}}$.

InG5, we first use Lemma 1 to make sure that there is only one pair of (Z, K) with one password corresponding to a partial transcript in $\Lambda_{\mathcal{H}}$. With Lemma 1, events AskH1-withU and AskH1-withS happen only if the adversary guesses the correct password, which means the success probability of active attacks is upper-bounded by the password guessing possibility. Lemma 2 shows that the probability of AskH1-passive is negligible, which means that no passive adversary can break the session key's semantic security with non-negligible probability. Then it is easy to see that the desired result follows from Lemmas 1 and 2. □

Lemma 1. *For some partial transcript $(\{S, \{A_j\}\}, \{X^*, B\}, \{Y\})$, if there are two elements PW_0 and PW_1 such that $(\Gamma, S, \{A_j\}, X^*, B, Y, Z_b, K_b)$ are in $\Lambda_{\mathcal{H}}$ with $Z_b = DH_{PW_b}(A_1, B)$ and $K_b = DH_g((X^*/Z_b), Y)$, one can solve the DIADH problem with probability 1/2.*

Lemma 2. *For some passive partial transcript* $(\{S, \{A_j\}\}, \{X^*, B\}, \{Y\})$*, if there is an element PW such that* $(\Gamma, S, \{A_j\}, X^*, B, Y, Z_b, K_b)$ *are in* $\Lambda_{\mathcal{H}}$ *with* $Z = DH_{PW}(A_1, B))$ *and* $K = DH_g((X^*/Z), Y)$*, one can solve the SCDH problem with probability 1/2.*

Theorem 2. *The NAPAKE protocol is anonymous against the server.*

Proof. (Sketch)Consider the event S_Guess_Auth in the definition above, i.e., S authenticates itself to the user and guesses the correct b, which means that S sends a correct authenticator $Auth_S$ to the user U_b for a partial transcript $(\{S, A_0, A_1\}, \{X^*, B\}, \{Y\})$, where A_0, A_1, and Y are chosen by S, and $\{X^* = g^x \cdot A_b^{r_c}, B = PW_b^{r_c}\}$ are computed by U_b. Namely, for such a transcript, S computes a correct $Z = \mathrm{DH}_{PW_b}(A_b, B)$ and $K = Y^x$. Let $e_0 = \log_{PW_0} A_0$, $e_1 = \log_{PW_1} A_1$.

If S is honest, then $e_0 = e_1$ is known to S, and S is sure to be able to compute the correct Z and K to authenticate itself. In such a situation, (X^*, B) is a valid encryption of g^x with respect to both A_0 and A_1. Therefore, the event S_Guess_Auth happens only with probability 1/2, and thus $Adv_{P,\mathcal{D}}^{U\text{-}anoy}(S) = 0$.

If $e_0 \neq e_1$ are known to S, he can compute $Z_0 = B^{e_0}$ and $Z_1 = B^{e_1}$. S has to choose one to generate $Auth_S$, and it happens to be correct only with probability 1/2 due to the randomness of X^* and B. So $Adv_{P,\mathcal{D}}^{U\text{-}anoy}(S) = 0$.

If S is malicious while e_0 and e_1 are unknown to S, the probability that S can compute a correct authenticator is upper-bounded by solving the CDH problem. Thus $Adv_{P,\mathcal{D}}^{U\text{-}anoy}(S)$ is still negligible provided CDH assumption holds. □

4.3 Efficiency

In NAPAKE, client C_i needs to do 4 modular exponentiations with 1 on-line, and server S does $n+3$ modular exponentiations with 2 on-line. Compared with TAP $(t = 1)$, NAPAKE costs 1 (resp., 2) more modular exponentiation for C_i (resp., S), and thus less efficient than TAP with $t = 1$. However, NAPAKE is different from TAP in essence. The TAP protocol is a key transport protocol, where the session key is chosen and totally controlled by the server. The NAPAKE protocol is a key agreement protocol, where the session key is determined by both the client and the server, and none of them can control the key alone. Moreover, the NAPAKE protocol has an advantage that the server can reuse the first message for all users in Γ. That is, the server doesn't have to do the n modular exponentiations in each execution of the protocol. TAP $(t = 1)$ doesn't have this property, because the data Y in the second message must not to be the same in different runs. The n modular exponentiations would waste the computational resource of the server greatly in APAKE and TAP, if an adversary tries to connect the server continually. However in NAPAKE the first message can be reused so the threat of DoS attack would be reduced dramatically.

With respect to the communication efficiency, our protocol requires a bandwidth of $(n + 1)|\mathrm{hash}|$ and $3|q|$, which is linear to the size of the user group as TAP $(t = 1)$ and APAKE.

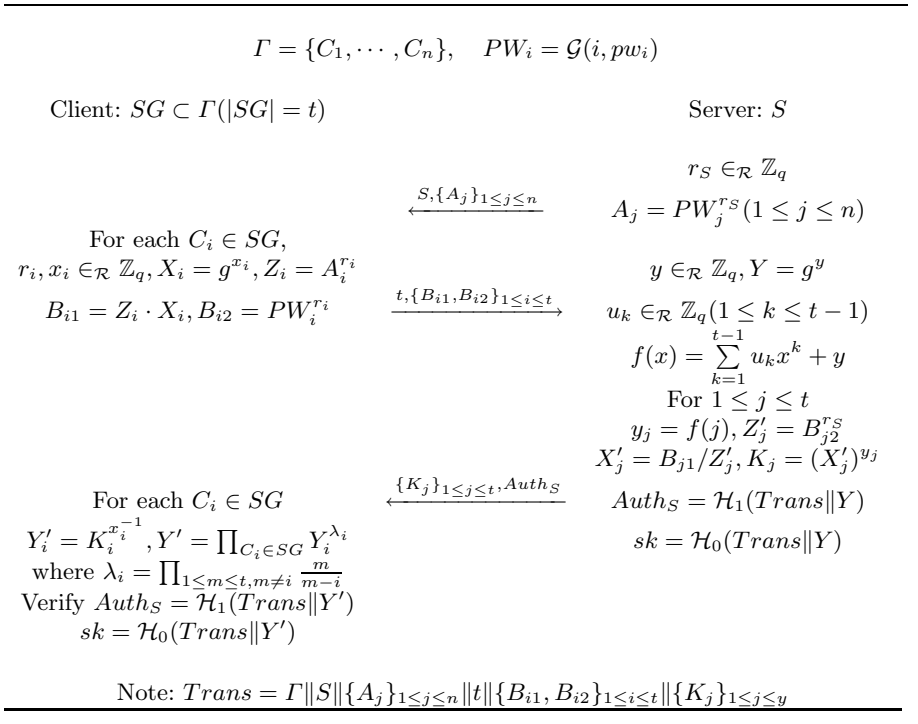


Fig. 3. The D-NAPAKE protocol

4.4 Extension of NAPAKE

Based on the proposed NAPAKE protocol, a new distributed anonymous PAKE, named D-NAPAKE, is presented in Fig.3. The D-NAPAKE also uses a secret share scheme to distribute the session key in SG, and the shared secret is $Y = g^y$, which is chosen by the server. The secret is transported to SG in a so-called Diffie-Hellman way, not the previous hiding code way in the k-out-of-n APAKE protocol and the TAP protocol. With such method the D-NAPAKE protocol is secure against the afore-mentioned impersonating attack. Furthermore, the D-NAPAKE protocol can also resist the off-line dictionary attack proposed in section 2.3. The reason is that, although D-NAPAKE also adopts the secret-share method, no one can make use of the transcript and the partial secrets $\{Y_j\}$ to guess the passwords of the clients in SG, because of the random factors of r_s, r_i and x_i.

5 Conclusion

In this paper, we proposed a new anonymous password-based authenticated key exchange protocol, named NAPAKE, which is proved secure under the SCDH and DIADH assumptions in the random oracle model.

As for the distributed anonymous PAKE, we analyzed the vulnerabilities of the Viet et al.'s k-out-of-n APAKE protocol and Shin et al.'s TAP ($t \geq 2$) protocol. We have given two attacks against the TAP protocol. One is an impersonating attack breaking the unilateral authentication of the scheme. The other is an off-line dictionary attack from a malicious insider, which make the adversary be able to guess all clients' passwords. And the second one is also applied to Viet et al.'s k-out-of-n APAKE. An extension of our protocol to the distributed setting can resist the proposed two attacks.

References

1. Abdalla, M., Pointcheval, D.: Simple Password-Based Encrypted Key Exchange Protocols. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer, Heidelberg (2005)
2. Abdalla, M., Pointcheval, D.: Interactive Diffie-Hellman Assumptions with Applications to Password-based Authentication. In: S. Patrick, A., Yung, M. (eds.) FC 2005. LNCS, vol. 3570, pp. 341–356. Springer, Heidelberg (2005)
3. Abdalla, M., et al.: Provably secure password-based authentication in TLS. In: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, pp. 35–45. ACM Press, New York (2006)
4. Bao, F., Deng, H.R., Zhu, H.F.: Variations of Diffie-Hellman Problem. In: Qing, S., Gollmann, D., Zhou, J. (eds.) ICICS 2003. LNCS, vol. 2836, pp. 301–312. Springer, Heidelberg (2003)
5. Bresson, E., Chevassut, O., Pointcheval, D.: Security Proofs for an Efficient Password-Based Key Exchange. In: Proceedings of the 10th ACM Conference on Computer and Communications Security 2003, pp. 241–250. ACM Press, New York (2003)
6. Bresson, E., Chevassut, O., Pointcheval, D.: New Security Results on Encrypted Key Exchange. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 145–158. Springer, Heidelberg (2004)
7. Boyd, C., Mathuria, A.: Protocols for authentication and key establishment. Springer, Heidelberg (2003)
8. Chai, Z.C., Cao, Z.F., Lu, R.X.: Efficient Password-Based Authentication and Key Exchange Scheme Preserving User Privacy. In: Cheng, X., Li, W., Znati, T. (eds.) WASA 2006. LNCS, vol. 4138, pp. 467–477. Springer, Heidelberg (2006)
9. MacKenzie, P.: On the Security of the SPEKE Password-authenticated Key Exchange Protocol. In: IACR ePrint archive, http://eprint.iacr.org/2001/057/
10. Shin, S., Kobara, K., Imai, H.: A Secure Threshold Anonymous Password-Authenticated Key Exchange Protocol. In: Miyaji, A., Kikuchi, H., Rannenberg, K. (eds.) IWSEC 2007. LNCS, vol. 4752, pp. 444–458. Springer, Heidelberg (2007)
11. Tzeng, W.G.: Efficient 1-Out-n Oblivious Transfer Schemes. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 159–171. Springer, Heidelberg (2002)
12. Viet, D.Q., Yamamura, A., Hidema, T.: Anonymous Password-Based Authenticated Key Exchange. In: Maitra, S., Veni Madhavan, C.E., Venkatesan, R. (eds.) INDOCRYPT 2005. LNCS, vol. 3797, pp. 244–257. Springer, Heidelberg (2005)

Group Key Management: From a Non-hierarchical to a Hierarchical Structure

Sébastien Canard and Amandine Jambert

Orange Labs R&D, 42 rue des Coutures, BP6243, F-14066 Caen Cedex, France
{sebastien.canard,amandine.jambert}@orange-ftgroup.com

Abstract. Since the very beginnings of cryptography many centuries ago, key management has been one of the main challenges in cryptographic research. In case of a group of players wanting to share a common key, many schemes exist in the literature, managing groups where all players are equal or proposing solutions where the group is structured as a hierarchy. This paper presents the first key management scheme suitable for a hierarchy where no central authority is needed and permitting to manage a graph representing the hierarchical group with possibly several roots. This is achieved by using a HMAC and a non-hierarchical group key agreement scheme in an intricate manner and introducing the notion of virtual node.

Keywords: Key Management, Access Control, Hierarchy.

1 Introduction

Key management scheme is one of the fundamental cryptographic primitive after encryption and digital signature. Such scheme allows e.g. two parties to securely exchange information among them. A running direction of research on key management is to generalize two party key agreement schemes to multi party setting, where a group of users try to create cryptographic keys together.

There are currently two main approaches regarding this generalization, depending on the structure of the group. In some cases, all members of the group are considered equals and each of them participates approximately at the same level to the construction of a cryptographic key that is finally shared by all members: this is called "group key management". Many papers exist in the literature in this case and their aim is to make the better generalization of the seminal Diffie-Hellman paper, dealing with authentication or group's dynamicity.

The second approach deals with hierarchy-based access control where members of the group are related one to another by a subordination relation while trying to access some protected documents. In this case, the group is most of the time represented as an oriented graph with no oriented cycle. In this setting, there is one key per group member and the main issue is then to provide a hierarchy of the keys in such a way that it is possible for a group member to derive from her own key all the keys that are lower in the graph. In this case, the dynamicity of the group concerns either the possibility to add or delete nodes in

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 213–225, 2008.

the graph, or the capacity to modify the key of a particular node. Ideally, these modifications imply the modification of a minority of node keys.

1.1 Related Work

The first work on this problem of key management in a hierarchy was by Akl and Taylor in 1983 [1]. Since then a large number of papers have been published [2, 4, 6, 7, 8, 9, 10, 12, 13, 17, 19, 21, 23, 25, 26, 27, 28, 29] and they can be divided into several families.

The first family contains the original paper of Akl and Taylor and its different improvements [1, 7, 10, 13, 17, 21]. These protocols use a Central Authority (CA) to generate keys and related public data. The dynamicity of the graph is not always possible in these proposals, and even in this case, a modification implies the recalculation of the keys of some predecessors. The second family is based on Sibling Intractable Function Family (SIFF) [12, 27]. While these solutions use a CA for generation and dynamism of the graph, their low complexity is quite attractive. The main problem comes from the difficulty to decide if a practical algorithm to generate such function exists or not (even in the literature [2, 22, 28]). The third group of papers uses polynomial interpolation [6, 9, 29] but [6] and [9] do not consider the dynamicity of the group, and the way to update keys in [29] is relatively inefficient. In the last group of papers [2, 4, 8, 19, 26, 28], the keys are randomly generated and the role of the CA is to provide the public link between them. Different possibilities are proposed and the best ones only use low cost operations as hash functions or xor operations. Two other papers [23, 25] use many modular exponentiations and thus induce a high complexity. Note that the solution in [25], even if presented with a CA, can be described without.

Mainly all these proposals use a Central Authority and only consider the case of a rooted graph. It is thus an open problem to describe an efficient graph key management in a multi-rooted oriented graph where (i) no Central Authority is needed and (ii) in which we can manage dynamic graphs.

1.2 Our Contribution

Our main idea in the construction of a graph key management is that we use at the same time two different solutions, depending on the structure of the subgraph we are considering. More precisely, the method to compute the key of a node in the graph depends on the number of fathers this node has. If there is one father, we use a Message Authentication Code (MAC) function on input the key of the father, a counter enumerating (approximately) the number of children of the father and a security constant.

The case where a node has several fathers cannot be treated as the case of one father and we thus adopt a different approach which consists in using group key agreement for a non-hierarchical group (in our case, the group of the fathers). More precisely, we introduce the concept of Refreshable and Replayable Group Key Agreement (R&R-GKA) schemes where the main difference with a traditional GKA scheme is that the internal state information is not truly composed of ephemeral secret information using random data, as it is the case

in existing GKA schemes. Moreover, we require an additional algorithm to replay the creation of the shared key using one private information and some public data, and we finally need a refresh method that permits to renew the shared key with a minimal effect on player's keys.

Our second trick is used when several fathers have several children in common. In that case, we introduce a virtual node between fathers and children so as to speed up the generation phase by using the "one-father method".

1.3 Organization of the Paper

The paper is organized as follows. After the present introduction, we set up in Section 2 our model for key management in an oriented graph. The cryptographic primitives we will use later are given in Section 3. Section 4 presents our scheme and its security arguments. Finally, we provide a conclusion in Section 5 and the bibliography afterward.

2 Problem and Model

The problem of access control in a hierarchy appears when users get different rights on common resources. As an example, workers in a company use common resources but according to their positions or their departments, they are not allowed to access the same documents. Another example could be on-line newspapers: different subscriptions lead to different rights.

For our part, we study the cryptographic aspect of access control. We represent the hierarchy by a graph and we look at access control as a problem of key management in that graph.

2.1 Notation

We consider an oriented *graph* denoted $G = \{N, A\}$ where $N = \{n_1, n_2, ..., n_l\}$, of cardinality l, is the set of *nodes* (in the following a node is denoted either n_i or simply i) and $A = \{a_1, a_2, ..., a_m\}$, of cardinality m, is the set of *edges*, such that there is no oriented cycles. An edge $a \in A$ corresponds to a couple of nodes (n_i, n_j), representing the fact that there is an edge going from node n_i to node n_j. n_i is called the father and n_j is the child. We denoted by F_i (resp. C_i) the number of fathers (resp. children) of the node n_i. The set of fathers of node n_i is denoted by $\mathcal{F}_i = \{f_i[1], \cdots, f_i[F_i]\}$ and the set of children of a node n_j is denoted $\mathcal{C}_j = \{c_j[1], \cdots, c_j[C_j]\}$.

A *path* $P = \{a_1, \cdots, a_k\}$ of cardinality k is a set of edges where for all $i \in \{1, \cdots, k-1\}$, if $a_i = (n_{i_0}, n_{i_1})$ and $a_{i+1} = (n_{i_2}, n_{i_3})$, then $n_{i_1} = n_{i_2}$: we also talk of the path from the first node to the last one.

If there is a path from node n_i to node n_j, we say that n_i is an *ascendant* of n_j and that n_j is a *descendant* of n_i. We denote by $\mathcal{D}_i$ the set of descendant nodes of node n_i and by $\mathcal{A}_j$ the set of ascendant nodes of node n_j. Note that $\mathcal{F}_i \subset \mathcal{A}_j$ and $\mathcal{C}_i \subset \mathcal{D}_j$.

Each node j represents a subgroup of members that share the same secret cryptographic key, named the node key and denoted k_{n_j} (or simply k_j) related to a public value pv_{n_j} (or simply pv_j). In the following, we consider a subgroup as a unique entity to avoid some authentication problems for which it exists well-known techniques. As we consider oriented graphs, we have a hierarchy between nodes. As a consequence, a node key k_{n_j} should be computable by all members of subgroups/nodes that belong to $\mathcal{A}_j$.

2.2 Actors and Procedures

We present in this section a formal definition of a graph key management scheme for a graph G. A graph key management scheme implies a set $\mathcal{P}$ of l players denoted $\mathcal{P}_1, \cdots, \mathcal{P}_l$. Each player $\mathcal{P}_i$ corresponds to a node i in the graph. In the following, we consider that the graph representation G is known by all players of the system.

Definition 1. *A Graph Key Management scheme (noted GKM) consists in the following algorithms:*

- *Setup is an algorithm which on input a security parameter τ generates the set of parameters of the system Γ. We now consider that the security parameter τ belongs to Γ.*
- *UserSetup is an algorithm which on input the set of parameters Γ provides each player in $\mathcal{P}$ with a long-lived key pair (sk_i, pk_i). From now on, Γ includes the public keys pk_i of all players.*
- *KeyGeneration is an algorithm which launches a protocol between all players $\mathcal{P}_1, \cdots, \mathcal{P}_l$, each of them taking on input the parameters Γ of the system and the long-lived key pair (sk_i, pk_i). Each player secretly outputs the first instance of the key related to its node, denoted $k_i[0]$. The algorithm outputs the first instance of some related public elements denoted $PE[0]$.*
- *KeyDerivation is an algorithm which on input the parameters Γ, a node j, a player $\mathcal{P}_i$ and an instance ρ provides the player $\mathcal{P}_i$ using the ρ-th instance of her node key $k_i[\rho]$ and the corresponding public elements $PE[\rho]$ with either an error message $\perp$ if $i \notin \mathcal{A}_j$ or the corresponding ρ-th instance of the key of node j, that is $k_j[\rho]$.*
- *KeyRefresh is an algorithm which on input the node j that needs to be refreshed launches a protocol between all players $\mathcal{P}_1, \cdots, \mathcal{P}_l$. Each player takes on input the parameters Γ, the current instance ρ, their corresponding node key $k_i[\rho]$ and the corresponding public elements $PE[\rho]$ and secretly outputs the new instance of the node key, denoted $k_i[\rho + 1]$. The algorithm outputs the new instance of some related public elements denoted $PE[\rho + 1]$.*

Remark 1. The efficiency of the KeyRefresh algorithm is a really important issue and if a particular node needs to be refreshed, this procedure should not (and needs not to) modify all the keys in the graph. The best configuration is when only the keys of the descendant nodes are modified. Note also that, for simplicity reasons, we consider in our model that all the keys change their version during this procedure, even if the new version may be equal to the previous one for some particular nodes.

2.3 Security Properties

A Graph Key Management scheme must have the Key Recovery security property. This corresponds to the fact that any coalition of players can't recover the key of a node which does not belong to their descendants.

The Key Recovery property corresponds to the following Experiment.

Experiment $Exp^{\mathsf{keyrecovery}}_{GKM,\mathcal{A}}$:

1. the challenger $\mathcal{C}$ initializes the system and sends the graph to $\mathcal{A}$.
2. $\mathcal{A}$ interacts with the system by generating and refreshing (player) keys, corrupting players and/or keys. At any time of the experiment, it must remain at least one key which is not corrupted. A key is considered as corrupted if at least one of its ascendant is corrupted or if the player corresponding to this node has beforehand been corrupted.
3. $\mathcal{A}$ finally outputs the identifier of the graph key management π, a node i, an instance ρ and an uncorrupted node key k.

We define the success of an adversary $\mathcal{A}$ for this experiment as:

$$Succ^{\mathsf{keyrecovery}}_{GKM,\mathcal{A}}(\tau) = Pr\left[k = k_i[\pi, \rho]\right].$$

Definition 2 (Key Recovery). *We say that a GKM scheme satisfies the Key Recovery property if* $Succ^{\mathsf{keyrecovery}}_{GKM,\mathcal{A}}(\tau)$ *is negligible.*

Remark 2. Note that this security model is stronger than the one given in e.g. [2] since this is the adversary who chooses the node he wants to focus on. In [2], a challenger chooses one particular node and the adversary has to output the key of this node. Note also that it is not possible to use a decisional experiment in graph key management where the aim of the adversary is to distinguish a true key from a random one (as it is done for many other key agreement primitives) since it is enough for the adversary to corrupt a descendant node and checks the consistency of the key derivation to win such game.

3 Useful Tools

3.1 The HMAC Functions

A cryptographic message authentication code (MAC) is a cryptographic tool used to authenticate a message and belongs to the family of symmetric cryptography. A *MAC scheme* is composed of a key generation algorithm KeyGen which permits to generate the MAC key denoted K. The code generation algorithm MAC accepts as input the secret key K and an arbitrary-length message m and outputs the message authentication code for message m, under the secret key K: $\Sigma = \mathsf{MAC}(K, m)$. Finally, the code verification algorithm VerMAC takes as input a message m, the secret key K and a message authentication code $\Sigma \in \mathcal{C}$ and outputs 1 if $\Sigma = \mathsf{MAC}(K, m)$ and 0 otherwise.

To be considered as secure, a MAC scheme should resist to existential forgery under chosen-plaintext attacks (EF-CMA), which means that even if an adversary $\mathcal{A}$ has access to an oracle which possesses the secret key and generates MACs for messages chosen by the adversary, $\mathcal{A}$ is unable to guess the MAC for a message it did not query to the oracle.

In our graph key management scheme (see Section 4), the used MAC scheme needs furthermore the pseudorandomness property, which says that an adversary is unable to distinguish the output of a Pseudo-Random Function (PRF) from a true random value. As a consequence, we will use the HMAC construction [20] which has been proved to be a PRF by Bellare [3].

3.2 The Notion of Refreshable and Replayable Group Key Agreement

A Group Key Agreement (GKA) scheme is a mechanism which permits to establish a cryptographic key shared by a group of participants, based on each one's contribution, over a public network. It exists several GKA schemes in the literature, using either an authenticated mode [5] or not [15, 16, 18, 24]. Note that it is possible to transform any unauthenticated protocol to an authenticated one using generic methods [11, 14].

In fact, in this paper, we need a GKA with some additional properties that are naturally verified by many of these schemes. We thus introduce the notion of Refreshable and Replayable Group Key Agreement (R&R-GKA) scheme. The main difference with a traditional GKA scheme is that the internal state information is not truly composed of ephemeral secret information using random data. Here, each player has a long-lived key to participate to the protocol but also another "personal" secret key used to create the shared one and replacing the ephemeral secret information. This new secret key can not be considered as "long-lived" since it can be refreshed when necessary. Moreover, a R&R-GKA has the following properties, which are reached most of the time by GKA schemes:

- it is a contributory Group Key Agreement protocol (GKA) [16],
- we require an additional deterministic algorithm which accepts previously fixed inputs and which is (once initialized) replayable by any player using his private information and some public data,
- it should contains a refresh method that permits to renew the shared key with a minimal effect on player's personal secret keys.

Procedures. More formally, we have the following definition, which is derived from [5]. Let $\mathcal{P}$ be the set of potential players for the GKA, that is, $\mathcal{P} = \{\mathcal{P}_1, \cdots, \mathcal{P}_l\}$.

Definition 3. *A R&R-GKA scheme consists in the following algorithms:*

- *Setup is an algorithm which on input τ generates the set of parameters of the system Γ. We now consider that the security parameter τ belongs to Γ.*
- *UserSetup is an algorithm which on input Γ provides each player in $\mathcal{P}$ with a long-lived key pair (sk_i, pk_i). Γ now includes the players public keys pk_i.*

- *KeyGeneration is an algorithm which on input a set $\mathcal{I} \subset \mathcal{P}$ of players secretly provides each player in $\mathcal{I}$ a first instance of a personal secret key $k_i[\mathcal{I}, 0]$ related to $\mathcal{I}$. This algorithm then launches a protocol between all players in $\mathcal{I}$, each of them taking on input Γ, the long-lived key pair (sk_i, pk_i) and their personal secret key $k_i[\mathcal{I}, 0]$. Each player secretly outputs the first instance of the shared secret key of the set $\mathcal{I}$ denoted $K[\mathcal{I}, 0]$. The algorithm also outputs the first instance of some related public elements denoted $PE[\mathcal{I}, 0]$.*
- *KeyRefresh is an algorithm which on input a set $\mathcal{I} \subset \mathcal{P}$ and a subset $\mathcal{J} \subset \mathcal{I}$ of players, secretly provides each player in $\mathcal{J}$ with a new instance of her personal secret key related to $\mathcal{I}$ denoted $k_i[\mathcal{I}, \rho + 1]$, if ρ is the current instance. Each player in $\mathcal{I} \setminus \mathcal{J}$ sets $k_i[\mathcal{I}, \rho + 1] = k_i[\mathcal{I}, \rho]$. This algorithm then launches a protocol between all players in $\mathcal{I}$, each of them taking on input the set of parameters Γ, the long-lived key pair (sk_i, pk_i), the two instances of their personal secret key $k_i[\mathcal{I}, \rho]$ and $k_i[\mathcal{I}, \rho + 1]$, $K[\mathcal{I}, \rho]$ and $PE[\mathcal{I}, \rho]$. Each player secretly outputs the new instance of the shared secret key of the set $\mathcal{I}$ denoted $K[\mathcal{I}, \rho + 1]$. The algorithm also outputs the new instance of the public elements denoted $PE[\mathcal{I}, \rho + 1]$.*
- *KeyRetrieve is an algorithm which on input a set $\mathcal{I} \subset \mathcal{P}$, a player $\mathcal{P}_i \in \mathcal{I}$ and an instance ρ, provides the player $\mathcal{P}_i$ taking on input the parameters Γ, the ρ-th instance of her personal secret key $k_i[\mathcal{I}, \rho]$ and the corresponding public elements $PE[\mathcal{I}, \rho]$ with the corresponding ρ-th instance of the common secret for $\mathcal{I}$, that is $K[\mathcal{I}, \rho]$.*

Note that the UserSetup procedure is done only once whereas the KeyGeneration one can be done several times, possibly in a concurrent manner.

Security property. It is commonly believed that the best security property for group key agreement is the Key Independence one (also known as Authenticated Key Exchange (AKE) property), where the aim of the adversary is to distinguish a true shared key from a random value. In this paper, we need to be sure that an adversary can not learn a non-corrupted instance of the personal secret key of a player. Since the adversary has access to the KeyRetrieve method, this is obviously related to the Key Recovery property, which says that the adversary can not compute a non-corrupted instance of the shared key.

4 Our Key Management Scheme

In this section, we present our solution of key management in an oriented graph structure. We first give an overview and then detail all procedures. Note that it is possible to use our method either in a centralized or in a distributed mode. In the first case, the roots generate all the keys and finally distribute them to all nodes. In the latter case, all nodes participate in the generation of the keys. In both cases, it is possible to construct the keys of the hierarchy while all players are not necessarily connected all the time.

4.1 Overview of Our Solution

One of our main ideas in the construction of a graph key management is that we use at the same time two different solutions, depending on the structure of the subgraph we are considering. More precisely, the method to compute the key of a child depends on the number of fathers this child has. We thus describe the two possible methods.

The case of one father. In this case, we use a simple HMAC function. Let s_i be a counter specific to the node i. This counter represents the number of times the node i has computed a new key using his own. s_i is related to the number of children, the number of refresh and potentially the dynamicity of the graph (see Section 4.5) and is maintained by the node. For each new child, the node i computes the HMAC function using its key k_i and the message corresponding to the concatenation of the counter s_i and a random constant number $C \in \{0,1\}^\tau$ specific to the graph. After that, this counter is incremented for the next child.

The case of several fathers. Here, we adopt a different approach which consists in using a non-hierarchical group key agreement (GKA) scheme. Let us consider a node i having several fathers $f_i[1], \ldots, f_i[F_i]$, where F_i is the number of fathers. Each father will be a player in the GKA scheme. By construction, each node is consequently related to a node key which will be used as a personal secret key in the GKA scheme.

As this shared value should be first computed interactively but also needs to be recalculated non-interactively, it should be possible for a father to use his node key and some public values to compute off-line the key of his child: we consequently need a R&R-GKA scheme such as described in the previous section.

Virtual nodes. The problem with the above technique is firstly that the known group key agreement schemes are deterministic and secondly that it implies many computations for all actors. Our second trick is used when two or more fathers have several children in common. In that case, we introduce a virtual node between fathers and children so as to speed up the generation phase.

This virtual node v is inserted between the fathers (f_1, f_2 and f_3) and the children (c_1 and c_2), as shown in Figure 1.

This new node is also related to a cryptographic key denoted k_v, computed from the keys of the fathers using the above method based on group key agreement schemes. Next, from this virtual key k_v it is possible to compute the keys for all children using the "one father" method, since this virtual node becomes the unique virtual father of several children.

4.2 Detail Procedures

We are now able to describe in details the different algorithms and protocols of our Graph Key Management scheme. Let τ be the security parameter, M be a secure MAC such as defined in the previous section and GKA be a secure

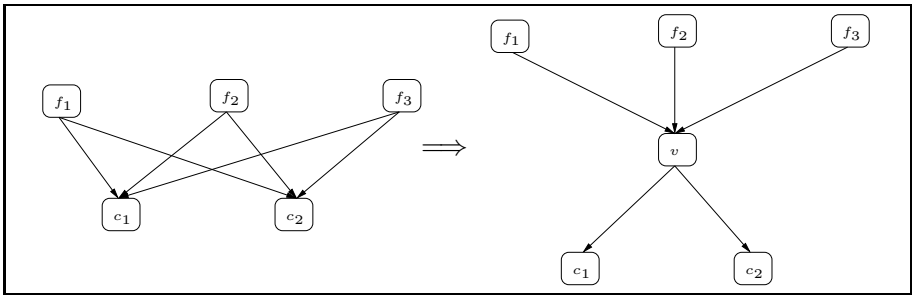


Fig. 1. The case of several fathers having several children

R&R-GKA scheme such as defined previously. Let $G = \{N, A\}$ be a graph where $N = \{1, 2, \cdots, l\}$ and $A = \{a_1, a_2, \cdots, a_m\}$. In the following, a node is tag as *keyed* when its key has been computed.

- $\mathsf{Setup}(1^\tau)$: this algorithm consists first in choosing at random a value $C \in \{0,1\}^\tau$ and second in executing the $GKA.\mathsf{Setup}(1^\tau)$ procedure which outputs $GKA.\Gamma$. The output of this algorithm is then $\Gamma = (C, GKA.\Gamma)$. This procedure also modifies the graph to insert virtual nodes, such as explained above and described in Figure 1. We denote $\tilde{G} = \{\tilde{N}, \tilde{A}\}$ the new graph with, by convention, $\tilde{N} = \{1, 2, \cdots, \tilde{l}\}$ and $\tilde{A} = \{\tilde{a}_1, \cdots, \tilde{a}_m\}$. Note that taken on input the initial graph G, the new graph $\tilde{G}$ is unique.
- $\mathsf{UserSetup}(\Gamma)$: it consists in executing the $GKA.\mathsf{UserSetup}(GKA.\Gamma)$ procedure which provides each player with a long-lived key-pair (sk_i, pk_i). All public keys are included in $GKA.\Gamma$ and thus in Γ.
- $\mathsf{KeyGeneration}()$: for each node $i \in \tilde{N}$, there are several cases.
  - i has no father in the graph: the node key $k_i[0]$ is chosen at random in $\{0,1\}^\tau$. There is no corresponding public value in this case.
  - i has one father $f \in \tilde{N}$ in the graph: let s_f be the number of current keyed children of f. Then

$$k_i[0] = M.\mathsf{MAC}(k_f[0], C \| s_f + 1).$$

    The new number of keyed children $s_f + 1$ for node f concatenated with the focused node i corresponds to the related public information $pk_i = s_f + 1 \| i$ in this case.
  - i has F_i fathers $(f_1, \cdots, f_{F_i})$: they execute the $GKA.\mathsf{KeyGeneration}$ procedure on input $\mathcal{I} = \{f_1, \cdots, f_{F_i}\}$ where each $f \in \mathcal{I}$ is "given" their node key $k_f[0]$ as a personal secret key $k_f[\mathcal{I}, 0]$. During the execution of this algorithm, the protocol between all players in $\mathcal{I}$ is launched. The key $k_i[0]$ of the node i is then the output of this protocol, that is $K[\mathcal{I}, 0]$. The related public element is then $pk_i = PE[\mathcal{I}, 0] \| i$ where $PE[\mathcal{I}, 0]$ is outputted by the $GKA.\mathsf{KeyGeneration}$ algorithm.

  At the end, at each node corresponds a key $k_i[0]$ and the algorithm outputs the first instance of the public element $PE[0]$ corresponding to the set of all public information $pk_i[0]$ outputted in the second and third cases.

– KeyDerivation(Γ, j, i, ρ): we need first to choose the best path between the nodes i and j. The choice of the smallest one (using standard graph shortest path finder algorithms) is not necessarily the best one. Obviously, in terms of computational efficiency, the case of one father (computation of a MAC) is more efficient than the case of several fathers (execution of a GKA protocol). Consequently, we should choose the path where the number of requests to the GKA is the smallest one. This path can be found either by exhaustive search or using a shortest path finder for weighted graphs. We now consider that this algorithm exists (note that it can be executed only once at the creation of the graph).

  Then, for each node v in the path between the node i and the descendant node j, this algorithm works as follows
  - if v has one father f: let $s_v \| v$ be the part of the public element $PE[\rho]$ corresponding to the focused node v. Then, computes

$$k_v[\rho] = M.\mathsf{MAC}(k_f[\rho], C \| s_v).$$

  - if v has F_v fathers $(f_1, \cdots, f_{F_v})$: let $pk_v = PE[\mathcal{V}, \rho] \| v$ be the part of the public element $PE[\rho]$ corresponding to the node v with $\mathcal{V} = \{f_1, \cdots, f_{F_i}\}$. Let $f \in \mathcal{V}$ be the father for which the key is known. This key can come from either the input of the algorithm or by derivation using one of the two methods. Then, executes the GKA.KeyRetrieve procedure on input $\mathcal{V}$, f and ρ. f takes as inputs $GKA.\Gamma$, $k_f[\mathcal{V}, \rho] = k_f[\rho]$, $PE[\mathcal{V}, \rho]$ and obtains the corresponding instance of the key $k_v[\rho]$.

– KeyRefresh(j): since we use virtual nodes, the targeted node has necessarily one father f (either a "true" father or a virtual node). We necessarily have $k_f[\rho + 1] = k_f[\rho]$ if ρ is the current instance. Thus, if we denote by s_f the number of times this father has computed a key for one of his children (either during the KeyGeneration procedure or a previous KeyRefresh one), then computes

$$k_j[\rho + 1] = M.\mathsf{MAC}(k_f[\rho + 1], C \| s_f + 1).$$

  The corresponding public information becomes $pk_j = s_f + 1 \| j$.

  Now that the key of the targeted node has been refreshed, the case of his own child has to be studied. There are then two cases for the new targeted node i, depending on the number of fathers the node has. If there is only one, we can do again what we have done for the first refresh.

  If there are several fathers $(f_1, \cdots, f_{F_i})$, let $pk_i = PE[\mathcal{I}, \rho] \| i$ be the part of the public element $PE[\rho]$ corresponding to the focused node i and where $\mathcal{I} = \{f_1, \cdots, f_{F_i}\}$. Let $\mathcal{F} \subset \mathcal{I}$ be the set of fathers for which the key has been previously refreshed. Then, executes the GKA.KeyRefresh procedure on input $\mathcal{I}$ and $\mathcal{F}$, where by assumption, each element $f \in \mathcal{F}$ has already received a new instance of its node key $k_f[\mathcal{I}, \rho + 1] = k_f[\rho + 1]$. This key is next used as a personal secret key in the GKA.KeyRefresh procedure. Again, for each $v \in \mathcal{I} \setminus \mathcal{F}$, we have $k_v[\rho + 1] = k_v[\rho]$. During the execution of this algorithm, the protocol between all players in $\mathcal{I}$ is launched. The new

instance of the key $k_i[\rho+1]$ of the node i is then the output of this protocol, that is $K[\mathcal{I}, \rho+1]$. The related public information is $pk_i = PE[\mathcal{I}, \rho+1]\|i$ where $PE[\mathcal{I}, \rho+1]$ is outputted by the $GKA.\mathsf{KeyGeneration}$ algorithm. The new instance of the public elements is then the set of all pk_i.

4.3 Security Considerations

Theorem 1. *Our key management scheme verifies the key recovery property under the existential unforgeability and the pseudorandomness of the HMAC and the key recovery property of the Group Key Management scheme.*

Proof. The idea of the proof is to play two different games with a possible adversary against the key recovery of our graph key management scheme. In the first game, we design a machine winning the EF-CMA experiment of the MAC scheme and in the second game, our machine tries to win the key recovery experiment for the R&R-GKA scheme. We thus flip a coin and play one of the two games. In case of success, we end and otherwise, we flip another coin. We are sure to succeed with probability 1/2. Due to space limitation, the complete proof is not given here.

Remark 3. During the KeyGeneration procedure, it is possible for a player to cheat by not giving the right key to one of her descendant. In order to detect such fraud, it is possible to add a key confirmation procedure where each node publishes a label related to its secret node key.

4.4 Efficiency Considerations

It is possible to instantiate our generic construction with the Group Key Agreement scheme presented in [16], where the solution is based on the use of a tree. While this solution is not the most efficient one regarding the complexity point of view, it fits very well our needs.

During the key generation phase of our construction, the case where there is only one father is very efficient since only needing a HMAC operation. When there are several fathers, we first insert the virtual node and then compute the corresponding node key using e.g. [16]. For each virtual node v, each father computes $\log_2(F_v) + 1$ modular exponentiations in a group of prime order. We should add the so-called blinded keys [16], which corresponds to $\log_2(F_v) - 1$ modular exponentiations in a group of prime order for the whole group.

4.5 The Dynamic Case

In case of a dynamic graph, we need to add two new procedures.

AddNode. The key of the j-th child of a node is computed from his father's one thanks to a specific counter s_f. In the static case, this counter is set to the number of current children plus the number of refreshes. In the dynamic case, this counter is also incremented when a new node is added to this father.

DeleteNode. We need first to modify the graph. The targeted node is deleted and the different links between ascendants and descendants are created. Note that if a path already exists from an ascendant to a descendant, there is no need to create a new one. In the second step, we refresh the keys in the sub-graph with all fathers of the deleted node as root(s). This step uses techniques described in either the KeyGeneration procedure or the KeyRefresh one and may use the dynamicity of the R&R-GKA scheme (deletion of a group member) if necessary.

5 Conclusion

This paper describes the first key management scheme suitable for multi-rooted oriented graphs with no oriented cycle without needing the presence of a central authority. In the most general setting, this scheme can be used for access control in a group with a hierarchical structure. Our construction mainly takes advantage of the use of a group key agreement designed for a non-hierarchical structure. We finally use virtual nodes in our graph so as to speed up the key generation phase.

Acknowledgments. We are grateful to Marc Girault and Jacques Traoré for their suggestions, and to anonymous referees for their valuable comments.

References

1. Akl, S.G., Taylor, P.D.: Cryptographic solution to a problem of access control in a hierarchy. In: ACM (ed.) ACM Trans. Comput. Syst. (TOCS 1983), vol. 1, pp. 239–248 (1983)
2. Atallah, M.J., Frikken, K.B., Blanton, M.: Dynamic and efficient key management for access hierarchies. In: ACM CCS 2005, pp. 190–202 (2005)
3. Bellare, M.: New proofs for nmac and hmac. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 602–619. Springer, Heidelberg (2006)
4. Birget, J., Zou, X., Noubir, G., Ramamurthy, B.: Hierarchy-based access control in distributed environments. In: IEEE International Conference on Communications, vol. 1, pp. 229–233 (2001)
5. Bresson, E., Chevassut, O., Pointcheval, D.: Provably secure authenticated group diffie-hellman key exchange. ACM Trans. Inf. Syst. Secur. 10(3), 10 (2007)
6. Chang, C.-C., Lin, I.-C., Tsai, H.-M., Wang, H.-H.: A key assignment scheme for controlling access in partially ordered user hierarchies. In: AINA 2004, p. 376. IEEE Computer Society, Los Alamitos (2004)
7. Chick, G.C., Tavares, S.E.: Flexible access control with master keys. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 316–322. Springer, Heidelberg (1990)
8. Chou, J.-S., Lin, C.-H., Lee, T.-Y.: A novel hierarchical key management scheme based on quadratic residues. In: Cao, J., Yang, L.T., Guo, M., Lau, F. (eds.) ISPA 2004. LNCS, vol. 3358, pp. 858–865. Springer, Heidelberg (2004)
9. Das, M.L., Saxena, A., Gulati, V.P., Phatak, D.B.: Hierarchical key management scheme using polynomial interpolation. SIGOPS 39(1), 40–47 (2005)
10. De Santis, A., Ferrara, A.L., Masucci, B.: Cryptographic key assignment schemes for any access control policy. Inf. Process. Lett. 92(4), 199–205 (2004)

11. Desmedt, Y., Lange, T., Burmester, M.: Scalable authenticated tree based group key exchange for ad-hoc groups. In: Dietrich, S., Dhamija, R. (eds.) FC 2007 and USEC 2007. LNCS, vol. 4886, pp. 104–118. Springer, Heidelberg (2007)
12. Hardjono, T., Zheng, Y., Seberry, J.: New solutions to the problem of access control in a hierarchy. Technical Report Preprint 93-2 (1993)
13. He, M., Fan, P., Kaderali, F., Yuan, D.: Access key distribution scheme for level-based hierarchy. In: PDCAT 2003, pp. 942–945 (2003)
14. Katz, J., Yung, M.: Scalable protocols for authenticated group key exchange. J. Cryptol. 20(1), 85–113 (2007)
15. Kim, Y., Perrig, A., Tsudik, G.: Group key agreement efficient in communication. IEEE Trans. Comput. 53(7), 905–921 (2004)
16. Kim, Y., Perrig, A., Tsudik, G.: Tree-based group key agreement. ACM Trans. Inf. Syst. Secur. 7(1), 60–96 (2004)
17. Kuo, F.H., Shen, V.R.L., Chen, T.S., Lai, F.: Cryptographic key assignment scheme for dynamic access control in a user hierarchy. In: IEE Proceedings Computers and Digital Techniques, vol. 146, pp. 235–240 (1999)
18. Lee, S., Kim, Y., Kim, K., Ryu, D.-H.: An efficient tree-based group key agreement using bilinear map. In: Zhou, J., Yung, M., Han, Y. (eds.) ACNS 2003. LNCS, vol. 2846. Springer, Heidelberg (2003)
19. Lin, C.-H.: Hierarchical key assignment without public-key cryptography. Computers & Security 20, 612–619 (2001)
20. Krawczyk, H., Bellare, M., Canetti, R.: Hmac: Keyed-hashing for message authentication. In: RFC 2104 (1997)
21. MacKinnon, S.J., Taylor, P.D., Meijer, H., Akl, S.G.: An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. IEEE Trans. Comput. 34(9), 797–802 (1985)
22. Ragab Hassen, H., Bouabdallah, A., Bettahar, H., Challal, Y.: Key management for content access control in a hierarchy. Comput. Netw. 51(11), 3197–3219 (2007)
23. Ray, I., Narasimhamurthi, N.u.: A cryptographic solution to implement access control in a hierarchy and more. In: SACMAT 2002, pp. 65–73. ACM, New York (2002)
24. Steiner, M., Tsudik, G., Waidner, M.: Key agreement in dynamic peer groups. IEEE Transactions on Parallel and Distributed Systems 11(8), 769–780 (2000)
25. Wu, J., Wei, R.: An access control scheme for partially ordered set hierarchy with provable security (2004/295) (2004), http://eprint.iacr.org/
26. Zhang, Q., Wang, Y.: A centralized key management scheme for hierarchical access control. In: IEEE GLOBECOM 2004, vol. 4, pp. 2067–2071 (2004)
27. Zheng, Y., Hardjono, T., Pieprzyk, J.: Sibling intractable function families and their applications. In: Matsumoto, T., Imai, H., Rivest, R.L. (eds.) ASIACRYPT 1991. LNCS, vol. 739, pp. 124–138. Springer, Heidelberg (1993)
28. Zhong, S.: A practical key management scheme for access control in a user hierarchy. Computers & Security 21, 750–759 (2002)
29. Zou, X., Karandikar, Y., Bertino, E.: A dynamic key management solution to access hierarchy. Int. J. Netw. Manag. 17(6), 437–450 (2007)

Scan Based Side Channel Attacks on Stream Ciphers and Their Counter-Measures

Mukesh Agrawal[1], Sandip Karmakar[2], Dhiman Saha[2], and Debdeep Mukhopadhyay[3]

[1] B.Tech. Student, [2] MS Student, [3] Assistant Professor
Dept. of Computer Science and Engineering
Indian Institute of Technology Kharagpur, India
{mukesh,sandipk,dhimans,debdeep}@cse.iitkgp.ernet.in

Abstract. Scan chain based attacks are a kind of side channel attack, which targets one of the most important feature of todays hardware - the test circuitry. Design for Testability (DFT) is a design technique that adds certain testability features to a hardware design. On the other hand, this very feature opens up a side channel for cryptanalysis, rendering crypto-devices vulnerable to scan-based attack. Our work studies scan attack as a general threat to stream ciphers and arrives at a general relation between the design of stream ciphers and their vulnerability to scan attack. Finally, we propose a scheme which we show to thwart the attacks and is more secure than other contemporary strategies.

1 Introduction

The widespread use of low-power hand-held/portable devices have necessitated the use of stream ciphers for cryptographic security. Stream ciphers [1] are much less power consuming and much faster than the block ciphers. The security of stream ciphers has been examined by several researchers. However, nowadays it is not only important for an algorithm to be mathematically robust but also they should be resistant against attacks which exploit implementation weaknesses. This class of cryptanalysis are known as Side Channel Attacks(SCA). They use the information leaked by side channels like power, timing etc. One such side channel attack is scan chain based attack. This technique exploits the fact that most of modern day ICs are designed for testability. Scan chains are one of the most popular methods to test a design. In this scheme all flip-flops (FFs) are connected in a chain and the states of the FFs can be scanned out through the chain. Scan testing equips a user with two very powerful features namely controllability and observability. Controllability refers to the fact that the user can set the FFs to a desired state, while observability refers to the power to observe the content of the FFs. Both these desirable features of a scan-chain testing methodology can be fatal from cryptographic point of view. In cryptographic algorithms, knowledge of intermediate values of the cipher can seriously reduce the complexity of breaking it. It has been shown in literature ([2],[3]) that attackers can very effectively use this scan technique to completely break a cryptosystem.

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 226–238, 2008.

Such kinds of attack can have profound practical importance as the security of the system can be compromised using unsophisticated methods. For example, keys can be extracted from popular pay TV set-top boxes via scan-chains at a cost of few dollars. Naturally, the attack can be prevented by eliminating the test capabilities from the design. However, this increases the risk of shipping chips with defects, which may hamper the normal functionality of the designs. Hence, research is needed to study the scan-chain based attacks against designs of standard ciphers with the objective of developing suitable DFT techniques for cryptographic hardware.

In this paper we have attacked Trivium, a hardware based stream cipher enlisted in phase 3 of the **eStream** [4] project. We have generalized the attack to any stream cipher and have highlighted the reason as to why such implementations crumble against scan based attacks. To prevent such attacks, various secure scan-chain design have been proposed. [2] proposed the technique in which inverters are inserted at random points of a scan chain. However, we show in this paper that the inverter based scheme can still be attacked. [3] provided a state machine based technique to achieve the same. But the architecture is difficult to incorporate in existing design flow. The motivation of this work is to propose a secure scan chain mechanism which is resistant against the existing attacks, at the same time provides high quality testability.

The paper is organized as follows: Section 2 gives a brief description of scan-chains and the scan based attacks on stream ciphers, Section 3 describes a case-study on a scan based attack on Trivium stream cipher, while Section 4 generalizes scan based attack on stream ciphers. A suitable counter-measure against scan chain based attacks is proposed in Section 5. Finally, Section 6 concludes the paper. The specification of Trivium is discussed in the Appendix.

2 Preliminaries

2.1 Scan Chains and Scan Based Attacks

Scan chains are a DFT technique kept with the objective to test designs by providing a simple way to set and observe every flip-flop in an Integrated Circuit. A special signal called scan enable is added to a design. When this signal is asserted, every flip-flop in the design is connected as a chain of registers. One input pin provides the data to this chain, and one output pin is connected to the output of the chain. On each clock event, an input pattern can be scanned-in to the registers. Then after a normal run of the device, the updated contents of the registers are scanned out of the scan chain. A standard scan chain is shown in Fig. 1. The test data is given through the scan_in line and the output pattern is scanned out through the scan_out line as shown in the figure.

The notion of scan based attacks and its prevention on crypto hardware appeared in many contemporary works. [5] has shown scan based attack on dedicated hardware implementation of the Data Encryption Standard(DES). Among some secure scan chain designs are Scan_enable integrity [6], Scan chain scrambling [7],

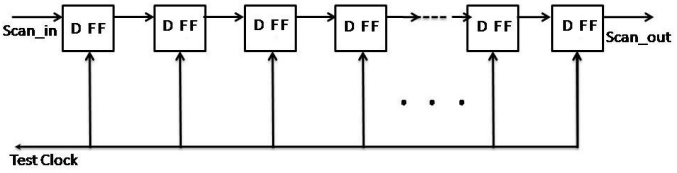


Fig. 1. Design of a Scan Chain

tree based scan chain with a compactor [8], Spy Flip-flop[9], Flipped-scan [2], Circularscan [10], Lock and Key [11] and Secure Scan [3]. A comparative study on the existing scan chain techniques have been presented in [12]. A scan attack on the Advanced Encryption Standard(AES) is discussed in [3], while [2] mounts an attack on the RC4 stream cipher.

A scan attack has two phases, namely, the phase when an attacker ascertains the internal structure of the scan chain, followed by deciphering the cryptogram. Our study reveals that stream ciphers are generally vulnerable to the first phase of a scan attack, while vulnerability to the second phase depends specifically on the robustness of the algorithm. To be precise, the second part of the attack depends on whether it is possible to track back to previous state of the cipher from the knowledge of the current state.

Next, we give a scan based attack on Trivium stream cipher as a case study. Trivium is a hardware based stream cipher enlisted in phase 3 of the **eStream** [13] project. It may be noted that although the following attack is targeted for the implementation of Trivium in [14], the attack may be adapted to other implementations as well. However, as any side channel analysis the attack is dependent on the implementation.

3 A Case Study on the Trivium Stream Cipher

3.1 Objective of the Attacker

The aim is to obtain the message stream from the stream of ciphertexts. He observes the cryptogram $c_1, c_2, c_3, \cdots, c_l$. He then gets the possession of the device and his intention is to obtain the plaintexts $m_1, m_2, m_3, \cdots, m_l$ using scan-chain based side channel analysis.

3.2 Attack on Trivium

Here we show the attack on the implementation of Trivium suggested in [14]. Trivium has a total of 288 internal state registers. The control circuitry in [14] requires an 11-bit counter which adds 11 more registers while 3 registers are required for the temporary variables. So the scan chain for trivium has a total of 302 registers.

As already mentioned in Section 2, a scan attack consists of two phases, ascertaining which bit corresponds to which register and deciphering the cryptogram.

Scan based attacks try to exploit the information gained about the internal states of the cipher through the scan chain. All hardware with scan chain based DFT have a scan-in and scan-out line. This allows the user to scan in a desired pattern in test mode, run the circuit in normal mode and then scan out the pattern to verify the functioning of the device. In case of crypto devices this feature equips an attacker with a tool by virtue of which he can gain knowledge about the internal intermediate states of the crypto algorithm. He exploits this knowledge to break the cryptosystem. Literature shows that these attacks are quite easy to implement and do not require any sophisticated set-up.

Ascertaining Bit Correspondence. As far as Trivium is concerned the first phase of the attack can be broken down in 3 parts i.e., ascertaining the location of the internal state bits, the counter bits and the temporary registers. Once the attacker gets hold of the crypto-chip, he tries to scan out the data to get the state of the registers. What the attacker does not know is the correspondence between the actual bits and the pattern he has scanned out. However, by some clever use of key and IV setup and by using certain properties of the state update function he can easily find the exact positions of the bits in the scanned out pattern. The following procedure details the steps the attacker takes:

1. **Ascertaining location of counter bits**
   The attacker can ascertain the positions of the counter bits by exploiting the Key-IV input pattern. As per the design in [14] the user is needed to give a padded input pattern as, $(Key_{80}, 0^{13}, IV_{80}, 0^{112}, 1^3)$ through the Key_IV padding line. Here Key_{80} represents the 80 bits of the key while 0^{13} denotes a sequence of 13 0's, similarly the other notations may be explained. The attacker inputs a pattern of all zeros i.e., 0^{288}. He then runs the system in normal mode for $(2^{10} - 1)$ cycles. Since the internal state of the cipher is set to all zeros, so according to Equation 7 (refer Appendix), running the cipher will have no change on the 288-bit internal state register and the temporary registers which always remain in an all zero state. The only thing that changes is the 11-bit counter. Since the system runs for $(2^{10} - 1)$ cycles, so 10 bits of the counter will be set to 1. The attacker then scans out the pattern and the bits which are 1, are concluded to be the 10 counter bits. The 11^{th} counter bit can be determined by scanning in the scanned out pattern again and running the system for 1 cycle. This sets the 11^{th} counter bit to 1. The attacker now scans out the pattern to get the bit position. The attacker requires a total of 288 clock cycles to scan-in the pattern and the same number of clock cycles to scan-out the pattern.

   It might be noted that to mount the scan attack the attacker need not know which bit of the counter corresponds to which bit-position but only the positions of all the counter-bits as a whole. The number of clock-cycles required is $(288 + 2^{10} - 1 + 288 + 288 + 1 + 288) = 2176$.

2. **Ascertaining the internal state bits**
   This part of the attack is straight-forward. The attacker here again tries to exploit the key-IV input pattern. He first resets the circuit and then gives

a pattern in which only $key_1 = 1$ and remaining bits are 0 during Key-IV setup (Equation 5) .i.e.,

$$(s_1, s_2, s_3, s_4, \cdots, s_{288}) \leftarrow (1, 0^{79}, 0^{13}, 0^{80}, 0^{112}, 0^3).$$

He then runs the system for 1 clock-cycle to load the first key bit, after which he scans out the entire pattern in test-mode. The output pattern will have 1s in two positions. Out of these one will correspond to the counter LSB which the attacker already knows while the other corresponds to s_1. So the bit position of s_1 in the output pattern is ascertained. He proceeds likewise setting to 1 only the bit he wants to find in the input pattern and then running the corresponding number of clock-cycles in normal mode. Typically, to find the i^{th} bit position he has to run the system for i clock periods. He thus finds the bit positions for all the 288 state bits in the output pattern. The number of clock-cycles required is $\sum_{i=1}^{288} i + 288^2 = 124560$.

3. **Ascertaining the temporary register bits**
The attacker has already determined the positions of $(288 + 11) = 299$ internal registers. He then finds the temporary registers t_1, t_2, t_3. The following equations can be derived from Equation 6,

$$t_1 \leftarrow s_{66} \oplus s_{91}.s_{92} \oplus s_{93} \oplus s_{171}$$
$$t_2 \leftarrow s_{162} \oplus s_{175}.s_{176} \oplus s_{177} \oplus s_{264}$$
$$t_3 \leftarrow s_{243} \oplus s_{286}.s_{287} \oplus s_{288} \oplus s_{69}$$

The attacker sets the input pattern in such a way that when he runs the cipher in normal mode after key-IV load, only one of the temporary registers changes their value to 1 while the others are 0. For example, the attacker can set $s_{66} = 1$ i.e.,

$$(s_1, s_2, s_3, s_4, \ldots, s_{288}) \leftarrow (0^{65}, 1, 0^{12}, 0^{13}, 0^{80}, 0^{112}, 0^3).$$

He then loads the Key-IV pattern in normal mode and runs for 1 more clock cycle which will set $t_1 = 1$. He scans out the pattern and as he already knows all other bit positions he can get the position of t_1. This requires 2×288 clocks, 288 for scanning in and 288 for scanning out. He repeats the process to get t_2. Once he obtains t_1 and t_2, the position of t_3 becomes evident. Thus, he requires a total of 4×288 clocks.

This concludes the first phase of the attack with the attacker now having the knowledge of all the bits positions in the internal state of the cipher. He then advances to the next phase where he attempts to decipher the cryptogram from the knowledge of the internal state.

3.3 Deciphering the Cryptogram

In a stream cipher, we XOR the plain text bit to the key stream bit to get the ciphertext bit. So, if we have key stream bit K_i and cipher text C_i, plaintext bit P_i can be obtained as,

$$P_i = K_i \oplus C_i$$

The attacker had scanned out the internal state of Trivium after getting hold of the device. Now, he has ascertained the position of state bits in the scan-chain. This information will be used to decipher the cryptogram and to obtain the plaintext. We proceed by knowing the previous state from the current state. Table 1 gives a clear picture about the relation of present and previous states of Trivium. As is clear from the encryption algorithm, current state is a right shift of previous state with first bit being a non-linear function of some other bits. So, our task remains to calculate 'a','b' and 'c'. Observe the following equations:

$$t_1 = s_{66} \oplus s_{93} \tag{1}$$

$$t_1 = t_1 \oplus s_{91} \cdot s_{92} \oplus s_{171} \tag{2}$$

$$(s_{94}, s_{95}, \cdots, s_{177}) \leftarrow (t_1, s_{94}, \cdots, s_{176}) \tag{3}$$

Equations 1 and 2 can be combined to get,

$$t_1 = s_{66} \oplus s_{93} \oplus s_{91} \cdot s_{92} \oplus s_{171} \tag{4}$$

This should be noted that if we give a clock at this configuration of Trivium, register s_{94} gets loaded with t_1 and other bits are shifted to their right. So, we can say that what is s_{67} now, must have been s_{66} in the previous state and what is s_{93} now, must have been s_{92} in the previous state and so on. Hence, from Equation 3 and 4 and by referring to Table. 1 we have the following equation:

$$s_{94} = a \oplus s_{67} \oplus s_{92} \cdot s_{93} \oplus s_{172}$$
$$\Rightarrow a = s_{94} \oplus s_{67} \oplus s_{92} \cdot s_{93} \oplus s_{172}$$

Similarly, 'b' and 'c' can be deduced by the following set of equations:

$$s_{178} = b \oplus s_{163} \oplus s_{176} \cdot s_{177} \oplus s_{265}$$
$$\Rightarrow b = s_{178} \oplus s_{163} \oplus s_{176} \cdot s_{177} \oplus s_{265}$$

And,

$$s_1 = c \oplus s_{244} \oplus s_{287} \cdot s_{288} \oplus s_{70}$$
$$\Rightarrow c = s_1 \oplus s_{244} \oplus s_{287} \cdot s_{288} \oplus s_{70}$$

Hence, we can compute all the previous states given a single current state of the internal registers. Once obtained a state, one can easily get the key stream bit for the l successive time units by following Equation 7. The key stream bit when xored with the ciphertext bit of the state produces the corresponding plaintext bit.

Table 1. Internal states of Trivium

Present State	Previous State
$(s_1, s_2, \cdots, s_{93})$	$(s_2, s_3, \cdots, s_{93}, a)$
$(s_{94}, s_{95}, \cdots, s_{177})$	$(s_{95}, s_{96}, \cdots, s_{177}, b)$
$(s_{178}, s_{179}, \cdots, s_{288})$	$(s_{179}, s_{180}, \cdots, s_{288}, c)$

3.4 Attack Simulation

The stream cipher Trivium was implemented in *Verilog HDL*. A scan-chain was inserted in the design. The entire attack was simulated and verified on the Spartan3 xc3s5000-fg900 FPGA platform. The results of the simulations entirely matched with the attack scenario presented above.

4 Generalization of the Attack on Stream Ciphers

The strategy applied above to break Trivium can be extended to any stream cipher. In general, the structure of stream ciphers make them vulnerable only to the first phase of the attack. This is because all of them have a key and/or IV setup phase where the user is allowed to load the key or IV from outside. This can be exploited to ascertain the bit correspondence between the internal registers and the key-IV bits. However, Stream ciphers also have some bits that are not loaded from outside. These bits are internally assigned to either zeros or ones. Unfortunately, this is not good enough. These bits also have to be included in the scan-chain for the purpose of testability and may be over-written by the scan input and read through scan-out line. This shall determine their positions in the scan chain. However, the second phase of the attack cannot be generalized over all stream ciphers. The second phase of the attack relies on the particular algorithm. As already mentioned, this depends on whether we can deduce the previous state of the cipher from the knowledge of its current state. The case-study reveals the fact that an insecure scan-chain can be fatal even to the modern stream ciphers. So we need to implement a secure scan-chain which shall help us to test the device without revealing the internal states to an unauthorized person. Next section describes such a counter-measure.

5 Prevention Mechanism

One of the recent techniques in implementing secure scan-chains is the Flipped-scan [2] technique. In this scheme inverters are introduced at random points in the scan-chain. Security lies in the fact that an attacker cannot guess the positions of the inverters with a probability significantly greater that $\frac{1}{2}$. However, this scheme is vulnerable to an attack which we call the reset attack. The reset attack is elaborated next.

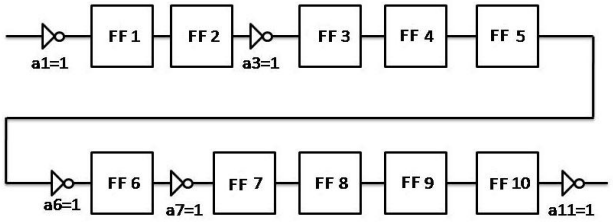


Fig. 2. Example Demonstrating Attack on Flipped-Scan

5.1 Reset Attack on Flipped-Scan Mechanism

In standard VLSI design, each FF is accompanied by either a synchronous or asynchronous RESET which initializes the FF to zero. An attacker can assert the reset signal and obtain the scan-out pattern by operating the crypto chip in test mode. The scan-out pattern would look like series of 0's interleaved with series of 1's. The places where polarity is reversed are the locations where an inverter has been inserted. The following example illustrates the attack.

Example 1. Suppose there is a scan-chain of 10 D-FFs; and inverters are placed at position numbers 1, 3, 6, 7 and 11, out of 11 possible positions as shown in Fig. 2. We apply a reset signal and scan out the pattern. We will get a pattern equal to $X_1, X_2, \cdots, X_{10}$, where $X_i = a_{i+1} \oplus a_{i+2} \cdots \oplus a_{11}$, where a_i is 1 if there is an inverter in the i_{th} link and $a_i = 0$ otherwise.

We will get this pattern in the order $X_{10}, X_9, \cdots, X_1$. In this example,

$$\begin{aligned} X_{10} &= a_{11} \\ X_9 &= a_{10} \oplus a_{11} \\ X_8 &= a_9 \oplus a_{10} \oplus a_{11} \\ &\vdots \\ X_1 &= a_2 \oplus a_3 \cdots a_{11} \end{aligned}$$

From the above set of equations it follows that we will get a sequence $X = \{0, 0, 1, 1, 1, 0, 1, 1, 1, 1\}$. As previously stated, inverters have been inserted at those positions where polarity is reversed in the pattern. Here, the positions where inverters were placed are 3,6,7 and 11. Presence of inverter at the 11^{th} position was detected using the fact that first bit obtained while scanning out is 1. It can be easily deduced whether there is an inverter in the first position or not by feeding in a bit and observing if the output toggles or not. Thus, inverters are placed at positions 1,3,6,7 and 11. By the above procedure, one can always ascertain the location of inverters. Therefore, the design in [2] is no longer secure.

In the following section we give a scheme which overcomes this threat against the reset attack while maintaining high level of security against conventional scan chain based attacks.

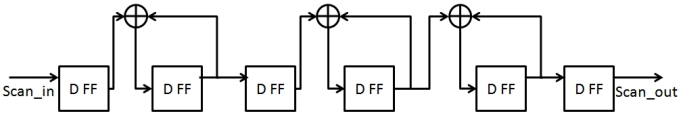


Fig. 3. Xor-Chain Architecture

5.2 Description of Xor-Chain

We propose a prevention scheme which is based on insertion of xor gates at random points in the scan chain. We call the scheme *Xor-Chain*. One of the inputs to the xor gate is the present input of the flip-flop from the one preceding it in the chain while the other input is the current output of the flip-flop. Actually each xor-gate serves as a data-dependent inverter which conditionally inverts the present input based on the past input. Clearly, this configuration passes the reset attack on the flipped scan architecture, as in case of reset, scan-chain output will be all zeros, i.e., the xor gates become transparent.

Fig. 3 shows a xor-chain architecture having six D-FFs. In this structure xor-gates are inserted before 2^{nd}, 4^{th} and 5^{th} FFs. In order to test a circuit, we first reset the chip and then feed in a pattern to the chain architecture.

5.3 Testability

We next inspect whether the proposed xor chain architecture may be used to test the circuit. For this purpose the following result needs to be satisfied by the proposed scan chain.

Theorem 1. *Let $X = \{X_1, X_2, \cdots, X_n\}$ be the vector space of inputs to the xor-chain and $Y = \{Y_1, Y_2, \cdots, Y_n\}$ be the vector space of outputs from the xor-chain. Then there is a one-to-one correspondence between X and Y, if the xor chain is reset before feeding in the pattern X.*

Proof. The proof has essentially two parts. First we try to prove the one-to-one correspondence between the input and what gets into the FFs.

Let the existence of an xor gate be denoted by a variable $a_i \in (0, 1)$ such that if $a_i = 1$ then there is an xor gate before the i^{th} FF. Otherwise, if $a_i = 0$ there is no xor gate. At any instant of time t the state of the internal FFs of the xor-chain can be expressed as follows:

$$
\begin{aligned}
S_1^t &= X_{n-t} \oplus S_1^{t-1}.a_1 \\
S_2^t &= S_1^{t-1} \oplus S_2^{t-1}.a_2 \\
S_3^t &= S_2^{t-1} \oplus S_3^{t-1}.a_3 \\
&\cdots \\
S_n^t &= S_{n-1}^{t-1} \oplus S_n^{t-1}.a_n
\end{aligned}
$$

where,

$$\begin{aligned} n &\Longrightarrow \text{ the total number of FFs,} \\ S_i^t &\Longrightarrow \text{the state(value) of the } i^{th} \text{ FF in the } t^{th} \text{ clock cycle,} \\ X_{n-t} &\Longrightarrow \text{the input to the xor-chain at the } t^{th} \text{ clock cycle.} \end{aligned}$$

Solving the above system of equations by multiplying these equations with appropriate values of a_i and subsequent xoring, we get the following expression:

$$a_1.S_2^t \oplus a_1.a_2.S_3^t \oplus \ldots (a_1.a_2...a_{n-1}).S_n^t = S_1^t \oplus X_{n-t} \oplus (a_1.a_2...a_n).S_n^{t-1}$$

For testing, before scanning in a pattern, the circuit is always reset. Hence, all FFs are initially holding 0. In the above equation, S_n^{t-1} is the state of the n^{th} FF at the $(t-1)^{th}$ clock cycle. However, that should be 0 because of the initial reset. So the term $(a_1.a_2...a_n).S_n^{t-1}$ is always 0 independent of the values of $a_1, a_2, ...a_n$. Thus the resultant equation can be rewritten as:

$$X_{n-t} = a_1.S_2^t \oplus a_1.a_2.S_3^t \oplus \ldots (a_1.a_2...a_{n-1}).S_n^t \oplus S_1^t$$

It can be inferred from above that given the states of all the FFs at an instant (i.e., $S_1^t, ..., S_n^t$) and the configuration of the xor-chain(i.e., $a_1, ..., a_n$) one can find out what the input to the chain was. Thus we can conclude that given a fixed X and $(a_1, a_2, .., a_n)$, the states of internal FFs have a one-to-one mapping with vector space X.

Proceeding in a similar way it can be established that the states of the FFs and the pattern scanned out will also bear a one-to-one correspondence. Thus the pattern scanned in and scanned out of the xor-chain will also have a one-to-one relation among themselves.

Thus the proposed xor chain architecture can be used for testability as we can have a unique output pattern for a given input pattern.

5.4 Security Analysis

In order to mount a scan attack on a crypto hardware in general and stream cipher in particular, an attacker must first ascertain the structure of the scan-chain. So the main aim of any prevention mechanism is to thwart such an attempt. Similar, to the philosophy of flipped scan chain, security of our scheme relies on the fact that if the positions of the xor gates are unknown to the attacker, then it is computationally infeasible for him to break the structure of the scan-chain. As a rule of thumb in designing we insert $m = \lfloor n/2 \rfloor$ xor gates in a scan-chain with n FFs. We show in the following that even if the attacker has the knowledge of the number of xor gates (m) among the number of FFs, the probability of successfully determining the structure is about $1/2^n$.

Theorem 2. *For an xor scan-chain with n FFs and $m = \lfloor n/2 \rfloor$ xor gates, the probability to guess the correct structure by an attacker with knowledge of n is nearly $1/2^n$.*

Table 2. Hardware Overhead Comparison

Prevention Mechanism	No. of insertions	Total Transistor Count
Flipped Scan	$\lfloor (n+1)/2 \rfloor$	$(n+1)$
Xored Scan	$\lfloor n/2 \rfloor$	$5n$

Proof. Let the number of FFs be n and the number of xor gates be m. Hence, the number of structures possible is ${}^{n}C_{m}$, where ${}^{n}C_{m}$ means the number of ways of choosing m unordered elements from n elements. Let us assume that the attacker uses the knowledge that the number of xor gates is half the number of FFs, so that we have $m = \lfloor n/2 \rfloor$. Therefore, the probability to guess the correct structure is now $1/{}^{n}C_{\lfloor n/2 \rfloor}$.

In order to compute the bound, we use the fact the maximum value of ${}^{n}C_{r}$ is when $r = \lfloor n/2 \rfloor$. We combine this with the fact that the maximum binomial coefficient (given by the above value) must be greater that the average of all binomial coefficients, i.e., when r runs from 0 to n.

But, ${}^{n}C_{\lfloor n/2 \rfloor} > 2^{n}/(n+1) \Rightarrow {}^{n}C_{\lfloor n/2 \rfloor} > 2^{n-log_2(n+1)}$. And for large values of n, $n - log_2(n+1) \approx n$. Hence, the probability of guessing the correct scan structure is nearly $1/2^{n}$

5.5 Hardware Overhead

The xor-chain model requires insertion of *xor* gates at random points of a scan chain. In terms of gate count, the xor-chain is more costly than its nearest counterpart *flipped-scan*. Table 2 compares the hardware overhead of the proposed prevention scheme with [2].

6 Conclusion

In this paper, we have shown that hardware designs of stream ciphers can be subjected to attacks when tested using scan chains. We demonstrate such an attack on a recent hardware based phase-3 stream cipher of eStream, named Trivium. The attack shows that stream ciphers, because of their inherent design methodology are vulnerable to scan based attacks. The paper in search of counter-measures against scan attacks shows that a contemporary strategy named Flipped Scan Chain may be attacked. Finally, a new protection mechanism using xor chains has been suggested and shown to be resistant against known attacks without compromising testability.

References

1. Stallings, W.: Cryptography and Network Security: Principles and Practice. Pearson Education, London (2002)
2. Sengar, G., Mukhopadhyay, D., Chowdhury, D.R.: Secured flipped scan-chain model for crypto-architecture. IEEE Trans. on CAD of Integrated Circuits and Systems 26(11), 2080–2084 (2007)

3. Yang, B., Wu, K., Karri, R.: Secure scan: A design-for-test architecture for crypto chips. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 25(10), 2287–2293 (2006)
4. eSTREAM, European network of excellence for cryptology (ecrypt) stream cipher project
5. Yang, B., Wu, K., Karri, R.: Scan based side channel attack on dedicated hardware implementations of data encryption standard. In: ITC 2004: Proceedings of the International Test Conference on International Test Conference, Washington, DC, USA, pp. 339–344. IEEE Computer Society, Los Alamitos (2004)
6. Hely, D., Bancel, F., Flottes, M.-L., Rouzeyre, B.: Test control for secure scan designs. In: ETS 2005: Proceedings of the 10th IEEE European Symposium on Test, Washington, DC, USA, pp. 190–195. IEEE Computer Society, Los Alamitos (2005)
7. Hely, D., Flottes, M.-L., Bancel, F., Rouzeyre, B., Berard, N., Renovell, M.: Scan design and secure chip. In: IOLTS 2004: Proceedings of the 10th IEEE International On-Line Testing Symposium, Washington, DC, USA, p. 219. IEEE Computer Society, Los Alamitos (2004)
8. Mukhopadhyay, D., Banerjee, S., RoyChowdhury, D., Bhattacharya, B.B.: Cryptoscan: A secured scan chain architecture. In: ATS 2005: Proceedings of the 14th Asian Test Symposium on Asian Test Symposium, Washington, DC, USA, pp. 348–353. IEEE Computer Society, Los Alamitos (2005)
9. Hély, D., Bancel, F., Flottes, M.-L., Rouzeyre, B.: A secure scan design methodology. In: DATE 2006: Proceedings of the conference on Design, automation and test in Europe, 3001 Leuven, Belgium, pp. 1177–1178. European Design and Automation Association (2006)
10. Arslan, B., Orailoglu, A.: Circularscan: A scan architecture for test cost reduction. date 02, 21290 (2004)
11. Lee, J., Tehranipoor, M., Patel, C., Plusquellic, J.: Securing scan design using lock and key technique. In: DFT 2005: Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Washington, DC, USA, pp. 51–62. IEEE Computer Society, Los Alamitos (2005)
12. Hely, D., Bancel, F., Flottes, M.-L., Rouzeyre, B.: Secure scan techniques: A comparison. In: IOLTS 2006: Proceedings of the 12th IEEE International Symposium on On-Line Testing, Washington, DC, USA, pp. 119–124. IEEE Computer Society, Los Alamitos (2006)
13. Trivium, http://www.ecrypt.eu.org/stream/triviump3.html
14. Chelton, W., Good, T., Benaissa, M.: Review of stream cipher candidates from a low resource hardware perspective, http://www.ecrypt.eu.org/stream/triviump3.html

Appendix

A Trivium Specifications

The following description is taken from [14].

A.1 Key and IV Setup

First, the key and IV are used to initialize the internal states of the cipher which are then updated using Equation 6 but without generating the key-stream.

$$\begin{aligned}(s_1, s_2, s_3, ..., s_{93}) &\leftarrow (K_1, K_2, K_3, ..., K_{80}, 0, 0, ..., 0)\\ (s_{94}, s_{95}, s_{96}, ..., s_{177}) &\leftarrow (IV_1, IV_2, IV_3, ..., IV_{80}, 0, 0, ..., 0)\\ (s_{178}, s_{179}, s_{180}, ..., s_{288}) &\leftarrow (0, 0, ..., 1, 1, 1)\end{aligned} \tag{5}$$

$$\begin{aligned}&\text{for } i = 1 \text{ to } 4 \times 288 \text{ do}\\ &\quad t_1 \leftarrow s_{66} \oplus s_{91}.s_{92} \oplus s_{93} \oplus s_{171}\\ &\quad t_2 \leftarrow s_{162} \oplus s_{175}.s_{176} \oplus s_{177} \oplus s_{264}\\ &\quad t_3 \leftarrow s_{243} \oplus s_{286}.s_{287} \oplus s_{288} \oplus s_{69}\\ &\quad (s_1, s_2, s_3, ..., s_{93}) \leftarrow (t_3, s_1, s_2, ..., s_{92})\\ &\quad (s_{94}, s_{95}, s_{96}, ..., s_{177}) \leftarrow (t_1, s_{94}, s_{95}, ..., s_{176})\\ &\quad (s_{178}, s_{179}, s_{180}, ..., s_{288}) \leftarrow (t_2, s_{178}, s_{179}, ..., s_{287})\\ &\text{end for}\end{aligned} \tag{6}$$

A.2 KeyStream Generation

The state update function is given by Equation 7. The variable z_i denotes the key-stream bit while N denotes the number of bits to be generated and $N \leq 2^{64}$.

$$\begin{aligned}&\text{for } i = 1 \text{ to } N \text{ do}\\ &\quad t_1 \leftarrow s_{66} \oplus s_{93}\\ &\quad t_2 \leftarrow s_{162} \oplus s_{177}\\ &\quad t_3 \leftarrow s_{243} \oplus s_{288}\\ &\quad z_i \leftarrow t_1 \oplus t_2 \oplus t_3\\ &\quad t_1 \leftarrow t_1 \oplus s_{91}.s_{92} \oplus s_{171}\\ &\quad t_2 \leftarrow t_2 \oplus s_{175}.s_{176} \oplus s_{264}\\ &\quad t_3 \leftarrow t_3 \oplus s_{286}.s_{287} \oplus s_{69}\\ &\quad (s_1, s_2, s_3, s_4, ..., s_{93}) \leftarrow (t_3, s_1, s_2, s_3, s_4, ..., s_{92})\\ &\quad (s_{94}, s_{95}, s_{96}, s_{97}, ..., s_{177}) \leftarrow (t_1, s_{94}, s_{95}, s_{96}, s_{97}, ..., s_{176})\\ &\quad (s_{178}, s_{179}, s_{180}, s_{181}, ..., s_{288}) \leftarrow (t_2, s_{178}, s_{179}, s_{180}, s_{181}, ..., s_{287})\\ &\text{end for}\end{aligned} \tag{7}$$

Floating Fault Analysis of Trivium

Michal Hojsík[1] and Bohuslav Rudolf[2]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway
michal.hojsik@ii.uib.no
[2] National Security Authority, Na Popelce 2/16, 150 06 Prague 5, Czech Republic
b.rudolf@nbu.cz

Abstract. One of the eSTREAM final portfolio ciphers is the hardware-oriented stream cipher Trivium. It is based on 3 nonlinear feedback shift registers with a linear output function. Although Trivium has attached a lot of interest, it remains unbroken by passive attacks.

At FSE 2008 a differential fault analysis of Trivium was presented. It is based on the fact that one-bit fault induction reveals many polynomial equations among which a few are linear and a few quadratic in the inner state bits. The attack needs roughly 43 induced one-bit random faults and uses only linear and quadratic equations.

In this paper we present an improvement of this attack. It requires only 3.2 one-bit fault injections in average to recover the Trivium inner state (and consequently its key) while in the best case it succeeds after 2 fault injections. We termed this attack floating fault analysis since it exploits the floating model of the cipher. The use of this model leads to the transformation of many obtained high-degree equations into linear equations.

The presented work shows how a change of the cipher representation may result in much better attack.

Keywords: Trivium, stream cipher, differential fault analysis.

1 Introduction

The year 2008 is the last year of the European project ECRYPT. Within the project a search for new stream ciphers, eSTREAM project, took place. After 3 project phases within 4 years the final results were announced on Eurocrypt 2008. The eSTREAM committee has pointed out four ciphers within each of the two profiles (hardware/software oriented ciphers) and published them as the eSTREAM portfolio.

At the very beginning of eSTREAM, 34 ciphers were proposed and evaluated but only some have made their way up to the final phase or were even included to the final portfolio. Among these (portfolio) ciphers, the stream cipher Trivium is somehow special. First of all its design is indeed very simple. It brings us again to the question how simple can a secure cipher be. Secondly, Trivium is the fastest cipher among all proposals in many tested architectures (see e.g. [15]

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 239–250, 2008.

or [14] for more details) and although Trivium was largely analysed, it remains unbroken by passive attacks.

Earlier this year at FSE 2008, a differential fault attack on Trivium was presented [2]. It is based on the fact, that an injection of a one-bit fault (a bit flip) into a Trivium inner state reveals to an attacker a few linear and a few quadratic equations in the inner state bits. The attack requires roughly 43 fault injections at random positions and it assumes that all fault injections are performed into the same inner state. This can be achieved in the chosen-ciphertext attack scenario assuming that the initialisation vector is a part of the cipher input. In this case, an attacker will always use the same cipher input (cipher text and initialisation vector) which will lead to the same cipher inner state during the decryption. This would allow him to perform the fault injection to the same inner state during the deciphering process. Hence the attack can be described as chosen-ciphertext fault injection attack.

In this paper, we present an essential improvement of this attack and we describe a novel approach to the differential fault analysis of Trivium. Using this approach, we have obtained much better results in the sense of number of fault injections needed. Where the original attack needs 43 fault injections, our approach reveals the secret key after 3.2 fault injections in average (over $10,000$ experiments). In the best cases we have obtained the secret key after only 2 fault injections.

The main idea behind our attack is a simple way of transforming polynomial equations obtained during the attack into linear equations. This procedure is based on what we call *the floating description* of Trivium and on some fault propagation properties of Trivium's inner state evolution.

Formally, we significantly extend the number of variables by denoting every new inner state bit as a new variable, while not forgetting its connections given by the Trivium inner state evolution. Consequently many keystream difference equations become linear or have low degree. We use equations up to degree 4 and we linearise them subsequently using already revealed inner state bits, which are obtained by the use of Gauss-Jordan elimination for the linear equations.

In the beginning, we start off the equations system by the keystream equations and the inner state bit connections and afterwards we use fault injection to obtain more equations. After each fault injection and the following equations processing, we search the inner state bit sequences for a time t in which the Trivium inner state IS_t would contain a sufficient number of known bits. Since shifting of the Trivium inner state (seen as an interval of the inner state bit sequence) in time reminded us of floating of the inner state, we call this description *the floating description* and the attack *the floating fault analysis*. After finding such a time t, for which the inner state IS_t is known, we clock Trivium backwards until we obtain an initial state from which we can directly read used secret key and IV.

The rest of the paper is organised as follows. In Sec. 2 we review the related work, while Sec. 3 describes Trivium in the floating notation. In Sec. 4 we summarise attack prerequisites, followed by the attack description in Sec. 5. The paper is concluded by Sec. 6.

2 Related Work

As far as we known, the stream cipher Trivium has been, despite its simplicity, secure against all non side-channel attacks. At this point, we would like to recall some related work on Trivium as well as some results on stream cipher side-channel analysis.

New methods of solving systems of sparse quadratic equations are applied by Raddum to Trivium in [4]. The attack has complexity of $O(2^{162})$. Maximov and Biryukov in [5] tried to solve the system of equations given by Trivium keystream by guessing some inner state bits, which would result in a reduction of degrees of obtained equations. The complexity of their attack is $O(c \cdot 2^{83.5})$, where c is the time needed to solve a sparse system of linear equations. In [6], presented at SASC 2007, Babbage pointed out different possible approaches to the analysis of Trivium. Thuran and Kara presented a model of the Trivium initialisation part as an 8-round function in [7]. Their linear approximation of 2-round Trivium has bias 2^{-31}. Differential cryptanalysis is applied to the initialisation part of Trivium in [8]. Recently at SASC 2008, Fisher, Khazaei and Meier presented a new method for key recovery attacks [13]. They successfully applied their attack to Trivium with a reduced number of initialisation steps (672 out of the original 1152 steps). In the same paper they also provide evidence that the proposed attack is not applicable on Trivium with full initialisation.

Since the presented attack is a fault analysis attack, i.e. a side-channel attack, we would also like to mention some previous work on the side-channel analysis of stream ciphers. An overview on passive side-channel attacks on stream ciphers can be found in [9], while fault attacks on stream ciphers are described in [10]. Recently, authors of [12] have theoretically analysed ciphers in phase 3 of the eSTREAM project with respect to many types of side-channel analysis. Early this year at FSE 2008, authors of [2] have described differential fault analysis of Trivium. They have presented an attack that requires 280 keystream bits and in average 43 fault injections to reveal the used secret key. The authors have also designed a simple method for fault position determination. As already mentioned, our attack is an extension of their work which leads to a rapid reduction in the number of fault injections needed.

3 Trivium Description in the Floating Model

The stream cipher Trivium is a bit-oriented additive synchronous stream cipher with 80-bit secret key and 80-bit initialisation vector (IV). Trivium (as other stream ciphers) can be divided into two parts: the *initialisation algorithm*, which turns a secret key and an initialisation vector into the inner state of Trivium, and the *keystream generation algorithm*, which produces the keystream (one bit per step).

Necessarily condition for a simple description of our attack is the use of the following notation (first described in [3]). In this notation the cipher inner state registers are represented by the binary sequences they produce instead of describing them as finite length NLFSRs. We will refer to this notation as *the*

floating notation or *the floating description*. We have chosen this name since in this notation a Trivium inner state is an interval which is "floating" on the inner state bit sequences as time goes on.

The stream cipher Trivium consists of 3 non-linear feedback shift registers. The sequences produced by the first, the second and the third register will be denoted by $\{x_n\}$, $\{y_n\}$ and $\{z_n\}$ respectively. Since Trivium registers are of lengths 93, 84 and 111 these sequences will be indexed from -93, -84 and -111 onwards. So at time t the Trivium inner state is equal to

$$IS_t = (x_{t-1}, \ldots, x_{t-93}, y_{t-1}, \ldots, y_{t-84}, z_{t-1}, \ldots, z_{t-111}).$$

At the beginning of the initialisation part of Trivium, an 80-bit secret key $K = (k_1, \ldots, k_{80})$ is used to initialise the sequence $\{x_n\}_{n=-93}^{\infty}$ so that $x_{-i} = k_i$, $i = 1, \ldots, 80$ and an 80-bit initialisation vector $IV = (u_1, \ldots, u_{80})$ is used to initialise the sequence $\{y_n\}_{n=-84}^{\infty}$ so that $y_{-i} = u_i$, $i = 1, \ldots, 80$. Finally z_{-109}, z_{-110} and z_{-111} are set to one and all previously unset x_n, y_n and z_n are set to zero for all $n < 0$. We will refer to this state as to the *initial state*.

For $n \geq 0$, the following recursions are used to compute new inner state bits x_n, y_n and z_n:[1]

$$\begin{aligned} x_n &= x_{n-69} + z_{n-66} + z_{n-111} + z_{n-110}z_{n-109}, \\ y_n &= y_{n-78} + x_{n-66} + x_{n-93} + x_{n-92}x_{n-91}, \\ z_n &= z_{n-87} + y_{n-69} + y_{n-84} + y_{n-83}y_{n-82}. \end{aligned} \tag{1}$$

The only difference between the initialisation part and the keystream generation part of Trivium is the keystream production function present in the latter one, while the former consists of 1152 recursion steps. For the sake of notation simplicity, we will denote the keystream sequence by $\{o_n\}_{n=1152}^{\infty}$ (i.e. it will be indexed from 1152 onwards). It is computed as

$$o_n = x_{n-66} + x_{n-93} + y_{n-69} + y_{n-84} + z_{n-66} + z_{n-111}, \ n \geq 1152. \tag{2}$$

In the classical description, the cipher is represented by its 288 bit inner state $IS = (s_1, \ldots, s_{288})$ which is updated each cipher clock. This description can be found e.g. in the cipher specification [1] and we will refer to this as the *static notation* or *static description*.

In the rest of the paper we will start our investigations at some random, but fixed time t. For simplicity, we will denote this time as time $t = 0$. So the inner state at this time will be $IS_0 = (x_{-1}, \ldots, x_{-93}, y_{-1}, \ldots, y_{-84}, z_{-1}, \ldots, z_{-111})$. (This was denoted by $IS_{t_0} = (s_1, \ldots, s_{288})$ by authors of [2].) From now on, by x_{-1} we mean the value of x_n for $n = t - 1$ for the fixed but unknown time t and not the value k_1 as set in the initialisation part of Trivium.

[1] All additions in this paper are carried modulo 2.

4 Attack Prerequisites

The attack presented in this paper is a differential fault analysis attack, meaning that an attacker has to be able to (repeatedly) insert a fault into a cipher inner state IS_0. As recently mentioned in Sect. 3, this is an arbitrary but fixed Trivium inner state. In our case, inserting a fault means a bit flip on an unknown random position. The inner state after the fault injection will be denoted by IS'_0.

We also assume that an attacker is able to obtain N consecutive keystream bits after the fault injection, where in our implementation we have successfully used $N = 800$. This keystream will be referred to as the *faulty keystream* and will be denoted by $\{o'_n\}$. Further we assume that the attacker has also access to the first N consecutive bits of so called *proper* keystream, $\{o_n\}$, which is generated from the proper inner state IS_0.

The last assumption we make is that the attacker is able to repeat the fault injection to the same inner state IS_0 (each time inserting a fault into a new random position). This means, that the attacker has to run Trivium more than once with the same secret key and IV to reach the same inner state. This can be achieved in the chosen-ciphertext scenario, assuming that the initialisation vector is a part of the cipher input. In this case, the attacker will always use the same cipher input (cipher text and initialisation vector) which will lead to the same cipher inner state during the decryption. This would allow him to perform the fault injection to the same inner state during the deciphering process.

All together, these are the prerequisites of our attack:

1. Attacker is able to obtain the first N consecutive bits of the keystream $\{o_n\}$ produced out of the inner state IS_0.
2. Attacker is able to inject exactly one fault (a bit flip) into the inner state IS_0 into an unknown random position.
3. Attacker is able to obtain the first N consecutive bits of the keystream $\{o'_n\}$ produced out of the faulty inner state IS'_0.
4. Attacker is able to repeat the fault injection into a random position of IS_0 M times.

In our implementation of the attack, we have used $N = 800$ and the attack reveals the secret key in average after 3.2 fault injections. During our experiments (we have run the attack 10000 times) the attack succeeded with the probability 2% for $M = 2$, 78.5% for $M = 3$, 99.8% for $M = 4$ and for $M = 5$ the attack always revealed the secret key.

5 Floating Fault Analysis of Trivium

The authors of [2] described a differential fault analysis of Trivium based on the static model. In this paper we take the advantage of the floating model and we show that this model leads to much better results.

Before describing the attack itself, let us introduce some more notation. For each of the sequences $\{x_n\}$, $\{y_n\}$, $\{z_n\}$ and $\{o_n\}$ we define a *delta sequence*

$\{\delta x_n\}$, $\{\delta y_n\}$, $\{\delta z_n\}$ and $\{\delta o_n\}$ respectively as a difference between the proper sequence (a sequence without the fault injection) and the faulty sequence (the sequence after the fault injection). All faulty variables are marked by a prime. Using (1), (2) and the following equation describing the difference induced by multiplication

$$\delta(ab) = a'b' + ab = \delta a \cdot b + a \cdot \delta b + \delta a \cdot \delta b$$

we obtain the following equations describing the delta sequences:

$$\delta o_n = \delta x_{n-66} + \delta x_{n-93} + \delta y_{n-69} + \delta y_{n-84} + \delta z_{n-66} + \delta z_{n-111} \quad (3)$$

$$\delta x_n = \delta x_{n-69} + \delta z_{n-66} + \delta z_{n-111} + \delta z_{n-109} \cdot z_{n-110} + \quad (4)$$
$$+ z_{n-109} \cdot \delta z_{n-110} + \delta z_{n-109} \cdot \delta z_{n-110}$$

$$\delta y_n = \delta y_{n-78} + \delta x_{n-66} + \delta x_{n-93} + \delta x_{n-91} \cdot x_{n-92} + \quad (5)$$
$$+ x_{n-91} \cdot \delta x_{n-92} + \delta x_{n-91} \cdot \delta x_{n-92}$$

$$\delta z_n = \delta z_{n-87} + \delta y_{n-69} + \delta y_{n-84} + \delta y_{n-82} \cdot y_{n-83} + \quad (6)$$
$$+ y_{n-82} \cdot \delta y_{n-83} + \delta y_{n-82} \cdot \delta y_{n-83}.$$

According to the assumptions 1 and 3 in Sec. 4 an attacker is able to obtain the proper keystream $\{o_n\}$ as well as the faulty keystream $\{o'_n\}$. During the attack he will compute the delta keystream $\{\delta o_n\}$ for each fault injection and express its bits as expressions in variables $\{x_n\}, \{y_n\}$ and $\{z_n\}$ using equations (3), (4), (5) and (6). We will refer to this equations as the delta keystream equations. Afterwards he will try to solve these equations.

5.1 Faults in the Floating Model and the Corresponding Delta-Equations

We claim that using the floating notation, an attacker will obtain many more linear and quadratic equations than with the static notation, using the same attack model and having the same assumptions. Why is there a difference between the static model and the floating model? In fact there is no difference in the obtained equations - they are equivalent. The difference is in the representation, in our viewpoint. In the static model, at time t for some $t > 0$, the floating model variable x_t (or y_t or z_t equivalently) would be expressed as a polynomial in bits of the fixed initial state $IS_0 = (x_{-1}, \ldots, x_{-93}, y_{-1}, \ldots, y_{-84}, z_{-1}, \ldots, z_{-111}) = (s_1, \ldots, s_{288})$. Hence some of the obtained delta keystream equations which are in fact linear in the floating variables ($\{x_n\}$, $\{y_n\}$, $\{z_n\}$), are polynomial in the static initial state variables ($(s_1, \ldots, s_{288})$). In other words, δo_n is in the static model often a linear combination of nonlinear terms $\{\delta x_n\}$, $\{\delta y_n\}$, $\{\delta z_n\}$ which are expressed in the variables $(s_1, \ldots, s_{288})$. On the other hand, in the floating model we have clearly many more variables. Instead of 288 variables of the fixed inner state in the static model, we have $3N + 288$ variables in the floating model, which gives us 2688 variables for $N = 800$ as we have used in our implementation (288 initial state variables plus 3 new variables for each Trivium step). Since we do not forget about the connections between variables, the floating model can be seen as a useful extension of the static model.

Another important difference between these two models is the following: In the static model we fix the set of variables as bits of the inner state into which the fault injections are performed. So before the single injected fault spreads over the inner state and produces many linear equations, the expressions for the new inner state bits become polynomial and therefore in fact linear delta keystream equations contain high-degree polynomials. In the floating model we decide which inner state we would like to compute according to the obtained equations. So we wait until the fault spreads over the inner state and only then do we try to compute the inner state for the best possible time. More precisely, during the attack we try to determine values of all the variables $\{x_n\}$, $\{y_n\}$ and $\{z_n\}$ and we wait until there are enough known variables in an interval of a single inner state, regardless of the actual position of this state in time. For illustration, in our experiments during the attack we have obtained enough equations in the bits of inner state IS_t which is reached after approximately 300 steps of Trivium, i.e. we have obtained a Trivium inner state that appears roughly 300 steps after the fault injection. Afterwards we clock Trivium backwards until we reach a state IS_u similar to the initial state. Then the secret key equals to $(x_{u-1}, \ldots, x_{u-80})$ and $IV = (y_{u-1}, \ldots, y_{u-80})$.

In the floating model, the attack starts with the initial delta inner state $\{\delta x_n\}_{n=-93}^{-1}$, $\{\delta y_n\}_{n=-84}^{-1}$, $\{\delta z_n\}_{n=-111}^{-1}$, where all the bits are equal to zero except one (the bit where the fault injection occurred). So for example if the fault was injected into $x_i, i \in \{-93, \ldots, -1\}$ (a bit of the first register), the initial delta state would be

$$\begin{aligned} (\delta x_{-93}, \ldots, \delta x_{i-1}, \delta x_i, \delta x_{i+1}, \ldots, \delta x_{-1}) &= (0, \ldots, 0, 1, 0, \ldots, 0) \\ (\delta y_{-84}, \ldots, \delta y_{-1}) &= (0, \ldots, 0) \\ (\delta z_{-111}, \ldots, \delta z_{-1}) &= (0, \ldots, 0). \end{aligned}$$

Afterwards we inductively use equations (4), (5) and (6) to express $\{\delta x_n\}, \{\delta y_n\}$ and $\{\delta z_n\}$ for $n \geq 0$ as polynomials in variables $\{x_n\}$, $\{y_n\}$, $\{z_n\}$. These are afterwards used to represent bits of known sequence $\{\delta o_n\}$ as terms in the variables $\{x_n\}$, $\{y_n\}$, $\{z_n\}$, i.e. they are used to create the delta keystream equations.

Now let's look closer on these equations. When do they contain non-linear dependencies? From (3) it follows, that δo_n is a linear combination of values $\{\delta x_n\}, \{\delta y_n\}$ and $\{\delta z_n\}$. So it will contain non-linear terms if and only if any of these values will be non-linear. Let's examine e.g. the case of δx_n. For some $j > 0$, δx_j depends non-linearly on $\{x_n\}, \{y_n\}, \{z_n\}$ if and only if at least one of the following conditions is satisfied:

1. at least one of the values δx_{j-69}, δz_{j-66} or δz_{j-111} is non-linear in $\{x_n\}$, $\{y_n\}$, $\{z_n\}$,
2. δz_{j-109} or δz_{j-110} (or both of them) has degree at least 1 as a polynomial in $\{x_n\}$, $\{y_n\}$ and $\{z_n\}$.

Clearly, case 1 is only a transition of a non-linearity from other terms so a new non-linearity is created only in the case 2. As described in the above example, the starting delta inner state is all zero except one bit and consequently there are

Table 1. The average number of equations after different numbers of fault injections. FI stands for fault injection(s).

	# equations before/after eq. processing			
	degree 1	degree 2	degree 3	degree 4
Before FI	800/800	2400/2400	0	0
After 1 FI	825/992	2466/2350	35/2	57/1
After 2 FI	1017/1232	2419/2236	36/3	57/1
After 3 FI	1258/2396	2298/484	37/1	56/0
After 4 FI	2402/2685	498/6	8/0	12/0

only few non-linearity creations and transitions for small values of j. Although afterwards both cases occur more often, we are still able to obtain low degree equations thanks to the simple substitution we use to eliminate non-linearities (we substitute already known variables into the higher degree equations). Moreover many variables are known directly from the delta keystream equations, since they appear as a linear equation with only one term.

Tab. 1 shows the average number of obtained equations of degree up to four after one, two, tree and four fault injections. Each table entry contains two values, where the first one stands for the number of equations right after the fault injection while the second one stands for the number of equations after they were processed by methods described in Sect. 5.3. The average is computed over 10000 experiments.

5.2 Fault Position Determination

During the fault injection phase of the attack, a fault (a bit flip) is injected at a random position of the actual Trivium inner state. In order to be able to proceed with the attack, we need to know this position. Authors of [2] have described a very simple fault position determination technique. In this paper, we will use their technique as described in section 5.3. of [2]. Briefly, the technique is based on the fact that the distribution of ones in the delta keystream uniquely determines the position of the induced fault within the inner state. After the fault injection, an attacker computes the delta keystream $\{\delta o_n\}$ and determines the fault position (by a single table look-up) according to the distance between the first occurrences of non-zero bits in $\{\delta o_n\}$.

This technique is deterministic and leads to the right result for all possible fault positions, assuming that exactly one fault was injected. In the attack description, we will refer to this technique as the *fault_position_determination()*.

5.3 The Equations System and Its Processing

All the equations used during the attack are in variables $\{x_n\}$, $\{y_n\}$ and $\{z_n\}$. The corresponding system will contain equations from 3 different sources:

– The first set of equations is given by the proper keystream $\{o_n\}$ for $n = 0, \ldots, N-1$. In the floating description, all keystream equations are linear

and have the form of Eq. (2). These are the 800 linear equations in the first line of Tab. 1.
- The second set of equations comes from the Eq. (1) which describes the connections between the variables $\{x_n\}$, $\{y_n\}$ and $\{z_n\}$ and these are the 2400 quadratic equations in the first line of Tab. 1.
- The last set of equations are those coming from the fault injections and we term them delta keystream equations.

During the attack we try to solve the actual equation system by the use of two simple methods. The first one is the Gauss-Jordan elimination which we apply to the system of obtained linear equations and we term this procedure *Gauss()*. The second used technique is the substitution. By the term *Substitution()* we will denote a procedure which substitutes values of already known variables into the equations of higher degree (in our implementation we use equations up to degree 4). Since the number of revealed variables after each fault injection is fairly high, the substitution reduces the degree of many non-linear equations. Hereby we utilise the higher degree equations as a potential source of new linear equations and this is the only way how we use them.

5.4 The Attack Algorithm

The attack algorithm is described by Alg. 1. We have used equations up to degree 4 and $N = 800$. Afterwards the attack has revealed the secret key after 3.2 fault injections in average.

Since Eq. (3), (4), (5) and (6) are simple, we do not need any precomputations compared to the attack from [2]. After we determine the fault position at step 10 of Alg. 1 using the function *fault_position_determination()*, we compute symbolic delta equations (described by Eq. 3) for the actual fault position. Afterwards we insert these equations into our equations system using the values of the actual delta-keystream $\{\delta o_n\}$ computed at step 9 of Alg. 1 as the right-hand-side.

Table 2 shows the average maximal number of known variables in a single inner state IS_t (maximum over all possible inner states, i.e. over all positions of the inner state in time t) after 1, 2, 3, 4 and 5 fault injections as well as the estimated probability that the attack will succeed after a given number of fault injections. We see that the attack will succeed after 2 fault injections only with probability 2%, while after 4 fault injections the attack had revealed the secret key in 99.8% of all the cases.

Table 2. The average maximal number of known variables in a single inner state and the estimated probability of success after different numbers of fault injection. Average made over 10,000 experiments with random key, IV and fault position.

# fault injections	1	2	3	4	5
# known variables in a single state	30	70.4	245.5	287.5	288
probability of success	0	0.02	0.79	0.99	1

Algorithm 1. Floating Fault Attack

get Trivium in an unknown fixed inner state IS_0
obtain the first N consecutive bits of $\{o_n\}$, starting from inner state IS_0
insert keystream equations (Eq. 2 in Sect. 3) into the eq. system
insert connection equations (Eq. 1 in Sect. 3) into the eq. system
while for all t IS_t not known **do**
  reset Trivium to the state IS_0
  insert a fault into IS_0
  obtain the first N consecutive bits of the faulty keystream $\{o'_n\}$
  $\delta o_n \leftarrow o_n + o'_n,\ n = 0, \ldots, N-1$
  $e \leftarrow fault\_position\_determination(\{\delta o_n\})$
  compute delta keystream equations for e and insert them into the eq. system
  repeat
    do *Substitution()*
  until it keeps changing the equation system
  do *Gauss()*
  if new variables obtained by *Gauss()* **then**
    goto 12
  end if
end while
// at this point we have a known inner state IS_t
run Trivium backwards starting with IS_t until an inner state similar to the initial state is reached
read the secret key from the reached initial state

5.5 Implementation and Complexity

In the floating model, the number of variables depends on the number of Trivium steps performed. In our implementation we have used $N = 800$ which gives us 2400+288 (initial state) = 2688 variables. Clearly, all obtained equations are very sparse. However, we have used the straight-forward implementation of the Gauss-Jordan elimination. Due to the implementation complexity, we use only equations up to degree 4.

According to the nature of the proposed algorithm, it is indeed very hard to do any theoretical complexity analysis. Hence we have used the average number of procedure calls to do the following complexity estimate. The most complex procedure used in the attack algorithm is the Gauss-Jordan elimination. We suppose that for a $m \times n$ matrix it has the complexity of $O(nm^2)$ operations. Since the number of columns equals 2688 and if we suppose that the number of linear equations is in average lower than 1024, we gain the running time of roughly $2^{11.4} \cdot 2^{20} = 2^{31.4}$ simple operations per an average Gauss-Jordan elimination used in the attack. We have performed 10000 runs of the attack and in the average one attack makes $80 \doteq 2^{6.3}$ calls of the *Gauss()* procedure. All together we can retrieve the secret key in the time of $2^{31.4} \cdot 2^{6.3} = 2^{37.7}$ simple operations. Since the average running time of the attack was only 40.3 seconds on our desktop computer with AMD Athlon 64 X2 Dual-Core 3800+ processor, the complexity estimate can be seen as the upper limit.

5.6 Effort for Further Improvements

During our experiments, we have tried to stop the attack at the point when we have found an inner state IS_u (for any u) with at least $288 - G$ known bits for a given G. Then we could use the brute force to determine the rest of the inner state bits. We have tried $G = 20$ and $G = 30$ but in the average over 10000 experiments, these attacks led to very similar results.

Another way to improve our attack could be the trick described in [5]. Namely, we tried to do an educated guess of up to 30 variables in the following way: after the second fault injection when we have already gathered many high-degree equations, we have guessed those variables which maximised the number of higher-degree terms eliminated by this guess. In this way we obtain not only 30 new known variables, but also more low-degree equations. However experiments have shown that the contribution of this smart guess to the final number of fault injections needed is in fact neglectable (even if applied on different places of the attack algorithm).

6 Conclusion

The paper describes a new approach to the fault analysis of Trivium. We have taken advantage of the so called floating model and have proposed and implemented an attack which can reveal the secret key after 3.2 fault injections in average using 800 keystream bits. In the best cases, we were able to reconstruct the secret key only after 2 fault injections.

Our results show how important it is for cryptanalysis to choose the right model for cipher representation.

References

1. De Cannière, C., Preneel, B.: Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/30 (2005), http://www.ecrypt.eu.org/stream
2. Hojsik, M., Rudolf, B.: Differential Fault Analysis of Trivium. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 158–172. Springer, Heidelberg (2008)
3. ECRYPT discussion forum, http://www.ecrypt.eu.org/stream/phorum/read.php?1,448
4. Raddum, H.: Cryptanalytic Results on Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/039 (2006), http://www.ecrypt.eu.org/stream
5. Maximov, A., Biryukov, A.: Two Trivial Attacks on Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/006 (2007), http://www.ecrypt.eu.org/stream
6. Babbage, S.: Some Thoughts on Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/007 (2007), http://www.ecrypt.eu.org/stream
7. Turan, M.S., Kara, O.: Linear Approximations for 2-round Trivium. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/008 (2007), http://www.ecrypt.eu.org/stream

8. Biham, E., Dunkelman, O.: Differential Cryptanalysis in Stream Ciphers. COSIC internal report (2007)
9. Rechberger, Ch., Oswald, E.: Stream Ciphers and Side-Channel Analysis. In: SASC 2004 - The State of the Art of Stream Ciphers, Workshop Record, pp. 320–326 (2004), http://www.ecrypt.eu.org/stream
10. Hoch, J.J., Shamir, A.: Fault Analysis of Stream Ciphers. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 240–253. Springer, Heidelberg (2004)
11. Biham, E., Granboulan, L., Nguyen, P.: Impossible Fault Analysis of RC4 and Differential Fault Analysis of RC4. In: SASC 2004 - The State of the Art of Stream Ciphers, Workshop Record, pp. 147–155 (2004), http://www.ecrypt.eu.org/stream
12. Gierlichs, B., et al.: Susceptibility of eSTREAM Candidates towards Side Channel Analysis. In: SASC 2008 - The State of the Art of Stream Ciphers, Workshop Record, pp. 123–150 (2008), http://www.ecrypt.eu.org/stream
13. Fisher, S., Khazaei, S., Meier, W.: Chosen IV Statistical Analysis for key Recovery Attacks on Stream Cipher. In: SASC 2008 - The State of the Art of Stream Ciphers, Workshop Record, pp. 33–41 (2008), http://www.ecrypt.eu.org/stream
14. Hwang, D., et al.: Comparison of FPGA - Targeted Hardware Implementations of eSTREAM Stream Cipher Candidates. In: SASC 2008 - The State of the Art of Stream Ciphers, Workshop Record, pp. 151–162 (2008), http://www.ecrypt.eu.org/stream
15. Good, T., Benaissa, M.: Hardware Performance of eSTREAM Phase-III Stream Cipher Candidates. In: SASC 2008 - The State of the Art of Stream Ciphers, Workshop Record, pp. 163–174 (2008), http://www.ecrypt.eu.org/stream

Algebraic Methods in Side-Channel Collision Attacks and Practical Collision Detection

Andrey Bogdanov[1], Ilya Kizhvatov[2], and Andrey Pyshkin[3]

[1] Horst Görtz Institute for Information Security
Ruhr-University Bochum, Germany
abogdanov@crypto.rub.de
[2] University of Luxembourg, Luxembourg
ilya.kizhvatov@uni.lu
[3] Technical University Darmstadt, Germany
pychkine@cdc.informatik.tu-darmstadt.de

Abstract. This paper presents algebraic collision attacks, a new powerful cryptanalytic method based on side-channel leakage which allows for low measurement counts needed for a successful key recovery in case of AES. As opposed to many other side-channel attacks, these techniques are essentially based on the internal structure of the attacked cryptographic algorithm, namely, on the algebraic properties of AES. Moreover, we derived the probability distributions of Euclidean distance for collisions and non-collisions. On this basis, a statistical framework for finding the instances of side-channel traces leaking most key information in collision attacks is proposed.

Additionally to these theoretical findings, the paper also contains a practical evaluation of these side-channel collision attacks for a real-world microcontroller platform similar to many smart card ICs. To our best knowledge, this is the first real-world study of collision attacks based on generalized internal collisions. We also combined our methods with ternary voting [1] which is a recent multiple-differential collision detection technique using profiling, where neither plaintexts, ciphertexts nor keys have to be known in the profiling stage.

Keywords: Side-channel attacks, collision attacks, algebraic cryptanalysis, multiple-differential collision attacks, ternary voting, AES, DPA.

1 Introduction

Motivation. The motivation of this paper is to develop a framework minimizing the number of online measurements needed for a successful key recovery in a real-world noisy environment. This is to a certain extent equivalent to extracting a maximum amount of key information from the given side-channel signal, which is the central question of side-channel cryptanalysis. In practice, this setting is important in such cases where the attacker has very restricted access to the device due to organizational policies or where only few cryptographic operations with the same key are allowed, which is used as a side-channel countermeasure

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 251–265, 2008.

in some real-world systems. An independent line of motivation we pursue is to come up with an efficient and practical alternative to such well-known side-channel techniques as differential power analysis (DPA) [2] and template attacks [3], [4] which would be free of their main natural limitations: Dependency on a certain leakage model for DPA and the necessity of thoroughly characterizing the attacked device for template attacks.

Side-channel *collision attacks* are a well-suited base for the solution of these problems due to the inherently low numbers of needed measurements, the absence of any concrete leakage model, and the possibility to build collision templates without detailed knowledge of the target.

Collision attacks. Basic side-channel collision attacks [5] were improved in [6] by introducing the notion of *generalized collisions* that occur if two S-boxes at some arbitrary positions of some arbitrary rounds process an equal byte value within several runs. However, [6] treats only the *linear collisions* of AES which are generalized collisions that occur in the first AES round only. Moreover, the results in [6] as well as those in [5] assume that the collision detection is absolutely reliable, while there are significant error probabilities in real-world scenarios. Though this problem was approached in [1] by introducing multiple-differential collision detection (binary and ternary voting), a sound real-world evaluation of these methods is still lacking.

Our contribution. Additionally to linear collisions, we consider *nonlinear collisions* that are defined as generalized collisions comprising several rounds. They deliver extra information contained in further AES rounds. Each such collision can be considered as a nonlinear equation over a finite field. The set of all detected collisions corresponds to a system of nonlinear equations with respect to the key, which can be solved using techniques closely related to the algebraic cryptanalysis of AES with a reduced number of rounds which are referred to as *algebraic collision-based key recovery*.

For collision detection, the Euclidean distance is used. We obtain probability distributions of this statistic in the univariate Gaussian noise model. We show that for large numbers of points in the side-channel trace these two Euclidean distances can be approximated by normal distributions with different parameters. This allows us to define a statistical metric for the time instants of the trace leaking most information for collision detection. It turns out that these points are quite different from those leaking key data in standard DPA.

Combining these improvements, we achieve a considerable reduction of the number of online measurements needed for a successful key recovery. We implemented the attacks for an Atmel AVR ATMega16 microcontroller. The practical results can be found in Table 1. In a version of the attack, we additionally use collisions from ternary voting, a multiple-differential collision detection technique from [1]. Neither plaintexts, ciphertexts nor keys have to be known in the profiling stage.

This indicates that the algebraic collision attacks on AES without profiling are superior to standard CPA in terms of number of measurements needed.

Table 1. Summary of results: Hamming-distance based CPA, basic collision attack (on our ATmega16 AES implementation) without profiling and stochastic methods with profiling (on an ATM163 AES implementation [4]) vs. collision attacks based on FL-collisions with and without profiling for $C_{\text{offline}} \leq 2^{40}$ (P – success probability, C_{online} – number of online measurements, $C_{\text{profiling}}$ – number of profiling measurements, C_{offline} – number of offline operations for key recovery)

	P	C_{online}	$C_{\text{profiling}}$
HD-based CPA	0.8	61	0
Basic collision attack [5]	0.85	300	0
Stochastic methods [4] for ATM163	0.82	10	200
FL-collisions, this paper	0.76	16	0
FL-collisions, this paper	0.72	12	625

Rather surprisingly, the efficiency of our collision techniques *without profiling* is comparable to the stochastic methods [4] *with profiling* (one of best known template-based attacks) for low numbers of profiling curves. Moreover, if profiling is allowed for collision attacks, the number of online measurements can be further reduced. Note that all the implemented collision techniques (both with and without profiling) do use the knowledge of the time instances leaking most information.

2 Preliminaries

2.1 Basic Notation

All collision attacks have two stages: an *online stage*, where measurements on the target device implementing the attacked cryptographic algorithm are performed, and an *offline stage*, where the cryptographic key is obtained from the traces acquired in the online stage. Additionally, a collision attack can be enhanced to have a *profiling stage*, where some profiling traces are obtained from some implementation of the attacked cryptographic algorithm.

In this paper we perform our collision attacks at the example of AES. We use the following notation to represent its variables. $K = \{k_j\}_{j=1}^{16}$, $k_j \in \text{GF}(2^8)$ is the 16-byte user-supplied key (the initial AES subkey). $X = \{x_j\}_{j=1}^{16}$, $Y = \{y_j\}_{j=1}^{16}$ and $Z = \{z_j\}_{j=1}^{16}$, $x_j, y_j, z_j \in \text{GF}(2^8)$ are the first, next to the last and last 16-byte AES subkeys, respectively. AES plaintexts are denoted by $P^i = \{p_j^i\}_{j=1}^{16}$, $p_j^i \in \text{GF}(2^8)$ and ciphertexts by $C^i = \{c_j^i\}_{j=1}^{16}$, $c_j^i \in GF(2^8)$, where $i = 1, 2, \ldots$ is the number of AES execution.

Collision-based key recovery methods for AES are mainly parametrized by the number γ of random plaintexts and/or ciphertexts needed to obtain the cryptographic key with success probability P. In our collision attacks, γ can be chosen between 4 and 20 in the majority of cases. We are interested in success probabilities $P \geq 0.5$.

2.2 Linear Collision-Based Key Recovery

Given a linear collision (within the first round of AES), one obtains a linear equation with respect to the key over GF(2^8) of the form

$$S(p_{j_1}^{i_1} \oplus k_{j_1}) = S(p_{j_2}^{i_2} \oplus k_{j_2}), \text{ or } k_{j_1} \oplus k_{j_2} = p_{j_1}^{i_1} \oplus p_{j_2}^{i_2} = \Delta_{j_1,j_2} \text{ for } j_1 \neq j_2.$$

In the example of Figure 1, one has the following equation: $k_4 \oplus k_{11} = p_4^1 \oplus p_{11}^2 = \Delta_{4,11}$. S-boxes where collisions occurred (*active* S-boxes) are marked by the numbers of collisions they account for. The input byte p_j^i is characterized by its position $j \in \{1, \ldots, 16\}$ within the plaintext block and the number $i = 1, 2, \ldots$ of the plaintext block it belongs to.

If D collisions have been detected, they can be interpreted as a system of linear binomial equations over GF(2^8):

$$\begin{cases} k_{j_1} \oplus k_{j_2} & = \Delta_{j_1,j_2} \\ \cdots & \\ k_{j_{2D-1}} \oplus k_{j_{2D}} & = \Delta_{j_{2D-1},j_{2D}} \end{cases}$$

This system cannot have the full rank due to the binomial form of its equations. Moreover, for small numbers of inputs to AES the system is not connected and it can be divided into a set of h_0 smaller independent (with disjunct variables) connected subsystems with respect to the parts of the key. Each subsystem has one free variable. Let h_1 be the number of all missing variables, and $h = h_0 + h_1$. Then the system has 2^{8h} solutions. That is, $C_{\text{offline}} = 2^{8h}$ guesses have to be performed, which is the offline complexity of the attack. Each key hypothesis is then tested using a known plaintext-ciphertext pair to rule out incorrect candidates. C_{offline} quickly becomes feasible as the number of distinct inputs grows. The probability that $C_{\text{offline}} \leq 2^{40}$ ($h \leq 5$) is about 0.85 for $\gamma = 6$, if all collisions are detected. Note that the question of reliable collision detection was not treated in [6].

2.3 Direct Binary Comparison Using Side-Channel Signal

There are ways of deciding if two S-boxes accept equal inputs using side-channel information obtained from the *implementation* of the attacked cryptographic

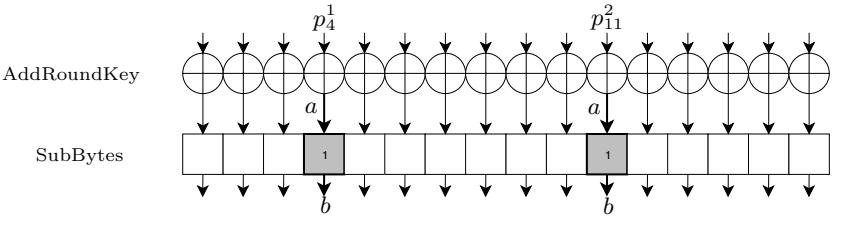


Fig. 1. A linear collision for some pair of runs

algorithm. For instance, typical side channels are the power consumption of devices as well as their electromagnetic radiation, which manifest some data and key dependency in many cases.

Given two side-channel traces $\tau_1 = (\tau_{1,1}, \ldots, \tau_{1,l}) \in \mathbb{R}^l$, $\tau_2 = (\tau_{2,1}, \ldots, \tau_{2,l}) \in \mathbb{R}^l$, respectively corresponding to some pair of S-box executions with inputs a_1 and a_2, it has to be decided whether $a_1 = a_2$ for collision detection. In this paper we use the Euclidean distance based binary comparison test T (as in [5] and [1]) for this purpose:

$$T(\tau_1, \tau_2) = \begin{cases} 0 \text{ (no collision)}, & \text{if } H(\tau_1, \tau_2) < W \\ 1 \text{ (collision)}, & \text{if } H(\tau_1, \tau_2) \geq W, \end{cases}$$

where W is a decision threshold and H is the following statistic:

$$H(\tau_1, \tau_2) = 1/\sum_{j=1}^{l} (\tau_{1,j} - \tau_{2,j})^2.$$

Test T is characterized by the following type I and II error probabilities[1]:

$$\alpha = \Pr\{T(\tau_1, \tau_2) = 0 | a_1 = a_2\}, \beta = \Pr\{T(\tau_1, \tau_2) = 1 | a_1 \neq a_2\}.$$

2.4 Ternary Voting: Indirect Comparison of Traces Using Profiling

As already mentioned in Subsection 2.1, collision detection can be made more efficient if profiling is allowed. One approach to such template-based collision detection is the ternary voting proposed in [1]: Test T on two *target traces* τ_1 and τ_2 for inputs a_1 and a_2 can be amplified by using further N *reference traces* $\{\pi_i\}_{i=1}^N$, $\pi_i = (\pi_{i,1}, \ldots, \pi_{i,l}) \in \mathbb{R}^l$, which correspond to some random unknown inputs $b_i \in \mathrm{GF}(2^8)$ and have been acquired on a similar implementation prior to the online stage. The main idea of ternary voting is to indirectly compare τ_1 and τ_2 through the pool of reference traces $\{\pi_i\}_{i=1}^N$. The ternary voting test can be defined as follows:

$$V(\tau_1, \tau_2) = \begin{cases} 0 \text{ (no collision)}, & \text{if } G(\tau_1, \tau_2) < U \\ 1 \text{ (collision)}, & \text{if } G(\tau_1, \tau_2) \geq U, \end{cases}$$

where $G(\tau_1, \tau_2) = \sum_{i=1}^N F(\tau_1, \tau_2, \pi_i)$ with $F(\tau_1, \tau_2, \pi_i) = T(\tau_1, \pi_i) \cdot T(\tau_2, \pi_i)$, U is some decision threshold, and T is the binary comparison test as defined in Subsection 2.3. The key observation is that the distributions of $G(\tau_1, \tau_2)$ for $a_1 = a_2$ and for $a_1 \neq a_2$ will be different. Typically, for sufficiently large N's $G(\tau_1, \tau_2)$ will be higher for $a_1 = a_2$ than for $a_1 \neq a_2$. To decide if there has been a collision, the attacker needs to statistically distinguish between these two distributions. We use ternary voting to amplify direct binary comparison, combining the sets of collisions detected by both methods, see Section 5.2.

[1] Note that α and β strongly depend on the statistical properties of the traces (among many other factors, on the noise amplitude) and the choice of W.

3 Algebraic Collision-Based Key Recovery

In this section we identify (Subsection 3.1) types of nonlinear generalized collisions enabling efficient algebraic representation that give rise to efficient key recovery. Then the corresponding systems of nonlinear equations are constructed (Subsection 3.2) and solved (Subsection 3.3). We first assume that all collisions are detected correctly. We deal with collision detection errors in Subsection 3.4 and Section 4.

3.1 Nonlinear Collisions

FS-Collisions. Generalized collisions in the first two AES rounds occurring between bytes of the first two rounds are called *FS-collisions*. If input bytes $a_{j_1}^{i_1}$ and $a_{j_2}^{i_2}$ of two S-boxes collide, one has the simple linear equation over $GF(2^8)$:

$$a_{j_1}^{i_1} \oplus a_{j_2}^{i_2} = 0.$$

If a_j^i lies in the S-box layer of the first round, then $\alpha_j^i = k_j \oplus p_j^i$, for some i, j. Otherwise, one has

$$a_j^i = x_j \oplus m_{(j-1)\ div\ 4} \cdot b^i_{(j-1)\ mod\ 4+1} \oplus m_{j\ div\ 4} \cdot b^i_{j\ mod\ 4+5} \oplus$$
$$m_{(j+1)\ div\ 4} \cdot b^i_{(j+1)\ mod\ 4+9} \oplus m_{(j+2)\ div\ 4} \cdot b^i_{(j+2)\ mod\ 4+13},$$

where $m = (m_0, m_1, m_2, m_3) = (02, 03, 01, 01)$[2] and $b_j^i = S(k_j \oplus p_j^i)$.

We distinguish between the following three types of FS-collisions: linear collisions in the first round, nonlinear collisions between the first two rounds, and nonlinear collisions within the second round. These three collision types are illustrated in Figure 2. Collision 1 occurs between two bytes of the first round, linearly binding k_1 and k_{13}. Collision 2 occurs between the S-box number 7 of the second round and the S-box number 1 of the first round. It binds 6 key bytes: k_1, k_3, k_8, k_9, k_{14}, and k_7. Collision 3 algebraically connects two MixColumn expressions on 8 key bytes after the S-box layer with two bytes of the second subkey in a linear manner. The algebraic expressions in this example are the following:

$$1: \ k_1 \oplus p_1^1 = k_{13} \oplus p_{13}^1$$
$$2: \ k_1 \oplus p_1^2 = S^{-1}(s_7^2) =$$
$$x_7 \oplus 01 \cdot S(k_2 \oplus p_2^2) \oplus 02 \cdot S(k_8 \oplus p_8^2) \oplus 03 \cdot S(k_9 \oplus p_9^2) \oplus 01 \cdot S(k_{14} \oplus p_{14}^2)$$
$$3: \ s_7^3 = s_{16}^3,$$
$$k_7 \oplus 01 \cdot S(k_3 \oplus p_3^3) \oplus 02 \cdot S(k_8 \oplus p_8^3) \oplus 03 \cdot S(k_9 \oplus p_9^3) \oplus 01 \cdot S(k_{14} \oplus p_{14}^3) =$$
$$x_{16} \oplus 03 \cdot S(k_4 \oplus p_4^3) \oplus 01 \cdot S(k_5 \oplus p_5^3) \oplus 01 \cdot S(k_{10} \oplus p_{10}^3) \oplus 02 \cdot S(k_{15} \oplus p_{15}^3)$$

Note that there are also mirrored collisions occurring between the S-boxes of the last round (number 10) and the round next to the last one (number 9). Such collisions are called *LN-collisions*.

[2] Here and below any byte $uv = u \cdot 16 + v = \sum_{i=0}^{7} d_i \cdot 2^i$ is interpreted as the element of $GF(2^8) = GF(2)[\omega]$ using a polynomial representation $\sum_{i=0}^{7} d_i \cdot \omega^i$, where $\omega^8 + \omega^4 + \omega^3 + \omega + 1 = 0$ holds.

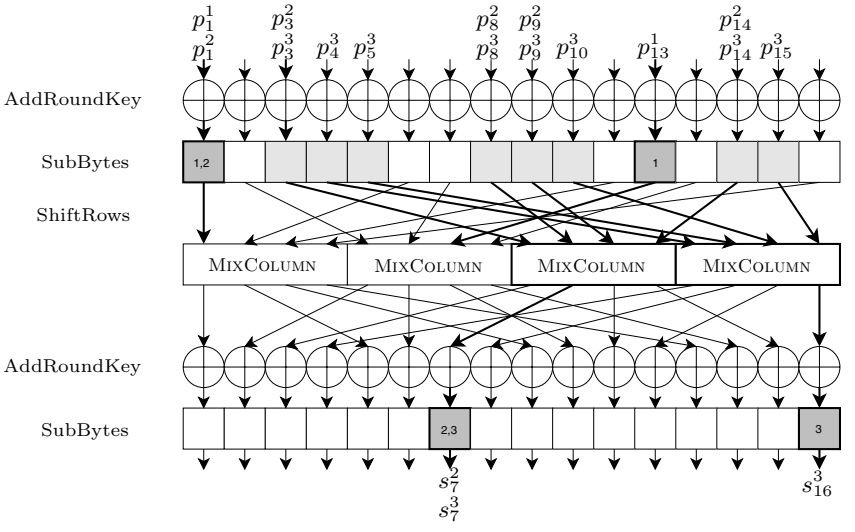


Fig. 2. FS-collisions

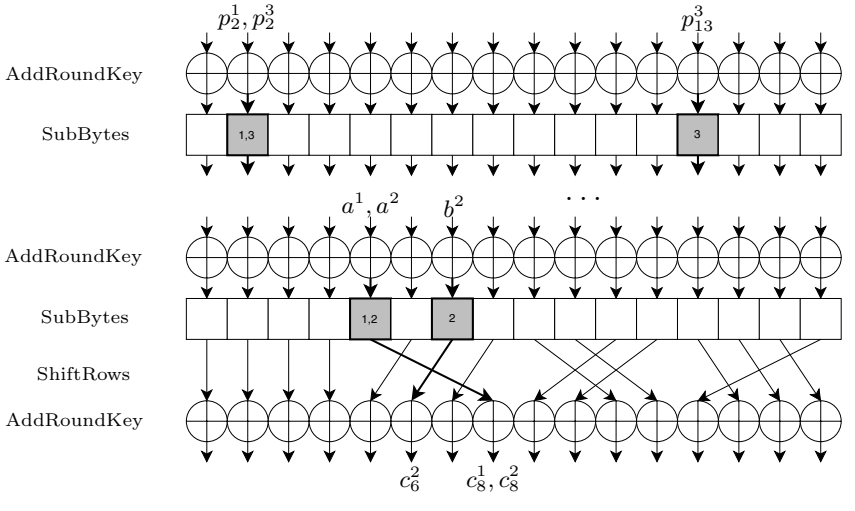


Fig. 3. FL-collisions

FL-Collisions. *FL-collisions* are generalized collisions between bytes of the first and last rounds. If plaintexts as well as ciphertexts are known, an FL-collision leads to a simple nonlinear equation. Linear collisions within the first round as well as those within the last round can be additionally used.

Figure 3 illustrates these three types of collisions. Collision 1 occurs between the 2nd byte of the first round and the 5th byte of the last round for some input and output with p_2^1 and c_8^1 (note that the bytes do not have to belong to the same input/output pair). Input $y_2 \oplus a^1$ to S-box 5 in the last round can be expressed as

Table 2. Solving equation systems for FS-collisions over $GF(2)$

Inputs, γ	5	5	4	4
Success probability π	0.425	0.932	0.042	0.397
Offline complexity (time), s	142.8	7235.8	71.5	6456.0
Memory limit, MB	500	500	500	500
Number of variables	896	896	768	768
Linear/quadratic equations	$96+8D/3276$	$96+8D/3276$	$96+8D/2652$	$96+8D/2652$

$S^{-1}(z_8 \oplus c_8^1)$ using the corresponding ciphertext and last subkey bytes. Collision 2 of Figure 3 is a linear collision within the last AES round. Collision 3 is a standard linear collision within the first AES round. The following equations result from these collisions:

$$\begin{aligned} 1:&\ k_2 \oplus p_2^1 = y_5 \oplus a^1 = S^{-1}(z_8 \oplus c_8^1), S(k_2 \oplus p_2^1) = z_8 \oplus c_8^1 \\ 2:&\ y_5 \oplus a^2 = y_7 \oplus b^2,\ z_8 \oplus c_8^2 = z_6 \oplus c_6^2 \\ 3:&\ k_2 \oplus p_2^3 = k_{13} \oplus p_{13}^3. \end{aligned}$$

3.2 Constructing Systems of Equations for FS- and FL-Collisions

Equations for FS-collisions. The application of the Faugère F4 algorithm to a system of equations constructed in Subsection 3.1 for FS-collisions gives results that are superior to the linear collision attack and are summarized in Table 2.

The system of nonlinear equations is considered over GF(2). For γ inputs there are 128 variables of the first subkey K, 128 variables of the second subkey X and $128 \cdot \gamma$ intermediate variables for the output bits of the first round S-box layer. The collision-independent part of the system consists of S-box equations for the first round and linear equations for the key schedule. Since the AES S-box can be implicitly expressed as 39 equations of degree 2 [7], we have $39 \cdot 16 \cdot \gamma$ quadratic equations over GF(2) connecting the inputs and outputs of the first round S-boxes, and $4 \cdot 39 = 156$ quadratic and $12 \cdot 8 = 96$ linear equations connecting K and X using the key schedule relations. Each of the three types of FS-collisions adds 8 linear equations to the system, resulting in $8 \cdot D$ equations if D collisions occurred.

Equations for FL-collisions. FL-collisions can be also obviously expressed as a system of quadratic equations over $GF(2)$. Now we show how to derive a system of quadratic equations over $GF(2^8)$ for these collisions. One way is to use the BES expression [7]. However one would have 8 variables per one key byte in this case. We describe a simpler system, which has only 32 variables.

It is clear that linear collisions in the first or the last round can be interpreted as linear equations over $GF(2^8)$. Let us consider a nonlinear FL-collision of type 1 (see example above). Its algebraic expression is given by $S(k_{j_1} \oplus p_{j_1}^{i_1}) = z_{j_2} \oplus c_{j_2}^{i_2}$ for some $j_1, j_2 \in \{1, \ldots, 16\}$, $i_1, i_2 = 1, 2, \ldots$ Recall that the AES S-box is the composition of the multiplicative inverse in the finite field $GF(2^8)$, the $GF(2)$-linear mapping, and the XOR-addition of the constant 63. The $GF(2)$-linear

mapping is invertible, and its inverse is given by the following polynomial over $GF(2^8)$:

$$f(x) = 6e \cdot x^{2^7} + db \cdot x^{2^6} + 59 \cdot x^{2^5} + 78 \cdot x^{2^4} + 5a \cdot x^{2^3} + 7f \cdot x^{2^2} + fe \cdot x^2 + 05 \cdot x.$$

Hence we have

$$(k_{j_1} \oplus p_{j_1}^{i_1})^{-1} = f(z_{j_2}) \oplus c_{j_2}^{i_2} \oplus 63) = f(z_{j_2}) \oplus f(c_{j_2}^{i_2} \oplus 63).$$

If we replace $f(z_{j_2})$ by a new variable u_{j_2}, we obtain the quadratic equation

$$(k_{j_1} \oplus p_{j_1}^{i_1})(u_{j_2} \oplus f(c_{j_2}^{i_2} \oplus 63)) = 1,$$

which holds with probability $\frac{255}{256}$. The following proposition follows:

Proposition 1. *Solutions to the equation* $S(k_{j_1} \oplus p_{j_1}^{i_1}) = z_{j_2} \oplus c_{j_2}^{i_2}$ *coincides with solutions to the equation*

$$(k_{j_1} \oplus p_{j_1}^{i_1})(u_{j_2} \oplus f(c_{j_2}^{i_2} \oplus 63)) = 1$$

under the change of variables $u_{j_2} = f(z_{j_2})$ *with a probability of* $\frac{255}{256}$.

Moreover, if $z_{j_2} \oplus z_{j_3} = \Delta_{j_2,j_3} = c_{j_2}^{i_2} \oplus c_{j_3}^{i_3}$, then we have

$$f(z_{j_2}) \oplus f(z_{j_3}) = u_{j_2} \oplus u_{j_3} = f(\Delta_{j_2,j_3}).$$

Thus, we derive for FL-collisions the system $\mathbb{S}$ of quadratic equations over $GF(2^8)$ in 32 variables $\mathcal{K} = \{k_j, u_j\}_{1 \le j \le 16}$. Furthermore, each equation of the resulting system has only two variables. We call such equations *binomial*.

We say that a subset of variables $\mathcal{K}' \subset \mathcal{K}$ is connected, if for any non-trivial partition of $\mathcal{K}' = A \cup B$ there is an equation in $\mathbb{S}$ in two variable $v \in A$ and $w \in B$. Thus $\mathcal{K}$ can be devided into disjoint subsets $\mathcal{K}_i$ with respect to $\mathbb{S}$, where $\mathcal{K}_i$ is either connected or singleton. Each $\mathcal{K}_i$ corresponds to an unique subsystem $\mathbb{S}_i$, and we call $(\mathcal{K}_i, \mathbb{S}_i)$ a chain.

3.3 Solving Systems for FS- and FL-Collisions

Solving equations for FS-collisions. The system of nonlinear equations for FS-collisions is solved in the following way. First the system is passed to the F4 algorithm without modifications. If it is not solvable, one guesses the largest connected linear component as in linear collision-based recovery (that is, 8 bits per connected component, see Subsection 2.2), adds the corresponding linear equation to the system and tries to solve the system again. The memory limit for the Magma program was set to 500 MB. It can be seen from Table 2 that for 5 inputs most ($> 93\%$) instances of the FS-system can be solved within several hours on a PC. For 4 inputs, less systems are solvable (about 40%) within approx. 2 hours on a standard PC under Linux.

Solving equations for FL-collisions. FL-collisions lead, as a rule, to better results. Each equation binds only two $GF(2^8)$-variables, since one deals with binomial equations introduced in Subsection 3.2 for FL-collisions. There are 32 variables $\mathcal{K}$ over $GF(2^8)$. The algebraic relations on these variables are much simpler, since one has both plaintext and ciphertext bytes (more information related to the detected collisions).

Moreover, for the system we have a set of independent chains. Let $(\mathcal{K}_i, \mathbb{S}_i)$ be a chain, and $v \in \mathcal{K}'$. Since $\mathcal{K}'$ is connected, there exists a relation between v and any other variable of $\mathcal{K}'$. It is not hard to prove that this relation can be expressed as linear or quadratic equation in two variables.Further, some chain can have a non-linear equation such that the corresponding variables still be connected also without this equation. In this case we call this chain a cycle. For any cycle the system has at most two solutions. Thus, there are often nonlinear subsystems solvable independently (see the example below).

On average, there are about 1.02 independently solvable subsystems covering 30.08 out of 32 $GF(2^8)$-variables for $\gamma = 5$ inputs and 0.99 cycles covering 20.08 out of 32 $GF(2^8)$-variables for $\gamma = 4$ inputs. Statistically there are 43.58 collisions for $\gamma = 5$ inputs and 29.66 collisions for $\gamma = 4$ inputs.

Table 3 contains the results for applying the F4 algorithm to FL-systems of nonlinear equations averaged over 10000 samples. After resolving the nonlinear subsystems using F4, as for FS-collisions, we guess variables defining the remaining bytes in a way similar to the linear key-recovery. With $C_{\text{offline}} \leq 2^{40}$, practically all FL-systems are solvable for 5 inputs, an FL-system being solvable with probability 0.85 for $\gamma = 4$ inputs.

3.4 Key Recovery Robust to Type I Collision Detection Errors

Both algebraic and linear collision-based key recovery methods can be made tolerant to non-zero type I error probabilities of collision detection: A non-zero value of the type I collision detection error probability α is equivalent to omitting some collisions. If the number of inputs γ is somewhat increased, this is easily tolerated by the methods, since the number of true (apriori) collisions grows quadratically with the increase of the number of inputs.

Under the assumption that the type II error probability β is negligibly low, we performed this trade-off between α, γ and success probability P for FS-, FL- and linear collisions. The results can be found in Figure 4 and show that the FL-collision based method is superior to FS- and linear collision based methods

Table 3. Solving equation systems over $GF(2^8)$ for FL-collisions

Inputs, γ	5	4
Success probability π	1.00	0.85
Offline complexity (operations)	$\leq 2^{40}$	$\leq 2^{40}$
Memory limit, MB	500	500
Number of variables	32	32
Average number of equations	43.58	29.66

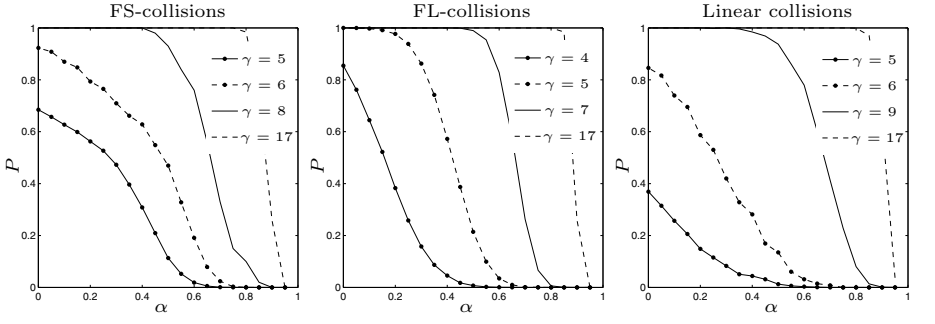


Fig. 4. Success probability P against type I error probability α in FS-, FL- and linear collision-based key recovery for different numbers γ of random inputs

even in the presence of strong noise. For example, while type I error probabilities up to $\alpha = 0.65$ are well tolerated with only $\gamma = 7$ inputs for FL-collisions, one needs at least $\gamma = 8$ or $\gamma = 9$ inputs to achieve a comparable tolerance level for FS- and linear collisions, respectively.

4 Towards Reliable Collision Detection in Practice

Probability distribution of Euclidean distance. Given two traces $\tau_1 = (\tau_{1,1}, \ldots, \tau_{1,l}) \in \mathbb{R}^l$ and $\tau_2 = (\tau_{2,1}, \ldots, \tau_{2,l}) \in \mathbb{R}^l$, we assume that each point $\tau_{i,j}$ can be statistically described as $\tau_{i,j} = s_{i,j} + r_{i,j}$, where $s_{i,j}$ is signal constant (without noise) for the given time point i as well as some fixed input to the S-box, and $r_{i,j}$ is Gaussian noise due to univariate normal distribution[3] with mean 0 and some variance σ^2 remaining the same for all time instances in our rather rough model. Let τ_1 and τ_2 correspond to some S-box inputs a_1 and a_2.

If $a_1 = a_2$, the corresponding deterministic signals are equal (that is, $s_{1,j} = s_{2,j}$ for all j's) and one has:

$$1/H(\tau_1, \tau_2)_{a_1=a_2} = \sum_{j=1}^{l} (\tau_{1,j} - \tau_{2,j})^2 = \sum_{j=1}^{l} \xi_j^2 = 2\sigma^2 \sum_{j=1}^{l} \eta_j^2,$$

where $\xi_j = r_{1,j} - r_{2,j}$, $\xi_j \sim \mathcal{N}\left(0, 2\sigma^2\right)$ and $\eta_j \sim \mathcal{N}(0, 1)$. That is, statistic $1/H(\tau_1, \tau_2)_{a_1=a_2}$ follows the chi-square distribution with l degrees of freedom up to the coefficient $2\sigma^2$. As the chi-square distribution is approximated by normal distribution for high degrees of freedom, one has the following

[3] The real measured power consumption is often due to the generic multivariate normal distribution. However, almost all entries of the corresponding covariance matrix are close to zero. Thus, the model with independent multivariate normal distribution seems to be quite realistic.

Proposition 2. *Statistic* $1/H(\tau_1,\tau_2)_{a_1=a_2} = \sum_{j=1}^{l}(\tau_{1,j}-\tau_{2,j})^2$ *for* $\tau_i = (\tau_{i,1},\ldots,\tau_{i,l}) \in \mathbb{R}^l$ *with* $\tau_{i,j} \sim \mathcal{N}\left(s_{i,j},\sigma^2\right)$ *can be approximated by normal distribution* $\mathcal{N}(2\sigma^2 l, 8\sigma^4 l)$ *for sufficiently large* l*'s.*

Alternatively, if $a_1 \neq a_2$, one has

$$1/H(\tau_1,\tau_2)_{a_1\neq a_2} = \sum_{j=1}^{l}(\tau_{1,j}-\tau_{2,j})^2 = \sum_{j=1}^{l}\left(\delta_j^{(1,2)}+\xi_j\right)^2 = 2\sigma^2\sum_{j=1}^{l}\nu_j^2,$$

where $\delta_j^{(1,2)} = \tau_{1,j}-\tau_{2,j}$, $\xi_j = r_{1,j}-r_{2,j}$, $\xi_j \sim \mathcal{N}\left(0,2\sigma^2\right)$ and $\nu_j \sim \mathcal{N}\left(\delta_j^{(1,2)}/\sqrt{2}\sigma, 1\right)$. That is, statistic $1/H(\tau_1,\tau_2)_{a_1\neq a_2}$ follows the *noncentral* chi-square distribution with l degrees of freedom and $\lambda = \sum_{j=1}^{l}\left(\delta_j^{(1,2)}/\sqrt{2}\sigma\right)^2$ up to the coefficient $2\sigma^2$. Again, we have an approximation using

Proposition 3. *Statistic* $1/H(\tau_1,\tau_2)_{a_1\neq a_2} = \sum_{j=1}^{l}(\tau_{1,j}-\tau_{2,j})^2$ *for* $\tau_i = (\tau_{i,1},\ldots,\tau_{i,l}) \in \mathbb{R}^l$ *with* $\tau_{i,j} \sim \mathcal{N}\left(s_{i,j},\sigma^2\right)$ *can be approximated by normal distribution* $\mathcal{N}\left(2\sigma^2(l+\lambda), 8\sigma^4(l+2\lambda)\right)$ *with* $\lambda = \sum_{j=1}^{l}\left(\delta_j^{(1,2)}/\sqrt{2}\sigma\right)^2$ *for sufficiently large* l*'s.*

Selection of most informative trace points. In the direct binary comparison, we try to distinguish between the distributions $1/H(\tau_1,\tau_2)_{a_1\neq a_2}$ and $1/H(\tau_1,\tau_2)_{a_1=a_2}$. As described above these statistics approximately follow normal distribution for large numbers of trace points. That is, to efficiently distinguish between these two statistics it is crucial to decrease their variances while keeping the difference of their means high. For this purpose, to increase the success probability of the Euclidean distance test, we propose to discard points of traces with small minimal contribution to the difference of means.

To illustrate this method of point selection, we assume for the moment that $\delta_j^{(1,2)} = 0$ for $j > l/2$ and $\delta_j^{(1,2)} \neq 0$ for $j \leq l/2$ with l even, that is, the second half of the trace does not contain any data dependent information. Then we can discard the second halves of the both traces τ_1 and τ_2 in the direct binary comparison function and compute two related statistics on the rest of the points:

$$1/H'(\tau_1,\tau_2)_{a_1=a_2} = \sum_{j=1}^{l/2}(\tau_{1,j}-\tau_{2,j})^2, 1/H'(\tau_1,\tau_2)_{a_1\neq a_2} = \sum_{j=1}^{l/2}(\tau_{1,j}-\tau_{2,j})^2.$$

This will adjust the means and variances of the approximating normal distributions: $\mathcal{N}\left(\sigma^2 l, 4\sigma^4 l\right)$ and $\mathcal{N}\left(2\sigma^2(l/2+\lambda), 8\sigma^4(l/2+2\lambda)\right)$, respectively. Note that the difference of means remains unaffected and equal to $2\sigma^2\lambda$. At the same time both variances are reduced, one of them by factor 2, which allows one to distinguish between these two distributions more efficiently and, thus, to detect collisions more reliably.

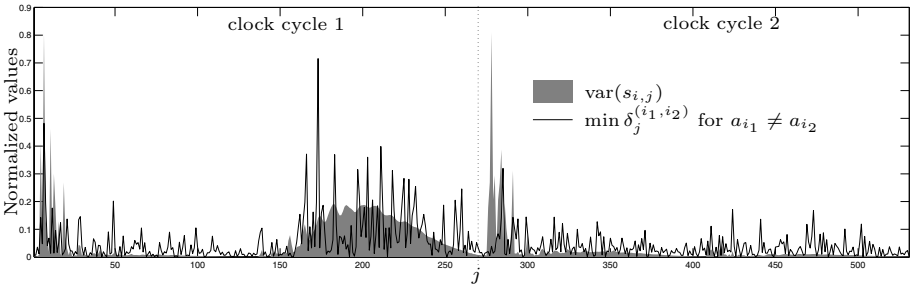


Fig. 5. Informative points for collision detection and DPA

More generally speaking, for AES we have to reliably distinguish between inputs in each (a_{i_1}, a_{i_2}) of the $\binom{256}{2}$ pairs of byte values, $a_{i_1}, a_{i_2} \in \mathrm{GF}(2^8)$. Thus, the most informative points j of the traces are those with maximal minimums of $\delta_j^{(i_1,i_2)}$ over all pairs of different inputs, that is, points j with maximal values of

$$\min_{a_{i_1} \neq a_{i_2}} \delta_j^{(i_1,i_2)}.$$

We estimated these values for all time instances j of our AES implementation and compared this to the signal variance in the same time points, $\mathrm{var}(s_{i,j})$, $a_i \in \mathrm{GF}(2^8)$, which is known to be a good indicator of the points leaking information in DPA. This comparison is represented in Figure 5 for two clock cycles of our 8-bit table look-up operation.

5 Experimental Validation

5.1 AES Implementation and Measurement Equipment

We performed our attacks for a typical AES implementation on the Atmel ATmega16 microcontroller, an RISC microcontroller from the 8-bit AVR family with a Harvard architecture. 8-bit AVR microcontrollers are widely used in embedded devices. To run a collision attack, the attacker has to know when the AES S-boxes are executed. So we measured the power consumption of the table look-ups corresponding to the relevant S-box applications. These include instances in SubBytes, ShiftRows, and MixColumns operations.

The microcontroller was clocked at 3.68 MHz and supplied with an operating voltage of 5V from a standard laboratory power source. The variations of the power consumption were observed on a shunt resistor of 5.6 Ohm inserted into the ground line of the microcontroller. The measurements were performed with a LeCroy WaveRunner 104MXi DSO equipped with ZS1000 active probe. The DSO has 8-bit resolution and 1 GHz input bandwidth (with the specified probe). The acquisitions were performed at the maximum sampling rate of 10 GS/s

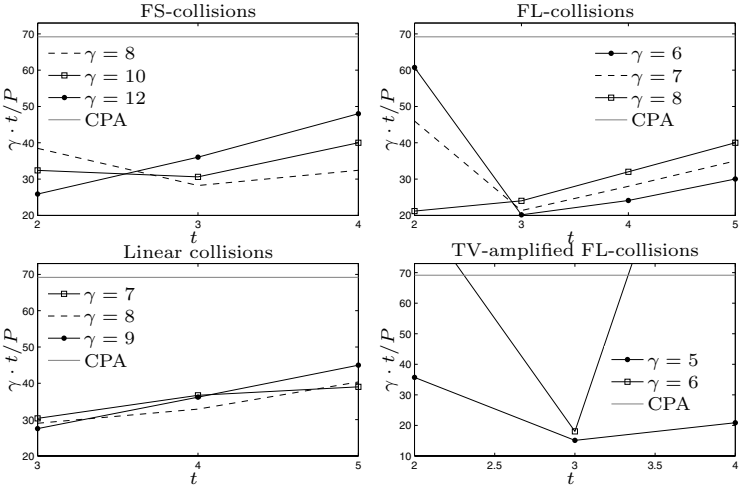


Fig. 6. Performance of collision attacks based on FS-, FL- and linear collisions without profiling as well as of FL-collision based attacks with profiling (ternary voting with 625 profiling measurements) vs. Hamming-distance based CPA on the same traces

without any input filters. We stress that this measurement setup is *not* noise-optimized[4].

5.2 Attack Scenarios and Results

Performance metric. We use the following efficiency metric to compare the performance of all these attacks: $t \cdot \gamma / P$, where t is the number of averagings, γ is the number of different inputs, and P is the success probability of the attack. In case of the Hamming-distance based CPA we apply a similar metric: n/p^{16}, where n is the number of measurements needed to determine a single-byte chunk of the AES key with probability p. These metrics characterize the expected number of measurements needed to recover the whole 16-byte key. The performance results for CPA can be found in Figure 6.

Collision attacks without profiling. In the online stage, an attacker observes executions of AES for γ random inputs, each repeated t times. Then, the $t \cdot \gamma$ traces are averaged t times and one obtains γ averaged traces for γ

[4] As in case of DPA [8], collision detection methods tend to be sensitive to the preprocessing of measured signals. To denoise the traces, we proceed in two steps. First, the traces are decimated by applying a low-pass filter to the original traces and subsequently resampling them at a lower rate. Additionally to noise reduction, this weakens time jitter. Second, the decimated traces are denoised by applying a wavelet decomposition at a certain level, thresholding the detail coefficients, and subsequent wavelet reconstruction. Our experiments show that symlets proposed by Daubechies (first of all, the 'sym3' wavelet) are most suitable for this operation.

random inputs. These are used to detect collisions – linear, FS- or FL-collisions (see Sections 2 and 3) depending on the key-recovery method – with the direct binary comparison (see Subsection 2.3). All key-recovery methods need AES plaintexts to be known. Additionally, if the attack is based on FL-collisions, the corresponding AES ciphertexts have to be known.

Note that since the adaptive threshold W is used in the direct binary comparison which eliminates all type II errors, many true collisions are omitted due to the increased type I error probability by shifting W to the right. That is, for given γ and α, the success probability P of the whole attack follows the dependencies illustrated in Figure 4 for different key-recovery methods.

Profiling-amplified collision attacks. If profiling is possible prior to the online stage, the ternary voting method (see Subsection 2.4) can be used to detect additional collisions, which are omitted by the direct binary comparison due to the application of the adaptive threshold. Taking additional collisions is equivalent to the increase of α, which further improves the performance metric $t \cdot \gamma / P$. In our profiling-amplified attacks we used $N = 10^5$ profiling S-box traces τ_i which is equivalent to about $10^5/160 = 625$ executions of AES in the profiling stage. See Figure 6 for concrete results of collision attacks with profiling.

References

1. Bogdanov, A.: Multiple-differential side-channel collision attacks on AES. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 30–44. Springer, Heidelberg (2008)
2. 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)
3. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 51–62. Springer, Heidelberg (2003)
4. Lemke-Rust, K.: Models and Algorithms for Physical Cryptanalysis. PhD thesis, Ruhr University Bochum (2007)
5. Schramm, K., Leander, G., Felke, P., Paar, C.: A collision-attack on AES: Combining side channel- and differential-attack. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 163–175. Springer, Heidelberg (2004)
6. Bogdanov, A.: Improved side-channel collision attacks on AES. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 84–95. Springer, Heidelberg (2007)
7. Cid, C., Murphy, S., Robshaw, M.: Algebraic Aspects of the Advanced Encryption Standard. Springer, Heidelberg (2006)
8. Charvet, X., Pelletier, H.: Improving the DPA attack using wavelet transform. In: NIST Physical Security Testing Workshop (2005)

New Related-Key Boomerang Attacks on AES

Michael Gorski and Stefan Lucks

Bauhaus-University Weimar, Germany
{Michael.Gorski,Stefan.Lucks}@uni-weimar.de

Abstract. In this paper we present two new attacks on round reduced versions of the AES. We present the first application of the related-key boomerang attack on 7 and 9 rounds of AES-192. The 7-round attack requires only 2^{18} chosen plaintexts and ciphertexts and needs $2^{67.5}$ encryptions. We extend our attack to nine rounds of AES-192. This leaves to a data complexity of 2^{67} chosen plaintexts and ciphertexts using about $2^{143.33}$ encryptions to break 9 rounds of AES-192.

Keywords: block ciphers, AES, differential cryptanalysis, related-key boomerang attack.

1 Introduction

The Advanced Encryption Standard (AES) [8] has become one of the most used symmetric encryption algorithm in the world. Differential cryptanalysis [6] is one of the most powerful attacks on block ciphers like the AES. It recovers subkey bits for the first or the last rounds, while using differential properties of the underlying cipher. Variants of this attack such as the impossible differential attack [2], the truncated differential attack [16], the higher order differential attack [16], the differential-linear attack [17], the boomerang attack [21], the amplified boomerang attack [12] and the rectangle attack [3] were introduced.

The boomerang attack [21] is a strong extension of differential cryptanalysis to break more rounds than differential attacks can do, since the cipher is treated as a cascade of two sub-ciphers, using short differentials in each sub-cipher. These differentials are combined in an adaptive chosen plaintext and ciphertext attack to exploit properties of the cipher that have a high probability.

Related-key attacks [1, 15] apply differential cryptanalysis to ciphers using different but related keys and consider the information that can be extracted from encryptions under these keys. Ciphers with a weak key schedule are vulnerable to this kind of attack. The idea of related-key differentials was presented in [13], while two encryptions under two related-keys are used. Biryukov [7] propose a boomerang attack on the AES-128 which can break up to 5 and 6 out of 10 rounds. The related-key boomerang attack was published first in [4], but was not used to attack the AES.

We present the first related-key boomerang attack on 7 rounds of AES-192 using 4 related keys. Our related-key boomerang attack can also break 9 rounds using 256 related keys. It uses less data and less time than existing attacks on

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 266–278, 2008.

Table 1. Existing attacks on round reduced AES-192

Attack	# rounds	# keys	data / time	source
Impossible Differential	7	1	2^{92} / 2^{186}	[19]
Square	7	1	2^{32} / 2^{184}	[18]
Partial Sums	7	1	$19 \cdot 2^{32}$ / 2^{155}	[9]
Partial Sums	7	1	$2^{128} - 2^{119}$ / 2^{120}	[9]
Related-Key Differential-Linear	7	2	2^{22} / 2^{187}	[22]
Related-Key Differential-Linear	7	2	2^{70} / 2^{130}	[22]
Related-Key Impossible	7	2	2^{111} / 2^{116}	[11]
Related-Key Impossible	7	32	2^{56} / 2^{94}	[5]
Related-Key Boomerang	7	4	2^{18} / $2^{67.5}$	Section 4
Partial Sums	8	1	$2^{128} - 2^{119}$ / 2^{188}	[9]
Related-Key Impossible	8	2	2^{88} / 2^{183}	[11]
Related-Key Rectangle	8	4	$2^{86.5}$ / $2^{86.5}$	[10]
Related-Key Rectangle	8	2	2^{94} / 2^{120}	[14]
Related-Key Differential-Linear	8	2	2^{118} / 2^{165}	[22]
Related-Key Impossible	8	32	2^{116} / 2^{134}	[5]
Related-Key Impossible	8	32	2^{92} / 2^{159}	[5]
Related-Key Impossible	8	32	$2^{68.5}$ / 2^{184}	[5]
Related-Key Rectangle†	9	256	2^{86} / 2^{181}	[4]
Related-Key Rectangle	9	64	2^{85} / 2^{182}	[14]
Related-Key Boomerang	9	256	2^{67} / $2^{143.33}$	Section 5
Related-Key Rectangle	10	256	2^{125} / 2^{182}	[14]
Related-Key Rectangle	10	64	2^{124} / 2^{183}	[14]

† the attack with some flaws corrected by Kim et al. [14].

the same number of reduced rounds. Table 1 summarizes existing attacks on AES-192 and our new attacks on 7 and 9 rounds.

In Section 2 we give a brief description of the AES. In Section 3 we describe the related-key boomerang attack. In Section 4, we present a new related-key boomerang attack on 7-round of AES-192. Our 9 round attack is presented in Section 5. We conclude the paper in Section 6.

2 Description of the AES

The AES [8] is a block cipher using data blocks of 128 bits with 128, 192 or 256-bit cipher key. A different number of rounds is used depending on the length of the cipher key. The AES has 10, 12 and 14 rounds when a 128, 192 or 256-bit

cipher key is used respectively. The plaintexts are treated as a 4 x 4 byte matrix, which is called state. A round applies four operations to the state:

- SubBytes (SB) is a non-linear byte-wise substitution applied on every byte of the state matrix in parallel.
- ShiftRows (SR) is a cyclic left shift of the i-th row by i bytes, where $i \in \{0, 1, 2, 3\}$.
- MixColumns (MC) is a multiplication of each column by a constant 4 x 4 matrix.
- AddRoundKey (AK) is a XORing of the state and a 128-bit subkey which is derived from the cipher key.

An AES round function applies the SB, SR, MC and AK operation in order. Before the first round a whitening AK operation is applied and the MC operation is omitted in the last round because of symmetry. We concentrate on the 192-bit version of the AES in this paper and refer to [8] for more details on the other versions. Let W_i be a 32-bit word, then the 192-bit cipher key is represented by $W_0||W_1||W_2||\ldots||W_5$. The 192-bit key schedule algorithm works as follows:

- For $i = 6$ till $i = 51$
  - If $i \equiv 0 \mod 6$, then $W_i = W_{i-6} \oplus SB(RotByte(W_{i-1})) \oplus Rcon(i/6)$,
  - else $W_i = W_{i-6} \oplus W_{i-1}$.

where *Rcon* denotes fixed constants depending on its input and *RotByte* represents a byte-wise left shift. The whitening key is $W_0||W_1||W_2||W_3$, the subkey of round 1 is $W_4||W_5||W_6||W_7$, the subkey of round 2 is $W_8||W_9||W_{10}||W_{11}$ and so one. The bytes coordinates of a 4 x 4 state matrix are labeled as:

x_0	x_4	x_8	x_{12}
x_1	x_5	x_9	x_{13}
x_2	x_6	x_{10}	x_{14}
x_3	x_7	x_{11}	x_{15}

3 The Related-Key Boomerang Attack

We now describe the related-key boomerang attack [4] in more detail. But first, we have to give some definitions.

Definition 1. *Let P, P' be two bit strings of the same length. The bit-wise xor of P and P', $P \oplus P'$, is called the* difference *of P, P'. Let a be a known and $*$ an unknown non-zero byte difference.*

Definition 2. *$\alpha \to \beta$ is called a* differential *if α is the plaintext difference $P \oplus P'$ before some non-linear operation $f(\cdot)$ and β is the difference after applying these operation, i.e, $f(P) \oplus f(P')$. The probability p is linked on a differential saying that an α difference turns into a β difference with probability p. The backward direction, i.e., $\alpha \leftarrow \beta$ has probability $\hat{p}$.*

Two texts (P, P') are called a *pair*, while two pairs (P, P', O, O') are called a *quartet*. Regularly, the differential probability decreases the more rounds are included. Therefore two short differential covering only a few rounds each will be used instead of a long one covering the whole cipher. Related-keys are used to exploit some weaknesses of the key schedule to enhance the probability of the differentials being used. We call such differentials related-key differentials. We split the related-key boomerang attack into two steps. The *related-key boomerang distinguisher step* and the *key recovery step*. The related-key boomerang distinguisher is used to find all plaintexts sharing a desired difference that depends on the choice of the related-key differential. These plaintexts are used in the key recovery step afterwards to recover subkey bits for the initial round key.

Distinguisher Step. During the *distinguisher step* we treat the cipher as a cascade of two sub-ciphers $E_K(P) = E^1_K(P) \circ E^0_K(P)$, where K is the key used for encryption and decryption. We assume that the related-key differential $\alpha \rightarrow \beta$ for E^0 occurs with probability p, while the related-key differential $\gamma \rightarrow \delta$ for E^1 occurs with probability q, where α, β, γ and δ are differences of texts. The backward direction $E^{0^{-1}}$ and $E^{1^{-1}}$ of the related-key differential for E^0 and E^1 are denoted by $\alpha \leftarrow \beta$ and $\gamma \leftarrow \delta$ and occur with probability $\hat{p}$ and $\hat{q}$ respectively. The related-key boomerang distinguisher involves four different unknown but related-keys $K_a, K_b = K_a \oplus \Delta K^*, K_c = K_a \oplus \Delta K'$ and $K_d = K_a \oplus \Delta K^* \oplus \Delta K'$, where ΔK^* and $\Delta K'$ are known cipher key differences. The attack works as follows:

1. Choose a pool of s plaintexts P_i, $i \in \{1, \ldots, s\}$ uniformly at random and compute a pool $P'_i = P_i \oplus \alpha$.
2. Ask for the encryption of P_i under K_a, i.e., $C_i = E_{K_a}(P_i)$ and ask for the encryption of P'_i under K_b, i.e., $C' = E_{K_b}(P'_i)$.
3. Compute the new ciphertexts $D_i = C_i \oplus \delta$ and $D'_i = C'_i \oplus \delta$.
4. Ask for the decryption of D_i under K_c, i.e., $O_i = E^{-1}_{K_c}(D_i)$ and ask for the decryption of D'_i under K_d, i.e., $O'_i = E^{-1}_{K_d}(D'_i)$.

– For each pair (O_i, O'_j), $i, j \in \{1, \ldots, s\}$

5. If $O_i \oplus O'_j$ equals α store the quartet (P_i, P'_j, O_i, O'_j) in the set M.

A pair (P_i, P'_j), $i, j \in \{1, \ldots, s\}$ with the difference α satisfies the differential $\alpha \rightarrow \beta$ with the probability p. The output of E_0 is A_i and A'_j, i.e., $E^0_{K_a}(P_i) = A_i$ and $E^0_{K_b}(P'_j) = A'_j$ have a certain difference $\beta = A_i \oplus A'_j$ with probability p. Using the ciphertexts C_i and C'_j we can compute the new ciphertexts $D_i = C_i \oplus \delta$ and $D'_j = C'_j \oplus \delta$. Let $B_i = E^{1^{-1}}_{K_c}(D_i)$ and $B'_j = E^{1^{-1}}_{K_d}(D'_j)$ are the decryption of D_i and D'_j with $E^{1^{-1}}_{K_i}$ $i \in \{c, d\}$. A difference δ turns into a difference γ after passing $E^{1^{-1}}_{K_i}$ with probability $\hat{q}$. Since $\delta = C_i \oplus D_i$ and $\delta = C'_j \oplus D'_j$ we know that $\gamma = A_i \oplus B_i$ and $\gamma = A'_j \oplus B'_j$ with probability $\hat{q}^2$. Since we also know, that $A_i \oplus A'_j = \beta$ with probability p, it follows that $(A_i \oplus B_i) \oplus (A_i \oplus A'_j) \oplus (A'_j \oplus B'_j) = \gamma \oplus \beta \oplus \gamma = \beta = (B_i \oplus B'_j)$ holds with probability $p \cdot \hat{q}^2$. A β difference turns into an α difference after passing the differential $E^{0^{-1}}_{K_i}$ with probability $\hat{p}$. Thus,

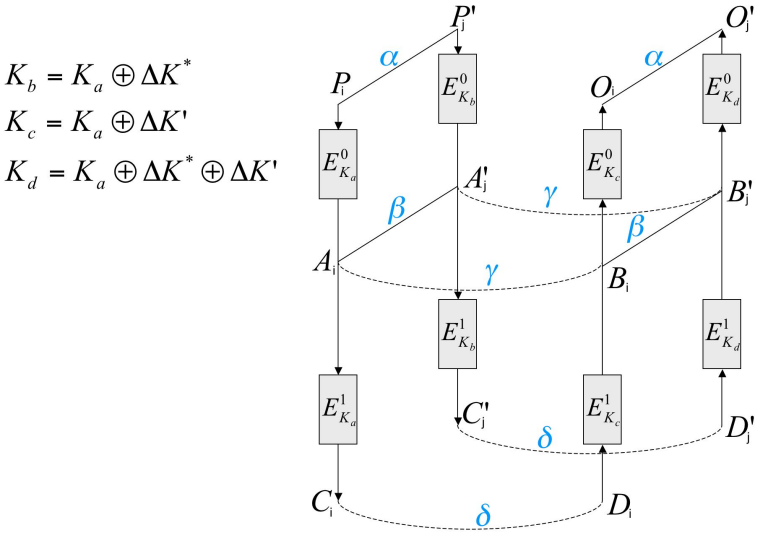


Fig. 1. The related-key boomerang distinguisher

a pair of plaintexts (P_i, P'_j) with $P_i \oplus P'_j = \alpha$ generates a new pair of plaintexts (O_i, O'_j) where $O_i \oplus O'_j = \alpha$ with probability $p \cdot \hat{p} \cdot \hat{q}^2$. A quartet containing these two pairs is defined as:

Definition 3. *A quartet* (P_i, P'_j, O_i, O'_j) *which satisfies* $P_i \oplus P'_j = \alpha = O_i \oplus O'_j$, $A_i \oplus A'_j = \beta = B_i \oplus B'_j$, $A_i \oplus B_i = \gamma = A'_j \oplus B'_j$, $C_i \oplus D_i = \delta = C'_j \oplus D'_j$ *is called a* ***correct related-key boomerang quartet*** *which occurs with probability* $Pr_c = p \cdot \hat{p} \cdot \hat{q}^2$. *A quartet* (P_i, P'_j, O_i, O'_j) *which only satisfies the condition* $P \oplus P'_j = \alpha = O_i \oplus O'_j$ *is called a* ***false related-key boomerang quartet****.*

Figure 1 displays the structure of the related-key boomerang distinguisher step. Any attacker who applies a related-key boomerang distinguisher does not know the internal states A_i, A'_j, B_i, B'_j, since he can only apply a chosen plaintext and ciphertext attack on the cipher. The set M which is the output of the related-key boomerang distinguisher, therefore contains correct and false related-key boomerang quartets. It is impossible to form another distinguisher which separates the correct and the false related-key boomerang quartets, since the interior differences β and γ cannot be computed.

Key Recovery Step. The second step of the related-key boomerang attack is the *key recovery step*. From now on, an attacker operates on the set M that was stored by the related-key boomerang distinguisher. Let k_a, k_b, k_c and k_d be some key bits of the last round keys derived from the cipher keys K_a, K_b, K_c and K_d. Let $d_k(C)$ be the one round partially decryption of C under the key bits k. The key bits are related as $k_b = k_a \oplus \Delta k^*$, $k_c = k_a \oplus \Delta k'$ and $k_d = k_a \oplus \Delta k^* \oplus \Delta k'$, where Δk^* and $\Delta k'$ are differences of the last round key bits. These differences

are derived from the cipher key difference ΔK^* and $\Delta K'$. The key recovery step works as follows:

- For each key-bit combination of k_a
  1. Initialize a counter for each key-bit combination with zero.
  - For all quartets (P, P', O, O') stored in M
    2. Ask for the encryption of P, P', O, O' under K_a, K_b, K_c and K_d respectively and obtain the ciphertext quartet C, C', D, D'. Decrypt the ciphertexts C, C', D, D' under k_a, k_b, k_c, k_d, i.e., $\bar{C} = d_{k_a}(C)$, $\bar{C'} = d_{k_b}(C'), \bar{D} = d_{k_c}(D)$ and $\bar{D'} = d_{k_d}(D')$.
    3. Test whether the differences $\bar{C} \oplus \bar{D}$ and $\bar{C'} \oplus \bar{D'}$ have a desired difference an attacker would expect depending on the related-key differential being used. Increase a counter for the used key-bits if the difference is fulfilled in both pairs.
4. Output the key-bits k_a with the highest counter as the correct one.

Four cases can be distinct in Step 3, since M contains correct and false related-key boomerang quartets and the key-bit combination k_a can either be correct or false. A correct related-key boomerang quartet encrypted with the correct key bits will have the desired difference needed to pass the test in Step 3 with probability 1. Hence, the counter for the correct key bits is increased. The three other cases are: a correct related-key boomerang quartet is used with false key bits (Pr_{cK_f}), a false related-key boomerang quartet is used with the correct key-bits (Pr_{fK_c}) or a false related-key boomerang quartet is used with a false key-bit combination (Pr_{fK_f}). We assume that the cipher acts like a random permutation. In these cases we assume that

$$Pr_{cK_f} = Pr_{fK_c} = Pr_{fK_f} =: Pr_{filter}.$$

The probability that a quartet in one of the three undesirable cases is counted for a certain key bit combination is Pr_{filter}. The related-key differentials have to be chosen such that the counter of the correct key bits is significantly higher than the counter of each false key bit combination. If the differentials have a high probability the key recovery step outputs the correct key-bits in Step 4 with a high probability much faster than exhaustive search.

4 Related-Key Boomerang Attack on 7-Round AES-192

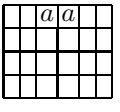
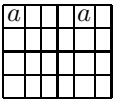
In this section we mount a key recovery attack on 7-round AES-192 using 4 related keys. The cipher is represented as $E = E^1 \circ E^0$. E^0 is a differential containing rounds 1 to 4 and including the whitening key addition as well as the key addition of round 4. E^1 is a differential covering rounds 5 to 7. After applying the related-key boomerang distinguisher for $E^1 \circ E^0$ using the related-key differentials E^0 and E^1 we apply it to recover 8 key-bits of the seventh round-keys. We assumed, that the S-Box acts like a random permutation. Thus, all S-Box output differences will have the same probability for a given input difference. The notation used in our attack will be defined as:

- K_a, K_b, K_c, K_d unknown cipher keys (192 bit).
- $K_{ai}, K_{bi}, K_{ci}, K_{di}$ unknown round keys of round i, where $i \in \{0, 1, 2, \ldots, 12\}$ (128 bit).
- $\Delta K^*, \Delta K'$ known cipher key differences (192 bit).
- $\Delta K_i^*, \Delta K_i'$ known subkey differences of round i (128 bit).
- P_i, P_j', O_i, O_j' plaintexts.
- C_i, C_j', D_i, D_j' ciphertexts.
- $E^0_{K_i}(\cdot)$ 4-round AES-192 encryption from round 1 to 4 under key K_i, where $i \in \{a, b, c, d\}$.
- $E^{1^{-1}}_{K_i}(\cdot)$ 3-round AES-192 decryption from round 7 to 5 under key K_i, where $i \in \{a, b, c, d\}$.
- a is a known non-zero byte difference.
- b is an output difference of S-Box for the input difference a.
- c, d are unknown non-zero byte differences.
- $*$ is a variable unknown non-zero byte differences.

The Structure of the Keys. In our attack we use four related but unknown keys K_a, K_b, K_c and K_d. Let K_a be the unknown key an attacker would like to recover. The relation that is required for the attack is:

$$K_b = K_a \oplus \Delta K^*, \quad K_c = K_a \oplus \Delta K'$$
$$K_d = K_a \oplus \Delta K^* \oplus \Delta K'$$

ΔK^* is the cipher key difference used for the first related-key differential E^0 and $\Delta K'$ is the cipher key difference used for the second related-key differential E^1. An attacker only knows the differences ΔK^* and $\Delta K'$ but does not know the keys. He chooses the cipher key differences as:

$\Delta K^* =$  and $\Delta K' =$ 

Using the key schedule algorithm of AES-192 we can use the cipher key differences ΔK^* and $\Delta K'$ to derive the round key differences $\Delta K_0^*, \ldots, \Delta K_8^*$ and $\Delta K_0', \ldots, \Delta K_8'$ respectively.[1] These values are shown in Figure 2 and 3.

The difference b can be one of $2^7 - 1$ values, because of the symmetry of the XOR operation and the fact that an a difference can be one of $2^8 - 1$ differences. If two texts forming an a difference passing the S-Box only one of $2^7 - 1$ differences can occur.

The Related-Key Differential E^0 for rounds $1 - 4$. The input difference α of E^0 has a non-zero difference in bytes 8 and 12. These differences are of value a with the probability 2^{-16}. This is the probability that two randomly chosen non-zero bytes are of value a. The whitening key addition AK_0 generates a zero difference in each byte of the state matrix. These zero differences remain

[1] These related keys are also used in [10].

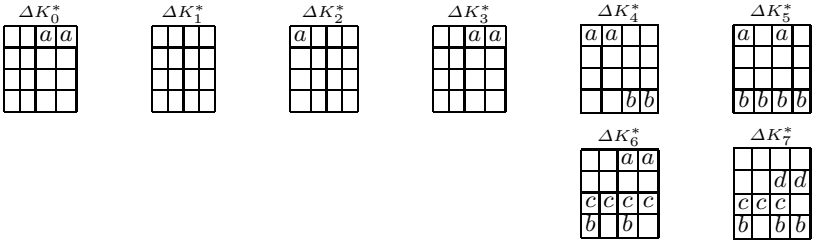


Fig. 2. Round key differences derived from ΔK^*

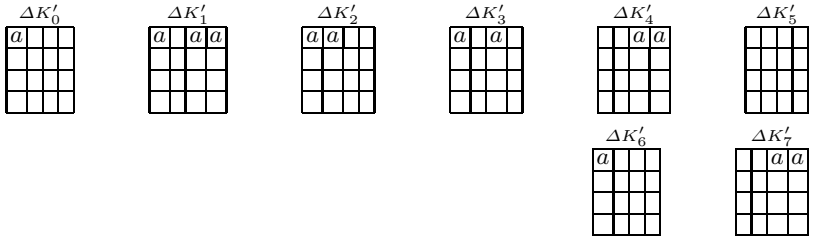


Fig. 3. Round key differences derived from $\Delta K'$

until AK_2 is applied, since ΔK_1^* has only zero differences and does not alter the differences in the state matrix. AK_2 generates an a difference in byte 0, which is transformed into a non-zero difference after SB_3. MC_3 creates a non-zero difference in bytes 0,1,2 and 3, while AK_3 inserts an a difference in bytes 8 and 12. After applying SR_4 we just have one non-zero byte in column 0 and 1 and two non-zero bytes in column 2 and 3. Four non-zero bytes remain after MC_4 in column 0 and 1 with probability one, while we do not know which bytes of column 2 and 3 are non-zero. These bytes are labeled with ?. Then AK_4 places an a difference in byte 12. We call β_{out} the difference obtaining after passing the related-key differential E^0. The probability of the differential E_0, i.e., the transformation of an α difference into a β_{out} difference is given by

$$Pr(\alpha \rightarrow \beta_{out}) = 2^{-16}.$$

The related-key differential E^0 is shown in Figure 4.

The Related-Key Differential $E^{1^{-1}}$ for rounds $7-5$. The input difference δ consists of a non-zero difference in byte 0 and two a differences in bytes 8 and 12. This differences vanish after AK_7^{-1}, since $\Delta K'_7$ has two a differences in bytes 8 and 12 while the other bytes of $\Delta K'_7$ have a zero difference. Only the non-zero difference in byte 0 remains. SB_7^{-1} generates an a difference in byte 0 with probability 2^{-8} since we assume that the S-Box acts like a random permutation. If this occurs the text difference after SB_7^{-1} is equal to the subkey difference $\Delta K'_6$. Hence, all bytes have a zero difference after applying AK_6^{-1}. All bytes will

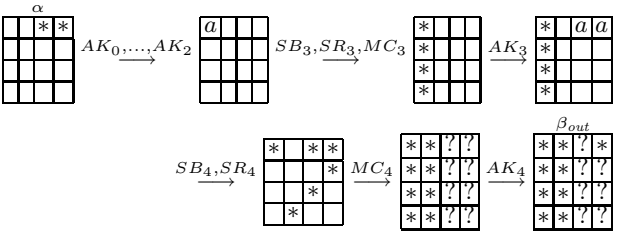


Fig. 4. The related-key differential E^0

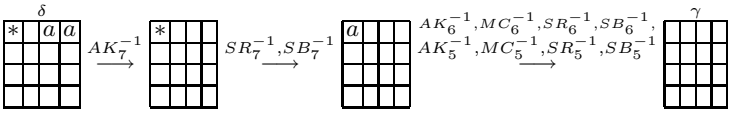


Fig. 5. The related-key differential $E^{1^{-1}}$

also have a zero difference after AK_5^{-1}, since $\Delta K_5'$ has a zero difference in each byte. We call the text difference after applying $E^{1^{-1}}$ γ which consists of 16 zero bytes. The probability of $E^{1^{-1}}$ is $\Pr(\gamma \leftarrow \delta) = 2^{-8}$. The related-key differential $E^{1^{-1}}$ is shown in Figure 5.

The Related-Key Differential $E^{0^{-1}}$ for rounds $4-1$. For the following steps we need that the output difference β_{out} of the related-key differential E^0 is equal to the input difference β_{in} for the related-key differential $E^{0^{-1}}$. Note that β_{in} and β_{out} are not only equal in the same positions of non-zero differences but are also equal in each byte. We will shown how to construct such a case. From the boomerang condition inside the cipher for two differences γ_1 and γ_2 we know that $\beta_{out} \oplus \gamma_1 \oplus \gamma_2 = \beta_{in}$ holds with some probability. Since γ_1 and γ_2 are equal in each byte, we simply write γ. Thus the above condition reduces to $\beta_{out} \oplus \gamma \oplus \gamma = \beta_{out} = \beta_{in}$. Using the differentials above, the differences β_{in} and β_{out} are equal with probability one. Note that these difference occur only with some probability, which will be described more detailed later.

Let A, A', B, B' be the internal state after SR_4 when encrypting P, P', O, O' under K_a, K_b, K_c, K_d respectively. We use the same notation as in Figure 1. Since MC is linear γ can be expressed as $\gamma = K_{a4} \oplus MC_4(A) \oplus K_{c4} \oplus MC_4(B) = K_{a4} \oplus K_{c4} \oplus MC_4(A \oplus B)$ and as $\gamma = K_{b4} \oplus MC_4(A') \oplus K_{d4} \oplus MC_4(B') = K_{b4} \oplus K_{d4} \oplus MC_4(A' \oplus B')$. Considering these equations we obtain $A \oplus A' = B \oplus B'$. Thus MC_4 can be undone with the probability 1. This means that we know exactly that after MC_4^{-1} only the bytes 0,7,8,10,12 and 13 are non-zero, while all other bytes are zero. SB_4^{-1} then transforms a non-zero difference into an a difference with probability 2^{-8}. Regarding bytes 8 and 12 we have the probability 2^{-16} of doing so. The resulting a differences in bytes 8 and 12 are canceled out by AK_3^{-1}. After that MC_3^{-1} generates a non-zero with a fixed position from four non-zero bytes with probability 2^{-24}. We have only one a difference after SB_3^{-1}

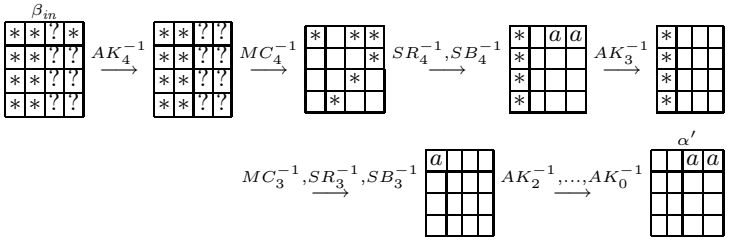


Fig. 6. The related-key differential $E^{0^{-1}}$

in byte 0 with probability $2^{-24} \cdot 2^{-8} = 2^{-32}$. This a difference is canceled out by AK_2^{-1}. We call α the difference that is the output of the related-key differential $E^{0^{-1}}$. α has an a difference in the bytes 8 and 12. The differential $E^{0^{-1}}$ has the probability $\Pr(\alpha \leftarrow \beta_{in}) = 2^{-16} \cdot 2^{-32} = 2^{-48}$ and is shown in Figure 6.

The Attack. The attack first applies a related-key boomerang distinguisher to obtain all correct and false boomerang quartets which are stored in M. A key-search is then applied on M to find 1 byte of the seventh round keys. Let k_{a7} be an 8-bit subkey in the position of byte 0 of the seventh round key K_{a7}. Let $d_{7k_{i7}}(X)$, $i \in \{a, b, c, d\}$ be the seventh round partially decryption of X under the 8-bit subkey k_i. The attack is as follows:

1. Choose $2^{49.5}$ structures $S_1, S_2, \ldots, S_{2^{49.5}}$ of 2^{16} plaintexts P_i, $i \in \{1, 2, \ldots, 2^{16}\}$, where all bytes are fixed except for bytes 8 and 12. Ask for encryption of P_i under K_a to obtain the ciphertexts C_i, i.e., $C_i = E_{K_a}(P_i)$.
2. Compute $2^{49.5}$ structures $S'_1, S'_2, \ldots, S'_{2^{49.5}}$ of 2^{16} plaintexts $P'_i = P_i$. Ask for encryption of the P'_i under K_b, where $K_b = K_a \oplus \Delta K^*$ to obtain the ciphertexts C'_i, i.e., $C'_i = E_{K_b}(P'_i)$.
3. Compute $2^{49.5}$ structures $S^*_1, S^*_2, \ldots, S^*_{2^{49.5}}$ of 2^{16} ciphertexts D_i, i.e, $D_i = C_i \oplus \delta$ where δ is a fixed difference with any non-zero byte difference in byte 0 and two a differences in bytes 8 and 12. Ask for decryption of D_i under K_c to obtain the plaintexts O_i, i.e., $O_i = E^{-1}_{K_c}(D_i)$.
4. Compute $2^{49.5}$ structures $S'^*_1, S'^*_2, \ldots, S'^*_{2^{49.5}}$ of 2^{16} ciphertexts D'_i, i.e., $D'_i = C'_i \oplus \delta$ where δ is as in Step 3. Ask for decryption of D'_i under K_d to obtain the plaintexts O'_i, i.e., $O'_i = E^{-1}_{K_d}(D'_i)$.
5. Store only those quartets (P_i, P'_j, O_i, O'_j), $i, j \in \{1, 2, \ldots, 2^{16}\}$ in the set M where $O_i \oplus O'_j$ have an a difference in bytes 8 and 12, while the remaining byte differences are zero.
6. For each 8-bit key k_{a7} compute $k_{b7} = k_{a7}, k_{c7} = k_{a7}$ and $k_{d7} = k_{a7}$.
   For each quartet passing the test in Step 5:
   - 6.1. Ask for encryption of (O_i, O'_j) under K_c, K_d to obtain (D_i, D'_j) and compute (C_i, C'_j) respectively.
   - 6.2. Partially decrypt a ciphertext quartet (C_i, C'_j, D_i, D'_j), i.e., $\bar{C}_i = d_{7k_{a7}}(C_i)$, $\bar{C}'_j = d_{7k_{b7}}(C'_j)$, $\bar{D}_i = d_{7k_{c7}}(D_i)$ and $\bar{D}'_j = d_{7k_{d7}}(O'_j)$.

6.3. Increase the counter for the used 8-bit subkey k_{a7} by one if $\bar{C}_i \oplus \bar{D}_i$ and $\bar{C}'_j \oplus \bar{D}'_j$ have an a-difference in byte 0.

7. Output the 8-bit subkey k_{a7} which counts at least two quartets as the correct one.

Analysis of the Attack. Two pools of 2^{16} plaintexts can be combined to approximately $(2^{16})^2 = 2^{32}$ quartets. Using $2^{49.5}$ structures we obtain $\#PP \approx 2^{49.5} \cdot 2^{32} = 2^{81.5}$ quartets in total. A correct related-key boomerang quartet occurs with probability $Pr_c = \Pr(\alpha \rightarrow \beta_{out}) \cdot (\Pr(\gamma \leftarrow \delta))^2 \cdot \Pr(\alpha \leftarrow \beta_{in}) = 2^{-16} \cdot (2^{-8})^2 \cdot 2^{-48} = 2^{-80}$ since all related-key differential conditions are fulfilled. A random permutation of a difference $O_i \oplus O'_j$ has 14 zero byte difference with probability $Pr_f = 2^{-112}$. Thus, after Step 5 we have about $\#C = \#PP \cdot Pr_c = 2^{81.5} \cdot 2^{-80} = 2^{1.5}$ correct and $\#F = \#PP \cdot Pr_f = 2^{81.5} \cdot 2^{-112} = 2^{-30.5}$ false related-key boomerang quartets. The data and time complexity of Step 6 to 7 is negligible compared to the other steps before, since we expect to have only $2^{1.5}$ quartets stored.

A false combination of quartets and key bits is counted in Step 6.3 with the probability $Pr_{filter} = 2^{-16}$. This is the probability that an active byte with an unknown non-zero difference has an a difference after SB_7^{-1}.

At least $\#CK_c = 2^{1.5}$ correct related-key boomerang quartets and additionally $\#FK_c = \#F \cdot Pr_{filter} = 2^{-30.5} \cdot 2^{-16} = 2^{-46.5}$ false related-key boomerang quartets are counted with the correct key bits. About $\#CK_c + \#FK_c = 2^{1.5} + 2^{-46.5} \approx 3$ quartets are counted in Step 6.3 for the correct key bits.

About $\#CK_f = \#C \cdot Pr_{filter} = 2^{1.5} \cdot 2^{-16} = 2^{-14.5}$ correct related-key boomerang quartets and $\#FK_f = \#F \cdot Pr_{filter} = 2^{-30.5} \cdot 2^{-16} = 2^{-46.5}$ false related-key boomerang quartets are counted with the false key bits, which are approximately $\#CK_f + \#FK_f = 2^{-14.5} + 2^{-46.5} = 2^{-14.5}$ counts for each false key bit combination.

Using the Poisson distribution we can compute the success rate of our attack. The probability that the number of remaining quartets for each false key bit combination is larger than 1 is $Y \sim Poisson(\mu = 2^{-14.5})$, $\Pr(Y \geq 2) \approx 0$. Therefore the probability that our attack outputs false key bits as the correct one is very low. We expect to have a count of 2^2 quartets for the correct key bits. The probability that the number of quartets counted for the correct key bits is larger than 1 is $Z \sim Poisson(\mu = 3)$, $\Pr(Z \geq 2) \approx 0.8$.

The data complexity of this attack is determined by Steps 1, 2, 3 and 4 which is about $2^{18} = 2^2 \cdot 2^{16}$ adaptive chosen plaintexts and ciphertexts for each structure. We do not have to compute all structures simultaneously, thus we keep only four structures in memory, which reduces the memory requirements of our attack. The time complexity is about $2^{67.5} = 2^{49.5} \cdot 2^2 \cdot 2^{16}$ seven round AES-192 encryptions. Our attack has a success rate of 0.8.

5 Related-Key Boomerang Attack on 9-Round AES-192

Our related-key boomerang attack can be extend to attack 9 rounds of AES-192 using 256 related keys. The data complexity is of 2^{67} chosen plaintexts and

ciphertexts and the time complexity is about $2^{143.33}$ nine round AES encryptions. The details are described in a full version of the paper available on ePrint or upon request.

6 Conclusion

In this paper we improved attacks on 7 and 9 round reduced versions of AES-192. This is the first application of the related-key boomerang attack on the AES. Our 7 round attack has a data complexity of 2^{18} chosen plaintexts and ciphertexts. Its time complexity is of $2^{67.5}$ seven round AES-192 encryptions. We also presented a 9 round related-key boomerang attack which needs only 2^{67} chosen plaintexts and ciphertexts and has a time complexity of $2^{143.33}$ nine round encryptions. Our attacks are the best attacks on seven and nine rounds of AES-192 in terms of data and time complexity known so far. The AES remains still unbroken but we have shown that up to 7 rounds practical attacks are available yet.

Acknowledgements

The authors would like to thank Thomas Peyrin, Ewan Fleischmann and the anonymous reviewers for many helpful comments.

References

[1] Biham, E.: New Types of Cryptanalytic Attacks Using Related Keys. J. Cryptology 7(4), 229–246 (1994)
[2] Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. J. Cryptology 18(4), 291–311 (2005)
[3] Biham, E., Dunkelman, O., Keller, N.: The Rectangle Attack - Rectangling the Serpent. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 340–357. Springer, Heidelberg (2001)
[4] Biham, E., Dunkelman, O., Keller, N.: Related-Key Boomerang and Rectangle Attacks. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 507–525. Springer, Heidelberg (2005)
[5] Biham, E., Dunkelman, O., Keller, N.: Related-Key Impossible Differential Attacks on 8-Round AES-192. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 21–33. Springer, Heidelberg (2006)
[6] Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. J. Cryptology 4(1), 3–72 (1991)
[7] Biryukov, A.: The Boomerang Attack on 5 and 6-Round Reduced AES. In: Dobbertin, H., Rijmen, V., Sowa, A. (eds.) AES 2005. LNCS, vol. 3373, pp. 11–15. Springer, Heidelberg (2005)
[8] Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer, Heidelberg (2002)
[9] Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., Whiting, D.: Improved Cryptanalysis of Rijndael. In: Schneier [20], pp. 213–230

[10] Hong, S., Kim, J., Lee, S., Preneel, B.: Related-Key Rectangle Attacks on Reduced Versions of SHACAL-1 and AES-192. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 368–383. Springer, Heidelberg (2005)
[11] Jakimoski, G., Desmedt, Y.: Related-Key Differential Cryptanalysis of 192-bit Key AES Variants. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 208–221. Springer, Heidelberg (2004)
[12] Kelsey, J., Kohno, T., Schneier, B.: Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. In: Schneier [20], pp. 75–93
[13] Kelsey, J., Schneier, B., Wagner, D.: Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. In: Han, Y., Quing, S. (eds.) ICICS 1997. LNCS, vol. 1334, pp. 233–246. Springer, Heidelberg (1997)
[14] Kim, J., Hong, S., Preneel, B.: Related-Key Rectangle Attacks on Reduced AES-192 and AES-256. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 225–241. Springer, Heidelberg (2007)
[15] Knudsen, L.R.: Cryptanalysis of LOKI91. In: Seberry, J., Zheng, Y. (eds.) AUSCRYPT 1992. LNCS, vol. 718, pp. 196–208. Springer, Heidelberg (1993)
[16] Knudsen, L.R.: Truncated and Higher Order Differentials. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 196–211. Springer, Heidelberg (1995)
[17] Langford, S.K., Hellman, M.E.: Differential-Linear Cryptanalysis. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 17–25. Springer, Heidelberg (1994)
[18] Lucks, S.: Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Keys. In: AES Candidate Conference, pp. 215–229 (2000)
[19] Phan, R.C.-W.: Impossible differential cryptanalysis of 7-round Advanced Encryption Standard (AES). Inf. Process. Lett. 91(1), 33–38 (2004)
[20] Schneier, B. (ed.): FSE 2000. LNCS, vol. 1978. Springer, Heidelberg (2001)
[21] Wagner, D.: The Boomerang Attack. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 156–170. Springer, Heidelberg (1999)
[22] Zhang, W., Zhang, L., Wu, W., Feng, D.: Related-Key Differential-Linear Attacks on Reduced AES-192. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 73–85. Springer, Heidelberg (2007)

New Impossible Differential Attacks on AES

Jiqiang Lu[1,⋆], Orr Dunkelman[2,⋆⋆], Nathan Keller[3,⋆⋆⋆], and Jongsung Kim[4,†]

[1] Information Security Group, Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK and Department of Mathematics and Computer Science, Eindhoven University of Technology,
5600 MB Eindhoven, The Netherlands
lvjiqiang@hotmail.com
[2] École Normale Supérieure
Département d'Informatique,
45 rue d'Ulm, 75230 Paris, France
orr.dunkelman@ens.fr
[3] Einstein Institute of Mathematics, Hebrew University.
Jerusalem 91904, Israel
nkeller@math.huji.ac.il
[4] Center for Information Security Technologies(CIST), Korea University
Anam Dong, Sungbuk Gu, Seoul, Korea
joshep@cist.korea.ac.kr

Abstract. In this paper we apply impossible differential attacks to reduced round AES. Using various techniques, including the early abort approach and key schedule considerations, we significantly improve previously known attacks due to Bahrak-Aref and Phan. The improvement of these attacks leads to better impossible differential attacks on 7-round AES-128 and AES-192, as well as to better impossible differential attacks on 8-round AES-256.

Keywords: AES, Impossible differential cryptanalysis.

1 Introduction

The *Advanced Encryption Standard (AES)* [12] is a 128-bit block cipher with a variable key length (128, 192, and 256-bit keys are supported). Since its selection,

⋆ This author as well as his work was supported by a British Chevening / Royal Holloway Scholarship.

⋆⋆ The second author was supported by the France Telecome Chaire. Some of the work presented in this paper was done while this author was staying at K.U. Leuven.

⋆⋆⋆ This author is supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

† This author was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Advancement) (IITA-2006-(C1090-0603-0025)).

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 279–293, 2008.

AES gradually became one of the most widely used block ciphers. AES has received a great deal of cryptanalytic attention, both during the AES process, and even more after its selection.

In the single-key model, previous results can attack up to 7 rounds of AES-128 (i.e., AES with 128-bit key). The first attack is a SQUARE attack suggested in [14] which uses $2^{128} - 2^{119}$ chosen plaintexts and 2^{120} encryptions. The second attack is a meet-in-the-middle attack proposed in [15] that requires 2^{32} chosen plaintexts and has a time complexity equivalent to almost 2^{128} encryptions. Recently, another attack on 7-round AES-128 was presented in [1]. The new attack is an impossible differential attack that requires $2^{117.5}$ chosen plaintexts and has a running time of 2^{121} encryptions.

These results were later improved and extended in [19] to suggest impossible differential attacks on 7-round AES-192 (data complexity of 2^{92} chosen plaintexts and time complexity of 2^{162} encryptions) and AES-256 ($2^{116.5}$ chosen plaintexts and $2^{247.5}$ encryptions).

There are several attacks on AES-192 [1,13,14,17,18,19]. The two most notable ones are the SQUARE attack on 8-round AES-192 presented in [14] that requires almost the entire code book and has a running time of 2^{188} encryptions and the meet in the middle attack on 7-round AES-192 in [13] that requires 2^{34+n} chosen plaintexts and has a running time of $2^{208-n} + 2^{82+n}$ encryptions. Legitimate values for n in the meet in the middle attack on AES-192 are $94 \geq n \geq 17$, thus, the minimal data complexity is 2^{51} chosen plaintexts (with time complexity equivalent to exhaustive search), and the minimal time complexity is 2^{146} (with data complexity of 2^{97} chosen plaintexts).

AES-256 is analyzed in [1,13,14,17,19]. The best attack is the meet in the middle attack in [13] which uses 2^{32} chosen plaintexts and has a total running time of 2^{209} encryptions.

Finally, we would like to note the existence of many related-key attacks on AES-192 and AES-256. As the main issue of this paper is not related-key attacks, and as we deal with the single key model, we do not elaborate on the matter here, but the reader is referred to [20] for the latest results on related-key impossible differential attacks on AES and to [16] for the latest results on related-key rectangle attacks on AES.

The strength of AES with respect to impossible differentials was challenged several times. The first attack of this kind is a 5-round attack presented in [4]. This attack is improved in [10] to a 6-round attack. In [18], an impossible differential attack on 7-round AES-192 and AES-256 is presented. The latter attack uses 2^{92} chosen plaintexts (or $2^{92.5}$ chosen plaintexts for AES-256) and has a running time of 2^{186} encryptions (or $2^{250.5}$ encryptions for AES-256). The time complexity of the latter attack was improved in [19] to 2^{162} encryptions for AES-192.

In [1] a new 7-round impossible differential attack was presented. The new attack uses a different impossible differential, which is of the same general type as the one used in previous attacks (but has a slightly different structure). Using the new impossible differential leads to an attack that requires $2^{117.5}$ chosen plaintexts and has a running time of 2^{121} encryptions. This attack was later

improved in [19] to use $2^{115.5}$ chosen plaintexts with time complexity of 2^{119} encryptions.

The last application of impossible differential cryptanalysis to AES was the extension of the 7-round attack from [1] to 8-round AES-256 in [19]. The extended attack has a data complexity of $2^{116.5}$ chosen plaintexts and time complexity of $2^{247.5}$ encryptions.

We note that there were three more claimed impossible differential attacks on AES in [7,8,9]. However, as all these attacks are flawed [6].

In this paper we present a new attack on 7-round AES-128, a new attack on 7-round AES-192, and two attacks on 8-round AES-256. The attacks are based on the attacks proposed in [1,18] but use additional techniques, including the *early abort* technique and key schedule considerations.

Our improvement to the attacks on 7-round AES-128 from [1,19] requires $2^{112.2}$ chosen plaintexts, and has a running time of $2^{117.2}$ memory accesses. Our improvement to the attack on 7-round AES-192 from [18] has a data complexity of $2^{91.2}$ chosen plaintexts and a time complexity of $2^{139.2}$ encryptions. Since the first attack is also applicable to AES-192, the two attacks provide a data-time tradeoff for attacks on 7-round AES-192.

Table 1. A Summary of the Previous Attacks and Our New Attacks

Key Size	Number of Rounds	Complexity Data (CP)	Complexity Time	Attack Type & Source
128	7	$2^{117.5}$	2^{121}	Impossible Differential [1]
	7	$2^{115.5}$	2^{119}	Impossible Differential [19]
	7	2^{32}	2^{128}	Meet in the Middle [15]
	7	$2^{112.2}$	$2^{117.2}$ MA	Impossible Differential (App. B)
192	7	2^{92}	$2^{186.2}$	Impossible Differential [18]
	7	$2^{115.5}$	2^{119}	Impossible Differential [19]
	7	2^{92}	2^{162}	Impossible Differential [19]
	7	2^{34+n}	$2^{208-n} + 2^{82+n}$	Meet in the Middle [13]
	8	$2^{128} - 2^{119}$	2^{188}	SQUARE [14]
	7	$2^{113.8}$	$2^{118.8}$ MA	Impossible Differential (App. B)
	7	$2^{91.2}$	$2^{139.2}$	Impossible Differential (Sect. 4.1)
256	7	$2^{92.5}$	$2^{250.5}$	Impossible Differential [18]
	7	2^{32}	2^{208}	Meet in the Middle [13]
	7	$2^{115.5}$	2^{119}	Impossible Differential [19]
	8	$2^{116.5}$	$2^{247.5}$	Impossible Differential [19]
	8	$2^{128} - 2^{119}$	2^{204}	SQUARE [14]
	8	2^{32}	2^{209}	Meet in the Middle [13]
	7	$2^{113.8}$	$2^{118.8}$ MA	Impossible Differential (App. B)
	7	2^{92}	2^{163} MA	Impossible Differential (Sect. 4.1)
	8	$2^{111.1}$	$2^{227.8}$ MA	Impossible Differential (App. B)
	8	$2^{89.1}$	$2^{229.7}$ MA	Impossible Differential (Sect. 4.2)

CP – Chosen plaintext, MA – Memory Accesses.
Time complexity is measured in encryption units unless mentioned otherwise.

The best attack we present on 8-round AES-256 requires $2^{89.1}$ chosen plaintexts and has a time complexity of $2^{229.7}$ memory accesses. These results are significantly better than any previously published impossible differential attack on AES. We summarize our results along with previously known results in Table 1.

This paper is organized as follows: In Section 2 we briefly describe the structure of AES. In Section 3 we discuss the impossible differential attack by [18] (and its improvement from [19]) on 7-round AES-192. The improvement of Phan's attack on 7-round AES-192 along with its extension to 8-round AES-256 is presented in Section 4. In Appendix A we describe a technique which is repeatedly used in impossible differential attacks on AES. In Appendix B we describe the attack by Bahrak and Aref on 7-round AES-128, and its possible improvements and extensions (to 8-round AES-256). Appendix C outlines the impossible differentials used in this paper for the sake of completeness. We conclude the paper in Section 5.

2 Description of AES

The advanced encryption standard [12] is an SP-network that supports key sizes of 128, 192, and 256 bits. A 128-bit plaintext is treated as a byte matrix of size 4x4, where each byte represents a value in $GF(2^8)$. An AES round applies four operations to the state matrix:

- SubBytes (SB) — applying the same 8-bit to 8-bit invertible S-box 16 times in parallel on each byte of the state,
- ShiftRows (SR) — cyclic shift of each row (the i'th row is shifted by i bytes to the left),
- MixColumns (MC) — multiplication of each column by a constant 4x4 matrix over the field $GF(2^8)$, and
- AddRoundKey (ARK) — XORing the state with a 128-bit subkey.

We outline an AES round in Figure 1. In the first round, an additional AddRoundKey operation (using a whitening key) is applied, and in the last round the MixColumns operation is omitted. As all other works on AES, we shall assume that reduced-round variants also have the MixColumns operation omitted from the last round.

The number of rounds depends on the key length: 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. The rounds are

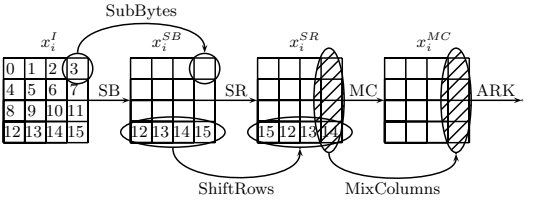


Fig. 1. An AES round

numbered $0, \ldots, Nr-1$, where Nr is the number of rounds ($Nr \in \{10, 12, 14\}$). For the sake of simplicity we shall denote AES with n-bit keys by AES-n, i.e., AES with 192-bit keys (and thus with 12 rounds) is denoted by AES-192.

The key schedule of AES takes the user key and transforms it into 11, 13, or 15 subkeys of 128 bits each. The subkey array is denoted by $W[0, \ldots, 59]$, where each word of $W[\cdot]$ consists of 32 bits. The first Nk words of $W[\cdot]$ are loaded with the user supplied key, i.e., $Nk = 4$ words for 128-bit keys, $Nk = 6$ words for 192-bit keys, and $Nk = 8$ for 256-bit keys. The remaining words of $W[\cdot]$ are updated according to the following rule:

- For $i = Nk, \ldots, 43/51/59$, do
  - If $i \equiv 0 \bmod Nk$ then $W[i] = W[i-Nk] \oplus SB(W[i-1] \lll 8) \oplus RCON[i/Nk]$,
  - Otherwise $W[i] = W[i-1] \oplus W[i-Nk]$,

where $RCON[\cdot]$ is an array of predetermined constants, and $\lll$ denotes rotation of the word by 8 bits to the left. We also note that for 256-bit keys, when $i \equiv 4 \bmod 8$ the update rule is $W[i] = W[i-8] \oplus SB(W[i-1] \lll 8)$.

2.1 The Notations Used in the Paper

In our attacks we use the following notations: x_i^I denotes the input of round i, while x_i^{SB}, x_i^{SR}, x_i^{MC}, and x_i^O denote the intermediate values after the application of SubBytes, ShiftRows, MixColumns, and AddRoundKey operations of round i, respectively. Of course, the relation $x_{i-1}^O = x_i^I$ holds.

We denote the subkey of round i by k_i, and the first (whitening) key is k_{-1}, i.e., the subkey of the first round is k_0. In some cases, we are interested in interchanging the order of the MixColumns operation and the subkey addition. As these operations are linear they can be interchanged, by first XORing the data with an equivalent key and only then applying the MixColumns operation. We denote the equivalent subkey for the altered version by w_i, i.e., $w_i = MC^{-1}(k_i)$.

We denote bytes of some intermediate state x_i or a key k_i (or w_i) by an enumeration $\{0, 1, 2, \ldots, 15\}$ where the byte $4m+n$ corresponds to the n'th byte in the m'th row of x_i, and is denoted by $x_{i,4 \cdot m+n}$. We denote the z'th column of x_i by $x_{i,Col(z)}$, i.e., $w_{0,Col(0)} = MC^{-1}(k_{0,Col(0)})$. Similarly, by $x_{i,Col(y,z)}$ we denote columns y and z of x_i. We define two more column related sets. The first is $x_{i,SR(Col(z))}$ which is the bytes in x_i corresponding to the places after the ShiftRows operation on column z, e.g., $x_{i,SR(Col(0))}$ is composed of bytes 0,7,10,13. The second is $x_{i,SR^{-1}(Col(z))}$ which is the bytes in the positions of column z after having applied the inverse ShiftRows operation.

3 The Phan Impossible Differential Attack on 7-Round AES-192

The security of AES against impossible differential attacks was challenged in two lines of research. The first presented in [4,10,18,19], and the second in [1,19].

Both lines use very similar impossible differentials as well as similar algorithms. In this section we present the first line of research, represented by the Phan attack [18] on 7-round AES-192. The second line of research, represented by the Bahrak-Aref attack [1] on 7-round AES-192 is considered in Appendix B.

The Phan attack, as well as all the other known impossible differential attacks on the AES, is based on the following 4-round impossible differential of AES, first observed in [4]:

Proposition 1. *Let $\Delta(x_i^I)$ denote the input difference to round i, and let $\Delta(x_{i+3}^{SR})$ denote the difference after the ShiftRows operation of round $i+3$. If the following two conditions hold:*

1. *$\Delta(x_i^I)$ has only one non-zero byte,*
2. *In $\Delta(x_{i+3}^{SR})$, at least one of the four sets of bytes $SR(Col(i))$, for the four different possible columns, is equal to zero,*

then $\Delta(x_i^I) \longrightarrow \Delta(x_{i+3}^{SR})$ is an impossible differential for any four consecutive rounds of AES. We outline one of these impossible differentials in Figure 3 (in Appendix C). We also note that if in round $i+3$ the order of MixColumns and AddRoundKey is swapped, then, one can consider the impossible differential $\Delta(x_i^I) \longrightarrow \Delta(x_{i+3}^O)$.

Proof. On the one hand, if $\Delta(x_i^I)$ has only one non-zero byte then $\Delta(x_{i+1}^I)$ has non-zero values in a single column, and therefore, $\Delta(x_{i+2}^I)$ has non-zero values in all the 16 bytes of the table (following the basic diffusion properties of AES, a fact used in many attacks on AES). On the other hand, if Condition (2) holds then $\Delta(x_{i+3}^I)$ has at least one zero column, and hence at least one of the four sets of bytes $SR^{-1}(Col(i))$ in $\Delta(x_{i+2}^I)$ is equal to zero, a contradiction.

3.1 The Phan Attack Algorithm on 7-Round AES-192

The algorithm of the Phan attack, as described in [18], is the following (depicted in Figure 2):

1. Encrypt 2^{60} structures of 2^{32} plaintexts each such that in every structure, the bytes of $SR^{-1}(Col(0))$ assume all the 2^{32} possible values and the rest of the bytes are fixed.
2. Select only the ciphertext pairs, corresponding to plaintexts in the same structure, for which the difference in bytes $SR(Col(2,3))$ is zero.
3. Guess k_6 and partially decrypt the remaining ciphertext pairs through round 6 to get x_6^I.
4. Using the guessed value of k_6, retrieve $k_{5,Col(0,1)}$ by the key schedule algorithm. For each remaining pair, decrypt x_6^I through $ARK^{-1} \circ MC^{-1} \circ SR^{-1} \circ SB^{-1} \circ MC^{-1}$ to get the difference $\Delta(x_4^{SR})$.[1] If the difference does not satisfy Condition (2) of Proposition 1, discard the pair.

[1] Note that the ARK^{-1} operation in the end of round 4 can be skipped since it does not affect the difference $\Delta(x_4^{SR})$.

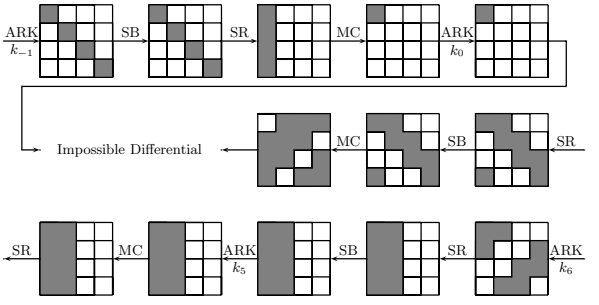


Fig. 2. The 7-Round Impossible Differential Attack on AES-192 by Phan

5. Consider the plaintext pairs corresponding to the remaining ciphertext pairs. Guess the value of $k_{-1,SR^{-1}(Col(0))}$ and partially encrypt each plaintext pair through $ARK \circ SB \circ SR \circ MC$ to get the difference $\Delta(x_1^I)$. If the difference satisfies Condition (1) of Proposition 1, discard the guess of $k_{-1,SR^{-1}(Col(0))}$.
6. If all the guesses of $k_{-1,SR^{-1}(Col(0))}$ are discarded for a guess of k_6, repeat Steps 3–5 with another guess of k_6. If a candidate of $k_{-1,SR^{-1}(Col(0))}$ remains, the rest of the key bits (or their equivalent) are exhaustively searched.

Step 1 of the attack consists of the encryption of 2^{92} chosen plaintexts. Step 2 of the attack takes 2^{92} memory accesses and proposes 2^{59} pairs for further analysis. Steps 3 and 4 take together 2^{188} two-round decryptions, and suggest 2^{29} pairs for Step 5 (for a given key guess). Step 5 takes 2^{185} 1-round encryptions.

Therefore, the data complexity of the attack is 2^{92} chosen plaintexts, and the time complexity is $2^{186.2}$ 7-round AES-192 encryptions. The attack requires 2^{157} bytes of memory, used for storing the discarded guesses of k_6 and $k_{-1,SR^{-1}(Col(0))}$.

4 Improving and Extending the Phan Attack

In this section we improve the Phan attack on 7-round AES-192 and extend it to an attack on 8-round AES-256.

4.1 Improvement of the Phan Attack on 7-Round AES-192

The improvement of the Phan attack is based on the early abort technique and key schedule considerations, as well as on a reuse of the data. Our approach reduces the data and time complexities of the attack significantly.

Reducing the number of guessed key material. We observe that the amount of guessed key bytes can be reduced for AES-192. This observation was made independently in [19] and was used there only to gain an immediate reduction in the time complexity of the Phan attack by factor 2^{24}. This follows the fact that the 16 subkeys bytes are needed by the attack: $k_{6,SR(Col(0,1))}$ and

$k_{5,Col(0,1)}$ (rather than the entire k_6 and $k_{5,Col(0,1)}$ as done in the original version of the Phan attack). In the Phan attack, the attacker needs to guess these 16 subkey bytes. However, using the key schedule of AES-192, the amount of guessed bytes can be reduced, as $k_{6,(10,11)}$ determine $k_{5,9}$, $k_{6,(10,13)}$ determine $k_{5,8}$, and $k_{6,(1,14)}$ determine $k_{5,12}$. Hence, it is sufficient to guess 13 key bytes instead of 16.

Reducing the Time Complexity of Steps 3–4 of Phan's Attack. The time complexity of Steps 3 and 4 of the attack can be further reduced. We first note that in the Phan attack the attacker can use four "output" differences for the impossible differential, i.e., requiring one of the four sets $SR(Col(0))$, $SR(Col(1))$, $SR(Col(2))$ or $SR(Col(3))$ of bytes to have a zero difference. Thus, the attacker repeats Steps 3–4 four times, each time under the assumption that the (shifted) column with zero difference is different. We shall describe the steps the attacker performs under the assumption that $x^{SR}_{4,SR(Col(0))}$ is zero.

In the improved attack, the attacker guesses the 80 bits of the key which compose $k_{6,SR(Col(0,1))}$ and $k_{5,Col(0)}$ (there are 2 bytes of $k_{5,Col(0)}$ which are known due to the key schedule). Then, all the remaining pairs are decrypted to find the differences in $\Delta x^{MC}_{4,SR^{-1}(Col(0))}$ (we note that the actual values of $x^{MC}_{4,SR^{-1}(Col(0,2,3))}$ are also known to the attacker). Under the assumption that the pair has a difference which satisfies Condition (2) of Proposition 1 for $x^{SR}_{4,SR(Col(0))}$, the attacker can immediately deduce the actual difference in each column of x^{MC}_4. This follows the fact that the MC operation is linear, and as the attacker knows for each column the byte with zero difference before the MC operation, and the difference in three bytes after the MC operation, she can determine the difference in the fourth byte of each column.

Once Δx^{MC}_4 is computed, the attacker knows the input differences to SubBytes of round 5 as well as the output differences (in all bytes), and thus, she can compute the exact inputs and outputs. Given an input and an output difference of the SubBytes operation, there is on average one pair of actual values that satisfies these differences.[2] Once the outputs are known, the attacker encrypts the values through Round 5 and retrieves the key bytes in $k_{5,Col(1)}$ suggested by this pair. Of course, if the suggested key disagrees with the known byte (recall that $k_{5,9}$ is known due to the key schedule) then the pair is discarded (for the specific 80-bit subkey guess). Otherwise, the pair is passed for further analysis in Steps 5–6 of the attack (for a specific guess of 104 bits of the key, 80 that were guessed and 24 that were computed).

The attacker starts with 2^{59} pairs, and for each 80-bit key value and shifted column, partially decrypts these pairs through three columns (two in one round, and then another one in the second round), and analyzes the fourth column. Hence, the time complexity of this step is roughly $2 \cdot 2^{59} \cdot 2^{80} \cdot 4 = 2^{142}$ 1-round encryptions, which are equivalent to $2^{139.2}$ 7-round encryptions.

[2] More accurately, for randomly chosen input and output differences we expect that about half of the combinations are not possible, about half propose two actual values, and a small fraction suggest four values.

Each of the $2^{59} \cdot 2^{80} \cdot 4 = 2^{141}$ partially decrypted pairs is expected to suggest one value for $k_{5,Col(1)}$. With probability $1 - 2^{-8}$ this value is discarded, and thus, for a given 104-bit guess, we expect $2^{141} \cdot 2^{-8}/2^{104} = 2^{29}$ pairs which are analyzed in Steps 5–6.

Optimizing Steps 5–6 of the Phan 7-Round Attack. Step 5 of the Phan attack can be performed efficiently using the hash table method described in [4]. A short description of this technique can be found in Appendix A. For each guess of the 104 key bits in k_5 and k_6 there are 2^{29} pairs, each suggesting 2^{10} values of the key to be removed. The time complexity of this step is $2^{104} \cdot 2^{29} \cdot 2^{10} = 2^{143}$ memory accesses. Therefore, it is expected that all the wrong guesses of the 104 guessed bits are discarded, and the attacker is left with the right value of 104 subkey bits. The rest of the key can be easily found using an exhaustive key search.

The memory complexity of the attack also can be significantly improved. We observe that there is no need to store the discarded values of the 136 guessed subkey bits. Instead, for each 80-bit guess of $k_{6,SR(Col(0,1))}$ and $k_{5,Col(0)}$, the attacker repeats Steps 3–4 and stores for each value of $k_{5,Col(1)}$ the pairs which can be used for analysis.

Therefore, the amount of memory required for the attack is smaller, as we mainly need to store the data. The memory complexity of the attack is therefore roughly 2^{65} bytes of memory.

Finally, we slightly reduce the data complexity (and thus the time complexity) of the attack. We observe that in the Phan attack, a wrong subkey for k_6 has probability $2^{-152.7}$ to remain after Step 5.[3] As the time complexity of the attack is already above 2^{130} encryptions, even if more subkeys remain, the attack can be completed by exhaustive key search without affecting the overall time complexity.

We first note that out of $W[24–29]$ (whose knowledge is equivalent to the knowledge of the key) the attacker already knows 96 bits (for a given 104-bit guess). Thus, as long as Step 5 does not suggest more than 2^{34} values for the 104-bit key, the exhaustive key search phase of the attack would be faster than 2^{130}. Hence, we can reduce the data complexity by a factor of $2^{0.8}$, which in turn reduces the time complexity of the attack by a similar factor.

Summarizing the improved attack, the data complexity of the attack is $2^{91.2}$ chosen plaintexts, the time complexity is $2^{139.2}$ encryptions. The memory complexity is 2^{65} bytes of memory. For AES-256, as the attacker cannot exploit the key schedule, the data complexity is 2^{92} chosen plaintexts and the time complexity is 2^{163} memory accesses.

4.2 Extension of the Phan Attack to 8-Round AES-256

The trivial extension of the Phan attack to 8-round AES-256 (by guessing the last round subkey, partial decryption, and application of the 7-round attack) leads to an attack whose time complexity is significantly higher than 2^{256}. By

[3] Due to space restrictions, the reader is refereed to [18] for the computation of this figure.

using key schedule arguments, changing the used impossible differentials, using a more advanced attack algorithm, and reusing the data, we can present an attack on 8-round AES-256.

Our attack still maintains the above general approach, i.e., the attack is of the form:

- Encrypt 2^{60} structures of 2^{32} plaintexts each such that in every structure, the bytes of $SR^{-1}(Col(0))$ assume all the 2^{32} possible values and the rest of the bytes are fixed.
- For each value of k_7 determine the pairs that are to be analyzed with this subkey guess.
- Apply the 7-round attack with the selected pairs.

To perform the actual attacks, we need to make several modifications in the internal 7-round attack. The first change is to use an impossible differential in which the two active columns in x_5^O are (2,3) (rather than (0,1)). As a result, in the 7-round attack there is no need to guess bytes from k_5 since these key bytes are known due to the key schedule, given the knowledge of k_7. Thus, in each iteration of the 7-round attack only 8 bytes from k_6 are guessed.

As we show later, $2^{90.7}$ chosen plaintexts are sufficient for the attack. Hence, we describe the results while taking this figure into account. As the partial decryption takes only $2^{90.7} \cdot 2^{128} = 2^{218.7}$ 1-round decryptions, which is less than the time complexity of the remainder of the attack, we do not optimize this step.

Analysis of Steps 3–4 of the 7-Round Attack in the 8-Round Attack The most time consuming steps of the new 8-round attack are Steps 3–4 of the 7-round attack. This step is repeated 2^{128} times, where each time the attacker has to analyze $2^{57.7}$ pairs under 2^{64} possible subkey guesses. However, the time complexity of these steps can be further reduced.

We observe that if $\Delta x^{SR}_{4,SR(Col(0))}$ has a zero difference (recall that the attack is repeated four times, once for each possible shifted column), and if x_4^{MC} has eight bytes with a zero difference, there are $2^8 - 1$ possible differences in each of the columns of x_4^{MC}. As there is a difference only in two bytes of each column, we deduce that there are only $2^{16} \cdot (2^8 - 1) \approx 2^{24}$ different pairs of actual values in the two active bytes in the pair (rather than 2^{32}). Thus, for $x^{MC}_{4,SR^{-1}(Col(2,3))}$ there are 2^{96} possible pairs of intermediate encryption values which satisfy the required differences. As we are dealing with the actual values, we can partially encrypt these values through the SubBytes operation, and the following ShiftRows operation and MC (applied to Columns (2,3) of x_5^{SR}). Given the value of $k_{5,Col(2,3)}$, the attacker is able to further compute the actual values which enter the SubBytes of round 6, and its outputs.

Hence, Steps 3–4 can be performed in a slightly different manner: For each guess of k_7, the attacker computes $k_{5,Col(1,2,3)}$. Then, for each of the 2^{96} possible actual values of $x^{SB}_{5,SR^{-1}(Col(2,3))}$, the attacker computes the respective values of $x^{SR}_{6,SR(Col(2,3))}$. Then, the attacker partially decrypts all the ciphertexts through round 7, and obtains x_6^O. For each of the $2^{57.7}$ expected pairs with zero

difference in $\Delta(x^{O}_{6,SR(Col(0,1))})$, the attacker then computes the equivalent key $w_{6,SR(Col(2,3))}$ suggested by the pair for each of the 2^{96} possible pairs of values of $x^{SR}_{6,SR(Col(2,3))}$. For each of the $2^{57.7}$ pairs, about $2^{96}/2^{64} = 2^{32}$ values of $w_{6,SR(Col(2,3))}$ are suggested.

An efficient implementation would therefore require only 2^{32} memory accesses for any remaining pair to retrieve this list of suggested $w_{6,SR(Col(2,3))}$ values. Then, each pair is added to the lists corresponding to the suggested values of $w_{6,SR(Col(2,3))}$. Steps 5 and 6 of the 7-round attack are repeated with these pairs (for a guess of k_7 and $w_{6,SR(Col(2,3))}$). We note that as there are $2^{57.7}$ pairs, each suggesting 2^{32} out of the 2^{64} keys, we expect each key to be suggested by $2^{25.7}$ pairs.

To further optimize the above attack, we note that the 2^{96} pairs of values of $x^{SR}_{6,SR(Col(2,3))}$ (computed from the 2^{96} possible pairs of values of $x^{MC}_{4,SR^{-1}(Col(2,3))}$) are not changed as long as the value of $k_{5,Col(2,3)}$ is not changed. Thus, an optimized implementation would try all possible values of k_7 in the order of the values of $k_{5,Col(2,3)}$. This reduces the total computational time of generating these 2^{96} pairs of values to about $2^{64} \cdot 2^{96}$ encryptions of two columns for two rounds, which is negligible with respect to the time complexity of the attack.

To conclude, this approach reduces the time complexity of this step to only $2^{128} \cdot 2^{57.7} \cdot 2^{32} \cdot 2 = 2^{218.7}$ memory accesses for a given shifted column with zero difference after round 4, or a total of $2^{220.7}$ memory accesses.

Reducing the Time Complexity of Step 5 of the 7-Round Attack. As in the 7-round attack, Step 5 (of the 7-round attack) can be performed efficiently using the hash table method described in [4]. The time complexity of this step is $2^{192} \cdot 2^{37.7} = 2^{229.7}$ memory accesses.

The data complexity of the attack can be reduced as the attack can tolerate wrong keys which remain with probability higher than 2^{-152}. Given the time complexity of the rest of the attack, it is sufficient to set the probability at 2^{-43}, i.e., we expect that out of the 2^{192} guesses of bytes in k_7 and w_6, only 2^{149} guesses remain. For each such guess, the attacker guesses the remaining 64 bits of w_6, computes k_6 from w_6, recovers the secret key, and tests it using trial encryptions. The time complexity of this step is close to $2^{149} \cdot 2^{64} = 2^{213}$ encryptions, which is negligible with respect to the other steps of the attack.

Another observation is that it is possible to reuse the data and repeat the 7-round attack using different pairs of columns. The attack can be repeated with $Col(1,3)$ or $Col(1,2)$ in round 5 instead of $Col(2,3)$. Thus, the attacker repeats the above analysis, assuming that there is no difference in columns 1 and 2 (or 1 and 3) of x^{O}_{5}. The attack algorithm is similar (with slight modifications of the columns and the bytes involved). Each of these attacks retrieves a candidate value for $k_{6,SR(Col(1,2))}$ (or $k_{6,SR(Col(1,3)}$) in the inner 7-round attack. As these subkeys share bits, if a candidate value is discarded in one of the attacks, it is sufficient to deduce that this value cannot be true. Hence, it is sufficient to use $2^{89.1}$ chosen plaintexts. The time complexity does not increase despite the 3 repetitions of the attack, as the data analyzed each time is reduced by a similar factor.

Summarizing the 8-round attack, the data complexity of the attack is $2^{89.1}$ chosen plaintexts, the time complexity is $2^{229.7}$ memory accesses, and the memory complexity is about 2^{101} bytes of memory (used mostly to store the table of 2^{96} pairs).

5 Summary and Conclusions

In this paper we improved the previously known impossible differential attacks on 7-round AES and presented new attacks on 8-round AES-256. This research shed more light on the security of AES, especially on the way to exploit the relatively slow diffusion in the key schedule algorithm.

We presented two attacks on 7-round AES. The first attack (applicable to AES-192 and AES-256) has a data complexity of about $2^{91.2}$ chosen plaintexts and a time complexity of $2^{139.2}$ encryptions for AES-192 (or 2^{163} memory accesses for AES-256). The second attack requires $2^{112.2}$ chosen plaintexts, and has a running time of $2^{117.2}$ memory accesses (when attacking AES-128, a slightly higher complexities are needed for AES-192 and AES-256).

We also presented two attacks on 8-round AES-256. The first and better one requires $2^{89.1}$ chosen plaintexts and has a time complexity of $2^{229.7}$ memory accesses. The second one has a slightly smaller running time, in exchange for much more data ($2^{111.1}$ chosen plaintexts and $2^{224.3}$ memory accesses).

References

1. Bahrak, B., Aref, M.R.: A Novel Impossible Differential Cryptanalysis of AES. In: Proceedings of the Western European Workshop on Research in Cryptology 2007, Bochum, Germany (2007)
2. Biham, E., Biryukov, A., Shamir, A.: Miss in the Middle Attacks on IDEA and Khufu. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 124–138. Springer, Heidelberg (1999)
3. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 12–23. Springer, Heidelberg (1999)
4. Biham, E., Keller, N.: Cryptanalysis of Reduced Variants of Rijndael (unpublished manuscript, 1999)
5. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer, Heidelberg (1993)
6. Chen, J.: Personal communications (August 2008)
7. Chen, J., Wei, Y., Hu, Y.: A New Method for Impossible Differential Cryptanalysis of 7-round Advanced Encryption Standard. In: Proceedings of International Conference on Communications, Circuits and Systems Proceedings 2006, vol. 3, pp. 1577–1579. IEEE, Los Alamitos (2006)
8. Chen, J., Hu, Y., Wei, Y.: A New Method for Impossible Differential cryptanalysis of 8-Round Advanced Encryption Standard. Wuhan University Journal of National Sciences 11(6), 1559–1562 (2006)

9. Chen, J., Hu, Y., Zhang, Y.: Impossible differential cryptanalysis of Advanced Encryption Standard. Science in China Series F: Information Sciences 50(3), 342–350 (2007)
10. Cheon, J.H., Kim, M., Kim, K., Lee, J.-Y., Kang, S.: Improved Impossible Differential Cryptanalysis of Rijndael and Crypton. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 39–49. Springer, Heidelberg (2002)
11. Daemen, J., Rijmen, V.: AES Proposal: Rijndael, NIST AES proposal (1998)
12. Daemen, J., Rijmen, V.: The design of Rijndael: AES — the Advanced Encryption Standard. Springer, Heidelberg (2002)
13. Demirci, H., Selçuk, A.A.: A Meet-in-the-Middle Attack on 8-Round AES. In: Proceedings of Fast Software Encryption 15. LNCS, vol. 5806, pp. 116–126. Springer, Heidelberg (2008)
14. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., Whiting, D.: Improved Cryptanalysis of Rijndael. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 213–230. Springer, Heidelberg (2001)
15. Gilbert, H., Minier, M.: A collision attack on 7 rounds of Rijndael. In: Proceedings of the Third AES Candidate Conference (AES3), New York, USA, pp. 230–241 (2000)
16. Kim, J., Hong, S., Preneel, B.: Related-Key Rectangle Attacks on Reduced AES-192 and AES-256. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 225–241. Springer, Heidelberg (2007)
17. Lucks, S.: Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Keys. In: Proceedings of the Third AES Candidate Conference (AES3), New York, USA, pp. 215–229 (2000)
18. Phan, R.C.-W.: Impossible Differential Cryptanalysis of 7-round Advanced Encryption Standard (AES). Information Processing Letters 91(1), 33–38 (2004)
19. Zhang, W., Wu, W., Feng, D.: New Results on Impossible Differential Cryptanalysis of Reduced AES. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 239–250. Springer, Heidelberg (2007)
20. Zhang, W., Wu, W., Zhang, L., Feng, D.: Improved Related-Key Impossible Differential Attacks on Reduced-Round AES-192. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 15–27. Springer, Heidelberg (2007)

A The Biham-Keller Technique for Efficiently Eliminating Wrong Subkeys

In [4] a technique for eliminating wrong subkey candidates in the round before the impossible differential is presented. The attack has two stages: In the precomputation stage, the attacker considers all possible pairs (z_1, z_2) of values of $x^{MC}_{0,Col(0)}$ that have difference in a single byte. For all these $2^{10} \cdot 2^{32} = 2^{42}$ pairs, the attacker computes the corresponding $x^{I}_{0,SR^{-1}(Col(0))}$ values (denoted by (w_1, w_2)), and stores in a table the values $(w_1 \oplus w_2, w_1)$. The table is sorted according to the $w_1 \oplus w_2$ values.

In the online stage, for each input pair, the attacker computes the XOR difference between the two plaintexts in the bytes $SR^{-1}(Col(0))$, and uses the table to detect the 2^{10} pairs of $x^{I}_{0,SR^{-1}(Col(0))}$ values corresponding to this difference. Since the AddRoundKey operation does not change the XOR difference between

the two plaintexts, by XORing the 2^{10} corresponding w_1 values with one of the plaintexts, the attacker gets a list of 2^{10} values of $k_{-1,SR^{-1}(Col(0))}$ that lead the plaintext pair to the input difference of the impossible differential at the beginning of Round 1.

These values are then marked in a list of all the possible $k_{-1,SR^{-1}(Col(0))}$ values. Once all the values in the list are marked, the attacker concludes that a contradiction occurred, and discards the value of the corresponding subkeys in the rounds after the impossible differential (i.e., in k_5, k_6, and k_7).

B The Bahrak-Aref Attack and Our Improvements

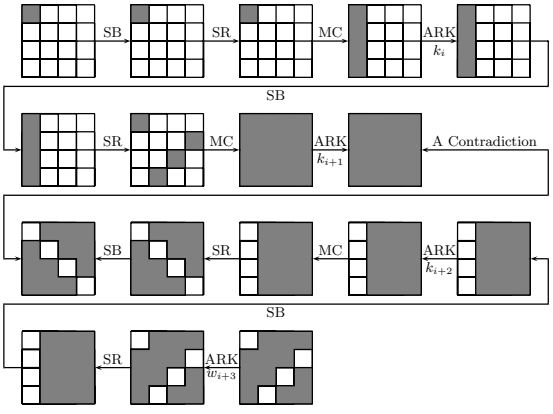
The algorithm of the BA attack, as described in [1], has the total time complexity of the attack is 2^{121} 7-round AES encryptions.[4] The data complexity of the attack is $2^{117.5}$ chosen plaintexts, and the memory complexity is 2^{109} bytes of memory required for storing the list of discarded key values.

We can improve this attack by using the following points (due to space consideration the full details are given in the full online version of this paper):

- For each candidate pair, we start by guessing the difference which is impossible, and derive from it the subkey which the pair suggests (rather than trying all subkeys for a given pair).
- In the attack algorithm there is a location where there are $2^{79.2}$ pairs, where there are only 2^{64} possible differences. As the analysis is "difference" based it is possible to analyze each difference at this stage, rather then each pair.
- It is possible to use 4 different differentials in the first round of the attack, thus increasing the number of subkey candidate discarded by a given pair (and thus reducing data complexity).
- It is also possible to repeat the attack four times according to the exact "output difference" of the impossible differential. Each of these trials have a shared 96-bit subkey value (out of 112 bits each trial analyzes). Thus, by carefully collecting the results, it is possible to even further reduce the data complexity.
- Using the subkey schedule algorithm it is possible to reduce some of the analysis work, by not analyzing subkey combinations which are impossible.
- Using the subkey schedule algorithm it is possible to perform the exhaustive search part efficiently (by guessing the last three bytes required for a trial encryption).
- Of course, the data complexity reduction also leads to a lower time complexity.

The total time complexity of the modified attack is $2^{117.2}$ memory accesses and data complexity of $2^{112.2}$ chosen plaintexts. For AES-192 and AES-256 there is a need for more data (to allow better discarding of wrong key candidates).

[4] In [1] one specific operation was considered as a full one-round decryption, while it is only 1/4 round in reality.



A gray box stands for a non-zero difference in the byte, while a white box stands for a zero difference.

Fig. 3. An Example for a 4-Round Impossible Differential of AES

For AES-192 $2^{113.8}$ chosen plaintexts are needed and the time complexity of the attack is $2^{118.8}$ memory accesses. For AES-256, similar result can be obtained, but the attacker would have to repeat his attack several times.

It is also possible to extend this attack to 8-round AES-256. Instead of guessing the last round subkey, we use similar methods as before, and slightly change the output of the impossible differential to exploit the key schedule algorithm. This along with the technique of guessing differences rather than keys and reapplying the attack 12 times (instead of only 4 as before), leads to an attack which uses $2^{111.1}$ chosen plaintexts, and has time complexity of $2^{227.8}$ memory accesses.

C The Basic Impossible Differential of AES

In Figure 3 we give the structure of the impossible differentials used in all the impossible differential attacks on AES.

Reflection Cryptanalysis of Some Ciphers

Orhun Kara*

TÜBİTAK UEKAE
National Research Institute of Electronics and Cryptology
Gebze 41470 Kocaeli/Turkey
orhun@uekae.tubitak.gov.tr

Abstract. In this paper, we provide a theoretical infrastructure of the reflection attack. In addition, we mount the reflection attack on some ciphers such as GOST, DEAL and a variant of DES. The attack method exploits certain similarities among round functions which have not been utilized in the previous self-similarity attacks. As an illustration, we introduce a chosen plaintext attack on full-round GOST under the assumption that its S-boxes are bijective. The attack works on approximately 2^{224} keys and its complexity is 2^{192} steps with 2^{32} chosen plaintexts. Also, we introduce a known plaintext attack on 30-round GOST, which works for any key. The key is recovered with 2^{224} steps by using only 2^{32} known plaintexts. As another example, we deduce that the reflection attack works on DEAL for certain keys. For instance, a 192-bit DEAL-key can be identified as a weak key by using approximately 2^{66} known plaintexts. Then, the key can be recovered with 2^{136} steps. The number of weak keys of 192-bit DEAL is roughly 2^{80}.

Keywords: Reflection attack, Slide Attack, Related Key Attack, Self-similarity, Block Cipher, Round Function, Round Key, Key Schedule, Cryptanalysis.

1 Introduction

The reflection attack was first introduced in FSE 2007 by Kara and Manap [22]. The authors mounted the reflection attack on Blowfish and described a new class of weak keys. Also, a brief description of the attack was given in its appendix [22]. However, the attack scenario is restricted through the specific high level structure of Blowfish itself. Some new descriptions of Blowfish were given by rearranging specific arithmetic operations on the round keys so as to impose certain conditions on the round keys. In this paper, we give a general view of the attack idea, consummating the specific attack on Blowfish, and supply several applications mounted on distinct ciphers such as GOST, DEAL and 2K-DES. In addition, we provide a theoretical infrastructure of the attack method.

The reflection attack is related to the slide attacks [10,11] and the related key attacks [3,28] as it is also a kind of self-similarity analysis. However, the

* Supported in part by the European Commission through the project FP-7 ICE under the project grant number 206546.

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 294–307, 2008.

principle and its assumptions are different from those in [10,11] or in [3,28]. Therefore, it is possible to apply the attack on some ciphers resistant to the previous self-similarity attacks. The reflection attack exploits certain similarities of some round functions of encryption process with those of decryption. This is the main difference from the previous self-similarity attacks, which exploit the similarities among the round functions only in encryption process. The main principle behind the reflection attack consists of exploiting a biased distribution of the fixed points of several intermediate rounds, and extending these properties to the full cipher. The reflection attack works especially well on ciphers containing involutions, since the fixed points of intermediate rounds are likely extended to the full cipher through the involutions.

The assumptions of the reflection attack are weaker than those of previous self-similarity attacks in some cases. For instance, reflection attacks are not limited to ciphers with simple key schedules. We give an example for the cipher DEAL, which has a complicated key schedule, and discover certain weak keys. A more interesting example is the reflection attack mounted on Blowfish [22].

We also apply the attack on GOST and 2K-DES (a variant of DES defined in [10]). They all are theoretical attacks with marginal workloads. We introduce a chosen plaintext attack on full-round GOST under the assumption that its S-boxes are bijective. The attack works on approximately 2^{224} keys and its complexity is 2^{192} steps with 2^{32} chosen plaintexts. In addition, we introduce a known plaintext attack on 30-round GOST which works for any key. The key is recovered with 2^{224} steps by using only 2^{32} known plaintexts. In another example, we recover 2K-DES key by using 2^{33} known plaintexts with a negligible time complexity. These attacks on GOST and 2K-DES are the best attacks known so far. Moreover, we detect that certain classes of DEAL keys are vulnerable to reflection attacks. For example, a 192-bit DEAL-key can be identified as a weak key by using approximately 2^{66} known plaintexts. Then, the key can be recovered with 2^{136} steps. The number of DEAL's weak keys is roughly 2^{80}. So, the workload of identifying a weak key is less than the cardinality of the weak-key class.

This paper is organized as follows. We introduce notations and summarize previous self-similarity analyses given in [10,11] and in [3,28] in Section 2. Then, we give a simple example of the reflection attack on GOST in section 3. The attack does not require any theoretical background. After this introductory example, the fundamental idea of the reflection attacks and the general statements are given in Section 3, including the assumptions and the description of a typical attack on Feistel networks. In the following three sections, we give some attack examples on three different ciphers. Finally, we conclude by discussing about the new security criteria of block ciphers.

2 Notation and Previous Self-similarity Analyses

We use the following notations throughout the paper. Let $E_K : GF(2)^n \rightarrow GF(2)^n$ be a block cipher of block length n defined by a key material K and D_K: $GF(2)^n \rightarrow GF(2)^n$ be its inverse mapping. Assume that E_K is a composition of some round functions:

$$E_K(x) = F_{k_r} \circ F_{k_{r-1}} \circ \cdots \circ F_{k_1}(x), x \in GF(2)^n,$$

where r is the number of rounds, $k_1, ..., k_r$ are the subkeys (round keys) and F_{k_i} is the i-th round function. Define the composite of $j - i + 1$ functions starting from i and denoted by $F_K[i, j]$ as

$$F_K[i, j] = F_{k_j} \circ \cdots \circ F_{k_i} \text{ for } 1 \leq i < j \leq r \tag{1}$$

and as identity map for $i > j$. Such functions can be called *intermediate functions*. Let $U_K(i, j)$ be the set of fixed points of the function $F_K[i, j]$. More explicitly,

$$U_K(i, j) = \{x \in GF(2)^n : F_K[i, j](x) = x\}.$$

One of the generic attack methods that exploits some degree of self-similarity is the slide attack [10,11]. In general, the typical slide attack can be applied if the sequence of round keys has a short period, such as 1, 2 or 4. For instance, if all the round keys are same, $k_i = K$, then the encryption function will be $E_K(x) = F_K^r(x) = y$. Let $F_K(x) = x'$. Encrypting x' we have $E_K(x') = y'$. Then, from these two encryptions we obtain two equations which are probably easy to solve: $F_K(x) = x'$ and $F_K(y) = y'$. Such (x, x') pairs are called *slid pairs*. The laborious part of the attack is to identify slid pairs. This basic attack can be generalized if the period of sequence of round keys is 2 (i.e., $k_i = k_{i+2}$) or 4 (i.e., $k_i = k_{i+4}$) [11].

Related key attacks proposed by Biham [3] and independently by Knudsen [28] are based on the powerful assumption that the attacker knows a relation between several keys and can access encryption function with these related keys. The goal is to find the keys. One of the most basic type of the relations defined over a pair of keys is that the i-th subkey of one key is equal to the $(i + 1)$-th subkey of the other key.

Self-similarity attacks are quite powerful cryptanalytic tools and there are several extensions of both slide attacks (e.g [4,11,17,14]) and related key attacks. In particular, they are serious attacks on block ciphers used in hash modes. Some related key attacks are combined with the well-known statistical attacks such as differential attack, linear attack, boomerang attack, square attack etc [15,5,6,7,20,26,24]. A recent study by Biham, Dunkelman and Keller unifies the classical related key attacks which use related key pairs and the composed attacks exploiting both self-similarities and statistical deviances [8].

3 Chosen Plaintext Attack on Full-Round GOST

GOST, the Russian encryption standard [36], is a 32 round 64 bit Feistel network with 256 bit key. 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 $k_0, ..., k_7, k_0, ..., k_7, k_0, ..., k_7, k_7, k_6, ..., k_1, k_0$. The round key is involved by the modular addition in the round function. We do not consider details of the round function. We only assume that it is bijective.

There is no known attack on a single key which is better than the exhaustive search for full-round GOST. A related key differential cryptanalysis is shown in [24]. The attack is impractical for properly chosen S-boxes. A slide attack has been mounted on 20 round GOST⊕, a variant of GOST defined in [11]. This attack uses 2^{33} known texts and 2^{65} memory space and the key is recovered in 2^{70} encryptions. Another related key differential attack given in [34] has been mounted on 21 round GOST. This attack has a data complexity of 2^{56} chosen plaintexts. A recent related key differential attack that uses the idea of Seki and Kaneko in [34], is mounted on GOST [25]. The attack is on full-round GOST and recovers 12 bits of the key with 2^{35} chosen plaintexts in 2^{36} steps. However, the attack is based on a powerful assumption that the attacker knows that the two related keys differ in only eight specific bits. Another recent study by Biham, Dunkelman and Keller is the improved slide attack mounted on GOST [4]. In this attack, they firstly recover the key for 24-round reduced GOST in 2^{63} steps and then extend the attack to 30-round GOST with a workload of 2^{254} encryptions by using almost the entire code book.

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 $F_K[1,8]$. In this work, we mount two reflection attacks on GOST. The first attack is a chosen plaintext attack on full-round GOST and it is successful if the key has certain properties. The number of such keys is roughly 2^{224}. The second attack is a known plaintext attack on 30-round GOST.

3.1 Description of the Attack

Assume there exists x such that x is a fixed point of both $F_K[1,8]$ and the swap operation S, i.e., $F_K[1,8](x) = x$ and $S(x) = x$. Then, x is also 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. If U is empty, then the attack is not applicable. Otherwise, for any x in U solve the equation $F_K[1,8](x) = x$ for K. Note that there are 2^{192} solutions and each of the solutions may be obtained by guessing $k_0, k_1, ..., k_5$ and then determining k_6 and k_7. Guessing $k_0, k_1, ..., k_5$, we construct a two-round Feistel network with unknown keys k_6 and k_7 and an input-output pair given as $(F_K[1,6](x), x)$. 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}, we obtain the inputs and then k_6 and k_7. Consequently, we obtain 2^{192} candidates for the key by solving $F_K[1,8](x) = x$. We recover the correct key by searching over all the candidates by roughly 2^{192} encryptions. We solve $F_K[1,8](x) = x$ for each $x \in U$. However, it is most likely that U is empty if there exists no fixed point of $F_K[1,8]$ with the equal halves. On the other hand, the number of keys satisfying that $\exists\ x$ such that $F_K[1,8](x) = x$ and $S(x) = x$ is roughly 2^{224}. Because, the

expected number of fixed points is one and the probability that any arbitrary value is a fixed point of S is 2^{-32}.

4 Generalizations of Reflection Attack

The reflection attack makes use of high level self-similarity. We compose a new function whose output matches with the output of encryption function on a large subset of input space by exploiting a biased distribution of fixed points of a properly chosen intermediate function and by extending its properties through certain involutions. The following statement plays a crucial role in the basic attack. The principle in the statement is depicted in Figure 1.

Lemma 1. *Let i, j be given such that $0 < j - i < i + j \leq r$. Assume that $F_{k_{i-t}} = F^{-1}_{k_{j+t}}$ for all $t : 1 \leq t < i$. If $F_K[i-t, i-1](x) \in U_K(i, j)$, then $x \in U_K(i-t, j+t)$ for all $t : 1 < t < i$. In addition, if $x \in U_K(i-t, j+t)$ for some $t : 1 < t < i$, then $F_K[i-t, i-1](x) \in U_K(i, j)$.*

Proof. Assume that $F_K[i-t, i-1](x) \in U_K(i, j)$. Then we have

$$\begin{aligned} F_K[i-t, j+t](x) &= F_K[j+1, j+t] \circ F_K[i, j] \circ F_K[i-t, i-1](x) \\ &= F_K[j+1, j+t] \circ F_K[i-t, i-1](x), \text{ since } F_K[i-t, i-1](x) \in U_K(i, j), \\ &= x, \text{ since } F_{k_{i-t}} = F^{-1}_{k_{j+t}} \text{ for all } t : 1 < t < i. \end{aligned}$$

Hence $x \in U_K(i-t, j+t)$. On the other hand, assume that $x \in U_K(i-t, j+t)$ for some $t : 1 < t < i$. Then the input of $F_K[i, j]$ is $F_K[i-t, i-1](x)$ and the output of $F_K[i, j]$ is $F_K^{-1}[j+1, j+t](x)$. However, $F_K^{-1}[j+1, j+t] = F_K[i-t, i-1]$ since we assume that $F_{k_{i-t}} = F^{-1}_{k_{j+t}}$. Hence, $F_K[i-t, i-1](x) \in U_K(i, j)$. □

The following corollary can lay the groundwork for an attack on product ciphers whose some round functions in the encryption are equal to some round functions in the decryption. The corollary works for the ciphers whose first $(i+j-1)$-round iteration has many fixed points. In this case, the first $i + j - 1$ rounds may be disregarded during the encryption of fixed points.

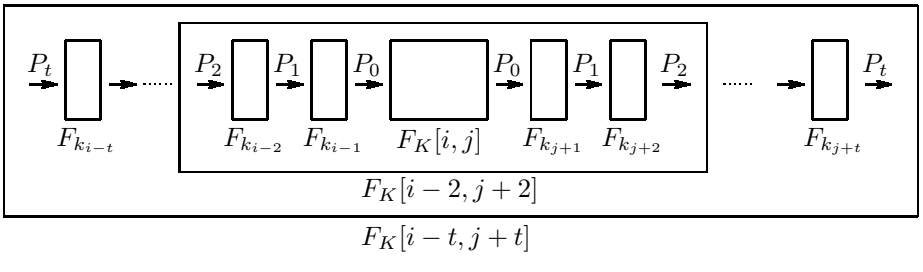


Fig. 1. The reflection property with palindromic round keys given in Lemma 1. The fixed point, P_0, of the intermediate function $F_K[i, j]$ is extended to the fixed point, P_t, of $F_K[i-t, j+t]$ through the involutions under the assumption that $F_{k_{i-t}} = F^{-1}_{k_{j+t}}$.

Corollary 1. *Let i, j be given such that $0 < j - i < i + j \leq r$. Assume that we have $F_{k_{i-t}} = F^{-1}_{k_{j+t}}$ for all $t : 1 < t < i$. Then, the encryption function E_K is equal to the function $F_K[i+j, r]$ on the pre-image set, $F_K^{-1}[1, i-1](U_K(i, j))$.*

Proof. The set $F_K^{-1}[1, i-1](U_K(i, j))$ is equal to $U_K(1, j+i-1)$ by Lemma 1. On the other hand, we have $F_K[1, j+i-1](x) = x$ for $x \in U_K(1, j+i-1)$ by definition. Thus,

$$E_K(x) = F_{k_r} \circ \cdots \circ F_{k_1}(x) = F_K[i+j, r] \circ F_K[1, i+j-1](x) = F_K[i+j, r](x). \square$$

Corollary 1 states that there exists another function which equals to the encryption function on some special subset of the encryption space. This function is probably much weaker than the encryption function since its number of rounds may be much less than r. Then, the attack given below recovers the round keys $k_{i+j}, ..., k_r$ by solving the system of equations

$$F_K[i+j, r](x) = E_K(x) = y. \tag{2}$$

We need m elements of $F_K^{-1}[1, i-1](U_K(i, j))$ for solving Equation 2. The expected number of elements in $F_K^{-1}[1, i-1](U_K(i, j))$ chosen randomly among t plaintexts is $(t \cdot |U_K(i, j)|)/2^n$. So, t should be approximately $(m \cdot 2^n)/(|U_K(i, j)|)$ in order to get about m elements of $F_K^{-1}[1, i-1](U_K(i, j))$. Each plaintext ciphertext pair gives an equation in the form of Equation (2). However, only approximately m of these equations are the correct equations. One may try all the plaintexts to solve Equation (2) exhaustively. However, if the conditional probabilities, $\Pr(F_K[1, i-1](x) \in U_K(i, j) \,|\, (x, E_K(x)))$, are large enough for some x, then it is more likely that the corresponding equations are correct. Choose ℓ plaintexts, $x_1, ..., x_\ell$, such that the sum of all the ℓ probabilities corresponding to x_i's is roughly m. This choice provides approximately m correct equations among these ℓ equations.

The correct equation set can be obtained by trying all subsets of m elements of $\{x_1, ..., x_\ell\}$. Since the search should be sorted according to the probabilities of subsets, we get an upper bound for the time complexity which is $\binom{\ell}{m} \cdot C$, where C is the time complexity of solving one equation set. This is a typical reflection attack. There are three main parameters which specify the complexity of the attack:

1. m: The number of required pairs (x, y) to solve Equation (2). By solving, we mean that there exists a unique solution if all of the m equations are correct and a contradiction (no solution) otherwise.
2. $|U_K(i, j)|$: The cardinality of $U_K(i, j)$.
3. $\Pr(F_K[1, i-1](x) \in U_K(i, j) \,|\, (x, E_K(x)))$: The probability that $F_K[1, i-1](x)$ is in $U_K(i, j)$ given $E_K(x)$.

The probability that $F_K[1, i-1](x)$ is in $U_K(i, j)$ is $(|U_K(i, j)|)/2^n$ for randomly chosen x. However, for given particular values of $E_K(x)$, the probability may be much greater or less than $(|U_K(i, j)|)/2^n$ depending on the structure of a cipher. The following statement summarize the attack.

Theorem 1. *Assume that we need m pairs to solve Equation (2) and let C be the time required for solving it in terms of the number of encryptions. Then, the attack to recover the round keys $k_{i+j}, ..., k_r$ has a data complexity of $(m \cdot 2^n)/|U_K(i,j)|$ known plaintexts. Assume the probabilities $\Pr(F_K[1,i-1](x) \in U_K(i,j) \,|\, (x, E_K(x)))$ are pre-calculated and the greatest ℓ probabilities are chosen among $(m \cdot 2^n)/|U_K(i,j)|$ plaintexts so that*

$$\sum_{s=1}^{\ell} \Pr(F_K[1,i-1](x_s) \in U_K(i,j) \,|\, (x_s, E_K(x_s))) \approx m. \tag{3}$$

Then, the attack has a time complexity which is at most $\binom{\ell}{m} \cdot C$ number of encryptions.

Let us note that the false alarm probability of the attack is disregarded in Theorem 1 since we assume that the solution set is empty if at least one of the equations is incorrect.

4.1 Reflection Attack on Feistel Networks

Let a plaintext $x \in GF(2)^n$ be given as $x = (x_0, x_1); x_0, x_1 \in GF(2)^{n/2}$. The Feistel structure can be stated as a recursive function defined as $x_i = R_{k_{i-1}}(x_{i-1}) \oplus x_{i-2}$ with the initial conditions given by $x = (x_0, x_1)$. The function $R : GF(2)^{n/2} \to GF(2)^{n/2}$ is the encryption function and $\oplus$ is the "XOR" operation. The i-th round operation is defined as

$$F_{k_i}(x_{i-1}, x_i) = (x_i, x_{i+1}) = (x_i, R_{k_i}(x_i) \oplus x_{i-1}) \tag{4}$$

for $i \leq r$. In general, the swap operation is excluded in the last round and (x_{r+1}, x_r) is the corresponding ciphertext. With some abuse of terminology, R is also called the round function. We call the stream $x_0, x_1, ..., x_r, x_{r+1}$ the *encryption stream* of $x = (x_0, x_1)$ with respect to K.

Proposition 1 ([13]). *For a natural number $m < r$, assume that $k_{m-i} = k_{m+i}$, $\forall i : 1 \leq i \leq \min\{r-m, m-1\}$. Let $x = (x_0, x_1)$ be encrypted and $x_0, x_1, ..., x_r, x_{r+1}$ be its encryption stream. If $R_{k_m}(x_m) = 0$, then $x_{m-i} = x_{m+i}$, $\forall i : 1 \leq i \leq \min\{r-m, m-1\}$. Conversely, if $x_{m-i} = x_{m+i}$ and $x_{m-i+1} = x_{m+i-1}$ for some i, then $R_{k_m}(x_m) = 0$.*

Proposition 1 has already been known during the studies on cycle structures of DES (see [13,32]). Hence, the notion of the fixed points of the weak keys of DES is well known. However, the studies focused on the algebraic properties of DES permutations and their short cycles rather than developing a key recovery attack [13,32,21,31]. The following corollary points out the opposite direction of this old phenomenon.

Corollary 2. *Assume that each round key k_i determines a round function R_{k_i} randomly. Let $x = (x_0, x_1)$ be encrypted and $x_0, x_1, ..., x_r, x_{r+1}$ be its encryption*

stream. Assume that the round number r is even, $r = 2r' > 4$, and $k_{r'-i} = k_{r'+i}$ $\forall i : 1 \le i < r'$. Let $s_{r'}$ be the cardinality of the pre-image set of the zero output, $R^{-1}_{k_{r'}}(0)$. Then, $\Pr(x_0 = x_r) = 2^{-\frac{n}{2}}(s_{r'} + 1 - 2^{-\frac{n}{2}} \cdot s_{r'})$ and

$$\Pr(R_{k_{r'}}(x_{r'}) = 0 \,|\, x_0 = x_r) = \frac{s_{r'}}{s_{r'} + 1 - 2^{-\frac{n}{2}} \cdot s_{r'}}.$$

Proof. Assume that the round function is random. Then, the probability that $x_0 = x_r$ is given as

$$\begin{aligned}\Pr(x_0 = x_r) &= \Pr(x_0 = x_r \,|\, R_{k_{r'}}(x_{r'}) = 0)\Pr(R_{k_{r'}}(x_{r'}) = 0) \\ &\quad + \Pr(x_0 = x_r \,|\, R_{k_{r'}}(x_{r'}) \neq 0)\Pr(R_{k_{r'}}(x_{r'}) \neq 0) \\ &= s_{r'} \cdot 2^{-n/2} + 2^{-n/2}(1 - s_{r'} \cdot 2^{-n/2}) = 2^{-\frac{n}{2}}(s_{r'} + 1 - s_{r'} \cdot 2^{-\frac{n}{2}}).\end{aligned}$$

Note that $\Pr(x_0 = x_r \,|\, R_{k_{r'}}(x_{r'}) \neq 0) = 2^{-n/2}$ since $r > 4$. On the other hand, $\Pr(x_0 = x_r \,|\, R_{k_{r'}}(x_{r'}) = 0) = 1$ by Proposition 1. Hence, we conclude that

$$\begin{aligned}\Pr(R_{k_{r'}}(x_{r'}) = 0 \,|\, x_0 = x_r) &= \frac{\Pr(x_0 = x_r \,|\, R_{k_{r'}}(x_{r'}) = 0) \cdot \Pr(R_{k_{r'}}(x_{r'}) = 0)}{\Pr(x_0 = x_r)} \\ &= \frac{s_{r'}}{s_{r'} + 1 - 2^{-\frac{n}{2}} \cdot s_{r'}}.\end{aligned}$$

□

The following theorem illustrates the property exploited by the attack on a Feistel network with palindromic round keys.

Theorem 2. *Assumptions are as in Corollary 2. Then the equality $x_0 = x_r$ implies that the equation*

$$x_1 = R_{k_r}(x_r) \oplus x_{r+1}. \tag{5}$$

is true with probability

$$\frac{s_{r'}}{s_{r'} + 1 - 2^{-\frac{n}{2}} \cdot s_{r'}}.$$

Proof. Assume that $x_0 = x_r$. Then by Corollary 2, we have $R_{k_{r'}}(x_{r'}) = 0$ with probability

$$\frac{s_{r'}}{s_{r'} + 1 - 2^{-\frac{n}{2}} \cdot s_{r'}}.$$

Thus, the equality $x_1 = x_{r-1}$ is true with the same probability by Proposition 1. On the other hand $x_{r+1} = R_{k_r}(x_r) \oplus x_{r-1}$. Thus, the probability that $x_1 = R_{k_r}(x_r) \oplus x_{r+1}$ is

$$\frac{s_{r'}}{s_{r'} + 1 - 2^{-\frac{n}{2}} \cdot s_{r'}}.$$

□

Reflection Attack on Feistels. Note that the parameters in Theorem 2 are all public except the last round key. (x_0, x_1) forms the plaintext and (x_{r+1}, x_r) forms the corresponding ciphertext. So, Theorem 2 leads to a straightforward attack: Encrypt some plaintexts and collect those such that $x_0 = x_r$. If the round keys satisfy that $k_{\frac{r}{2}-i} = k_{\frac{r}{2}+i}$, then the corresponding equations, $x_1 = R_{k_r}(x_r) \oplus x_{r+1}$, are

correct with probability roughly $s_{r'}/(s_{r'}+1)$ (this probability is almost one half if the round function is a permutation) for the collected plaintexts by Theorem 2. Most probably, these equations are easy to solve. The last round key is recovered by solving these equations. One may apply the attack several times with properly chosen parameters or use the key schedule to recover the main key.

5 Known Plaintext Attack on 30-Round GOST

Consider 30-round GOST by eliminating the first two rounds. Then, the encryption function, $E_K^{(30)}$, is given as

$$E_K^{(30)}(x) = F_K[8,1] \circ S \circ F_K^2[1,8] \circ F_K[3,8](x)$$

where $F_K[8,1]$ is the inverse of $F_K[1,8]$. Recall that S has 2^{32} fixed points, which are all the vectors whose left halves equal their right halves. Take S as the intermediate function. Then, $F_K[8,1] \circ S \circ F_K[1,8]$ has also 2^{32} fixed points by Lemma 1. Therefore, if we encrypt 2^{32} arbitrary plaintexts, then we expect that one of the ciphertexts is a fixed point of $F_K[8,1] \circ S \circ F_K[1,8]$. Assume that y is a fixed point of $F_K[8,1] \circ S \circ F_K[1,8]$ and x is the corresponding plaintext. Then, we have $E_K^{(30)}(x) = y = F_K[1,8] \circ F_K[3,8](x)$. Solve the equation, $y = F_K[1,8] \circ F_K[3,8](x)$ for K. Note that the equation has 2^{192} solutions. Guessing the subkeys, $k_2, k_3, ..., k_7$, we obtain a two-round Feistel network with keys k_0 and k_1, and an input/output pair given as $(F_K[3,8](x), F_K[8,3](y))$. Then, as in the case of chosen ciphertext attack, recover k_0 and k_1 by reversing the round functions. Then, one immediate check is whether y is a fixed point of $F_K[8,1] \circ S \circ F_K[1,8]$ by checking $F_K[1,8](y)$ has equal halves. All the 2^{32} plaintext/ciphertext pairs are checked and we expect that one of the ciphertexts is a fixed point and hence the corresponding equation, $y = F_K[1,8] \circ F_K[3,8](x)$, is correct. Then, the correct key will be one of the 2^{192} candidates. In conclusion, we recover the key with at most 2^{224} encryptions by using only 2^{32} known plaintexts.

Remark 1. It is believed that GOST is less secure without the twist in the order of round keys. In [11], it is concluded that the twist of GOST hinders the slide attacks. However, it is surprising that the reflection attack exploits this twist property of GOST.

6 Cryptanalysis of 2K-DES

2K-DES is one of the modified DES examples given in [10]. 2K-DES uses two independent 48 bit keys K_1 and K_2 and has a very simple key schedule. K_1 is used in the odd-numbered rounds and K_2 is used in the even-numbered rounds. The total number of rounds is 64. It is most likely that 2K-DES resists to the conventional differential [9] and linear attacks [30] due to its increased number of rounds. Biryukov and Wagner have proposed a slide attack with a complexity independent of the number of rounds [10]. The attack uses 2^{32} known plaintexts

and its time complexity is 2^{50} 2K-DES encryptions. Then, it is improved to 2^{33} steps by applying complementation slide techniques [11].

Observe that $k_{32-i} = k_{32+i}$ and $k_{33-i} = k_{33+i}$ for $i = 1, ..., 31$ (note that this condition is weaker than that of slide attack in [10]). Hence, one can apply reflection attack to both encryption function and decryption function. Assume that $s_{32} = s_{33} = 1$ for a given key.

We need to find one plaintext $x = (x_0, x_1)$ satisfying $x_{64} = x_0$ and another plaintext $x' = (x'_0, x'_1)$ satisfying $x'_{65} = x'_1$. The former gives the equation $x_1 = R_{K_2}(x_{64}) \oplus x_{65}$ and the latter gives $x'_{64} = R_{K_1}(x'_1) \oplus x'_0$. Each equation is true with a probability of nearly one half and one needs approximately 2^{32} known plaintexts to get approximately four equations by Theorem 2. Two of them are from the encryption direction, whereas the other two equations are from the decryption direction. Two equations deduced from $x_{64} = x_0$ give at most 2^{17} candidates for K_2 whereas other two equations deduced from $x'_{65} = x'_1$ give at most 2^{17} candidates for K_1. One may get the correct K_1 and K_2 by searching over these solution sets exhaustively with 2^{34} 2K-DES encryptions. As a result, the reflection attack on 2K-DES uses 2^{32} known plaintexts and recovers the keys in 2^{34} steps.

It is obvious that the attack can be improved further by increasing the amount of plaintexts. If we use 2^{33} plaintexts, then we expect four equations for each key and two of them to be correct. It is most likely that two correct equations out of four equations give a unique solution and we get no solution for the other two equations. Hence, the time complexity in encryption is $2 \cdot \binom{4}{2} \cdot C$ by Theorem 1 where $C = 2/64 = 2^{-5}$ encryption if the four equations are given. The equations can be obtained by 2^{33} checking whether the left half of a plaintext is equal to the right half of the corresponding ciphertext. Observe that the attack works for the keys whose round functions produce 0-string output.

7 Weak Keys of DEAL

DEAL is a 128 bit block cipher designed by Knudsen [27] and submitted to the AES contest. It is a Feistel network and accepts three different key sizes, namely 128-bit (for 6 rounds), 192-bit (for 6 rounds) and 256-bit (for 8 rounds). DEAL makes use of DES as its round function.

There are some impractical attacks on DEAL. The attack by Knudsen [27] is a meet-in-the-middle attack and requires unrealistically many chosen plaintexts and unrealistic amount of memory. In [29], Lucks uses similar techniques and mounts chosen ciphertext attack on DEAL. A trade-off is given between the number of plaintext/ciphertext pairs and the time complexity. In [23], Kelsey and Schneier discuss the existence of equivalent keys and mount a related key attack.

We mount the reflection attack on DEAL when the key satisfies some conditions. We briefly describe DEAL and explain the attack for 128 bit key-length. The attacks are quite similar for the other cases of key lengths.

DEAL-128 uses 128 bit key K, divided into two 64-bit parts as K_1 and K_2. The six round keys, $RK_1, ..., RK_6$, are computed by using DES as $RK_i = E_C(K_{(i-1 \bmod 2)+1} \oplus RK_{i-1} \oplus s_i)$ where E is the DES encryption, C is a 56-bit

public constant used as a DES key in the key schedule, $RK_0 = 0$ and s_i's are 64-bit constants. Only 56 bits of each RK_i is used in the i-th round of DEAL which we denote $RED(RK_i)$ (reduction of RK_i to 56 bits). Note that the final round ends with a swap.

Assume that $RED(RK_2) = RED(RK_6)$ and $RED(RK_3) = RED(RK_5)$. The probability that these equalities hold is roughly 2^{-112}. In this case, the last five rounds of DEAL has 2^{64} fixed points (without the last swap). Applying the reflection attack similar to 2K-DES, we obtain approximately eight equations for the first round encryption by collecting the plaintexts whose left parts are equal to the left parts of their corresponding ciphertexts among 2^{66} known plaintexts. This will be enough to decide that the equalities $RED(RK_2) = RED(RK_6)$ and $RED(RK_3) = RED(RK_5)$ hold, otherwise we would expect approximately four plaintexts whose left parts are equal to the left parts of their corresponding ciphertexts.

Four of the eight equalities are expected to come from the fixed points. Hence, we can recover 56 bits of RK_1 by making search over all possible values of $RED(RK_1)$ and checking whether approximately four of the equations, $E_{RK_1}(x) = y$, hold. Recovering 56 bits of RK_1 yields 56-bit information about the first 64 bit part of the main key, K_1. The remaining unknown key bits may be obtained by applying several attacks on 5-round DEAL (see [27,29]). However, the simplest way is just to make search on the remaining bits. So, the time complexity is around 2^{72} steps.

We have the same data complexity for DEAL-192 and DEAL-256. This data complexity is also the workload for identifying a weak key. On the other hand, the time complexities of the reflection attacks on DEAL-192 and DEAL-256 are approximately 2^{136} and 2^{200} steps, respectively (this is the complexity of searching the remaining bits after recovering 56 bits of a key). Note that we have three equalities instead of two when the key length is 256 bits. Hence, the probability that the equalities hold is roughly 2^{-168} in this case. As a result, the attack is successful only for 192-bit and 256-bit key lengths since the number of weak keys are roughly 2^{80} and 2^{88}, respectively and identifying a weak key costs less than the number of weak keys. Notice that such attacks which work on some subset of the key space can be considered successful if identifying a weak key costs less than the number of weak keys and recovering the key in this case costs less than the complexity of the exhaustive search. Indeed, the total cost of recovering a weak key among several keys, if exists, consists of the cost of identifying the weak key with the cost of recovering it. Therefore, the reflection attack is unsuccessful on 128-bit DEAL since the cost of identifying weak key exceeds the number of weak keys.

8 Discussion

Some security criteria have been imposed on the lengths of parameters of a stream cipher such as IV length [19] and internal state size [1,16]. The corresponding criterion on block ciphers is that the block length should be at least

as large as the key length if it is operated in a stream mode in order to supply resistance to tradeoff attacks. This is necessary also against distinguishing attacks. Besides, observe that relatively much smaller block length of GOST is also exploited in the reflection attack.

We illustrate some examples supporting that the assumptions about the independence of the round functions in the security proofs given in [35,2] are not only sufficient but also necessary. Some classifications of key schedules have been proposed in [12,18] according to the independence degree of round keys. It was argued that AES key schedule was surprisingly simple and a new key schedule was proposed for AES in [18]. On the other hand, the key scheduling process of Blowfish is complex (see [33]), but some self-similarity attacks work in some special cases [22,10]. This is due to the high degree of similarity of the functions used to produce round keys, whereas these functions themselves are highly complicated and nonlinear.

Acknowledgements. I thank Esen Akkemik, Hüseyin Demirci, Orr Dunkelman, Nezih Geçkinli, İsmail Güloğlu, Atilla Arif Hasekioğlu, Cevat Manap, Raphael C.-W. Phan and Ali Aydın Selçuk for their constructive comments.

References

1. Babbage, S.: Improved Exhaustive Search Attacks on Stream Ciphers. In: IEE Conference publication European Convention on Security and Detection, vol. 408, pp. 161–166. IEE (1995)
2. Bellare, M., Rogaway, P.: The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Heidelberg (2006)
3. Biham, E.: New Types of Cryptanalytic Attacks Using Related Keys. J. of Cryptology 7, 229–246 (1994)
4. Biham, E., Dunkelman, O., Keller, N.: Improved Slide Attacks. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 153–166. Springer, Heidelberg (2007)
5. Biham, E., Dunkelman, O., Keller, N.: Related-Key Boomerang and Rectangle Attacks. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 507–525. Springer, Heidelberg (2005)
6. Biham, E., Dunkelman, O., Keller, N.: New Cryptanalytic Results on IDEA. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 412–427. Springer, Heidelberg (2006)
7. Biham, E., Dunkelman, O., Keller, N.: A Simple Related-Key Attack on the Full SHACAL-1. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 20–30. Springer, Heidelberg (2006)
8. Biham, E., Dunkelman, O., Keller, N.: A Unified Approach to Related-Key Attacks. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 73–96. Springer, Heidelberg (2008)
9. Biham, E., Shamir, A.: Differential Cryptanalysis of Data Encryption Standard. Springer, Heidelberg (1993)
10. Biryukov, A., Wagner, D.: Slide Attacks. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 245–259. Springer, Heidelberg (1999)

11. Biryukov, A., Wagner, D.: Advanced Slide Attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 589–606. Springer, Heidelberg (2000)
12. Carter, G., Dawson, E., Nielsen, L.: Key Schedules of Iterated Block Ciphers. In: Boyd, C., Dawson, E. (eds.) ACISP 1998. LNCS, vol. 1438, pp. 80–89. Springer, Heidelberg (1998)
13. Coppersmith, D.: The Real Reason for Rivest's Phenomenon. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 535–536. Springer, Heidelberg (1985)
14. Courtois, N., Bard, G.V., Wagner, D.: Algebraic and Slide Attacks on KeeLoq. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 89–104. Springer, Heidelberg (2008)
15. Dunkelman, O., Keller, N., Kim, J.: Related-Key Rectangle Attack on the Full SHACAL-1. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 28–44. Springer, Heidelberg (2007)
16. Golić, J.: Cryptanalysis of Alleged A5 Stream Cipher. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 239–255. Springer, Heidelberg (1997)
17. Furuya, S.: Slide Attacks with a Known-Plaintext Cryptanalysis. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 214–225. Springer, Heidelberg (2002)
18. Henricksen, M.: Design, Implementation and Cryptanalysis of Modern Symmetric Ciphers. PhD Thesis, ISRC, Faculty of Information Technology, Queensland University of Technology (2005)
19. Hong, J., Sarkar, P.: Rediscovery of the Time Memory Tradeoff. In: Cryptology ePrint Archive, Report 2005/090 (2005)
20. Hong, S., Kim, J., Kim, G., Lee, S., Preneel, B.: Related-Key Rectangle Attacks on Reduced Versions of SHACAL-1 and AES-192. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 368–383. Springer, Heidelberg (2005)
21. Kaliski, B.S., Rivest, R.L., Sherman, T.: Is DES a Pure Cipher? (Results of More Cycling Experiments on DES). In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 212–222. Springer, Heidelberg (1986)
22. Kara, O., Manap, C.: A new class of Weak Keys for Blowfish. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 167–180. Springer, Heidelberg (2007)
23. Kelsey, J., Schneier, B.: Key-Schedule Cryptanalysis of DEAL. In: Heys, H.M., Adams, C.M. (eds.) SAC 1999. LNCS, vol. 1758, pp. 118–134. Springer, Heidelberg (2000)
24. Kelsey, J., Schneier, B., Wagner, D.: Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 237–251. Springer, Heidelberg (1996)
25. Ko, Y., Hong, S., Lee, W., Lee, S., Kang, J.: Related Key Differential Attacks on 27 Rounds of XTEA and Full-Round GOST. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 299–316. Springer, Heidelberg (2004)
26. Kim, J., Hong, S., Preneel, B.: Related-Key Rectangle Attacks on Reduced AES-192 and AES-256. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 225–241. Springer, Heidelberg (2007)
27. Knudsen, L.: DEAL - a 128-Bit Block Cipher, http://www.ii.uib.no/~larsr/aes.html
28. Knudsen, L.: Cryptanalysis of LOKI91. In: Zheng, Y., Seberry, J. (eds.) AUSCRYPT 1992. LNCS, vol. 718, pp. 196–208. Springer, Heidelberg (1993)
29. Lucks, S.: On the Security of 128-Bit Block Cipher DEAL. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 60–70. Springer, Heidelberg (1999)
30. Matsui, M.: Linear Cryptanalysis Method of DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)

31. Moore, J.H., Simmons, G.J.: Cycle Structure of the DES with Weak and Semi-Weak Keys. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 9–32. Springer, Heidelberg (1987)
32. Moore, J.H., Simmons, G.J.: Cycle Structure of the DES for Keys Having Palindromic (or Antipalindromic) Sequences of Round Keys. IEEE Transactions on Software Engineering 13, 262–273 (1987)
33. Schneier, B.: Description of a New Variable - Length Key, 64 Bit Block Cipher (Blowfish). In: Anderson, R. (ed.) FSE 1993. LNCS, vol. 809, pp. 191–204. Springer, Heidelberg (1994)
34. Seki, H., Kaneko, T.: Differential Cryptanalysis of Reduced Rounds of GOST. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012, pp. 315–323. Springer, Heidelberg (2001)
35. Vaudenay, S.: Decorrelation: A Theory for Block Cipher Security. J. of Cryptology 16(4), 249–286 (1985)
36. Zabotin, I.A., Glazkov, G.P., Isaeva, V.B.: Cryptographic Protection for Information Processing Systems. Cryptographic Transformation Algorithm. In: Government Standard of the USSR, GOST 28147-89 (1989)

A Differential-Linear Attack on 12-Round Serpent

Orr Dunkelman[1,⋆], Sebastiaan Indesteege[2,⋆⋆], and Nathan Keller[3,⋆⋆⋆]

[1] École Normale Supérieure
Département d'Informatique,
CNRS, INRIA
45 rue d'Ulm, 75230 Paris, France
orr.dunkelman@ens.fr
[2] Katholieke Universiteit Leuven
Department of Electrical Engineering ESAT/SCD-COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
sebastiaan.indesteege@esat.kuleuven.be
[3] Einstein Institute of Mathematics, Hebrew University.
Jerusalem 91904, Israel
nkeller@math.huji.ac.il

Abstract. Serpent is an SP Network block cipher submitted to the AES competition and chosen as one of its five finalists. The security of Serpent is widely acknowledged, especially as the best known attack so far is a differential-linear attack on only 11 rounds out of the 32 rounds of the cipher.

In this paper we introduce a more accurate analysis of the differential-linear attack on 11-round Serpent. The analysis involves both theoretical aspects as well as experimental results which suggest that previous attacks had overestimated complexities. Following our findings we are able to suggest an improved 11-round attack with a lower data complexity. Using the new results, we are able to devise the first known attack on 12-round Serpent.

1 Introduction

Serpent [1] is one of the five block ciphers chosen as AES finalists. The cipher has an SP Network structure repeating 32 rounds consisting of 4-bit to 4-bit S-boxes and a linear transformation. The block size is 128 bits, and the supported key size is of any length between 0 and 256 bits.

Since its introduction, Serpent was the target of extensive cryptanalytic efforts [5,6,7,9,13]. Despite that, the best previously known attack is on 11-round

⋆ The first author was supported by the France Telecome Chaire. Some of the work presented in this paper was done while the first author was staying at K.U. Leuven.
⋆⋆ F.W.O. Research Assistant, Fund for Scientific Research – Flanders (Belgium). Also supported by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).
⋆⋆⋆ The research presented in this paper was supported by the Adams fellowship.

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 308–321, 2008.

Serpent. In [13] a 256-bit key variant of 9-round Serpent is attacked using the amplified boomerang attack. The attack requires 2^{110} chosen plaintexts and its time complexity is 2^{252} 9-round Serpent encryptions.

In [5] the rectangle attack is applied to 256-bit key 10-round Serpent. The attack uses $2^{126.8}$ chosen plaintexts and has a time complexity of 2^{217} memory accesses.[1] The 10-round rectangle attack is improved in [7] and the improved attack requires $2^{126.3}$ chosen plaintexts with time complexity of $2^{173.8}$ memory accesses. A similar boomerang attack which requires almost the entire code book is also presented in [7].

In [6] a linear attack on 11-round Serpent is presented. The attack exploits a 9-round linear approximation with bias of 2^{-58}. The attack requires data complexity of 2^{118} known plaintexts and time complexity of 2^{214} memory accesses.

The linear approximation presented in [6] is combined with a differential in [9] to construct a differential-linear attack on 11-round Serpent. The data complexity of this attack is $2^{125.3}$ chosen plaintexts and the time complexity is about $2^{139.2}$ 11-round Serpent encryptions. The first attack on 10-round Serpent with 128-bit keys is also presented in [9]. The 10-round attack requires $2^{107.2}$ chosen plaintexts and $2^{125.2}$ 10-round Serpent encryptions.

We note that Serpent is also very common example in research about the use of multiple linear approximations in linear cryptanalysis [11,12]. This line of research actually shows that the use of multiple linear approximations can give a great advantage from the data complexity point of view, but not necessarily from the time complexity point of view.

In this paper we present a more accurate analysis of the 11-round attack from [9], showing that the attack requires less data than previously believed (namely, $2^{121.8}$ chosen plaintexts). This leads to an immediate reduction in the time complexity of the attack (to $2^{135.7}$ encryptions). We then switch the order of the differential and the linear parts in the differential-linear approximation. The new 9-round differential-linear approximation is used to construct a new 11-round attack that uses $2^{113.7}$ chosen ciphertexts and has a running time of $2^{137.7}$ memory accesses.

The reduced data and time complexities allow to extend the 11-round attack from [9] by one extra round, and obtain the first 12-round attack on Serpent. This attack requires $2^{123.5}$ chosen plaintexts and has a time complexity of $2^{249.4}$ encryptions.

Finally, we present a novel related-key attack applicable to a modified variant of Serpent in which the round constants are removed from the key schedule algorithm. We note that, while the removal of these constants changes the cipher into a more symmetric structure, the repeated core, i.e., 8-round Serpent, is still relatively secure. The (still) non-trivial key schedule and the strong keyed permutation make most related-key attacks are quite likely to fail.

We organize this paper as follows: In Section 2 we present a short description of Serpent. Section 3 describes the differential-linear technique. We present the

[1] In [5] a different number is quoted, but in [7] this mistake is identified, and the correct time complexity of the algorithm is presented.

differential-linear attacks of this paper (the improved 11-round attack and the new 12-round attack) in Section 4. Section 5 describes a related-key attack on a modified variant of Serpent. We summarize our results and compare them with previous results on Serpent in Section 6. The appendices contain the differentials and the linear approximation used in the attacks.

2 A Description of Serpent

In [1] Anderson, Biham and Knudsen presented Serpent. Serpent has a block size of 128 bits and it accepts 0–256 bit keys. Serpent is an SP Network block cipher with 32 rounds. Each round is composed of key mixing, a layer of S-boxes and a linear transformation. There is an equivalent bitsliced description which is more efficient and easier to describe.

In our description we adopt the notations of [1] in the bitsliced version. The intermediate value before round i is denoted by $\hat{B}_i$ (a 128-bit value), where the 32 rounds are numbered $0, 1, \ldots, 31$. Each $\hat{B}_i$ is composed of four 32-bit words X_0, X_1, X_2, X_3.

Serpent uses a set of eight 4-bit to 4-bit S-boxes. Each round function R_i uses a single S-box applied 32 times in parallel. For example, R_0 uses 32 copies of S_0 in parallel. The first copy of S_0 takes the least significant bits from X_0, X_1, X_2, X_3 and returns the output to these bits. The set of eight S-boxes is used four times. S_0 is used in round 0, S_1 is used in round 1, etc. After using S_7 in round 7, S_0 is used again in round 8, then S_1 in round 9, and so on. In the last round (round 31) the linear transformation is omitted and another key is XORed.

The cipher may be formally described by the following equations:

$$\begin{aligned} \hat{B}_0 &:= P \\ \hat{B}_{i+1} &:= R_i(\hat{B}_i) \qquad i = 0, \ldots, 31 \\ C &:= \hat{B}_{32} \end{aligned}$$

where

$$\begin{aligned} R_i(X) &= LT(\hat{\mathcal{S}}_i(X \oplus \hat{K}_i)) \qquad \text{For } i = 0, \ldots, 30 \\ R_i(X) &= \hat{\mathcal{S}}_i(X \oplus \hat{K}_i) \oplus \hat{K}_{32} \qquad \text{For } i = 31 \end{aligned}$$

where $\hat{\mathcal{S}}_i$ is the application of the S-box $S_{(i \bmod 8)}$ thirty two times in parallel, and LT is the linear transformation of Serpent.

As our attack do not use explictly the properties of the linear transformation or the key schedule algorithm, we omit their description and refer the interested reader to [1].

3 Differential-Linear Cryptanalysis

Differential cryptanalysis [2] analyzes ciphers by studying the development of differences through the encryption process. A differential attack is mostly concerned with an input difference Ω_P for which an output difference Ω_T holds with

high enough probability (even though there are variants which use the fact that the probability is zero).

Linear cryptanalysis [16] analyzes the cipher by approximating the encryption process in a linear manner. The attacker finds a linear approximation $\lambda_P \cdot P \oplus \lambda_T \cdot T$ which holds with probability $1/2 + q$ (q might be negative) and gathers many plaintexts and ciphertexts. By checking whether the approximation holds, one can deduce subkey information or distinguish the cipher from a random permutation.

In 1994, Langford and Hellman [15] showed that both kinds of analysis can be combined together in a technique called *differential-linear cryptanalysis*. The attack uses a differential that induces a linear relation between two intermediate encryption values with probability one. In [8,14] this technique is extended to the cases where the probability of the differential part is smaller than one.

We use notations based on [2,4] for differential and linear cryptanalysis, respectively. In our notations Ω_P, Ω_T are the input and output differences of the differential characteristic, and λ_T, λ_C are the input and output subsets (denoted by bit masks) of the linear approximation.

Let E be a block cipher described as a cascade of two sub-ciphers E_0 and E_1, i.e., $E = E_1 \circ E_0$. Langford and Hellman suggested to use a truncated differential $\Omega_P \rightarrow \Omega_T$ for E_0 with probability 1. To this differential they concatenate a linear approximation $\lambda_T \rightarrow \lambda_C$ for E_1 with probability $1/2+q$ (or bias q). Their attack requires that the bits masked in λ_T have a zero difference in Ω_T.

If we take a pair of plaintexts P_1 and P_2 that satisfy $P_1 \oplus P_2 = \Omega_P$, then after E_0, $\lambda_T \cdot E_0(P_1) = \lambda_T \cdot E_0(P_2)$. This follows from the fact that $E_0(P_1)$ and $E_0(P_2)$ have a zero difference in the masked bits according to the output difference of the differential.

Recall that the linear approximation predicts that $\lambda_T \cdot T = \lambda_C \cdot E_1(T)$ with probability $1/2+q$. Hence, $\lambda_T \cdot E_0(P_1) = \lambda_C \cdot E_1(E_0(P_1))$ with probability $1/2+q$, and $\lambda_T \cdot E_0(P_2) = \lambda_C \cdot E_1(E_0(P_2))$ with probability $1/2 + q$. As the differential predicts that $\lambda_T \cdot E_0(P_1) = \lambda_T \cdot E_0(P_2)$, then with probability $1/2 + 2q^2$, $\lambda_C \cdot C_1 = \lambda_C \cdot C_2$ where C_1 and C_2 are the ciphertexts corresponding to P_1 and P_2, respectively, i.e., $C_i = E_1(E_0(P_i))$.

This fact allows to construct differential-linear distinguishers based on encrypting many plaintext pairs and checking whether the ciphertexts agree on the parity of the output subset. The data complexity of the distinguishers is $O(q^{-4})$ chosen plaintexts. The exact number of plaintexts is a function of the desired success rate, and of the number of possible subkeys.

In [8] Biham, Dunkelman and Keller proposed a way to deal with differentials with probability $p < 1$. In case the differential is satisfied (probability p), the above analysis remains valid. The assumption for the remaining $1 - p$ of the pairs is that the input subset parities are distributed randomly. In that case, the probability that a pair with input difference Ω_P will satisfy $\lambda_C \cdot C_1 = \lambda_C \cdot C_2$ is $p(1/2 + 2q^2) + (1 - p) \cdot 1/2 = 1/2 + 2pq^2$.

Furthermore, in [8] it is shown that the attack can still be applicable if $\Omega_T \cdot \lambda_T = 1$, i.e., the differential predicts that there is a difference in approximated

bits. In this case, the analysis remains valid, but instead of looking for the instances for which $\lambda_T \cdot C_1 = \lambda_T \cdot C_2$, we look for the cases when $\lambda_T \cdot C_1 \neq \lambda_T \cdot C_2$. As the analysis remains the same given a pair of plaintexts with the input difference Ω_P, the probability that the pair disagrees on the output subset parity is $1/2 + 2pq^2$. Another interesting result is that the attack still applies even when $\Omega_T \cdot \lambda_T$ is unknown, as long as its value is fixed. The data complexity of the enhanced differential-linear attack is $O(p^{-2}q^{-4})$.

4 Differential-Linear Attacks on Serpent

We first recall the 11-round attack from [9] which we use as a starting point of our research. We then continue to improve the 11-round attack by reducing the data complexity by a factor of about 2^8. Our main results follow from a small change in the linear approximation, which takes into account the huge difference between the number of active S-boxes and the number of pairs. Finally, we extend the 11-round attack to 12 rounds.

4.1 The Previous Attack on 11-Round Serpent

The attack from [9] is a differential-linear attack using a 9-round differential-linear approximation for rounds 2–10 composed of a 3-round differential and a 6-round linear approximation. The input difference of the 3-round differential is Ω_P = 0000 0000 0000 0000 0000 0000 4005 0000_x which with probability 2^{-6} does not affect bits 1 and 117 at the entrance of round 5. The 6-round linear approximation starts with these bits, and the output mask is λ_C = 0000 1000 0000 0000 5000 0100 0010 0001_x. The bias of the approximation is 2^{-27}, and thus the total bias of the differential-linear approximation is $2pq^2 = 2 \cdot 2^{-6} \cdot (2^{-27})^2 = 2^{-59}$. We describe the differential and the approximation in Appendices A and B, respectively.

There are 5 active S-boxes in the round before the differential-linear approximation and 7 active S-boxes afterward. Thus, the attacker uses structures of 2^{20} chosen plaintexts each (covering the five active S-boxes), thus resulting in 2^{39} pairs (of which 2^{19} are expected to have difference Ω_P at the entrance to round 1). After generating sufficiently many such structures, the attacker uses the following algorithm: For each guess of the subkey in round 0 that enters the 5 active S-boxes, the attacker partially encrypts all the plaintexts, and finds all the plaintext pairs with input difference Ω_P. Then, for each of these pairs, and for each guess of the subkey in round 10, the attacker checks whether the partial decryption of the pair satisfies the approximation or not.

The last step is done in an optimized way using a table look-up. We note that for each pair only 7 S-boxes are decrypted. Thus, there are 28 bits from each of the two ciphertexts that are being decrypted (under a subkey guess of 28 bits). Thus, instead of repeatedly decrypting the same values under the same subkey guess, the attacker counts for each subkey guess of round 1 how

many times each of the 56-bit ciphertext values (the 28 bits from each of the two paired ciphertexts) appears. Then, by performing 2^{28} trial encryptions for each of these counters, the attacker is able to deduce how many pairs satisfy the approximation.

In [9] the above attack is applied using $2^{125.3}$ plaintexts (which compose $2^{124.3}$ pairs). The time complexity is mostly dominated by the division into pairs, i.e., the partial encryption of $2^{125.3}$ values under 2^{20} possible subkeys. Repeating the analysis done in [9] and using the success probability formula establishedin [17] we have found out that for $2^{122.3}$ pairs, the success probability of the attack is expected to be about 84%. Thus, the actual data and time complexity of the original 11-round attack is $2^{123.3}$ chosen plaintexts, and the time complexity is $2^{137.2}$ encryptions.

We have experimentally verified the differential-linear property with a 3-round differential and the first round of the linear approximation. While the expected bias for this shortened approximation is $2 \cdot 2^{-6}(\cdot 2^{-5})^2 = 2^{-15}$, we found out that the bias of the shortened differential-linear approximation is $2^{-13.75}$. We performed 100 experiments, where in each experiment 2^{36} pairs with input difference Ω_P were encrypted, and the intermediate encryption values were checked with respect to whether the parity of the output subset is the same or not. The standard deviation of the bias was $2^{-18.87}$.

The difference between the expected value and the actual value follows the fact that even when the differential is not satisfied, and a difference enters one of the approximated S-boxes, the output mask is still biased. This means that the assumption that for pairs which do not follow the differential, there is no bias from 1/2 with respect to whether the approximations hold simultaneously for the two intermediate values, does not hold.

By assuming the piling-up lemma [16] to hold for the remainder of the linear approximation, we expect that the actual bias of the 9-round differential-linear approximation is also $2^{1.25}$ times higher than 2^{-59}, i.e., the probability that two pairs with input difference Ω_P have the same parity in λ_C is $1/2+2^{-57.75}$. Taking this into account shows that the actual data complexity required for the original 11-round attack is $2^{121.8}$ chosen plaintexts, and that the actual time complexity is $2^{135.7}$ encryptions.

4.2 Further Improvements of the 11-Round Attack

We first note that the attack can be easily improved by using the optimization ideas performed in the original attack also in the differential side of the distinguishers ,i.e., in round 1. For each subkey guess, we can build a list of the pairs according to the value in the 20 bits which enter the five active S-boxes. Thus, let P_1 and P_2 be a pair under some subkey guess, and flip a bit which does not enter an active S-box in both plaintexts to obtain P_1' and P_2', respectively. It is obvious that P_1' and P_2' are actually a pair, without any need to partially encrypt them. Thus, it is possible to improve the attack from [9] to be $2^{20} \cdot 2^{121.8} = 2^{141.8}$ memory accesses rather than $2^{135.7}$ 11-round encryptions.

The second improvement is based on the observation that the attacker has to process $2^{121.8}$ plaintexts/ciphertexts, and thus, the time complexity of the partial decryption at round 11 (which is about $2^{28} \cdot 2^{28}$ partial decryptions and 2^{84} memory accesses for a subkey guess of round 1) can be increased without affecting the time complexity of the attack. This can be achieved by inverting the order of the differential and the linear approximation. If we use a 3-round differential for rounds 11–13 (with probability 2^{-6}) and the linear approximation for rounds 5–10 as before (but in the decryption direction), then the linear approximation can be improved (increasing its bias by a factor of 2), thus reducing the data complexity of the attack, and as a consequence the time complexity as well. The change in the linear approximation is changing one of the approximations in round 5 to activate more S-boxes in the round before in exchange for a higher bias. The new 3-round differential for rounds 11–13 is presented in Appendix A.

We experimentally verified that the bias in the number of pairs with ciphertext difference Ω_C having the same parity in the input of the differential is 2^{-7}. When we decrypted one more round, and applied the last round of the linear approximation we expected a bias of $2pq^2 = 2 \cdot 2^{-6} \cdot (2^{-6})^2 = 2^{-17}$. However, for 100 different keys, we have observed a bias of 2^{-14} (we used 2^{36} pairs in each experiment, and the mean value was $2^{-13.93}$ with standard deviation of $2^{-18.92}$). Assuming that the remaining rounds behave independently, the expected bias of the entire 9-round differential-linear approximation in the inverse direction is 2^{-54}.

The difference between the expected and the computed values follows from the correlation between the differential and the linear approximation. It appears[2] that in about half of the pairs satisfying the differential, the input difference in at least one of the five active S-boxes in the first round of the linear approximation is zero. As a result, the bias of the differential-linear approximation for these pairs is much higher, and this causes the higher bias of the overall differential-linear approximation.

Thus, the improved 11-round attack is as follows:

1. Select $N = 2^{113.7}$ ciphertexts, consisting of $2^{89.7}$ structures, each is chosen by selecting:
   (a) A ciphertext C_0.
   (b) The ciphertexts $C_1, \ldots, C_{2^{24}-1}$ which differ from C_0 by all the $2^{24} - 1$ possible (non-empty) subsets of the twenty four bits which enter the 6 active S-boxes in round 14.
2. Decrypt these ciphertexts under the unknown key K.
3. For each value of the 24 bits of K_{14} entering these 6 S-boxes:
   (a) Initialize an array of 2^{72} counters to zeroes.
   (b) Partially decrypt for each ciphertext the 6 active S-boxes in round 14 and find the pairs which satisfy the difference Ω_C after round 13.

[2] We have verified this claim experimentally.

(c) Given those $2^{112.7}$ pairs, perform for each ciphertext pair: Let $extract_{36}$ be the function that extracts the 36 bits which enter the 9 active S-boxes in round 4, then for each pair P, P' increment the counter corresponding to $extract_{36}(P)||extract_{36}(P')$.
(d) For every 36-bit guess of the subkey entering these S-boxes, compute the parity of the corresponding partial decrypted pairs, and store the most biased guess for these 36 subkey bits (along with the guess of K_{14}).

4. Output the subkey combination with the largest deviation from $N/2$.

The data complexity of the attack is $2^{113.7}$ chosen plaintexts. The time complexity of the attack is dominated mainly by Step 3 which is repeated 2^{24} times. For each of these guesses the attacker first identifies the pairs (using tables) and has to perform $2^{113.7}$ memory accesses to compute $extract_{36}$ for all the pairs (Step 3(c)) and about 2^{108} memory accesses in Step 3(d), which means that the total time complexity of the attack is $2^{137.7}$ memory accesses. Again, using the success formula found in [17], for $2^{113.7}$ chosen plaintexts, the probability that the right key has the largest bias is about 93%.

The memory complexity of the attack is $2^{24} \cdot 2^{72} = 2^{96}$ counters. As the attack is repeated 2^{24} times (once for each guess of K_{14}), we can either store all the data, i.e., $2^{113.7}$ values, or store for each such guess the number of pairs. The second way is more efficient, as it allows analyzing each structure independently of others, and discarding it once the analysis is done. This approach has no impact on the data complexity or the time complexity, but it reduces the memory complexity to 2^{96} counters, each of up to 64 bits, or a total of 2^{99} bytes.

4.3 12-Round Differential-Linear Attack

We now present a differential-linear attack on 12-round Serpent. The attack is based on the original 11-round attack (in the forward direction) and uses the fact that a pair which satisfies the input difference of the differential has at most 28 active S-boxes in round 0. Thus, it is possible to change the attack algorithm a bit and obtain a 12-round attack against Serpent with 256-bit keys.

We have tried all the possible input differences to round 1 that lead to the difference $LT^{-1}(\Omega_P) = 2000\ 0000\ 0000\ 01A0\ \ 0E00\ 4000\ 0000\ 0000_x$. This difference is not affected by S-boxes 2, 3, 19, and 23, i.e., these S-boxes do not affect the active bits of $LT^{-1}(\Omega_P)$. Thus, we can construct structures of plaintexts which take this fact into consideration and obtain a 12-round attack on Serpent:

1. Select $N = 2^{123.5}$ plaintexts, consisting of $2^{11.5}$ structures, each is chosen by selecting:
   (a) Any plaintext P_0.
   (b) The plaintexts $P_1, \ldots, P_{2^{112}-1}$ which differ from P_0 by all the $2^{112} - 1$ possible (non-empty) subsets of the bits which enter all S-boxes besides 2, 3, 19, and 23 in round 0.
2. Request the ciphertexts of these plaintext structures (encrypted under the unknown key K).

3. For each value of the 112 bits of K_0 entering these 28 S-boxes, partially encrypt all the plaintexts the first round, and apply the original 11-round attack.
4. Each trial of the key gives us $112 + 20 + 28 = 160$ bits of the subkeys (112 bits in round 0, 20 bits in round 1 and 28 bits in round 11), along with a measure for correctness. The correct value of the 160 bits is expected to be the most frequently suggested value (with more than 84% success rate).
5. The rest of the key bits are then recovered by auxiliary techniques.

The data complexity of the attack is $2^{123.5}$ chosen plaintexts. The time complexity of the attack is $2^{123.5} \cdot 2^{112} \cdot \frac{28}{384} = 2^{231.7}$ encryptions for the partial encryption in Step 3, and $2^{112} \cdot 2^{137.4} = 2^{249.4}$ for the repeated trials of the 11-round attack.[3]

4.4 10-Round Differential-Linear Attack on Serpent with 128-bit Keys

We can use the three improvements suggested earlier to improve the 10-round attack on Serpent. We recall the three improvements:

- Better analysis of the bias of the differential-linear approximation,
- Better analysis of the success probability,
- Changing the output mask.

We shall start with changing the output mask of the approximation. In the 10-round attack in [9], the last round of the approximation is omitted, and the new 8-round differential-linear approximation has a bias of $2 \cdot 2^{-6} \cdot (2^{-22})^2 = 2^{-49}$. The last round of the approximation is optimized for reducing the number of active S-boxes in the last round (to 5 S-boxes). However, as before, we may activate a few more S-boxes, and almost have no effect on the time complexity of the attack (by increasing the counters).

By changing the output mask of the last round (where S_1 is used) from $\lambda'_C = 0010\,0001\,0000\,1000\ \ 0100\,0000\,0000\,0000$ to $\lambda'_C = 0010\,0001\,0000\,1000\ \ 0300\,0000\,0000\,0000$, we increase the bias of the linear approximation by a factor of 2, i.e., the differential-linear approximation has a bias of 2^{-47}.

Taking into consideration the better transition between the differential and the linear approximation, we obtain that the actual bias is $2^{-45.75}$. Using the formula from [17], and taking into consideration that there are 5 active S-boxes before the differential, and 9 active S-boxes after the linear approximation, we need $2^{96.2}$ pairs, i.e., $2^{97.2}$ chosen plaintexts to achieve a success rate of 84%.

The time complexity of the attack is $2^{111.2}$ 10-round encryptions (the time complexity required for partial encryptions and locating all the pairs) and 2^{128} memory accesses for handling the tables.

We note that if the approximation is not changed, the data complexity of the 10-round attack is $2^{101.2}$ chosen plaintexts, and the time complexity is $2^{115.2}$ encryptions.

[3] We note that the time complexity of the 11-round attack is $2^{135.7}$ encryptions. As the number of plaintexts is $2^{1.7}$ times larger in this attack, the time complexity of one iteration of the 11-round attack in this case is $2^{1.7}$ times larger.

5 A Related-Key Attack on a Modified Serpent

It is a well known fact that ciphers that iterate the exact same round function over and over are susceptible to slide attacks and related-key attacks [3,10]. In Serpent the constants which modify the round function are found in two places: the different S-boxes (which are used in a cycle of 8 rounds), and the constants in the key schedule algorithm.

Removing the constants from the key schedule algorithm makes the cipher susceptible to related-key attacks which treat the cipher as an iteration of the same "round" function which is composed of 8 consecutive rounds. Even though there are several attacks on 8-round Serpent, it is highly unlikely to elevate them into attacks on the full Serpent, as 8-round Serpent is secure enough to prevent easy detection of the related-key plaintext pairs.

We present a related-key relation that holds with probability of 2^{-124}, and can be used to distinguish this simplified variant of Serpent from a random permutation (for 256-bit keys) with data complexity of about 2^{125} chosen plaintexts, and a negligible time complexity. We then use this relation to retrieve partial information about the keys.

Consider two related keys K and K' such that $K = (w_{-8}, \ldots, w_{-1})$ and $K' = (w_{-8} \lll 1, \ldots, w_{-1} \lll 1)$. For these two keys, all the corresponding subkeys k_i and k'_i respectively satisfy that $k_i = k'_i \lll 1$. Under these two keys we consider the plaintexts $P = (a, b, c, d)$ and $P' = (a \lll 1, b \lll 1, c \lll 1, d \lll 1)$. We denote such keys, plaintexts, or intermediate encryption values, i.e., two values such that the second is a rotate to the left by one bit of each 32-bit word independently, as satisfying the *rotation property*.

The rotation property is kept through the key addition, i.e., $P' \oplus K'_1$ is a rotate left by one bit word-wise of $P \oplus K$, and the S-boxes layer. The only problem is the linear transformation which contains cyclic rotations and shifts. The cyclic rotations do not affect the rotation property, so the only problem in extending the property is the shift operation. However, the property can bypass a shift with probability of 2^{-2}. Let X be a 32-bit word, and let $X' = X \lll 1$. Then, if the least significant bit of X is zero, and the least significant bit of $X' \ll m$ (the most significant bit of $X \ll m$) is 0 as well, then $X \ll m$ and $X' \ll m$ satisfy the rotation property. As in each linear transformation there are two such shifts, the probability that the rotation property is maintained after the linear transformation is 2^{-4}.

Serpent has 31 linear transformations, and thus, the probability that the rotation property remains from the plaintext till the ciphertext is 2^{-124}, while for two random permutations, one expects the probability of 2^{-128}. This property can be used to distinguish this variant of Serpent from a random permutation using about 2^{125} plaintexts. Given the pair that satisfies the rotation property it is also possible to deduce the equivalent of 4 bits of the key.

6 Summary

In this paper we studied differential-linear cryptanalysis of Serpent. We showed several improvements in the analysis of the previously best known results on

Table 1. Summary of Attacks on Serpent with Reduced Number of Rounds

Rounds	Type of Attack	Key Size	Complexity Data	Time	Memory
10	Rectangle [7]	192 & 256	$2^{126.3}$ CP	$2^{173.8}$ MA	$2^{131.8}$ B
	Boomerang [7]	192 & 256	$2^{126.3}$ AC	$2^{173.8}$ MA	2^{89} B
	Differential-Linear [9]	all	$2^{105.2}$ CP	$2^{123.2}$ En	2^{40} B
	Differential-Linear (Sect. 4.4)	all	$2^{101.2}$ CP	$2^{115.2}$ En	2^{40} B
	Differential-Linear (Sect. 4.4)	all	$2^{97.2}$ CP	2^{128} MA	2^{72} B
11	Differential-Linear [9]	192 & 256	$2^{125.3}$CP	$2^{172.4}$ En	2^{30} B
	Differential-Linear [9]	192 & 256	$2^{125.3}$ CP	$2^{139.2}$ En	2^{60} B
	Differential-Linear (Sect. 4.1)	192 & 256	$2^{121.8}$ CP	$2^{135.7}$ MA	2^{76} B
	Differential-Linear (Sect. 4.2)	192 & 256	$2^{113.7}$ CC	$2^{137.7}$ MA	2^{99} B
12	Differential-Linear (Sect. 4.3)	256	$2^{123.5}$ CP	$2^{249.4}$ En	$2^{128.5}$ B

En — Encryptions, MA — Memory Accesses, B — bytes, CP — Chosen Plaintexts
CC — Chosen Ciphertexts, AC — Adaptive Chosen Plaintexts and Ciphertexts.

11-round Serpent, and suggested a new 11-round attack with a much lower data complexity. Combining experimental results and the improved analysis, we presented the first attack on 12-round Serpent. The attack uses $2^{123.5}$ chosen plaintexts, and has a time complexity of $2^{249.4}$ encryptions.

Finally, we explored a related-key attack on a modified Serpent where the round constants are removed from the key schedule, and showed that despite the strong repeated cipher (8-round of Serpent), there are high probability related-key properties that can be used both for distinguishing and key recovery.

We summarize our new attacks, and selected previously published attacks against Serpent in Table 1.

References

1. Anderson, R., Biham, E., Knudsen, L.R.: Serpent: A Proposal for the Advanced Encryption Standard, NIST AES Proposal (1998)
2. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer, Heidelberg (1993)
3. Biham, E.: New Types of Cryptanalytic Attacks Using Related Keys. Journal of Cryptology 7(4), 229–246 (1994)
4. Biham, E.: On Matsui's Linear Cryptanalysis. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 341–355. Springer, Heidelberg (1995)
5. Biham, E., Dunkelman, O., Keller, N.: The Rectangle Attack – Rectangling the Serpent. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 340–357. Springer, Heidelberg (2001)
6. Biham, E., Dunkelman, O., Keller, N.: Linear Cryptanalysis of Reduced Round Serpent. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 16–27. Springer, Heidelberg (2002)
7. Biham, E., Dunkelman, O., Keller, N.: New Results on Boomerang and Rectangle Attacks. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 1–16. Springer, Heidelberg (2002)

8. Biham, E., Dunkelman, O., Keller, N.: Enhancing Differential-Linear Cryptanalysis. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 254–266. Springer, Heidelberg (2002)
9. Biham, E., Dunkelman, O., Keller, N.: Differential-Linear Cryptanalysis of Serpent. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 9–21. Springer, Heidelberg (2003)
10. Biryukov, A., Wagner, D.: Slide Attacks. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 245–259. Springer, Heidelberg (1999)
11. Collard, B., Standaert, F.-X., Quisquater, J.-J.: Improved and Multiple Linear Cryptanalysis of Reduced Round Serpent. In: Pei, D., Yung, M., Lin, D., Wu, C. (eds.) Inscrypt 2007. LNCS, vol. 4990, pp. 51–65. Springer, Heidelberg (2008)
12. Collard, B., Standaert, F.-X., Quisquater, J.-J.: Experiments on the Multiple Linear Cryptanalysis of Reduced Round Serpent. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 382–397. Springer, Heidelberg (2008)
13. Kelsey, J., Kohno, T., Schneier, B.: Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 75–93. Springer, Heidelberg (2001)
14. Langford, S.K.: Differential-Linear Cryptanalysis and Threshold Signatures, Ph.D. thesis (1995)
15. Langford, S.K., Hellman, M.E.: Differential-Linear Cryptanalysis. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 17–25. Springer, Heidelberg (1994)
16. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994)
17. Selçuk, A.A.: On Probability of Success in Linear and Differential Cryptanalysis. Journal of Cryptology 21(1), 131–147 (2008)
18. Wagner, D.: The Boomerang Attack. In: Knudsen, L.R. (ed.) FSE 1999. LNCS, vol. 1636, pp. 156–170. Springer, Heidelberg (1999)

A The Differential Characteristic

A.1 The Original 3-Round Differential

The 3-round truncated differential used in the original 11-round attack is as follows. The first round of the differential is round 2 (or any other round that uses S_2) with probability 2^{-5}:

$$\begin{array}{rll} \Omega_P = & 0000\ 0000\ 0000\ 0000\ \ 0000\ 0000\ 4005\ 0000 & \stackrel{S_2}{\rightarrow} \qquad \Pr = 2^{-5} \\ & 0000\ 0000\ 0000\ 0000\ \ 0000\ 0000\ A004\ 0000 & \stackrel{LT}{\rightarrow} \\ & 0040\ 0000\ 0000\ 0000\ \ 0000\ 0000\ 0000\ 0000 & \stackrel{S_3}{\rightarrow} \qquad \Pr = 2^{-1} \\ & 00X0\ 0000\ 0000\ 0000\ \ 0000\ 0000\ 0000\ 0000 & \end{array}$$

where $X \in \{2, 4, 6, 8, A_x, C_x, E_x\}$. After the linear transformation, we get the following truncated differential in S_4:

$$\begin{array}{rl} 0QT_30\ 0T_200\ 0T_100\ 0000\ \ 000Y_4\ 00Y_30\ W_2Y_20W_1\ Y_10Z0 & \stackrel{S_4}{\rightarrow} \\ 0??0\ 0?00\ 0?00\ 0000\ \ 000?\ 00?0\ ??0?\ ?0?0 & = \Omega_T, \end{array}$$

where ? is any possible difference and $Y_i \in \{0, 1\}, Z \in \{0, 2\}, W_i \in \{0, 8\}, T_i \in \{0, 4\}, Q \in \{0, 2, 4, 6\}$.

A.2 The 3-Round Differential in the Improved 11-Round Attack

The 3-round differential used in the improved 11-round attack is in the backward direction. The output difference is $\Omega_C = 0000\,0000\,0000\,0090\,0000\,0000\,0000\,0000_x$ which with probability of about 2^{-6} does not affect the bits in $LT(\lambda_C)$. This follows from the main following differential characteristic:

$$\begin{array}{rll} \Omega_C = & 0000\ 0000\ 0000\ 0090\ \ 0000\ 0000\ 0000\ 0000 & \stackrel{S_5^{-1}}{\rightarrow} \qquad \Pr = 2^{-2} \\ & 0000\ 0000\ 0000\ 0040\ \ 0000\ 0000\ 0000\ 0000 & \stackrel{LT^{-1}}{\rightarrow} \\ & 0000\ A004\ 0000\ 0000\ \ 0000\ 0000\ 0000\ 0000 & \stackrel{S_4^{-1}}{\rightarrow} \qquad \Pr = 2^{-3} \\ & 0000\ ?009\ 0000\ 0000\ \ 0000\ 0000\ 0000\ 0000 & \stackrel{LT^{-1}}{\rightarrow} \\ & 0Z_300\ T_2YZ_2R\ 0T_14Z_1\ 2080\ \ 0X_200\ 10X_10\ 01Q0\ 0W00 & \stackrel{S_3^{-1}}{\rightarrow} \qquad \Pr = 1 \\ & 0?00\ ????\ 0???\ ?0?0\ \ 0?00\ ?0?0\ 0??0\ 0?00 & = \Omega_T \end{array}$$

with probability 1, when $W \in \{0, 4\}, Q \in \{0, 2, 8, A_x\}, X_i \in \{0, 1\}, Z_i \in \{0, 8\}$, $T_i \in \{8, A_x\}, R \in \{8, C_x\}$.

We note that despite the fact that the probability of this differential is 2^{-5}, when counting all possible output differences, the probability that there is a difference in the bits covered by $LT(\lambda_C)$, the bias was found to be 0.007773, i.e., $1/128.7 \approx 2^{-7.007}$. This follows mostly from the cases where the differential does not follow the second round, i.e., the input difference to S-box 24 is not 9, as then there is an active S-box which affects the approximation with relatively high probability (with output difference 2). Thus, the "probability" of the differential can be assumed to be $p = 2^{-6}$.

B The Linear Approximation

The 6-round linear approximation used in the attack is as follows. It starts before the linear transformation of round 4 with $\lambda_T = 2006\ 0040\ 0000\ 0100\ \ 1000\ 0000\ 0000\ 0000_x$. In round 5 the following approximation[4] holds with bias -2^{-5}:

$$
\begin{array}{rll}
LT(\lambda_T) = & 0020\ 0000\ 0000\ 0000\ \ 0000\ 0000\ 0000\ 0002 & \xrightarrow{S_5} \quad \Pr = \frac{1}{2} - 2^{-5} \\
& 0040\ 0000\ 0000\ 0000\ \ 0000\ 0000\ 0000\ 0008 & \xrightarrow{LT} \\
& 0000\ 0000\ 0000\ 0000\ \ 0000\ 0000\ 8000\ 0000 & \xrightarrow{S_6} \quad \Pr = \frac{1}{2} - 2^{-3} \\
& 0000\ 0000\ 0000\ 0000\ \ 0000\ 0000\ 1000\ 0000 & \xrightarrow{LT} \\
& 0000\ 00A0\ 0001\ 0000\ \ 0000\ 0000\ 0000\ 0000 & \xrightarrow{S_7} \quad \Pr = \frac{1}{2} - 2^{-5} \\
& 0000\ 0010\ 0001\ 0000\ \ 0000\ 0000\ 0000\ 0000 & \xrightarrow{LT} \\
& 0000\ 0000\ 0000\ 0000\ \ 0000\ 1000\ 0B00\ 00A0 & \xrightarrow{S_0} \quad \Pr = \frac{1}{2} + 2^{-6} \\
& 0000\ 0000\ 0000\ 0000\ \ 0000\ 1000\ 0100\ 0010 & \xrightarrow{LT} \\
& 0010\ 000B\ 0000\ B000\ \ 0A00\ 0000\ 0000\ 0000 & \xrightarrow{S_1} \quad \Pr = \frac{1}{2} - 2^{-7} \\
& 0010\ 0001\ 0000\ 1000\ \ 0100\ 0000\ 0000\ 0000 & \xrightarrow{LT} \\
& 0000\ A000\ 0000\ 0000\ \ 1000\ 0B00\ 00B0\ 000B & \xrightarrow{S_2} \quad \Pr = \frac{1}{2} - 2^{-6} \\
& 0000\ 1000\ 0000\ 0000\ \ 5000\ 0100\ 0010\ 0001 & = \lambda_C.
\end{array}
$$

After the linear transformation of round 11, $LT(\lambda_C) = 000B\ 0000\ B000\ 0300\ 00B0\ 200E\ 0000\ 0010$, i.e., there are seven active S-boxes: 1, 8, 11, 13, 18, 23 and 28.

[4] For the improved attack we change the input bias in S-box 29 to E_x and the bias in that case is 2^{-4}.

New AES Software Speed Records

Daniel J. Bernstein[1] and Peter Schwabe[2,⋆]

[1] Department of Computer Science
University of Illinois at Chicago, Chicago, IL 60607–7045, USA
djb@cr.yp.to
[2] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, Netherlands
peter@cryptojedi.org

Abstract. This paper presents new speed records for AES software, taking advantage of (1) architecture-dependent reduction of instructions used to compute AES and (2) microarchitecture-dependent reduction of cycles used for those instructions. A wide variety of common CPU architectures—amd64, ppc32, sparcv9, and x86—are discussed in detail, along with several specific microarchitectures.

Keywords: AES, software implementation.

1 Introduction

This paper describes a new AES software implementation achieving extremely high speeds on various common CPUs. For example, on an UltraSPARC III or IV, this implementation's main loop takes only 193 cycles/block. The smallest cycle count previously claimed for AES software on *any* SPARC was 270 cycles/block by Helger Lipmaa's proprietary software.

Almost all of the specific techniques we use are well known. The main novelty in this paper lies in the analysis and combination of these techniques, producing surprisingly high speeds for AES. We have published our software to ensure verifiability of our results; we have, furthermore, placed the software into the public domain to maximize reusability of our results.

Section 2 reviews the standard structure of "32-bit" AES software implementations. Section 3 surveys techniques for reducing the number of CPU *instructions* required to compute AES. Section 4 explains how we reduced the number of CPU *cycles* required to compute AES on various platforms.

Thanks to Ruben Niederhagen and the anonymous reviewers for suggesting many improvements in our explanations.

⋆ The first author was supported by the National Science Foundation under grant ITR–0716498. He carried out parts of this work while visiting Technische Universiteit Eindhoven. The second author was supported by the European Commission through the ICT Programme under Contract ICT–2007–216499 CACE. Permanent ID of this document: b90c51d2f7eef86b78068511135a231f. Date: 2008.09.25.

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 322–336, 2008.

Which AES? This paper focuses on the most common AES key size, namely 16 bytes (128 bits). We plan to adapt our implementation later to support 32-byte (256-bit) keys. One might guess that AES is 40% slower with 32-byte keys, since 16-byte keys use 10 rounds while 32-byte keys use 14 rounds; actual costs are not exactly linear in the number of rounds, but 40% is a reasonable estimate.

There are several modes of operation of AES: cipher-block chaining (CBC), output feedback (OFB), counter mode (CTR), et al. There are also, in the literature, many different ways to benchmark AES software. This variability interferes with comparisons. Often a faster AES performance report is from a *slower* AES implementation measured with a less invasive benchmarking framework.

A big improvement in comparability has been achieved in the last few years by eSTREAM, a multi-year ECRYPT project that has identified several promising new stream ciphers. The eSTREAM benchmarking framework has been made public, allowing anyone to verify performance data; includes long-message and short-message benchmarks; and includes AES-CTR as a basis for comparison. The original AES-CTR implementation in the benchmarking framework is a reference implementation written by Brian Gladman; other authors have contributed implementations optimized for several architectures.

This paper reports cycle counts directly from the eSTREAM benchmarking framework, and uses exactly the same form of AES-CTR. Our software passes the extensive AES-CTR tests included in the benchmarking framework. By similar techniques we have also sped up Biryukov's LEX stream cipher [6].

See [10] for much more information on the eSTREAM project; [9] for the benchmarking framework; [4] for a more portable version of the benchmarking framework (including our software as of version 20080905); and [12] for more information about Gladman's AES software.

Bitslicing. The recent papers [23], [17], and [19] have proposed bitsliced AES implementations for various CPUs. The most impressive report, from Matsui and Nakajima in [19], is 9.2 cycles/byte for bitsliced AES on a Core 2.

Unfortunately, this speed is achieved only for 2048-byte chunks that have been "transposed" into bitsliced form. Transposition of ciphertext costs about 1 cycle/byte. More importantly, bitsliced encryption of a 576-byte Internet packet costs as much as bitsliced encryption of a 2048-byte packet, multiplying the cycle counts by approximately 3.5. Consequently this bitsliced implementation is not competitive in speed with the implementation reported in this paper. The very recent semi-bitsliced implementation in [14] uses much smaller chunks, only 64 bytes, but it is also non-competitive: it takes 19.81 cycles/byte on an Athlon 64.

Bitslicing remains of interest for several reasons: first, some applications encrypt long streams and do not mind padding to 2048-byte boundaries; second, some applications will use bitslicing on both client and server and can thus eliminate the costs of transposition; third, bitsliced implementations are inherently immune to the cache-timing attacks discussed in [3] and [21].

Other literature. Worley et al. in [26] report AES implementations for PA-RISC and IA-64. Schneier and Whiting in [24] report AES implementations for the Pentium, Pentium Pro, HP PA-8200, and IA-64. Weiss and Binkert in

[25] report AES implementations for the Alpha 21264. Aoki and Lipmaa in [1] report AES implementations for the Pentium II. Most of these CPUs are now quite difficult to find, but these papers—particularly [1]—are well worth reading for their discussions of AES optimizations.

More recent AES speed reports: Osvik in [20] covers Pentium III, Pentium 4, and Athlon. Lipmaa in [16] and [15] covers many CPUs. Matsui and Fukuda in [18] cover the Pentium III and Pentium 4. Matsui in [17] covers the Athlon 64.

[5], [2], and [8] report implementations for smaller CPUs using the ARM architecture. [13] reports implementations for graphics processors (GPUs). There is also an extensive literature on implementations of AES in hardware, in FPGAs, and in hardware-software codesigns.

2 A Short Review of AES

AES expands its 16-byte key into 11 "round keys" $r_0, \ldots, r_{10}$, each 16 bytes. Each 16-byte block of plaintext is xor'ed with round key r_0, transformed, xor'ed with round key r_1 (ending "round 1"), transformed, xor'ed with round key r_2 (ending "round 2"), transformed, etc., and finally xor'ed with round key r_{10} (ending "round 10") to produce a 16-byte block of ciphertext.

Appendix A is sample code in the C programming language for a typical round of AES. The round reads a 16-byte state stored in four 4-byte variables `y0`, `y1`, `y2`, `y3`; transforms the variables; xor's a 16-byte round key stored in four 4-byte variables; and puts the result into `z0`, `z1`, `z2`, `z3`.

The main work in the transformation is 16 table lookups indexed by the 16 bytes of `y0`, `y1`, `y2`, `y3`; beware that the last round of AES is slightly different. Each table lookup produces 4 bytes. In this sample code there are four tables interleaved in memory, with the jth entry of table i at address `table` $+ 4i + 16j$; `table` $+ 4i$ is precomputed as a byte pointer `table`i. For example,

```
p03 = (uint32) y0 << 4;
p03 &= 0xff0;
p03 = *(uint32 *) (table3 + p03);
```

in the sample code extracts the bottom byte of `y0`, multiplies it by 16, adds it to the byte pointer `table3`, and reads the 4 bytes at that address.

One can eliminate the table interleaving, and store the jth entry of table i at address `table` $+ 1024i + 4j$; then the shift distances `20`, `12`, `4`, `4` need to be changed to `22`, `14`, `6`, `2`. An intermediate possibility is to interleave the first two tables and interleave the second two tables, using shift distances `21`, `13`, `5`, `3`.

We do not describe the work required to invert this process, computing a 16-byte plaintext from a 16-byte ciphertext and a 16-byte key. In AES-CTR, the "plaintext" is actually a 16-byte counter; the encrypted counter serves as keystream that is xor'ed to the user's actual plaintext, producing counter-mode ciphertext. AES-CTR decryption is the same as AES-CTR encryption so it does not require inverting AES.

Relationship to `SubBytes` etc. An AES round is often described differently, as a series of four operations on 4×4 matrices: `SubBytes`, `ShiftRows`, `MixColumns`, and `AddRoundKey`. The last round skips `MixColumns`.

As part of the original AES proposal, Daemen and Rijmen described how to merge `SubBytes`, `ShiftRows`, and `MixColumns` into 16 lookups from 4 tables T_0, T_1, T_2, T_3, each containing 256 32-bit entries; `AddRoundKey` is nothing but xor'ing the round key r_i. See [7, Section 5.2]. As far as we know, the first implementation using this structure was written by Rijmen, Bosselaers, and Barreto. This is the structure used in the sample code in Appendix A, and the starting structure for the speedups described in the following sections.

3 Saving Instructions for AES

This section surveys several methods to reduce the number of CPU integer instructions, load instructions, etc. used for AES. Many of these methods take advantage of additional instructions and features provided by some CPUs.

Beware that not all of the methods can be combined. Furthermore, minimizing *cycles* is a much more subtle task than minimizing *instructions*. In Section 4 we discuss the cycle counts that we have achieved on various platforms, taking account of limited register sets, instruction latencies, etc.

3.1 Baseline (720 Instructions)

One round of AES can be decomposed into 16 shift instructions, 16 mask instructions, 16 load instructions for the table lookups, 4 load instructions for the round keys, and 16 xor instructions. See Appendix A. Overall there are 68 instructions, specifically 20 loads and 48 integer instructions.

Subsequent code examples in this section express CPU instructions using the `qhasm` language, `http://cr.yp.to/qhasm.html`. Each line that we display represents one CPU instruction.

All of the target platforms have shift instructions, mask instructions, load instructions, and xor instructions. Some platforms—for example, the x86 and amd64—do not support three-operand shift instructions (i.e., shift instructions where the output register is not the input register) but do have byte-extraction instructions that are adequate to achieve the same instruction counts. In this section we ignore register-allocation issues.

The 10-round main loop uses more than 680 instructions, for four reasons:

- Before the first round there are 4 extra round-key loads and 4 extra xors.
- The last round has 16 extra masks, one after each of the table lookups.
- For AES-CTR there are 4 loads of 4-byte plaintext words, 4 xors of keystream with plaintext, and 4 stores of ciphertext words.
- There are 4 extra instructions for miscellaneous tasks such as incrementing the AES-CTR input.

Overall there are 720 instructions, specifically 208 loads, 4 stores, and 508 integer instructions. The 508 integer instructions consist of 160 shift instructions, 176 mask instructions, 168 xor instructions, and 4 extra instructions.

In this count we ignore the costs of conditional branches; these costs are easily reduced by unrolling. We also ignore extra instructions needed to handle, e.g., big-endian loads on a little-endian architecture; almost all endianness issues can be eliminated by appropriate swapping of the AES code and tables.

We also ignore the initial costs of computing the 176 bytes of round keys from a 16-byte key. This computation involves hundreds of extra instructions—certainly a noticeable cost—but the round keys can be reused for all the blocks of a message. Round keys can also be reused for other messages if they are saved.

3.2 Table Structure and Index Extraction

Combined shift-and-mask instructions (−160 instructions). Some architectures allow a shift instruction `p02=y0>>4` and a mask instruction `p02&=0xff0` to be combined into a single instruction `p02=(y0>>4)&0xff0`. Replacing 160 shifts and 160 masks by 160 shift-and-mask instructions saves 160 instructions.

On the ppc32 architecture, for example, the `rlwinm` instruction can do any rotate-and-mask where the mask consists of consecutive bits.

Scaled-index loads (−80 instructions). On other architectures a shift instruction `p03<<=4` and a load instruction `p03=*(uint32*)(table3+p03)` can be combined into a single instruction. The instructions

```
p03 = (uint32) y0 << 4
p03 &= 0xff0
p03 = *(uint32 *) (table3 + p03)
```

for handling the bottom byte of `y0` can then be replaced by

```
p03 = y0 & 0xff
p03 = *(uint32 *) (table3 + (p03 << 4))
```

Similarly, the instructions

```
p00 = (uint32) y0 >> 20
p00 &= 0xff0
p00 = *(uint32 *) (table0 + p00)
```

for handling the top byte of `y0` can be replaced by

```
p00 = (uint32) y0 >> 24
p00 = *(uint32 *) (table0 + (p00 << 4))
```

In 10 rounds there are 40 top bytes and 40 bottom bytes.

The x86 architecture, for example, allows scaled indices in load instructions. The x86 scaling allows only a 3-bit shift, not a 4-bit shift, but this is easily accommodated by non-interleaved (or partially interleaved) tables.

Second-byte instructions (−40 instructions). All architectures support a mask instruction `p03=y0&0xff` that extracts the bottom byte of `y0`.

Some architectures—for example, the x86—also support a single instruction `p02=(y0>>8)&0xff` to extract the *second* byte of `y0`. In conjunction with scaled-index loads this instruction allows

```
p02 = (uint32) y0 >> 6
p02 &= 0x3fc
p02 = *(uint32 *) (table2 + p02)
```

to be replaced by

```
p02 = (y0 >> 8) & 0xff
p02 = *(uint32 *) (table2 + (p02 << 2))
```

saving another 40 instructions overall.

Padded registers (−80 instructions). Some architectures—e.g., sparcv9—do not have any of the combined instructions described above, but *do* have 64-bit registers. On these architectures one can expand a 4-byte value such as `0xc66363a5` into an 8-byte value such as `0x0c600630063a50` (or various other possibilities such as `0x0000c6063063a50`). If this expansion is applied consistently in the registers, the lookup tables (before the last round), and the round keys, then it does not cost any extra instructions.

The advantage of the padded 8-byte value `0x0c600630063a50` is that a single mask instruction produces the shifted bottom byte `a50`, and a single shift instruction produces the shifted top byte `c60`. Consequently the original eight shift-and-mask instructions, for extracting four shifted bytes from `y0`, can be replaced by six instructions:

```
p00 = (uint64) y0 >> 48
p01 = (uint64) y0 >> 32
p02 = (uint64) y0 >> 16
p01 &= 0xff0
p02 &= 0xff0
p03 = y0 & 0xff0
```

Expanded lookup tables, like scaled-index loads, thus save 80 instructions overall.

32-bit shifts of padded registers (−40 instructions). Some architectures—for example, sparcv9—have not only a 64-bit right-shift instruction but also a 32-bit right-shift instruction that automatically masks its 64-bit input with `0xffffffff`. This instruction, in conjunction with padded registers, allows

```
p02 = (uint64) y0 >> 16
p02 &= 0xff0
```

to be replaced by `p02 = (uint32) y0 >> 16`, saving 40 additional instructions.

3.3 Speedups for the Last Round

Byte loads (−4 instructions). As mentioned earlier, the last round of AES has 16 extra masks for its 16 table lookups. Four of the masks are `0xff`. All of

the target architectures allow these masks to be absorbed into single-byte load instructions. For example,

```
p00 = *(uint32 *) (table0 + p00)
p00 &= 0xff
```

can be replaced with `p00 = *(uint8*) (table0 + p00)` on little-endian CPUs or `p00 = *(uint8*) (table0 + p00 + 3)` on big-endian CPUs.

Two-byte loads (−4 instructions). Four of the masks are `0xff00`. These masks can be absorbed into two-byte load instructions if the table structure has `00` next to the desired byte. Often this structure occurs naturally as part of other table optimizations, and in any case it can be achieved by a separate table.

Masked tables (−8 instructions). The other eight masks are `0xff0000` and `0xff000000`. These masked values cannot be produced by byte loads and two-byte loads but can be produced by four-byte loads from separate tables whose entries are already masked.

Separate masked tables are also the easiest way to handle the distinction between padded 64-bit registers (see Section 3.2) and packed 32-bit AES output words.

Combined mask and insert (−16 instructions). A 4-byte result of the last round, such as `z0`, is produced by 4 xors with 4 masked table entries, where the masks are `0xff`, `0xff00`, `0xff0000`, `0xff000000`.

Some architectures have an instruction that replaces specified bits of one register with the corresponding bits of another register. For example, the ppc32 architecture has a `rlwimi` instruction that does this, optionally rotating the second register. The instruction sequence

```
p00 &= 0xff000000
p11 &=   0xff0000
p22 &=     0xff00
p33 &=       0xff
z0 ^= p00
z0 ^= p11
z0 ^= p22
z0 ^= p33
```

can then be replaced by

```
p00 bits 0xff0000 = p11 <<< 0
p00 bits   0xff00 = p22 <<< 0
p00 bits     0xff = p33 <<< 0
z0 ^= p00
```

(Note for C programmers: in C notation, `p00 bits 0xff0000 = p11` would be `p00 = (p00&0xff00ffff)|(p11&0xff0000)`.) This is another way—without using byte loads, and without constraining the table structure—to eliminate all the extra masks.

3.4 Further Speedups

Combined load-xor (−168 instructions). Often the result of a load is used solely for xor'ing into another register. Some architectures—for example, x86 and amd64—allow load and xor to be combined into a single instruction.

Byte extraction via loads (−160 . . . −320 integer instructions; +200 load/store instructions). Extracting four indices from `y0` takes at most 8 integer instructions, and on some architectures as few as 4 integer instructions, as discussed in Section 3.2.

A completely different way to extract four bytes from `y0`—and therefore to extract indices, on architectures allowing scaled-index loads—is to store `y0` and then do four byte loads from the stored bytes of `y0`. This eliminates between 4 and 8 integer instructions—potentially helpful on CPUs where integer instructions are the main bottleneck—at the expense of 5 load/store instructions.

One can apply this conversion to all 160 byte extractions. One can also apply it to some of the byte extractions, changing the balance between load instructions and integer instructions. The optimum combination is CPU-dependent.

Round-key recomputation (−30 load instructions; +30 integer instructions). In the opposite direction: Instead of loading 44 round-key words, say words $0, 1, 2, \ldots, 43$, one can load 14 round-key words, specifically words $0, 1, 2, 3, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40$, and compute the other round-key words by 30 xors, taking advantage of the AES round-key structure. This reduces the number of load instructions—potentially helpful on CPUs where loads are the main bottleneck—although it increases the number of integer instructions.

One can also use intermediate combinations, such as 22 loads and 22 xors. As before, the optimum combination is CPU-dependent.

Round-key caching (≈ −44 instructions). Each 16-byte block of input involves 44 round-key loads (or xors). The same round keys are used in the next block. On an architecture with many registers, some or all of the round keys can be kept in registers, moving these loads out of the main loop. The same type of savings appears if several blocks are handled in parallel.

Counter-mode caching (≈ −100 instructions). Recall that our AES software uses counter mode in exactly the form specified by eSTREAM. AES is applied to a 16-byte counter that is increased by 1 for every block of input.

Observe that 15 bytes of the counter remain constant for 256 blocks; just one byte of the counter changes for every block of input. All operations in the first round not depending on this byte are shared between 256 blocks. The resulting values `y0`, `y1`, `y2` and `y3` can be saved and reused in 256 consecutive blocks.

Similar observations hold for round 2: only one of the four 4-byte input words of round 2 changes every block. All computations not depending on this word can be saved and reused in 256 consecutive blocks.

This caching is perhaps the least well known of all AES software speedups. We learned it from an eSTREAM AES implementation by Hongjun Wu; we have not found it elsewhere in the literature.

4 Saving Cycles for AES

Minimizing instructions is not the same as minimizing cycles. A CPU that advertises "four instructions per cycle" actually performs *at most* four instructions per cycle; software that does not take account of microarchitecture-specific bottlenecks often runs much more slowly, sometimes below one instruction per cycle.

For example, a typical microarchitecture does not allow the results of an integer instruction to be used until the next cycle; multiple instructions in the same cycle must therefore be independent parallel operations. The results of a load instruction cannot be used until two or three cycles later, requiring even more independent instructions to be carried out in parallel. On "in-order" CPUs, parallel instructions need to write to different output registers; instruction scheduling is often heavily constrained by limits on the number of architectural registers. On "out-of-order" CPUs, controlling the precise scheduling of instructions can be extremely difficult.

This section reports the cycle counts that we have achieved on several specific CPUs. For each CPU we describe the important bottlenecks and the measures that we took to address those bottlenecks. We describe the CPUs in decreasing order of our cycle counts.

We report the speeds of our implementation and previous AES implementations measured by the eSTREAM benchmarking framework. The framework focuses on long-stream performance, but it also measures short-packet performance for the fastest software; our tables include the results for 576-byte packets.

4.1 Motorola PowerPC G4 7410, ppc32 Architecture

We measured our software on a computer named `gggg` in the Center for Research and Instruction in Technologies for Electronic Security (RITES) at the University of Illinois at Chicago. This computer has two 533MHz Motorola PowerPC G4 7410 processors; measurements used one processor. Resulting speeds for encrypting a long stream:

Software	Measurement	Cycles/byte
This paper	eSTREAM	14.57 (or 15.34 for 576 bytes)
Wu	eSTREAM	16.26
Bernstein	eSTREAM	17.84
Gladman	eSTREAM	26.74
OpenSSL 0.9.8c	`openssl speed aes`	29

Unpublished code by Denis Ahrens is claimed in [16] to use 25.06 cycles/byte on a PowerPC G4 7400, a very similar CPU to a PowerPC G4 7410, and 24.06 cycles/byte on a PowerPC G4 7457, a somewhat more powerful CPU.

Reducing instructions. For this CPU, our implementation uses 461 instructions in the main loop, specifically 180 load/store instructions, 279 integer instructions, and 2 branch instructions. We use the following techniques from

Section 3: combined shift-and-mask instructions; combined mask-and-insert; and counter-mode caching.

Reducing cycles. The PowerPC G4 7410 can dispatch at most 3 instructions per cycle. At most 2 of the instructions can be load/store or integer instructions, so our 459 non-branch instructions take at least $459/2 = 229.5$ cycles, i.e., 14.34 cycles/byte. At most 1 of the instructions can be a load/store instruction, but we have only 180 load/store instructions, so this is a less important bottleneck.

The G4 is, for most purposes, an in-order CPU, so load instructions have to be interleaved with arithmetic instructions. Results of load instructions are available after 3 cycles. Saving all possible callee-save registers makes 29 4-byte integer registers available for AES encryption. Detailed analysis shows that these are enough registers for almost perfect instruction scheduling, making the processor execute 2 instructions almost every cycle in the AES main loop. Our 233 cycles/loop are very close to the 229.5 cycles/loop lower bound for 459 instructions. We do not mean to suggest that 29 registers are ample; further registers would be useful for round-key caching.

4.2 Intel Pentium 4 f12, x86 Architecture

Warning: There are considerable performance differences between, e.g., a Pentium 4 f12, a Pentium 4 f29, a Pentium 4 f41, etc. "Pentium 4" is not an adequate CPU specification for performance measurements.

We measured our AES software on a computer named `fireball` in the Center for Research and Instruction in Technologies for Electronic Security (RITES) at the University of Illinois at Chicago. This computer has a single-core 1900MHz Intel Pentium 4 f12 processor. Resulting speeds for encrypting a long stream:

Software	Measurement	Cycles/byte
This paper	eSTREAM	14.13 (or 14.54 for 576 bytes)
Bernstein	eSTREAM	16.97
Wu	eSTREAM	18.23
OpenSSL 0.9.8g	`openssl speed aes`	21
Gladman	eSTREAM	26.48

Unpublished code by Matsui and Fukuda is claimed in [18] to use 15.69 cycles/byte on a "Pentium 4 Northwood," i.e., a Pentium 4 f2, and 17.75 cycles/byte on a "Pentium 4 Prescott," i.e., a Pentium 4 f3/f4. Unpublished code by Osvik is claimed in [20] to use 16.25 cycles/byte on an unspecified type of Pentium 4. Unpublished code by Lipmaa is claimed in [16] to use 15.88 cycles/byte on an unspecified type of Pentium 4. We have seen several other reports of Pentium 4 AES speeds above 20 cycles/byte.

Reducing instructions. For this CPU, our implementation uses 414 instructions in the main loop. We use the following techniques from Section 3: scaled-index loads; second-byte instructions; byte loads; two-byte loads; masked tables; combined load-xor; and counter-mode caching. We use some extra stores and

loads to handle the extremely limited number of general-purpose x86 integer registers. We compressed our total table size (including masked tables for the last round) to 4096 bytes; this improvement does not affect the eSTREAM benchmark results but reduces cache-miss costs in many applications.

Reducing cycles. There are several tricky performance bottlenecks on the Pentium 4. We recommend the manuals by Agner Fog [11] for much more comprehensive discussions of several x86 (and amd64) microarchitectures.

The most obvious bottleneck is that the Pentium 4 can do only one load per cycle. Our main loop has 177 loads, accounting for most—although certainly not all—of the 226 cycles that we actually use.

4.3 Sun UltraSPARC III, Sparcv9 Architecture

We measured our AES software on a computer named `icarus` at the University of Illinois at Chicago. This computer has eight 900MHz Sun UltraSPARC III CPUs; measurements used one CPU. Resulting speeds:

Software	Measurement	Cycles/byte
This paper	eSTREAM	12.06 (or 12.36 for 576 bytes)
Bernstein	eSTREAM	20.75
Gladman	eSTREAM	24.08
Wu	eSTREAM	28.88
OpenSSL 0.9.7e	`openssl speed aes`	35

We also measured our AES software on a computer named `nmisolaris10` in the NMI Build and Test Lab at the University of Wisconsin at Madison. This computer has two 1200MHz Sun UltraSPARC III Cu processors; measurements used one processor.

Software	Measurement	Cycles/byte
This paper	eSTREAM	12.03 (or 12.33 for 576 bytes)
Wu	eSTREAM	17.27
Bernstein	eSTREAM	25.08
Gladman	eSTREAM	25.08

Unpublished code by Lipmaa is claimed in [16] to use 16.875 cycles/byte on a "480 MHz SPARC," presumably an UltraSPARC II. Lipmaa discusses counter mode in [15] but does not report any speedups for the SPARC in this mode.

Reducing instructions. For this CPU, our implementation uses 505 instructions in the main loop, specifically 178 load/store instructions, 325 integer instructions, and 2 branch instructions. We use the following techniques from Section 3: padded registers; 32-bit shifts of padded registers; masked tables; and counter-mode caching.

Reducing cycles. An UltraSPARC CPU dispatches at most four instructions per cycle. Only one of these instructions can be a load/store instruction, so our

178 load/store instructions use at least 178 cycles. Furthermore, only two of these instructions can be integer instructions, so our 325 integer instructions use at least 162.5 cycles.

The simplest way to mask a byte is with an arithmetic instruction: for example, `&0xff00`. The SPARC architecture supports only 12-bit immediate masks, so three of the masks have to be kept in registers.

The UltraSPARC is an in-order CPU, except for store instructions. Proper instruction scheduling thus requires each load instruction to be grouped with two integer instructions. Only 24 8-byte integer registers are available, posing some challenges for instruction scheduling. We have built a simplified UltraSPARC simulator that accounts for 186 cycles with our current instruction scheduling; we are continuing to analyze the gaps between 178 cycles, 186 cycles, and the 193 cycles actually used by our main loop.

We have considered round-key recomputation (see Section 3) to trade some loads for integer instructions, but this makes scheduling even more difficult. With more registers we would expect to be able to reach approximately 170 cycles.

4.4 Intel Core 2 Quad Q6600 6fb, amd64 Architecture

We measured our AES software on a computer named `latour` in the Coding and Cryptography Computer Cluster (C4) at Technische Universiteit Eindhoven. This computer has a 2400MHz Intel Core 2 CPU with four cores; measurements used one core. Resulting speeds for encrypting a long stream:

Software	Measurement	Cycles/byte
This paper	eSTREAM	10.57 (or 10.79 for 576 bytes)
Wu	eSTREAM	12.27
Bernstein	eSTREAM	13.75
Gladman	eSTREAM	16.17
OpenSSL 0.9.8g	`openssl speed aes`	18

Unpublished code by Matsui and Nakajima is claimed in [19, Table 6] to use 14.5 cycles/byte (without bitslicing) on a Core 2. See also the discussion of bitslicing in Section 1.

Reducing instructions. For this CPU, our implementation uses 434 instructions in the main loop. We use the following techniques from Section 3: scaled-index loads; second-byte instructions; byte loads; two-byte loads; masked tables; combined load-xor; round-key recomputation; round-key caching; and counter-mode caching.

Reducing cycles. The Core 2 can dispatch three integer instructions per cycle but, like the Pentium 4, can dispatch only one load per cycle. We have often spent extra integer instructions to avoid loads and to improve the scheduling of loads. For example, we have kept round-key words in "XMM" registers, even though copying an XMM register to a normal integer register costs an extra integer instruction. Our main loop currently has 143 loads, accounting for most of our 169 cycles.

4.5 AMD Athlon 64 X2 3800+ 15/75/2, amd64 Architecture

We measured our AES software on a computer named `mace` in the Center for Research and Instruction in Technologies for Electronic Security (RITES) at the University of Illinois at Chicago. This computer has one 2000MHz AMD Athlon 64 X2 3800+ 15/75/2 CPU with two cores; measurements used one core. Resulting speeds for encrypting a long stream:

Software	Measurement	Cycles/byte
This paper	eSTREAM	10.43 (or 10.71 for 576 bytes)
Wu	eSTREAM	13.32
Bernstein	eSTREAM	13.40
Gladman	eSTREAM	18.06
OpenSSL 0.9.8g	`openssl speed aes`	21

Unpublished code by Matsui is claimed in [17] to use 10.62 cycles/byte on an Athlon 64. Unpublished code by Lipmaa is claimed in [16] to use 12.44 cycles/byte on an Athlon 64.

Reducing instructions. For this CPU, our implementation uses 409 instructions in the main loop. We use the following techniques from Section 3: scaled-index loads; second-byte instructions; byte loads; two-byte loads; masked tables; combined load-xor; and counter-mode caching.

Reducing cycles. The Athlon 64 can dispatch three instructions per cycle, including *two* load instructions. Loads and stores must be carried out in program order, placing a high priority on careful instruction scheduling.

Our Athlon-64-tuned software runs at 11.54 cycles/byte on the Core 2, and our Core-2-tuned software runs at 14.77 cycles/byte on the Athlon 64, illustrating the importance of microarchitectural differences.

References

1. Aoki, K., Lipmaa, H.: Fast implementations of AES candidates. In: AES Candidate Conference, pp. 106–120 (2000)
2. Atasu, K., Breveglieri, L., Macchetti, M.: Efficient AES implementations for ARM based platforms. In: SAC 2004: Proceedings of the 2004 ACM symposium on Applied computing, pp. 841–845. ACM, New York (2004), http://doi.acm.org/10.1145/967900.968073
3. Bernstein, D.J.: Cache-timing attacks on AES (2005), http://cr.yp.to/antiforgery/cachetiming-20050414.pdf
4. Bernstein, D.J.: Estreambench software package (2008), http://cr.yp.to/streamciphers/timings.html#toolkit-estreambench
5. Bertoni, G., Breveglieri, L., Fragneto, P., Macchetti, M., Marchesin, S.: Efficient software implementation of AES on 32-bit platforms. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 159–171. Springer, Heidelberg (2003)

6. Biryukov, A.: A new 128 bit key stream cipher: Lex (2005), http://www.ecrypt.eu.org/stream/papers.html
7. Daemen, J., Rijmen, V.: AES proposal: Rijndael (1999), http://www.iaik.tugraz.at/Research/krypto/AES/old/~rijmen/rijndael/rijndaeldocV2.zip
8. Darnall, M., Kuhlman, D.: AES software implementations on ARM7TDMI. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 424–435. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11941378_30
9. De Cannière, C.: The eSTREAM project: software performance (2008), http://www.ecrypt.eu.org/stream/perf
10. ECRYPT. The eSTREAM project (2008), http://www.ecrypt.eu.org/stream
11. Fog, A.: How to optimize for the Pentium family of microprocessors (2008), http://www.agner.org/assem/
12. Gladman, B.: AES and combined encryption/authentication modes (2006), http://fp.gladman.plus.com/AES/
13. Harrison, O., Waldron, J.: AES encryption implementation and analysis on commodity graphics processing units. In: Paillier and Verbauwhede [22], pp. 209–226
14. Könighofer, R.: A fast and cache-timing resistant implementation of the AES. In: Malkin, T.G. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 187–202. Springer, Heidelberg (2008)
15. Lipmaa, H.: AES ciphers: speed in no-feedback mode (2006), http://www.adastral.ucl.ac.uk/~helger/research/aes/nfb.html
16. Lipmaa, H.: AES/Rijndael: speed (2006), http://www.adastral.ucl.ac.uk/~helger/research/aes/rijndael.html
17. Matsui, M.: How far can we go on the x64 processors. In: Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 341–358. Springer, Heidelberg (2006), http://www.iacr.org/archive/fse2006/40470344/40470344.pdf
18. Matsui, M., Fukuda, S.: How to maximize software performance of symmetric primitives on Pentium III and 4 processors. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 398–412. Springer, Heidelberg (2005)
19. Matsui, M., Nakajima, J.: On the power of bitslice implementation on Intel Core2 processor. In: Paillier and Verbauwhede [22], pp. 121–134, http://dx.doi.org/10.1007/978-3-540-74735-2_9
20. Osvik, D.A.: Fast assembler implementations of the AES (2003), http://www.ii.uib.no/~osvik/pres/crypto2003.html
21. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: the case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11605805_1
22. Paillier, P., Verbauwhede, I. (eds.): CHES 2007. LNCS, vol. 4727. Springer, Heidelberg (2007)
23. Rebeiro, C., Selvakumar, A.D., Devi, A.S.L.: Bitslice implementation of AES. In: Pointcheval, D., Mu, Y., Chen, K. (eds.) CANS 2006. LNCS, vol. 4301, pp. 203–212. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11935070_14
24. Schneier, B., Whiting, D.: A performance comparison of the five AES finalists. In: AES Candidate Conference, pp. 123–135 (2000)
25. Weiss, R., Binkert, N.L.: A comparison of AES candidates on the Alpha 21264. In: AES Candidate Conference, pp. 75–81 (2000)
26. Worley, J., Worley, B., Christian, T., Worley, C.: AES finalists on PA-RISC and IA-64: implementations & performance. In: AES Candidate Conference, pp. 57–74 (2000)

A Review: One AES Round, in C

```
z0 = roundkeys[i * 4 + 0];
z1 = roundkeys[i * 4 + 1];
z2 = roundkeys[i * 4 + 2];
z3 = roundkeys[i * 4 + 3];

p00 = (uint32) y0 >> 20;
p01 = (uint32) y0 >> 12;
p02 = (uint32) y0 >> 4;
p03 = (uint32) y0 << 4;
p00 &= 0xff0;
p01 &= 0xff0;
p02 &= 0xff0;
p03 &= 0xff0;
p00 = *(uint32 *) (table0 + p00);
p01 = *(uint32 *) (table1 + p01);
p02 = *(uint32 *) (table2 + p02);
p03 = *(uint32 *) (table3 + p03);
z0 ^= p00;
z3 ^= p01;
z2 ^= p02;
z1 ^= p03;

p10 = (uint32) y1 >> 20;
p11 = (uint32) y1 >> 12;
p12 = (uint32) y1 >> 4;
p13 = (uint32) y1 << 4;
p10 &= 0xff0;
p11 &= 0xff0;
p12 &= 0xff0;
p13 &= 0xff0;
p10 = *(uint32 *) (table0 + p10);
p11 = *(uint32 *) (table1 + p11);
p12 = *(uint32 *) (table2 + p12);
p13 = *(uint32 *) (table3 + p13);
z1 ^= p10;
z0 ^= p11;
z3 ^= p12;
z2 ^= p13;

p20 = (uint32) y2 >> 20;
p21 = (uint32) y2 >> 12;
p22 = (uint32) y2 >> 4;
p23 = (uint32) y2 << 4;
p20 &= 0xff0;
p21 &= 0xff0;
p22 &= 0xff0;
p23 &= 0xff0;
p20 = *(uint32 *) (table0 + p20);
p21 = *(uint32 *) (table1 + p21);
p22 = *(uint32 *) (table2 + p22);
p23 = *(uint32 *) (table3 + p23);
z2 ^= p20;
z1 ^= p21;
z0 ^= p22;
z3 ^= p23;

p30 = (uint32) y2 >> 20;
p31 = (uint32) y2 >> 12;
p32 = (uint32) y2 >> 4;
p33 = (uint32) y2 << 4;
p30 &= 0xff0;
p31 &= 0xff0;
p32 &= 0xff0;
p33 &= 0xff0;
p30 = *(uint32 *) (table0 + p30);
p31 = *(uint32 *) (table1 + p31);
p32 = *(uint32 *) (table2 + p32);
p33 = *(uint32 *) (table3 + p33);
z3 ^= p30;
z2 ^= p31;
z1 ^= p32;
z0 ^= p33;
```

A New Class of Weak Encryption Exponents in RSA

Subhamoy Maitra and Santanu Sarkar

Indian Statistical Institute, 203 B T Road, Kolkata 700 108, India
{subho,santanu_r}@isical.ac.in

Abstract. Consider RSA with $N = pq$, $q < p < 2q$, public encryption exponent e and private decryption exponent d. We concentrate on the cases when $e(= N^{\alpha})$ satisfies $eX - ZY = 1$, given $|N - Z| = N^{\tau}$. Using the idea of Boneh and Durfee (Eurocrypt 1999, IEEE-IT 2000) we show that the LLL algorithm can be efficiently applied to get Z when $|Y| = N^{\gamma}$ and $\gamma < 4\alpha\tau \left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{(\frac{1}{4\tau} + \frac{1}{12\alpha})^2 + \frac{1}{2\alpha\tau}(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau})} \right)$. This idea substantially extends the class of weak keys presented by Nitaj (Africacrypt 2008) when $Z = \psi(p, q, u, v) = (p - u)(q - v)$. Further, we consider $Z = \psi(p, q, u, v) = N - pu - v$ to provide a new class of weak keys in RSA. This idea does not require any kind of factorization as used in Nitaj's work. A very conservative estimate for the number of such weak exponents is $N^{0.75-\epsilon}$, where $\epsilon > 0$ is arbitrarily small for suitably large N.

Keywords: Cryptanalysis, Factorization, Lattice, LLL Algorithm.

1 Introduction

RSA [13] is the most well known public key cryptosystem. The security of RSA depends on the hardness of factorization. Though RSA is quite secure if properly used, the extensive literature in RSA cryptanalysis identified different scenario where the security can be compromised. Before proceeding further, let us briefly explain the standard notations related to RSA cryptosystem: (i) primes p, q with same bit size, i.e., $q < p < 2q$. (ii) $N = pq$, $\phi(N) = (p - 1)(q - 1)$; (iii) e, d are such that $ed = 1 + t\phi(N)$, $t \geq 1$; (iv) N, e are available in public domain and the message M is encrypted as $C = M^e \bmod N$; (v) the secret key d is required to decrypt the message as $M = C^d \bmod N$.

There is detailed literature on RSA cryptanalysis and a survey on the attacks on RSA before the year 2000 is available in [3]. One very important result regarding RSA weak keys has been presented in [14], where it has been shown that N can be factored from the knowledge of N, e if $d < \frac{1}{3}N^{\frac{1}{4}}$. The important idea of [6] using lattice based techniques has also been exploited in great detail [4,5,1] to find weak keys of RSA when $d < N^{0.292}$. For very recent results on RSA cryptanalysis, one may refer to [7] and the references therein. Even with all these cryptanalytic ideas, RSA is still quite secure if used properly. In this

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 337–349, 2008.

direction, identifying new weak keys of RSA is always important so that RSA can be applied in a secured manner.

In [2], it has been shown that p, q can be found in polynomial time for every N, e satisfying $ex + y \equiv 0 \bmod \phi(N)$, with $x \leq \frac{1}{3}N^{\frac{1}{4}}$ and $|y| = O(N^{-\frac{3}{4}}ex)$; further some extensions considering the difference $p-q$ have also been considered. The work of [2] uses the result of [6] as well as the idea of CF expression [14] in their proof. The number of such weak keys has been estimated as $N^{\frac{3}{4}-\epsilon}$.

In a similar direction of [2], further weak keys are presented in [12]. The idea of [12] is as follows. Consider that e satisfies $eX - (p-u)(q-v)Y = 1$ with $1 \leq Y < X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$, $|u| < N^{\frac{1}{4}}$, $v = \left[-\frac{qu}{p-u}\right]$ ($[x]$ means the nearest integer of the real number x). If all the prime factors of $p-u$ or $q-v$ are less than 10^{50}, then N can be factored from the knowledge of N, e. The number of such weak exponents are estimated as $N^{\frac{1}{2}-\epsilon}$.

In [12], Continued Fraction (CF) expression is used to find the unknowns X, Y among the convergents of $\frac{e}{N}$. We immediately get improved results over [12] using the LLL [9] algorithm and our results are as follows.

- The bound on Y can be extended till N^{γ}, $\gamma < 4\alpha\tau\left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{(\frac{1}{4\tau} + \frac{1}{12\alpha})^2 + \frac{1}{2\alpha\tau}(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau})}\right)$, given $e = N^{\alpha}$ and $|N - (p-u)(q-v)| = N^{\tau}$.
- The only constraint on X is to satisfy the equation $eX-(p-u)(q-v)Y = 1$, which gives $X = \frac{1+(p-u)(q-v)Y}{e}$, i.e., $X = \lceil N^{1+\gamma-\alpha} \rceil$.
- In [12], the constraint $1 \leq Y < X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$ forces that the upper bound of e is $O(N)$. However, in our case the value of e can exceed this bound. Our results work for e upto $N^{1.875}$ for $\tau = \frac{1}{2}$.

In fact, our result is more general. Instead of considering some specific form $eX - (p-u)(q-v)Y = 1$, we consider equations like $eX - ZY = 1$, where $Z = \psi(p, q, u, v)$. Given $e = N^{\alpha}$ and the constraint $|N - Z| = N^{\tau}$, we can efficiently find Z using the LLL algorithm when $|Y| = N^{\gamma}$, where

$\gamma < 4\alpha\tau\left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{(\frac{1}{4\tau} + \frac{1}{12\alpha})^2 + \frac{1}{2\alpha\tau}(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau})}\right)$.

As an example, we consider $Z = \psi(p, q, u, v) = N - pu - v$ to present a new class of weak keys in RSA. This idea does not require any kind of factorization as used in [12]. We estimate a lower bound on the number of weak keys in this class as $N^{0.75-\epsilon}$.

The organization of the paper is as follows. In Section 2 we present our main result following the idea of [4,5]. The improvements over the results of [12] are presented in Section 3. A completely new class of weak keys when $Z = \psi(p, q, u, v) = N - pu - v$ is studied in Section 4. Section 5 concludes the paper.

2 Our Basic Technique

In this section we build the framework of our analysis related to weak keys. First we present a result based on continued fraction expression.

Lemma 1. *Let $N = pq$ be an RSA modulus with $q < p < 2p$. Consider that e satisfies the equation $eX - ZY = 1$ where $|N - Z| = N^\tau$. Then $\frac{Y}{X}$ is one of the convergents in the CF expansion of $\frac{e}{N}$ when $2XY < N^{1-\tau}$.*

Proof. We have $\frac{e}{N} - \frac{Y}{X} = \frac{eX-NY}{NX} = \frac{1-(N-Z)Y}{NX} \approx -\frac{(N-Z)Y}{NX}$. So, $|\frac{e}{N} - \frac{Y}{X}| \approx |\frac{(N-Z)Y}{NX}| = \frac{N^\tau Y}{NX} = \frac{N^{\tau-1}Y}{X}$. So, $\frac{Y}{X}$ will be one of the convergents of $\frac{e}{N}$ if $\frac{N^{\tau-1}Y}{X} < \frac{1}{2X^2} \Leftrightarrow 2XY < N^{1-\tau}$. □

We will use the above result later to demonstrate certain improvements over existing schemes. Next we present the following theorem which is the core of our results. For detailed ideas related to lattices, one may have a look at [4,5].

Theorem 1. *Let $N = pq$ be an RSA modulus with $q < p < 2p$. Consider that e $(= N^\alpha)$ satisfies the equation $eX - ZY = 1$ where $|N - Z| = N^\tau$, and $|Y| = N^\gamma$. Then we can apply LLL algorithm efficiently to get Z when $\gamma < 4\alpha\tau\left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{(\frac{1}{4\tau} + \frac{1}{12\alpha})^2 + \frac{1}{2\alpha\tau}(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau})}\right)$.*

Proof. We have $eX - ZY = 1$, which can also be written as $eX = 1 + NY + (Z - N)Y$. Hence, $1 + NY + (Z - N)Y = 0 \bmod e$. Thus, we have to find the solution of $f(x, y) = 1 + Nx + xy$ in $\mathbb{Z}_e$, where $x = Y, y = Z - N$ (the unusual assignment of Y to x is to maintain similar notation as in [4] in the following part of the proof).

We have to find x, y such that $1 + x(N + y) \equiv 0 \bmod e$, where $|x| = N^\gamma = e^{\frac{\gamma}{\alpha}}$ and $|y| = N^\tau = e^{\frac{\tau}{\alpha}}$. Let $X_1 = e^{\frac{\gamma}{\alpha}}, Y_1 = e^{\frac{\tau}{\alpha}}$. One may refer to [4, Section 4] for $det_x = e^{m(m+1)(m+2)/3} X_1^{m(m+1)(m+2)/3} Y_1^{m(m+1)(m+2)/6}$ and $det_y = e^{tm(m+1)/2} X_1^{tm(m+1)/2} Y_1^{t(m+1)(m+t+1)/2}$. Plugging in the values of X_1 and Y_1, we obtain, $det_x = e^{m^3(\frac{1}{3}+\frac{\gamma}{3\alpha}+\frac{\tau}{6\alpha})+o(m^3)}$, $det_y = e^{tm^2(\frac{1}{2}+\frac{\gamma}{2\alpha}+\frac{\tau}{2\alpha})+\frac{mt^2\tau}{2\alpha}+o(tm^2)}$. Now $det(L) = det_x det_y$ and we need to satisfy $det(L) < e^{mw}$, where $w = (m + 1)(m + 2)/2 + t(m + 1)$, the dimension of L. To satisfy $det(L) < e^{mw}$, we need $m^3(\frac{1}{3} + \frac{\gamma}{3\alpha} + \frac{\tau}{6\alpha}) + tm^2(\frac{1}{2} + \frac{\gamma}{2\alpha} + \frac{\tau}{2\alpha}) + \frac{mt^2\tau}{2\alpha} < \frac{m^3}{2} + tm^2$, ignoring the smaller terms. This leads to $m^2(\frac{1}{3} + \frac{\gamma}{3\alpha} + \frac{\tau}{6\alpha} - \frac{1}{2}) + tm(\frac{1}{2} + \frac{\gamma}{2\alpha} + \frac{\tau}{2\alpha} - 1) + \frac{t^2\tau}{2\alpha} < 0$. After fixing an m, the left hand side is minimized at $t = m(\frac{\alpha}{2\tau} - \frac{1}{2} - \frac{\gamma}{2\tau})$. Putting this value we have, $(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau}) + \gamma(\frac{1}{4\tau} + \frac{1}{12\alpha}) - \frac{\gamma^2}{8\alpha\tau} < 0$.

So, $\gamma < 4\alpha\tau\left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{(\frac{1}{4\tau} + \frac{1}{12\alpha})^2 + \frac{1}{2\alpha\tau}(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau})}\right)$. Similar to the idea in [4, Section 4], if the first two elements (polynomials $P_1(x, y), P_2(x, y)$) of the reduced basis out of the LLL algorithm are algebraically independent (i.e., nonzero resultant $res(P_1, P_2)$ which is a polynomial of y, say), then we get y by solving $res(P_1, P_2) = 0$. The value of y gives $Z - N$. (This actually happens with a high probability in practice as we have also checked by experimentation.) □

Based on Theorem 1, one can design

- a probabilistic polynomial time algorithm $\mathcal{A}$, which will take
- $N, e = N^\alpha$ as inputs
- and will provide the correct Z if

- $eX - ZY = 1$, with $|N - Z| = N^{\tau}$, and $Y = N^{\gamma}$, where $\gamma < 4\alpha\tau \left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{(\frac{1}{4\tau} + \frac{1}{12\alpha})^2 + \frac{1}{2\alpha\tau}(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau})} \right)$.
- and the resultant polynomial $res(P_1, P_2)$ on y is nonzero with integer solution (in practice the integer solution is correct with a high probability).

If Z is known, one can try to get further information on the primes. As example, in [12], $Z = \psi(p, q, u, v) = (p-u)(q-v)$, the knowledge of which presents a class of weak keys in RSA. In our further analysis, we use $Z = \psi(p, q, u, v) = N - pu - v$ in Section 4.

We now like to list the following points.

- For a fixed α, the value of γ decreases when τ increases and
- given a fixed τ, the value of γ increases with the increased value of α.

Below we present the numerical values of γ corresponding to α following Theorem 1 for three different values of τ, which are $\frac{1}{4}, \frac{1}{2}, \frac{3}{4}$.

In the work of [12], the value of τ has been taken as $\frac{1}{2}$. Thus we discuss some cases when $\tau = \frac{1}{2}$ to highlight the improvements we achieve over [12]. Note that for randomly chosen e's such that $e < \phi(N)$, the value of e will be $O(N)$ in most of the cases. In such a case, putting $\alpha = 1$ and $\tau = \frac{1}{2}$, we get that $\gamma < 0.284$.

When $\alpha < 1$ and $\tau = \frac{1}{2}$, the bound on γ will decrease and it will become 0 at $\alpha = \frac{1}{2}$. However, for randomly chosen e's such that $e < \phi(N)$, this will happen in negligibly small proportion of cases.

Most interestingly, the bound of γ will increase further than 0.284 when $\alpha > 1$. Wiener's attack [14] becomes ineffective when $e > N^{1.5}$ and the Boneh-Durfee attack [5] becomes ineffective when $e > N^{1.875}$. Similar to the result of [5], equations of the form $eX - ZY = 1$ cannot be used for $e > N^{1.875}$, since in such case no X will exist given the bound on Y. For $\tau = \frac{1}{2}$, we have presented the theoretical results for e reaching $N^{1.875}$ in Table 1. Experimental results will not reach this bound as we work with small lattice dimensions in practice, but even then the experimental results for e reach close to the value $N^{1.875}$ as we demonstrate results for $N^{1.774}$ in Section 3.3 for 1000-bit N.

Note that here we present a theoretical estimate on the bound of γ. These bounds may not be achievable in practice due to the large lattice dimensions. However, the experimental results are close to the theoretical estimates, which are presented in Sections 3.3, 4.1.

Table 1. The numerical upper bounds of γ (in each cell) following Theorem 1, given different values of α and τ

α	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.875
$\tau = \frac{1}{4}$	0.482	0.555	0.629	0.704	0.780	0.856	0.934	1.012	1.091	1.150
$\tau = \frac{1}{2}$	0.284	0.347	0.412	0.477	0.544	0.612	0.681	0.751	0.821	0.875
$\tau = \frac{3}{4}$	0.131	0.188	0.245	0.305	0.365	0.427	0.489	0.553	0.618	0.667

3 Improvements over the Work of [12]

In this section we present various improvements over the work of [12]. For this, first we present an outline of the strategy in [12]. Consider that e satisfies $eX - (p-u)(q-v)Y = 1$ with $1 \leq Y < X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$, $|u| < N^{\frac{1}{4}}$, $v = \left[-\frac{qu}{p-u}\right]$. If all the prime factors of $p-u$ or $q-v$ are less than 10^{50}, then N can be factored from the knowledge of N, e. The number of such weak exponents are estimated as $N^{\frac{1}{2}-\epsilon}$. The flow of the algorithm in [12] is as follows.

1. Continued Fraction algorithm is used to find the unknowns X, Y among the convergents of $\frac{e}{N}$.
2. Then Elliptic Curve Factorization Method (ECM [10]) is used to partially factor $\frac{eX-1}{Y}$, i.e., $(p-u)(q-v)$.
3. Next, an integer relation detection algorithm (LLL [9]) is used to find the divisors of B_{ecm}-smooth part of $\frac{eX-1}{Y}$ in a small interval.
4. Finally, if $p-u$ or $q-v$ is found, the method due to [6] is applied.

After knowing $(p-u)(q-v)$, if one gets the factorization of $p-u$ or $q-v$, then it is possible to identify $p-u$ or $q-v$ efficiently and the overall complexity is dominated by the time required for factorization. According to [12], if ECM [10] is used for factorization, and if all prime factors of $p-u$ or $q-v$ are less than 10^{50}, then getting $p-u$ or $q-v$ is possible in moderate time. Once $p-u$ or $q-v$ is found, as u, v are of the order of $N^{\frac{1}{4}}$, using the technique of [6], it is possible to find p or q efficiently.

3.1 The Improvement in the Bounds of X, Y

In [12] the bounds of X and Y are given as $1 \leq Y < X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$. Since, u, v are of $O(N^{\frac{1}{4}})$, we get that $(p-u)(q-v)$ is $O(N)$. When e is $O(N^{1+\mu})$, $\mu > 0$ and X is $O(N^{\nu})$, $0 < \nu \leq \frac{1}{4}$, the value of eX is $O(N^{1+\mu+\nu})$. In such a case, Y will be $O(N^{\mu+\nu})$, which is not possible as $Y < X$. Thus the values of e are bounded by $O(N)$ in the work of [12]. Next we generalize the bounds on X, Y.

The method of [12] requires $1 \leq Y < X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$. For $\tau = \frac{1}{2}$, our result in Lemma 1 gives that it is enough to have $2XY < N^{\frac{1}{2}}$ pointing out better bounds than [12] as follows:

- there is no need to have $Y < X$ when X, Y are of $O(N^{\frac{1}{4}})$, and
- the bound of either X or Y can be greater than $N^{\frac{1}{4}}$ when the other one is less than $N^{\frac{1}{4}}$.

We have already noted that the values of e are bounded by $O(N)$ in the work of [12]. In our case, this bound is increased too. The exponent e is of the order of $\frac{(p-u)(q-v)Y}{X}$, which is $O(\frac{N^{2-\tau}}{X^2})$. Given $\tau = \frac{1}{2}$, when X is $O(N^{\frac{1}{4}})$, we get the bound on e as $O(N)$, which is same as what given in [12]; however, our bound increases as X decreases.

3.2 Further Improvement over Section 3.1

With our results in Theorem 1, we get further bounds on X, Y. Note that $Y = N^{\gamma}$. Thus, $X = \lceil N^{1+\gamma-\alpha} \rceil$. This gives, $XY = N^{1+2\gamma-\alpha}$, where $\gamma < 4\alpha\tau \left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{(\frac{1}{4\tau} + \frac{1}{12\alpha})^2 + \frac{1}{2\alpha\tau}(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau})} \right)$, as in Theorem 1. Our result gives the following improvements.

- The bound on Y can be extended till N^{γ}. One may have a look at Table 1 for the numerical values of the theoretical bounds of γ given some α. The experimental results are presented in Section 3.3.
- X has to satisfy the equation $eX - (p-u)(q-v)Y = 1$, which gives $X = \lceil N^{1+\gamma-\alpha} \rceil$. The value of X becomes smaller as α increases.
- The constraint $1 \leq Y < X < 2^{-\frac{1}{4}} N^{\frac{1}{4}}$ in [12] forces that the upper bound of e has to be $O(N)$. However, in our case the value of e can exceed this bound and the result works for e upto $N^{1.875}$ theoretically as described in Section 2.

3.3 Experimental Results

We have implemented the programs in SAGE 2.10.1 over Linux Ubuntu 7.04 on a computer with Dual CORE Intel(R) Pentium(R) D CPU 2.80GHz, 1 GB RAM and 2 MB Cache. Below we start with a complete example.

Example 1. Let us consider that N, e are available. We choose a 1000-bit N which is a product of two 500-bit primes. Let N be
6965301839116252842151601289304534942713324759565286529653767647
8768155930430666799437161475057925109785426932854120387907825516
6760398939767348434594081678022491354913429428712241705242402188
3293512985226845866182664930993217767070732004829330778668319338
4022584310722922583086543088894619639637867 53
and e be a 1000-bit number
6131104587526715137731121962949288372067853010907049196083307936
8589835495733258743040864963707793556567754437239459230699951652
4114049172146335728027441527646511562295901427592480445339976342
3629012467921720937327176882146018075507772227029755907501291937
3001161776473105277852317642726755078398159 27.

Running our method as explained in Theorem 1, we get $(p-u)(q-v)$ as
6965301839116252842151601289304534942713324759565286529653767647
8768155930430666799437161475057925109785426932854120387907825516
6760398939767348434593866189401923798371559862126676390152959012
1188144136872190253575242273886606436919626785397201816501702315
9018457675859611961778476725358482825420001 03.

The factorization of $(p-u)(q-v)$ is as follows:
$3^4 \times 17^{24} \times 61681^{24} \times 2297 \times 141803 \times 345133$
$\times$1412745718201607004385882806363338596530456748 3679
$\times$1734403587161852748161040884417992576345006759868812178 08
2685037749671290446013065114219 1039.

This requires 112151.62 seconds (less than 1.3 days) using the ECM method of factoring.

In this case, $p-u$ is $3\times 17^{24}\times 61681^{24}\times 345133$ and the rest of the terms will give $q-v$. Since, $p-u < N^{\frac{1}{4}}$, p can be found using the idea of [6] in polynomial time.

Finally one can find p, q as
3232329308513348128043498783477144091769556249463616612811415899
2596559541241660031449960551292345720627446011227767334525515385
020845432088003209987 27,
2154886205675565418695665855651429394513037499642411075207610599
5064373027494440020966640367528230480418297340818511556350343590
0138969547253354733323 9 respectively.

In this example, X, Y are respectively 275 and 274 bit numbers as follows:
3035420144102701673311659229411748291628760686018968001955956890
2170379456331382787 and
2671883975804894456278842490580443758950128272095600125097638973
0227494745205672316. Note that these numbers are clearly greater than $N^{\frac{1}{4}}$ (in contrary to the bound presented in [12] where $1 \le Y < X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$) as N is a 1000 bit integer here.

Now we show that the technique of [12] will not work here. We calculate the CF expansion of $\frac{e}{N}$ and study all the convergents $\frac{Y}{X}$ with denominator $X < 2^{-\frac{1}{4}}N^{\frac{1}{4}}$. Except $X = Y = 1$, no $\frac{eX-1}{Y}$ is an integer. When $X = Y = 1$, we have $\frac{eX-1}{Y} = e-1$. Thus in this case, $(p-u)(q-v) = e-1$. As given in [12, Lemma 4], one needs to satisfy the condition $|(p-u)(q-v)-N| < 2^{-\frac{1}{2}}N^{\frac{1}{2}}$. Thus, in this example, one needs to satisfy $|e-1-N| < 2^{-\frac{1}{2}}N^{\frac{1}{2}}$, which is not true.

Next we go for different cases with $e = N^{\alpha}$ having varying α given the same p, q in Example 1. The experimental results are as follows where each run to find $(p-u)(q-v)$ requires less than 15 minutes. Note that comparing to Table 1, the results in Table 2 gives little lower values of γ. Further, we do not get the solutions for these p, q values when $\alpha = 1.8, 1.875$ as $1+\gamma-\alpha$ becomes very close to zero and hence X does not exist given the bound of Y. The maximum e for which we get a valid X is of size 1774 bits in this example. Thus the maximum value of α for which our method works in this example is 1.774. The value of γ in this example is 0.778 as Y is a 778-bit integer.

Table 2. The numerical values of γ given α found by experiment when N is of 1000 bits and p, q are as in Example 1. The lattice has the parameters $m = 7, t = 3, w = 60$.

α	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.875
γ	0.274	0.336	0.399	0.464	0.529	0.596	0.659	0.726	–	–

4 A New Class of Weak Keys

The problem with the idea of [12] is that one needs to factorize $(p-u)(q-v)$ to attack RSA and this is only possible when the factors of either $(p-u)$ or $(q-v)$

are relatively small. In this section we present a new class of weak keys where there is no involvement of factorization at all. For this, let us first refer to the following existing result from [11, Theorem 10].

Theorem 2. *[11] Let $N = pq$ be an RSA modulus with $q < p < 2q$. Let u be an (unknown) integer that is not a multiple of q. Suppose we know an approximation $\hat{P}$ of pu with $|pu - \hat{P}| \leq 2N^{\frac{1}{4}}$. Then N can be factorized in time polynomial in* $\log N$.

After finding X and Y from the continued fraction expression (as given in Lemma 1 of $\frac{e}{N}$, one can get $N - pu - v$ and hence $pu + v$. In our case, the $\hat{P}$ of Theorem 2 is $pu + v$. Following Theorem 2, if $|v| < N^{\frac{1}{4}}$ and u is not a multiple of q, then one can get p.

Next we present further extension on this class of weak keys using the idea of Theorem 1 and noting $Z = \psi(p, q, u, v) = pq - pu - v = N - pu - v$. In this case, $|N - Z| = |pu + v| = N^{\tau}$. When $Y = N^{\gamma}$ and $e = N^{\alpha}$ then $X = \lceil N^{1+\gamma-\alpha} \rceil$. This gives, $XY = N^{1+2\gamma-\alpha}$, where
$\gamma < 4\alpha\tau \left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{(\frac{1}{4\tau} + \frac{1}{12\alpha})^2 + \frac{1}{2\alpha\tau}(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau})} \right)$, as in Theorem 1.

It is clear that when $N^{1+2\gamma-\alpha}$ is greater than the bound $\frac{N}{pu+v} = N^{1-\tau}$ of Lemma 1, then we get a larger class of weak keys using the LLL method instead of the technique using continued fraction expression. This happens when $1 + 2\gamma - \alpha > 1 - \tau$, which is true taking the value of the upper bound on γ.

As an example, taking $\alpha = 1$ and $\tau = \frac{1}{2}$ the upper bound of γ is 0.284 and in this case XY will be $N^{0.568}$. However, the bound from Lemma 1 in this case is $N^{0.5}$. For larger values of α, the bound on XY will increase further.

4.1 Experimental Results

We consider the same N as used in Example 1.

Example 2. Let us first apply the continued fraction method as explained in Lemma 1. The public exponent e is a 1000-bit number
692419179490482244433191998806567583495808947875560402122448655074128690949704828809730335657675687024439533049589338170873599164174174034188975911152337295179208385986195123683049905106852500834430037397316270086424933658833703279759991267661961106695199420327753974414344960816633731258838407337775 1. Then the continued fraction of $\frac{e}{N}$ gives X, Y (respectively) as
168499666669691498716668844293872691710232152640878578006897564057916750516149153812903301083887475716938857701405775134549853035313960.

Then we find $pu + v$ as
145571170693594476334648107890902240200017752724641162397297681658278344761540286158290720682977007139784420982238075922044258786603243206710852708500229540011415961 60.

From $pu + v$ we get p using the idea of Theorem 2 as in this case $u = 2^{52}$ is not a multiple of q and $v = 2^{175}$ is less than $N^{\frac{1}{4}}$.

Next we give an example using the LLL technique that cannot be done using the technique of Lemma 1.

Example 3. The public exponent e is a 1000-bit integer as follows:
687516170030337570454608977725479760158853535391807125913113000153328299906575533758892504686028065396816041796620837316367632067332030043195254536198882701786034369526609680993429591977911304810695130485689008459936413100391578316422327846859295059053363440166896857407938814185106980946848053261475 5.

Using Theorem 1 we get Y (a 240 bit integer)
174398174704213885381621483955069353166685834872767501841198166276216919 3, and $pu + v$ (a 552 bit integer)
145571170693594476334648107890902240200017752724641162397297681658278344761540286158290720682977007139784420982238075922044258786603243206710852708500229540011415961 60.

In this case u, v are same as in Example 2 and hence p can be found using the idea of Theorem 2. It is to note that in this case X is a 241 bit integer 176684706477838432958329750074291851582748389687561895812160620129261979 0. Note that $\lceil \frac{N}{pu+v} \rceil$ is a 448-bit integer. Now $2XY$ should be less than 448 bit for a success using the technique of Lemma 1, which is not possible as X, Y are 241 and 240 bit integers respectively. Thus the idea of continued fraction method can not be applied here.

Now we like to point out that the weak key of Example 3 is not covered by the works of [14,15,2]. In this case, the decryption exponent d is a 999-bit integer and hence the bound of [14] that $d < \frac{1}{3}N^{\frac{1}{4}}$ will not work here.

Here $p - q > N^{0.48}$ and according to [15, Section 6] one can consider $\beta = 0.48$. If $d = N^{\delta}$, then according to [15, Section 6] the bound of δ will be $2 - 4\beta < \delta < 1 - \sqrt{2\beta - \frac{1}{2}}$ for RSA to be insecure. Putting $\beta = 0.48$, one can get $0.08 < \delta < 0.3217$, which is much smaller than 0.999 (in Example 3, $d \approx N^{0.999}$) and hence the weak keys of [15] does not cover our result.

In [2, Theorem 4, Section 4], it has been shown that p, q can be found in polynomial time for every N, e satisfying $ex + y = k\phi(N)$, with $0 < x \leq \frac{1}{3}\sqrt{\frac{\phi(N)}{e}}\frac{N^{\frac{3}{4}}}{p-q}$ and $|y| \leq \frac{p-q}{\phi(N)N^{\frac{1}{4}}}ex$. According to [2], convergents of the CF expression of $\frac{e}{N-\lfloor 2\sqrt{N} \rfloor}$ will provide $\frac{k}{x}$. For the parameters in Example 3, we calculated all the convergents with $x \leq \frac{1}{3}\sqrt{\frac{\phi(N)}{e}}N^{\frac{3}{4}}$ and we find that for each such k, x, $|ex - k\phi(N)| > \frac{p-q}{\phi(N)N^{\frac{1}{4}}}ex$. As $y = ex - k\phi(N)$, the bound on $|y|$ is not satisfied. Thus the weak key presented in Example 3 is not covered by the work of [2].

Next we go for different cases with $e = N^{\alpha}$ having varying α given the same p, q in Example 1. The experimental results are as follows where each run to find $N - pu - v$ requires less than 15 minutes. As the attack works for the values $u = 2^{52}$ and $v = 2^{175}$ given in Example 2, we get $N^{\tau} = pu + v$ and hence $\tau = 0.552$. In the first row of Table 3, we present the values of α where $e = N^{\alpha}$; the second row provides the theoretical upper bound on γ according

Table 3. The numerical values of γ given different values of α when $\tau = 0.552$ (both theoretical and experimental values) when N is of 1000 bits and p, q are as in Example 1. The lattice has the parameters $m = 7, t = 3, w = 60$.

α	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.72
γ (theoretical)	0.250	0.311	0.374	0.438	0.504	0.570	0.638	0.706	0.720
γ (experimental)	0.240	0.300	0.362	0.426	0.491	0.555	0.621	–	–

to Theorem 1, given the values of α in the first row and $\tau = 0.552$; the third row presents the experimental values of γ, which is obtained from the bit size of Y as found in the experiments. In experiments, we do not get the solutions for these p, q values when $\alpha \geq 1.7$ as $1 + \gamma - \alpha$ becomes very close to zero and hence X does not exist given the bound of Y. The maximum e for which we get a valid X is of size 1653 bits in this example. Thus the maximum value of α for which our method works in this example is 1.653. The value of γ in such a case is 0.656 as Y is a 656-bit integer.

We like to point out that one can exploit the techniques using sub-lattices given in [5] for improvement in the bound of γ than in Theorem 1 (where we use the idea of lattices following [4]). In practice, the idea of sub-lattices helps in getting the same result with less lattice dimension. During actual execution, for fixed N, e, u, v, Y, consider that t_1 is the time in seconds to run the LLL algorithm, t_2 is the time in seconds to calculate the resultant and t_3 is the time in seconds to find the integer root of the resultant; and let us refer this as a tuple $< (b_N, b_e, b_u, b_v, b_Y), t_1, t_2, t_3 >^a$, where b_N, b_e, b_u, b_v, b_Y are the bit-sizes of N, e, u, v, Y respectively; and $a = L$ for full rank lattice and $a = S$ for sub-lattice. Our examples are with lattice parameters $m = 7, t = 3$ and thereby giving the dimension 60 for full rank lattice (following the idea of [4]) and dimension 43 for sub-lattice (exactly following [5] the dimension should be 45, but due to the upper bounds X_1, Y_1 in Theorem 1, we get lower sub-lattice dimension). The examples are as follows:

$< (1000, 1000, 52, 175, 240), 20, 373, 4 >^L$, $< (1000, 1000, 52, 175, 240), 14, 377, 4 >^S$,
$< (2000, 1995, 104, 350, 465), 79, 1074, 16 >^L$, $< (2000, 1995, 104, 350, 465), 68, 1075, 15 >^S$, and
$< (9999, 9999, 520, 1750, 2350), 4722, 5021, 248 >^L$, $< (9999, 9999, 520, 1750, 2350), 4426, 5028, 198 >^S$.

As long as t_1 is much less than t_2, using sub-lattices (following [5]) instead of lattices (following [4]) will not provide significant improvement in total execution time. However, when t_1 becomes dominant, then the implementation using sub-lattices will provide faster execution.

4.2 Estimation of Weak Keys

In this section, we estimate the number of exponents for which our method works. We first present a simple analysis.

Lemma 2. *Consider RSA with $N = pq$, where p, q are primes such that $q < p < 2q$. Let e be the public encryption exponent that satisfies $eX - (N - pu - v)Y = 1$. Then for $X = Y = 1$, N can be factorized in poly$(\log N)$ time from the knowledge of N, e when u is not a multiple of q and $|v| < N^{\frac{1}{4}}$. The number of such weak*

keys e, such that $e < N$ is $N^{\frac{3}{4}-\epsilon}$, where $\epsilon > 0$ is arbitrarily small for suitably large N.

Proof. Given the equation $eX - (N - pu - v)Y = 1$, we consider the scenario when $X = 1, Y = 1$, i.e., when $e = N - pu - v + 1$. In such a case, from e, we will immediately get $pu + v$ and then following Theorem 2, one can get p in $O(poly(\log N))$ time when $|v| < N^{\frac{1}{4}}$ and u is not a multiple of q. Considering $1 < e < \phi(N)$, we get that $pu + v < N$. Considering p, q are of same bit size, i.e., $q < p < 2q$, one may find $\sqrt{N} < p < \sqrt{2N}$ and $\sqrt{\frac{N}{2}} < q < \sqrt{N}$. As, $|v| < N^{\frac{1}{4}}$ and v may be a negative integer, a conservative upper bound of u is $\sqrt{\frac{N}{2}}$ and clearly in such a case u is not a multiple of q. The total number of options of u, v pairs when $0 < u < \sqrt{\frac{N}{2}}$ and $|v| < N^{\frac{1}{4}}$ is $\sqrt{\frac{N}{2}} \times 2N^{\frac{1}{4}} = \sqrt{2}N^{\frac{3}{4}}$. As we have to consider those e's which are coprime to $\phi(N)$, we can only consider those u, v pairs such that $\gcd(N - pu - v + 1, \phi(N)) = 1$. Similar to the arguements of [2, Lemma 13] and [12, Theorem 6], the number of such u, v pairs is $N^{\frac{3}{4}-\epsilon}$, where $\epsilon > 0$ is arbitrarily small for suitably large N. □

The result of Lemma 2 will actually work in a similar manner for any X, Y which are bounded by a small constant as one can search those values of X, Y pairs to guess $pu + v$. Now we discuss a more general scenario.

Consider $\tau \leq 1 - \epsilon_1$ for some arbitrarily small positive constant ϵ_1. We have $\sqrt{N} < p < \sqrt{2N}$, $pu + v = N^{\tau}$ and $|v| < N^{\frac{1}{4}}$. Thus, it is enough to consider $u \leq \frac{1}{\sqrt{2}}N^{\frac{1}{2}-\epsilon_1}$. In such a case, $N - pu - v$ will be $c_1 N$ for some constant $0 < c_1 < 1$. Considering $e = c_2 N$, with $0 < c_2 < 1$, a constant, we find that X, Y are of the same order. As we consider $e = c_2 N$, we can estimate α as 1. Following Theorem 1 and putting $\alpha = 1$, we find that as τ goes towards 1 (i.e., ϵ_1 goes to 0), the value of $4\alpha\tau\left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{(\frac{1}{4\tau} + \frac{1}{12\alpha})^2 + \frac{1}{2\alpha\tau}(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau})}\right)$ goes towards 0. As $\gamma < 4\alpha\tau\left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{(\frac{1}{4\tau} + \frac{1}{12\alpha})^2 + \frac{1}{2\alpha\tau}(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau})}\right)$, and $|Y| = N^{\gamma}$, given that X, Y are of the same order, this puts a constraint on X also and we need to consider $X < N^{\epsilon_3}$ where

$\epsilon_3 = 4\alpha\tau\left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{(\frac{1}{4\tau} + \frac{1}{12\alpha})^2 + \frac{1}{2\alpha\tau}(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau})}\right)$, for $\alpha = 1$ and $\tau = 1 - \epsilon_1$.

Let us refer to the following result from [8].

Theorem 3. *[8] Consider a large integer N that has a divisor d_1 in an interval $(y, z]$ for a large y. Then N has exactly one divisor (almost certainly) in this interval if $z \approx y + \frac{y}{(logy)^{\log 4 - 1}}$.*

Now we present a technical result required for counting the weak keys.

Lemma 3. *Let $N = pq$ with $q < p < 2q$ and $0 \leq u, u' \leq \frac{1}{\sqrt{2}}N^{\frac{1}{2}-\epsilon_1}$. Let X be an integer with $1 \leq X < N^{\epsilon_3}$ such that $gcd(X, N - pu - v) = 1$ and $gcd(X, N - pu' - v') = 1$, where $|v|, |v'| < N^{\frac{1}{4}}$. Let $e \equiv X^{-1} \bmod N - pu - v$ and $e' \equiv X^{-1} \bmod N - pu' - v'$. If $e = e'$ then almost surely $u = u', v = v'$.*

Table 4. Numerical values of ϵ_3 (i.e., the bound of γ) following Theorem 1 corresponding to the values of τ when $\alpha = 1$

τ	0.5	0.6	0.7	0.75	0.8	0.85	0.9	0.95	0.97	0.99
ϵ_3	0.284	0.220	0.160	0.130	0.100	0.077	0.051	0.025	0.015	0.005

Proof. We have $e = e'$. So $N - pu - v$ and $N - pu' - v'$ both divides $eX - 1$. Since $u, u' \leq \frac{1}{\sqrt{2}} N^{\frac{1}{2}-\epsilon_1}$, and $|v|, |v'| < N^{\frac{1}{4}}$ so $N - pu - v$ and $N - pu' - v'$ are in the interval $I = [N - N^{1-\epsilon_1} - N^{\frac{1}{4}}, N + N^{\frac{1}{4}}]$. Let $y = N - N^{1-\epsilon_1} - N^{\frac{1}{4}}$ and $y' = N + N^{\frac{1}{4}}$. One can check that $y' < y + \frac{y}{(logy)^{\log 4-1}}$ for a fixed ϵ_1 and a large N. Thus following Theorem 3, $N - pu - v = N - pu' - v'$ holds almost surely.

Given $N - pu - v = N - pu' - v'$, we get $pu + v = pu' + v'$ iff $p(u - u') = v' - v$. Since p is $O(\sqrt{N})$ and v, v' are $O(N^{\frac{1}{4}})$, we get $u = u'$ and $v = v'$. □

Thus if any one (or both) of the pairs u, u' and v, v' are distinct, then e, e' are almost surely distinct once X is fixed. The number of such distinct u, v pairs is $\frac{1}{\sqrt{2}} N^{\frac{3}{4}-\epsilon_1}$. Fixing X, for each pair of values of u, v, the exponent e need to be coprime to $\phi(N)$. Similar to the arguements of [2, Lemma 13] and [12, Theorem 6], the number of such u, v pairs is $\frac{1}{\sqrt{2}} N^{\frac{3}{4}-\epsilon_1-\epsilon_2}$, where $\epsilon_2 > 0$ is arbitrarily small for suitably large N. Once again, we like to highlight the constraint that $X < N^{\epsilon_3}$, where the value of ϵ_3 goes towards 0 as ϵ_1 goes to zero. Following Table 4, we get that when ϵ_1 is 0.01, then $\epsilon_3 = 0.005$. Thus, in this case, for any $X < N^{0.005}$, we get approximately $\frac{1}{\sqrt{2}} N^{0.74-\epsilon_2}$ many weak keys.

Note that, the problem becomes more complicated when we consider the set of weak keys for two unequal values of X, say X', X''. Let $H_{X'}$ and $H_{X''}$ be two different sets of weak keys for $X' \neq X''$. However, it is not clear what is the intersection between $H_{X'}$ and $H_{X''}$. If it can be shown that for distinct values of X, the corresponding sets H_X does not have quite a large intersection, then the total number of weak keys will be much higher than what demonstrated in this section. Further in this section we have only considered $e < N$. As the result of Theorem 1 shows that our technique can be applied for $e > N$ too, that will provide much larger class of weak keys.

5 Conclusion

In this paper we present some new weak keys of RSA. We study the public exponents $e(= N^\alpha)$ when $eX - ZY = 1$ under the constraint $|N - Z| = N^\tau$. We show that the LLL algorithm can be efficiently applied to get $Z = \psi(p, q, u, v)$ when $|Y| = N^\gamma$ and $\gamma < 4\alpha\tau \left(\frac{1}{4\tau} + \frac{1}{12\alpha} - \sqrt{(\frac{1}{4\tau} + \frac{1}{12\alpha})^2 + \frac{1}{2\alpha\tau}(\frac{1}{12} + \frac{\tau}{24\alpha} - \frac{\alpha}{8\tau})} \right)$.

Some specific forms of $\psi(p, q, u, v)$ are then studied. We improve the results of [12] taking $\psi(p, q, u, v) = (p - u)(q - v)$. Further, we consider $\psi(p, q, u, v) = N - pu - v$ to provide a new class of weak keys in RSA. A very preliminary analysis of this class shows that the number of weak keys is at least $N^{0.75-\epsilon}$ and

it seems that a better analysis will provide a better bound than this. Further, the study of different forms of $\psi(p, q, u, v)$ may provide additional weak keys that are not known till date.

References

1. Blömer, J., May, A.: Low secret exponent RSA revisited. In: Silverman, J.H. (ed.) CaLC 2001. LNCS, vol. 2146, pp. 4–19. Springer, Heidelberg (2001)
2. Blömer, J., May, A.: A generalized Wiener attack on RSA. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 1–13. Springer, Heidelberg (2004)
3. Boneh, D.: Twenty Years of Attacks on the RSA Cryptosystem. Notices of the AMS 46(2), 203–213 (1999)
4. Boneh, D., Durfee, G.: Cryptanalysis of RSA with private key d less than $N^{0.292}$. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 1–11. Springer, Heidelberg (1999)
5. Boneh, D., Durfee, G.: Cryptanalysis of RSA with private key d less than $N^{0.292}$. IEEE Trans. on Information Theory 46(4), 1339–1349 (2000)
6. Coppersmith, D.: Small solutions to polynomial equations and low exponent vulnerabilities. Journal of Cryptology 10(4), 223–260 (1997)
7. Jochemsz, E.: Cryptanalysis of RSA variants using small roots of polynomials. Ph. D. thesis, Technische Universiteit Eindhoven (2007)
8. Ford, K., Tenenbaum, G.: The distribution of Integers with at least two divisors in a short interval (last accessed July 1, 2008), http://arxiv.org/abs/math/0607460
9. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen 261, 513–534 (1982)
10. Lenstra, H.W.: Factoring integers with elliptic curves. Annals of Mathematics 126, 649–673 (1987)
11. May, A.: New RSA vulnerabilities using lattice reduction methods. PhD thesis, University of Paderborn (2003) (last accessed July 1, 2008), http://wwwcs.upb.de/cs/ag-bloemer/personen/alex/publications/
12. Nitaj, A.: Another Generalization of Wiener's Attack on RSA. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 174–190. Springer, Heidelberg (2008)
13. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public key cryptosystems. Communications of ACM 21(2), 158–164 (1978)
14. Wiener, M.: Cryptanalysis of short RSA secret exponents. IEEE Transactions on Information Theory 36(3), 553–558 (1990)
15. de Weger, B.: Cryptanalysis of RSA with small prime difference. Applicable Algebra in Engineering, Communication and Computing 13(1), 17–28 (2002)

Two New Efficient CCA-Secure Online Ciphers: MHCBC and MCBC

Mridul Nandi

National Institute of Standards and Technology
mridul.nandi@gmail.com

Abstract. Online ciphers are those ciphers whose ciphertexts can be computed in real time by using a length-preserving encryption algorithm. HCBC1 and HCBC2 are two known examples of Hash Cipher Block Chaining online ciphers. The first construction is secure against chosen plaintext adversary (or called CPA-secure) whereas the latter is secure against chosen ciphertext adversary (or called CCA-secure). In this paper, we have provided simple security analysis of these online ciphers. We have also proposed two new more efficient chosen ciphertext secure online ciphers modified-HCBC (MHCBC) and modified-CBC (MCBC). If one uses a finite field multiplication based universal hash function, the former needs one less key and one less field multiplication compared to HCBC2. The MCBC does not need any universal hash function and it needs only one blockcipher key unlike the other three online ciphers where two independent keys (hash function and blockcipher) are required.

Keywords: online cipher, CBC, universal hash function, random permutation.

1 Introduction

In this paper we fix a finite group $(G, +)$ (e.g., $\{0,1\}^n$ with bitwise addition $\oplus$ for some fixed positive integer n). An element of G is called a block. A cipher over a domain $D \subseteq G^+ := \cup_{i \geq 1} G^i$ is a keyed function family $\{F_K\}_{K \in \mathcal{K}}$, where $\mathcal{K}$ is a key space and for each key K, $F_K : D \to D$ is a length-preserving permutation on D (i.e., for each $x \in D \cap G^i$, $F_K(x) \in G^i$). Any cipher over domain G is called a blockcipher. An important blockcipher is AES [5] with

domain $G = \{0,1\}^{128}$. The pseudorandom permutation or PRP and strong pseudorandom permutation [8] or SPRP are two well known security notions for ciphers. They are also called chosen plaintext secure or CPA-secure and chosen ciphertext secure or CCA-secure. Intuitively, a cipher is called PRP (or SPRP) if it is indistinguishable from the ideal cipher based on encryption queries (or both encryption and decryption queries respectively). An ideal cipher on D is chosen at random from $\mathsf{Perm}(D)$, the set of all permutations on D.[1] In this paper we

[1] If D is an infinite set then we define the ideal cipher runtime as queries being asked to it. A similar definition can be found for an ideal online cipher as defined in figure 1.

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 350–362, 2008.

are interested in online cipher with domain G^+ whose i^{th} ciphertext block $C[i]$ is computable from the first i plaintext blocks $M[1], \cdots, M[i]$. The above property is popularly known as online property. One can show that online ciphers can not be PRP since the online property itself can be used to distinguish it from an ideal cipher which actually does not have this property. The appropriate security notions of an online cipher are CPA-online secure and CCA-online secure [1]. Informally an online cipher is CPA-online secure (or CCA-online secure) if it is indistinguishable from an ideal online cipher (see figure 1) based on only encryption queries (or encryption and decryption queries both respectively). The possible candidates of online ciphers are in [3,4,6], most of which are different variants of Cipher Block Chaining modes or CBC. If $e \in \mathsf{Perm}(D)$ then the CBC based on e with an initial value IV is defined as $e^+(x_1, \cdots, x_m) = (y_1, \cdots, y_m)$ where $y_i = E_K(y_{i-1} + x_i)$, $1 \leq i \leq m$ and $y_0 = \text{IV}$. One can see that CBC based on any blockcipher is an online cipher. In [1] authors have shown that CBC with public or secret IV [3] and ABC [6] ciphers (another example of online cipher) are not CPA-secure online ciphers. In the same paper [1] a CPA-secure HCBC1 online cipher and CCA-secure HCBC2 online cipher have been proposed. These online ciphers need two keys (for a blockcipher and a universal hash family [7,10,11]).

Applicability of online ciphers. The known online ciphers Hash-CBC namely, HCBC1, HCBC2 and our constructions need current and previous plaintext block and previous cipher block to compute the current cipher block. Thus, online cipher could be used where encryption is made in an online manner with a very small amount of memory or buffer. It could be useful in scenarios where there is a constraint requiring length-preserving ciphertext. For example, one can think some ciphers dealing with fixed packet formats, legacy code and disk-sector encryption. In this situation, a length preserving PRP or SPRP can be used but potentially these may be costlier than online ciphers as these are stronger security notions than a CPA or CCA online secure.

Our contributions. In this paper we have provided simple as well as concrete security analysis of HCBC1 and HCBC2. The proof idea can also be used in other online ciphers. For example, we have used the same approach to have security proof of our new proposals MHCBC and MCBC. These two new online ciphers have many advantages over the previous ciphers. MHCBC needs a universal

Table 1. In this comparison universal hash function is based on the field multiplication as given in example 1. Here, $G = \{0,1\}^n$, the set of blocks and k_{BC} denotes the key size of the blockcipher. The number of multiplications and block-cipher (BC) invocations are given to encrypt a plaintext with m blocks.

Name of Online cipher	CPA-secure	CCA-secure	# field multiplication	# BC	Key-size
HCBC1 [2]	✓	×	m	m	$n + k_{\text{BC}}$
HCBC2 [2]	✓	✓	$2m$	m	$2n + k_{\text{BC}}$
MHCBC	✓	✓	m	m	$n + k_{\text{BC}}$
MCBC	✓	✓	0	$2m + 1$	k_{BC}

hash function from G to G whereas HCBC2 needs a universal hash function from G^2 to G, which makes it a potential efficient candidate. For example, if we use field multiplication based universal hash function then we need one less field multiplication and one less key (see example 1). Our second construction modified-CBC or MCBC does not need any universal hash function. It needs only one blockcipher key. One can definitely replace the universal hash function of MHCBC by a blockcipher (since an ideal blockcipher is Δuniversal hash function, see example 2) at the cost of an extra new independent blockcipher key which eventually causes an extra key-scheduling algorithm. In this paper we have shown that if we use same blockcipher key then MHCBC is not CCA-secure. Thus, the security of MCBC indeed is not straightforward from that of MHCBC. Table 1 provides a comparison of these four online ciphers.

2 Basic Definition and Results

We write $M = (M[1], \cdots, M[m])$ where $M[i] \in G$ is the i^{th} block of M. For $1 \le i \le j \le m$, $M[i..j]$ represents $(M[i], M[i+1], \cdots, M[j])$. If $j < i$ then by convention $M[i..j] = \lambda$, the empty string. $X \xleftarrow{*} S$ means that the random variable X is chosen uniformly from a finite set S and it is independent with all previously defined random variables. An equivalent phrase "at random" is also widely used in cryptology. Let $F : \mathcal{K} \times D \to R$ where $\mathcal{K}$ is a finite set. By abuse of notation we denote $\mathsf{f} \xleftarrow{*} F$ to mean that $\mathsf{f} := F_K$ where $K \xleftarrow{*} \mathcal{K}$. The interpolation probability of F for any fixed tuple $\tau := ((M_1, C_1), \cdots, (M_q, C_q))$ is the probability $\Pr[\mathsf{f}(M_1) = C_1, \cdots, \mathsf{f}(M_q) = C_q]$. We define the set of all non-empty prefixes of $M_1, \cdots, M_q \in G^+$ as $\mathbb{P}(M_1, \cdots, M_q)$ or $\mathbb{P}_M$. We denote $\sigma := \sigma(M_1, \cdots, M_q) = |\mathbb{P}(M_1, \cdots, M_q)|$. Clearly, $\sigma \le \sum_{i=1}^{q} \|M_i\|$. For $p \in G^+$ with $\|p\| = m$, we define $\mathrm{chop}(p) = p[1..m-1]$ and $\mathrm{last}(p) = p[m]$. In other words, $p = (\mathrm{chop}(p), \mathrm{last}(p))$ where $\mathrm{last}(p) \in G$ and $\mathrm{chop}(p) \in G^*$. Note that if $p \in G$ then $\mathrm{chop}(p) = \lambda$ and $\mathrm{last}(p) = p$. By our convention, $\mathrm{last}(\lambda) = \mathbf{0}$. Now we define $\mathbb{P}'_M = \mathbb{P}'(M_1, \cdots, M_q) := \{p' \in G^* : p' = \mathrm{chop}(p) \text{ for some } p \in \mathbb{P}_M\}$. Let $\mathbf{P}(N, k) := N(N-1)\cdots(N-k+1)$ for any positive integer $k < N$. A function $H : \mathcal{K} \times D \to G$ is called ε-Δuniversal hash function with domain D and key-space $\mathcal{K}$ if $\Pr[\mathsf{h}(x_1) = \mathsf{h}(x_2) + y] \le \varepsilon, \forall x_1 \ne x_2 \in D, y \in G$ where h denotes $H(K, \cdot)$ and K is chosen uniformly from $\mathcal{K}$. Now we state a simple and useful property of ε-Δuniversal hash function and provide two examples of universal hash function. An *uniform random permutation* or URP over G is a random variable $E^u \xleftarrow{*} \mathsf{Perm}(G)$ (it means that E^u is chosen uniformly and independently from $\mathsf{Perm}(G)$). It is an ideal candidate of a blockcipher. Due to limited space we omit most of the proofs and those can be found in the full version [9].

Lemma 1. *Let H be an ε-Δuniversal hash function and let $(x_1, y_1), \cdots, (x_\sigma, y_\sigma) \in D \times G$ be distinct. Then,* $\Pr[\mathsf{h}(x_i) + y_i = \mathsf{h}(x_j) + y_j,\ 1 \le i \ne j \le \sigma] \le \frac{\varepsilon\sigma(\sigma-1)}{2}$.

Example 1. Suppose $(G, +, \cdot)$ is a finite field. Let $D = G = \mathcal{K}$ then $H(K, x) = K \cdot x$ is $\frac{1}{N}$-Δuniversal hash function. This is true since the number of K's such that

$K\cdot(x_1-x_2)=y$ is at most one. Similarly $D=G^2=\mathcal{K}$. Let $H((K_1,K_2),(x,y))=K_1\cdot x+K_2\cdot y$ then it is a $\frac{1}{N}$-Δuniversal hash function.

Example 2. Let $E^u \xleftarrow{*} \mathsf{Perm}(G)$ then for any $x_1\neq x_2\in G$ and $y\in G\setminus\{\mathbf{0}\}$ we have $\Pr[E^u(x_1)=E^u(x_2)+y]=\frac{1}{N-1}$. If $y=\mathbf{0}$ then $\Pr[E^u(x_1)=E^u(x_2)+y]=\Pr[E^u(x_1)=E^u(x_2)]=0$. Hence E^u is $\frac{1}{N-1}$-Δuniversal hash function.

Proposition 1. *Let $X_1,Y_1,\cdots,X_k,Y_k$ be random variables taking values on G and $E^u \xleftarrow{*} \mathsf{Perm}(G)$. Let $\mathsf{COLL}_{\text{in}}$ denote the event that X_i's are not distinct, $\mathsf{COLL}_{\text{out}}$ denote the event that Y_i's are not distinct then*

$$\Pr[E^u(X_1)=Y_1,\cdots,E^u(X_k)=Y_k]\geq\frac{1-\Pr[\mathsf{COLL}_{\text{in}}]-\Pr[\mathsf{COLL}_{\text{out}}]}{\mathbf{P}(N,k)}. \quad (1)$$

We say a distinguisher is (q,σ)-CPA if it asks at most q encryption queries such that total number of blocks in all queries is at most σ. Similarly we can define (q,σ)-CCA distinguisher where it can ask both encryption and decryption queries. Let $F:\mathcal{K}\times D\to D$ be a cipher with finite key-space $\mathcal{K}$. Suppose an oracle algorithm or distinguisher $\mathcal{A}$ has access of a cipher with domain D. Now $\mathcal{A}^F\Rightarrow b$ represents the event that $\mathcal{A}$ outputs b after interacting with F_K where $K\xleftarrow{*}\mathcal{K}$. Similarly we define $\mathcal{A}^{F,F^{-1}}\Rightarrow b$. The CPA advantage of oracle algorithms for a blockcipher $E:\mathcal{K}\times G\to G$ is $\mathbf{Adv}^{\text{CPA}}_{\mathcal{A}}(E,E^u)=\Pr[\mathcal{A}^E\Rightarrow 1]-\Pr[\mathcal{A}^{E^u}\Rightarrow 1]$. The prp-insecurity of E is $\mathbf{Insec}^{\text{prp}}_{E}(q,\sigma)=\max_{\mathcal{A}}\mathbf{Adv}^{\text{CPA}}_{\mathcal{A}}(E,E^u)$ where maximum is taken over all (q,σ)-CPA distinguishers. CCA advantage of $\mathcal{A}$ and sprp-insecurity of a blockcipher can be defined similarly. Let $\mathcal{A}^{F,F^{-1}}$ be an oracle algorithm having access of an online cipher and its inverse. Responses of an ideal online cipher or uniform random online permutation (UROP) oracle Π^u and its inverse oracle $(\Pi^u)^{-1}$ is defined below in figure 1. Like insecurity of a blockcipher we can define $\mathbf{Insec}^{\text{prop}}_{F}(q,\sigma)$ and $\mathbf{Insec}^{\text{sprop}}_{F}(q,\sigma)$ where F is an online cipher. It is the maximum CPA or CCA advantage for (q,σ) distinguishers distinguishing F from the ideal online cipher Π^u.

The *longest common prefix* of $M,M'\in G^*$ (or $\mathsf{LCP}(M,M')$) is the block-sequence $N\in G^\ell$ such that N is a longest common prefix of M and M'. One can check that $\mathsf{LCP}(M,M')$ always exists and it is unique. The length of the longest common prefix is denoted by $\ell_{M,M'}$. Any element of the form $\tau=((M_1,C_1),\cdots,(M_q,C_q))\in\mathcal{T}:=((G^+)^2)^+$ is known as *qr-tuple* or *query-response tuple*[2] where M_i's and C_i's are block-sequences.

A qr-tuple $\tau=((M_1,C_1),\cdots,(M_q,C_q))$ is said to be **online-compatible** if $\ell_{M_i,M_j}=\ell_{C_i,C_j}$, for all $1\leq i,j\leq q$. Let $\mathcal{T}_{\text{oc}}$ be the set of all online-compatible qr-tuples. Let $\tau=((M_1,C_1),\cdots,(M_q,C_q))\in\mathcal{T}_{\text{oc}}$ be a qr-tuple then for any $p\in\mathbb{P}_M:=\mathbb{P}(M_1,\cdots,M_q)$ we define the **corresponding block-sequence** $q^p\in\mathbb{P}_C:=\mathbb{P}(C_1,\cdots,C_q)$ by $C_i[1..j]$ where $p=M_i[1..j]$[3]. Now we provide bounds of interpolation probability of UROP Π^u.

[2] Later we see that the pair (M_i,C_i) corresponds to a query-response pair where M_i is encryption query and C_i is response or C_i is decryption query and M_i is response.

[3] Clearly, $j=||p||$ but there can be more than one choices of i. So we need to check well defined-ness of q^p. Suppose $p=M_i[1..j]=M_{i'}[1..j]$ for some $i,i'\leq q$ and

Initially $\mathbb{P} = \emptyset$ and a function $\Gamma : \mathbb{P} \to \mathsf{Perm}(G)$.

On encryption query $M \in G^m$	On decryption query $C \in G^m$
for $i = 1$ to m	for $i = 1$ to m
$p = M[1..i-1]$;	$p = M[1..i-1]$;
if $p \in \mathbb{P}$ then $C[i] = \Gamma(p)(M[i])$;	if $p \in \mathbb{P}$ then $M[i] = \Gamma(p)^{-1}(C[i])$;
else	else
$\Pi_p^u \xleftarrow{*} \mathsf{Perm}(G)$;	$\Pi_p^u \xleftarrow{*} \mathsf{Perm}(G)$;
$\mathbb{P} \leftarrow \mathbb{P} \cup \{p\}$;	$\mathbb{P} \leftarrow \mathbb{P} \cup \{p\}$;
$\Gamma(p) = \Pi_p^u$;	$\Gamma(p) = \Pi_p^u$;
$C[i] = \Gamma(p)(M[i])$;	$M[i] = \Gamma(p)^{-1}(C[i])$;
endif	endif
endfor	endfor
return $C = C[1..m]$;	return $M = M[1..m]$;

Fig. 1. Responses of a UROP oracle Π^u and its inverse oracle $(\Pi^u)^{-1}$

Lemma 2. (Interpolation probability of UROP)
Let $\mathbf{Pr} := \Pr[\Pi^u(M_1) = C_1, \cdots, \Pi^u(M_q) = C_q] = 0$ *where* Π^u *is an UROP. Now* $\frac{1}{N^\sigma} \le \mathbf{Pr} \le \frac{1}{\mathbf{P}(N,\sigma)}$ *if* $((M_1, C_1), \cdots, (M_q, C_q))$ *is an* online-compatible, *otherwise* $\mathbf{Pr} = 0$, *where* σ *is the total number of blocks in all* q *plaintexts.*

Now we define an important object called view or transcript[4] of a distinguisher which actually contains all query-responses in the form of tuple.

Definition 1. (view or transcript of an adversary) *Let* $g : G^+ \to G^+$ *be an oracle and* $\tau = ((M_1, C_1), \cdots, (M_q, C_q)) \in \mathcal{T}$, *for some* $q \ge 1$.

(1) Chosen Plaintext Adversary *:* τ *is called a view of* $\mathcal{A}^g$ *and denoted as* $\mathsf{view}(\mathcal{A}^g)$ *if* $M_1, \cdots, M_q$ *are all the queries made by* $\mathcal{A}$ *and* $C_1, \cdots, C_q$ *are the corresponding responses.*

(2) Chosen Ciphertext Adversary *:* τ *is called a view of* $\mathcal{A}^{g,g^{-1}}$ *and denoted as* $\mathsf{view}(\mathcal{A}^{g,g^{-1}})$, *if* M_i *is the query and* C_i *is the response whenever the* i^{th} *query is* g*-query or if* C_i *is query and* M_i *is response whenever* i^{th} *query is* g^{-1}*-query (in both cases we have* $g(M_i) = C_i$*).*

Now we define some views, subsets of $\mathcal{T}_{\text{oc}}$, called bad views and bound the probability that a bad view occurs when a distinguisher is interacting with uniform random online permutation Π^u. These bad views will be considered as bad views for the online ciphers HCBC1, HCBC2, MHCBC and MCBC. Let $\tau = ((M_1, C_1), \cdots, (M_q, C_q))$ and $x_p = \mathrm{last}(p)$ and $y_p = \mathrm{last}(q^p)$.

hence $\ell_{M_i,M_{i'}} \ge j$. Since τ is online compatible, $\ell_{C_i,C_{i'}} = \ell_{M_i,M_{i'}} \ge j$. Thus, $C_i[1..j] = C_{i'}[1..j]$.

[4] The term "transcript" has been used in many literatures, but in this paper we mainly use the word view as it really signifies the view of the oracle which is obtained by the distinguisher after having query-responses.

$\mathcal{V}_{\text{bad},1} = \{\tau \in \mathcal{T}_{\text{oc}} :\ y_{p_1} = y_{p_2}\ \text{ or }\ y_{p_1} = \mathbf{0},\ \text{for some } p_1 \neq p_2 \in \mathbb{P}_M\}$
$\mathcal{V}_{\text{bad},2} = \{\tau \in \mathcal{T}_{\text{oc}} :\ (y_{p_1}, x_{p_1}) = (y_{p_2}, x_{p_2}) \text{ or } (y_{p_1}, x_{p_1}) = (\mathbf{0}, \mathbf{0}), p_1 \neq p_2 \in \mathbb{P}_M\}$
$\mathcal{V}_{\text{bad},3} = \{\tau \in \mathcal{T}_{\text{oc}} :\ y_{p_1} + x_{p_1} = y_{p_2} + x_{p_2} \text{ or } y_{p_1} + x_{p_1} = \mathbf{0}, p_1 \neq p_2 \in \mathbb{P}_M\}$.
$\mathcal{V}_{\text{bad},4} = \{\tau \in \mathcal{T}_{\text{oc}} :\ y_{p_1} + x_{p_1} = y_{p_2} + x_{p_2} \text{ or } y_{p_1} + x_{p_1} = \mathbf{0} \text{ or } \mathbf{1}, p_1 \neq p_2 \in \mathbb{P}_M\}$.

Proposition 2. *For any (q, σ) CPA-distinguisher $\mathcal{A}$ interacting with a uniform random online permutation Π^u,* $\Pr[\mathsf{view}(\mathcal{A}^{\Pi}) \in \mathcal{V}_{\text{bad},1}] \leq \frac{\sigma(\sigma-1)}{2N}$. *For any (q, σ) CCA-distinguisher $\mathcal{A}$ interacting with a uniform random online permutation Π^u and its inverse $(\Pi^u)^{-1}$,* $\Pr[\mathsf{view}(\mathcal{A}^{\Pi^u,(\Pi^u)^{-1}}) \in \mathcal{V}_{\text{bad},i}] \leq \frac{(\sigma+2)(\sigma+3)}{N}$, $i = 2, 3, 4$.

Proposition 3. (main tool of the paper)
Let F be an online cipher. Suppose for some $1 \leq k \leq 4$ and for all $\tau \in \mathcal{T}_{\text{oc}} \backslash \mathcal{V}_{\text{bad},i}$, the interpolation probability of F satisfies the following equation

$$\Pr[F(M_1) = C_1, \cdots, F(M_q) = C_q] \geq \frac{(1-\varepsilon)}{\mathbf{P}(N, \sigma)} \tag{2}$$

where $\tau = ((M_1, C_1), \cdots, (M_q, C_q))$ and σ is the total number of blocks in q plaintexts. Then F is $(q, \sigma, \varepsilon + \frac{(\sigma+2)(\sigma+3)}{2N})$-CCA secure when $i = 2, 3, 4$ or F is $(q, \sigma, \varepsilon + \frac{\sigma(\sigma-1)}{2N})$-CPA secure when $i = 1$.

The above proposition is true for any online ciphers and so it can be applied for any other online ciphers. Depending on the definition of the online cipher, the definition of bad views may have to be changed. The similar and a more general result can be found in [12].

3 Two Known Online Ciphers : HCBC1 and HCBC2

3.1 HCBC1 [1]

Given a permutation $\pi \in \mathsf{Perm}(G)$ and a hash function $h : G \to G$, we define $\mathsf{HCBC1}[\pi, h]$ online permutation. Let $x_i, y_i \in G$, $1 \leq i \leq m$, $y_0 = \mathbf{0}$. Now,

$$\mathsf{HCBC1}[\pi, h](x_1, \cdots, x_m) = (y_1, \cdots, y_m),\ y_i = \pi(h(y_{i-1}) + x_i),\ 1 \leq i \leq m.$$

Note that $\mathsf{HCBC1}[\pi, h]$ is an online permutation. The online property can be proved by induction as the i^{th} output block only depends on $(i-1)^{\text{th}}$ output block and i^{th}input block. It is also a permutation and its inverse is defined as

$$\mathsf{HCBC1}[\pi, h]^{-1}(y_1, \cdots, y_m) = (x_1, \cdots .x_m),\ x_i = \pi^{-1}(y_i) - h(y_{i-1}),\ 1 \leq i \leq m.$$

Let $E^u \xleftarrow{*} \mathsf{Perm}(G)$ be a URP or uniform random permutation, $e \xleftarrow{*} E$ be a blockcipher and $\mathsf{h} \xleftarrow{*} H$ be an ε-Δuniversal hash function from G to G. We define $\mathsf{H1} := \mathsf{HCBC1}[E^u, \mathsf{h}]$, $\mathsf{H1}' := \mathsf{HCBC1}[e, \mathsf{h}]$.

Interpolation probability of H1. We compute q-interpolation probability of H1 for a type-1 qr-tuple $\tau := ((M_1, C_1), \cdots, (M_q, C_q)) \in \mathcal{V}_{\text{good},1}$, i.e., we compute

$\Pr[\mathsf{H1}(M_1) = C_1, \cdots, \mathsf{H1}(M_q) = C_q]$. From the definition of HCBC1 one can verify the following equivalences.

$$\begin{aligned} &\mathsf{H1}(M_1) = C_1, \cdots, \mathsf{H1}(M_q) = C_q \\ \Leftrightarrow\ & E^u(\ \mathsf{h}(\mathrm{last}(q^{\mathrm{chop}(p)})) + \mathrm{last}(p)\ ) = \mathrm{last}(q^p)\ \ \forall p \in \mathbb{P}_M \end{aligned}$$

Thus, during the computation of the interpolation, $(\mathsf{h}(\mathrm{last}(q^{\mathrm{chop}(p)}))+\mathrm{last}(p))_{p\in\mathbb{P}_M}$ correspond to the set of inputs of E^u and $(\mathrm{last}(q^p))_{p\in\mathbb{P}_M}$ corresponds to the set of outputs of E^u. Since $\tau \in \mathcal{V}_{\mathrm{good},1}$, $\mathrm{last}(q^p)$'s are distinct for all $p \in \mathbb{P}_M$. Thus, all outputs of E^u are distinct. Now we prove that for all $p \in \mathbb{P}_M$, $(\mathrm{last}(q^{\mathrm{chop}(p)}), \mathrm{last}(p))$'s are distinct. Suppose for some p_1, p_2, $(\mathrm{last}(q^{p_1'}), \mathrm{last}(p_1)) = (\mathrm{last}(q^{p_2'}), \mathrm{last}(p_2))$ where $p_i' = \mathrm{chop}(p_i)$. Now $\mathrm{last}(q^{p_1'}) = \mathrm{last}(q^{p_2'})$ implies that $p_1' = p_2'$ and hence $p_1 = p_2$ (since $\mathrm{last}(p_1) = \mathrm{last}(p_2)$). By using lemma 1 we have $\Pr[(\mathsf{h}(\mathrm{last}(q^{\mathrm{chop}(p)})) + \mathrm{last}(p))$'s are distinct$] \geq 1 - \varepsilon\binom{\sigma}{2}$. Thus, all inputs of E^u are distinct with probability at least $1 - \varepsilon\binom{\sigma}{2}$. Moreover, the inputs and outputs are independent with E^u since h is independent with E^u. So $\Pr[E^u(\ \mathsf{h}(\mathrm{last}(q^{\mathrm{chop}(p)})) + \mathrm{last}(p)\ ) = \mathrm{last}(q^p)\ \forall p \in \mathbb{P}_M] \geq \frac{1-\varepsilon\binom{\sigma}{2}}{\mathbf{P}(N,\sigma)}$. So we have proved the following theorem. The second part of the theorem is followed from proposition 3.

Theorem 1. (interpolation probability of HCBC1)

$$\Pr[\mathsf{H1}(M_1) = C_1, \cdots, \mathsf{H1}(M_q) = C_q] \geq \frac{1 - \varepsilon\binom{\sigma}{2}}{\mathbf{P}(N,\sigma)} \tag{3}$$

for all $((M_1, C_1), \cdots, (M_q, C_q)) \in \mathcal{V}_{\mathrm{good},1}$ *where* H1 *is based on* ε*-*Δ*universal hash function. Let* $\mathcal{A}$ *be a* (q, σ)*-CPA distinguisher then we have*

$$\mathbf{Adv}^{\mathrm{CPA}}_{\mathcal{A}}(\mathsf{H1}) \leq \binom{\sigma}{2}(\varepsilon + \frac{1}{N}),$$

$$\mathbf{Adv}^{\mathrm{CPA}}_{\mathcal{A}}(\mathsf{H1}') \leq \binom{\sigma}{2}(\varepsilon + \frac{1}{N}) + \mathbf{Insec}^{\mathrm{CPA}}_{E}(\sigma).$$

3.2 HCBC2 [1]

Now we make similar study for HCBC2. We follow same notations as given in HCBC1 except that $h : G^2 \to G$, h is ε-Δuniversal hash from G^2 to G and $x_0 = \mathbf{0}$. $\mathsf{H2} := \mathsf{HCBC2}[E^u, \mathsf{h}]$, $\mathsf{H2}' := \mathsf{HCBC2}[e, \mathsf{h}]$ where

$$\mathsf{HCBC2}[\pi, h](x_1, \cdots, x_m) = (y_1, \cdots, y_m),\ y_i = \pi(h(x_{i-1}, y_{i-1}) + x_i) + h(x_{i-1}, y_{i-1})$$

for $1 \leq i \leq m$. $\mathsf{HCBC2}[\pi, h]^{-1}$ denotes its inverse online cipher.

Interpolation probability of H2. We compute q-interpolation probability of H2 for any given type-2 qr-tuple $\tau := ((M_1, C_1), \cdots, (M_q, C_q)) \in \mathcal{V}_{\mathrm{good},2}$. From the definition of HCBC2 we have the following equivalences.

$$\begin{aligned} &\mathsf{H2}(M_1) = C_1, \cdots, \mathsf{H1}(M_q) = C_q \\ \Leftrightarrow\ & E^u(z_{\mathrm{chop}(p)} + \mathrm{last}(p)\ ) = \mathrm{last}(q^p) + z_{\mathrm{chop}(p)}\ \ \forall p \in \mathbb{P}_M \end{aligned}$$

where $z_p = \mathsf{h}(\text{last}(p), \text{last}(q^p))$. Thus while computing the interpolation, $(z_{\text{chop}(p)} + \text{last}(p))_{p\in\mathbb{P}_M}$ are all inputs of E^u and $(z_{\text{chop}(p)} + \text{last}(q^p))_{p\in\mathbb{P}_M}$ are all outputs of E^u. Since $\tau \in \mathcal{V}_{\text{good},2}$, by using a similar argument, as given for $\mathcal{V}_{\text{good},1}$, we can show that $((\text{last}(\text{chop}(p)), \text{last}(q^{\text{chop}(p)})), \text{last}(p))$'s are distinct and $((\text{last}(\text{chop}(p)), \text{last}(q^{\text{chop}(p)})), \text{last}(q^p))$'s are distinct for all $p \in \mathbb{P}_M$. By using lemma 1 we have

$$\Pr[z_{\text{chop}(p)} + \text{last}(p))\text{'s are distinct}] \geq 1 - \varepsilon\binom{\sigma}{2}$$

$$\Pr[z_{\text{chop}(p)} + \text{last}(q^p))\text{'s are distinct}] \geq 1 - \varepsilon\binom{\sigma}{2}.$$

Thus, all inputs and outputs of E^u are distinct with probability at least $1-2\varepsilon\binom{\sigma}{2}$ and the inputs and outputs are independent with E^u (since h is independent with E^u). So by applying the proposition 1, $\Pr[E^u(z_{\text{chop}(p)} + \text{last}(p)\ ) = \text{last}(q^p) + z_{\text{chop}(p)} \quad \forall p \in \mathbb{P}_M] \geq \frac{1-2\varepsilon\binom{\sigma}{2}}{\mathbf{P}(N,\sigma)}$.

Theorem 2. (interpolation probability of HCBC2)

$$\Pr[\mathsf{H2}(M_1) = C_1, \cdots, \mathsf{H2}(M_q) = C_q] \geq \frac{1 - 2\varepsilon\binom{\sigma}{2}}{\mathbf{P}(N,\sigma)} \tag{4}$$

for all $((M_1, C_1), \cdots, (M_q, C_q)) \in \mathcal{V}_{\text{good},2}$ *and* $\mathsf{H2}$ *is based on* ε*-*Δ*universal hash function. Let* $\mathcal{A}$ *be a* (q, σ)*-CCA distinguisher then we have*

$$\mathbf{Adv}^{\text{CCA}}_{\mathcal{A}}(\mathsf{H2}) \leq \frac{(\sigma+2)(\sigma+3)}{2} \times (2\varepsilon + \frac{1}{N}),$$

$$\mathbf{Adv}^{\text{CCA}}_{\mathcal{A}}(\mathsf{H2}') \leq \frac{(\sigma+2)(\sigma+3)}{2} \times (2\varepsilon + \frac{1}{N}) + \mathbf{Insec}^{\text{CCA}}_{E}(\sigma).$$

4 Two New Online Ciphers : MHCBC and MCBC

4.1 MHCBC

With the same notation as in HCBC1 we define $1 \leq i \leq m$,

$\mathsf{MHCBC}[\pi, h](x_1, \cdots, x_m) = (y_1, \cdots, y_m),\ y_i = \pi(h(x_{i-1} + y_{i-1}) + x_i) + h(x_{i-1} + y_{i-1})$

$\mathsf{MHCBC}[\pi, h]^{-1}(y_1, \cdots, y_m) = (x_1, \cdots, x_m), x_i = \pi^{-1}(y_i - h(x_{i-1} + y_{i-1})) - h(x_{i-1} + y_{i-1}).$

$\mathsf{H3} := \mathsf{MHCBC}[E^u, \mathsf{h}].$

$\mathsf{H3}' := \mathsf{MHCBC}[e, \mathsf{h}].$

Note that MHCBC uses (G, G) universal hash function similar to HCBC1 and still it is CCA-secure. This is true since both plaintext and ciphertext blocks are xored and hence the adversary does not have any control on the XOR. In case of HCBC2, these two blocks are inputs of a (G^2, G) universal hash function.

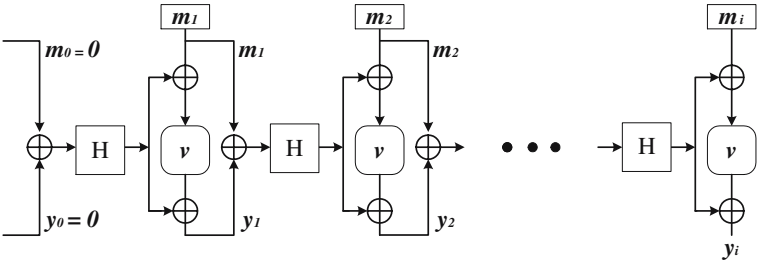


Fig. 2. MHCBC or Modified-Hash CBC online cipher

Now we will compute interpolation probability for type-3 qr tuples $\mathcal{V}_{\text{good},3}$. Recall that a qr-tuple $((M_1, C_1), \cdots, (M_q, C_q)) \in \mathcal{V}_{\text{good},3}$ if and only if $(\text{last}(p) + \text{last}(q^p))$'s are distinct for all $p \in \mathbb{P} \cup \{\lambda\}$. Basic idea of the computation pf interpolation is similar for online cipher constructions considered here. We first provide an equivalent representation of interpolation as interpolation of URP and hence it is sufficient to calculate interpolation probability of URP for a specific tuple. Then we calculate the probability that all inputs and outputs of URP. Finally by using proposition 1 we have interpolation probability.

Theorem 3. (interpolation probability of MHCBC)
For all $((M_1, C_1), \cdots, (M_q, C_q)) \in \mathcal{V}_{\text{good},3}$,

$$\Pr[\mathsf{H3}(M_1) = C_1, \cdots, \mathsf{H3}(M_q) = C_q] \geq \frac{1 - \varepsilon\binom{2\sigma}{2}}{\mathbf{P}(N, \sigma)} \tag{5}$$

Proof. From the definition of HCBC3 we have the following equivalences.

$$\begin{aligned} &\mathsf{H3}(M_1) = C_1, \cdots, \mathsf{H3}(M_q) = C_q \\ \Leftrightarrow\ &E^u(z_{\text{chop}(p)} + \text{last}(p)\ ) = \text{last}(q^p) + z_{\text{chop}(p)} \quad \forall p \in \mathbb{P}_M \end{aligned}$$

where $z_p = \mathsf{h}(\text{last}(p) + \text{last}(q^p))$. Thus while computing the interpolation, $(z_{\text{chop}(p)} + \text{last}(p))_{p \in \mathbb{P}_M}$ are all inputs of E^u and $(z_{\text{chop}(p)} + \text{last}(q^p))_{p \in \mathbb{P}_M}$ are all outputs of E^u. Moreover, $((\text{last}(\text{chop}(p)) + \text{last}(q^{\text{chop}(p)})), \text{last}(p))$'s and $((\text{last}(\text{chop}(p)) + \text{last}(q^{\text{chop}(p)})), \text{last}(q^p))$'s are distinct for all $p \in \mathbb{P}_M$ (since $\tau \in \mathcal{V}_{\text{good},3}$). By using lemma 1 we have

$$\Pr[z_{\text{chop}(p)} + \text{last}(p))\text{'s are distinct}] \geq 1 - \varepsilon\binom{\sigma}{2}$$

$$\Pr[z_{\text{chop}(p)} + \text{last}(q^p))\text{'s are distinct}] \geq 1 - \varepsilon\binom{\sigma}{2}.$$

Thus, all inputs and outputs of E^u are distinct with probability at least $1 - 2\varepsilon\binom{\sigma}{2}$ and these are independent with E^u (since h is independent with E^u). So by applying the proposition 1, $\Pr[E^u(z_{\text{chop}(p)} + \text{last}(p)\ ) = \text{last}(q^p) + z_{\text{chop}(p)} \quad \forall p \in \mathbb{P}_M] \geq \frac{1 - 2\varepsilon\binom{\sigma}{2}}{\mathbf{P}(N,\sigma)}$. □

Corollary 1. *Let $\mathcal{A}$ be a (q,σ)-CCA distinguisher then we have*

$$\mathbf{Adv}^{\mathrm{CCA}}_{\mathcal{A}}(\mathsf{H3}) \le \frac{(\sigma+2)(\sigma+3)}{2} \times (2\varepsilon + \frac{1}{N}),$$

$$\mathbf{Adv}^{\mathrm{CCA}}_{\mathcal{A}}(\mathsf{H3}') \le \binom{\sigma}{2}(2\varepsilon + \frac{1}{N}) + \mathbf{Insec}^{\mathrm{CCA}}_{E}(\sigma)$$

where $\mathsf{H3}$ *is based on ε-Δuniversal hash function. If we consider the finite field multiplication based universal hash function then* $\mathbf{Adv}^{\mathrm{CCA}}_{\mathcal{A}}(\mathsf{H3}') \le \frac{3(\sigma+2)(\sigma+3)}{2N} + \mathbf{Insec}^{\mathrm{CCA}}_{E}(\sigma)$.

4.2 MCBC or Modified-CBC

A Chosen Ciphertext Attack for a Variant of MHCBC

A simple replacement of H of MHCBC by v (note that, from example 2 we know that v is universal hash function) would not make CCA-secure. So define $\mathsf{H'3}(x_1,\cdots,x_m) = (y_1,\cdots,y_m)$ where $y_i = \pi(\pi(x_{i-1}+y_{i-1})+x_i) + \pi(x_{i-1}+y_{i-1})$, $1 \le i \le m$. It is easy to see that $\mathsf{H'3}^{-1}(\mathbf{0}) = v(\mathbf{0}) = v_0$ (known to us) and hence $\mathsf{H'3}(\mathbf{0}) = v_0 \oplus v(v_0)$. So $v(v_0)$ is also known to us and call it v_1. Now, $\mathsf{H'3}(v_0,v_1) = (0, v_1 \oplus v_0)$ always true where this is true with probability close to $1/N$ for the ideal online cipher. Thus we can have CCA-attack by making only three queries with four blocks.

CCA-security analysis of MCBC

Let $\pi \in \mathsf{Perm}(G)$ then we define MCBC or modified CBC online permutation as follows :

$$\mathsf{MCBC}[\pi](x_1,\cdots,x_m) = (y_1,\cdots,y_m),\ y_i = \pi(\pi(x_{i-1}+y_{i-1})+x_i) + K + x_i$$

$1 \le i \le m$ and $K = \pi(\mathbf{1})$. We write $\mathsf{MC} := \mathsf{MCBC}[E^u]$ and $\mathsf{MC'} := \mathsf{MCBC}[e]$ for a chosen blockcipher $e \xleftarrow{*} E$.

MCBC does not use any universal hash function since the underlying blockcipher is used in the place of the universal of hash function. Thus we are able

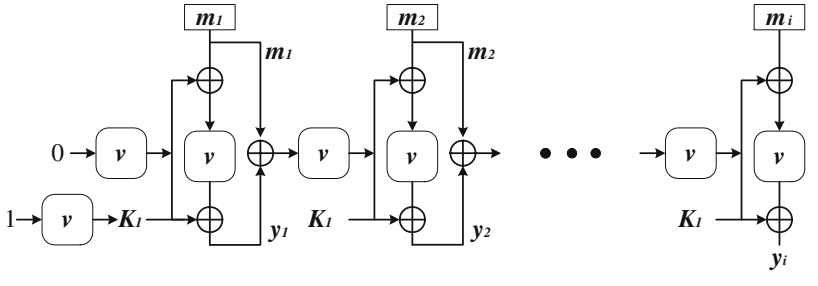


Fig. 3. MCBC or Modified-CBC online cipher

to remove extra key storage as well as an extra design of a universal hash function. The proof idea for MCBC is similar to MHCBC except the fact that we have to consider all inputs and outputs of the underlying blockciphers. We have to be a little bit careful while computing interpolation probability. We first see all inputs and outputs of the uniform random permutation E^u during the computations of interpolation probability of $\mathsf{MC}(M_1) = C_1, \cdots, \mathsf{MC}(M_q) = C_q$ where $((M_1, C_1), \cdots (M_q, C_q)) \in \mathcal{V}_{\text{good},4}$. Let $\mathbb{P} := \mathbb{P}(M_1, \cdots, M_q)$. Now one can check that

1. $K := E^u(\mathbf{1}),\ z_{\text{chop}(p)} := E^u(w_{\text{chop}(p)})$ and $(\text{last}(q^p) - K - z_{\text{chop}(p)})$ are all outputs of $E^u,\ p \in \mathbb{P}_M$ where $w_p := \text{last}(p) + \text{last}(q^p)$.
2. $\mathbf{0}, \mathbf{1},\ w_p,\ z_{\text{chop}(p)} + \text{last}(p),\ p \in \mathbb{P}_M$ are all inputs of E^u.

Since $((M_1, C_1), \cdots (M_q, C_q)) \in \mathcal{V}_{\text{good},4}$, for all $p \in \mathbb{P} \cup \{\lambda\}$, $(\text{last}(p) + \text{last}(q^p))$'s are distinct and different from $\mathbf{1}$. All inputs and outputs are completely determined once z_p and K is defined. Let A be the number of possible values of z_p and K such that

$$\mathbf{1},\ w_p,\ z_{\text{chop}(p)} + \text{last}(p),\ p \in \mathbb{P}_M \text{ are distinct and}$$

$$K,\ z_{\text{chop}(p)},\ \text{last}(q^p) - K - z_{\text{chop}(p)}\ p \in \mathbb{P}_M \text{ are distinct .}$$

We estimate A by counting the complement. The above conditions are not true due to following possibilities. Recall that $\sigma' = |\mathbb{P}'|$.

1. $w_p = \mathbf{1}$ or $w_{p_1} = w_{p_2}$ for some $p_1 \neq p_2 \in \mathbb{P}'$ and $p \in \mathbb{P}$. This is not possible since $((M_1, C_1), \cdots (M_q, C_q)) \in \mathcal{V}_{\text{good},4}$.
2. $z_{\text{chop}(p_1)} + \text{last}(p_1) = w_{p_2}$ or $z_{\text{chop}(p_1)} + \text{last}(p_1) = \mathbf{1}$ for some $p_1 \neq p_2$, $p_1 \in \mathbb{P}, p_2 \in \mathbb{P}'$. There are at most $N^{\sigma'} \times \sigma \times (\sigma' + 1) (\leq N^{\sigma'} \times \sigma^2)$ solutions.
3. Similarly, $z_{\text{chop}(p_1)} + \text{last}(p_1) = z_{\text{chop}(p_2)} + \text{last}(p_2)$ for some $p_1 \neq p_2 \in \mathbb{P}$. There are at most $N^{\sigma'} \times \sigma^2$ solutions.
4. $z_{\text{chop}(p)}$ and K are not distinct. There are at most $N^{\sigma'} \times \sigma^2$ solutions.
5. K or $z_{\text{chop}(p_1)}$ is same as $\text{last}(q^{p_2}) - K - z_{\text{chop}(p_2)}$ for some $p_1 \neq p_2 \in \mathbb{P}$. There are $N^{\sigma'} \times \sigma^2$ solutions.
6. $(\text{last}(q^p) - K - z_{\text{chop}(p)})$'s are not distinct. There are $N^{\sigma'} \times \sigma^2$ solutions.

So there are $5N^{\sigma'}\sigma^2$ cases where the above is not true. Thus the number of possible solutions is at least $N^{\sigma'+1} - 5N^{\sigma'}\sigma^2$ and hence $A \geq N^{\sigma'+1}(1 - \frac{5\sigma^2}{N})$. For each such solution of $z_{\text{chop}(p)}$ and K such that the above is true we have

$$\Pr[E^u(\mathbf{1}) = K, E^u(w_p) = z_p, E^u(z_{\text{chop}(p)} + \text{last}(p)) = \text{last}(q^p) - K - w_{\text{chop}(p)}, \forall p \in \mathbb{P}_M]$$

$$= \frac{1}{\mathbf{P}(N, \sigma + \sigma' + 1)}.$$

Summing over A solutions we have

$$\begin{aligned}\Pr[\mathsf{MC}(M_1) = C_1, \cdots, \mathsf{MC}(M_q) = C_q] &\geq \frac{A}{\mathbf{P}(N, \sigma + \sigma' + 1)} \\ &\geq \frac{N^{\sigma'+1}(1 - \frac{5\sigma^2}{N})}{\mathbf{P}(N, \sigma + \sigma' + 1)} \\ &\geq \frac{(1 - \frac{5\sigma^2}{N})}{\mathbf{P}(N, \sigma)}.\end{aligned}$$

Thus we have the following main theorem for MCBC.

Theorem 4. (interpolation probability of MCBC)
For all $((M_1, C_1), \cdots, (M_q, C_q)) \in \mathcal{V}_{\text{good},4}$

$$\Pr[\mathsf{MC}(M_1) = C_1, \cdots, \mathsf{MC}(M_q) = C_q] \geq \frac{1 - 5\sigma^2/N}{\mathbf{P}(N, \sigma)} \tag{6}$$

Corollary 2. *Let* $\mathcal{A}$ *be a* (q, σ)*-CCA distinguisher with* $\sigma \geq 6$ *then we have*

$$\mathbf{Adv}^{\text{CCA}}_{\mathcal{A}}(\mathsf{MC}) \leq \frac{6\sigma^2}{N},$$

$$\mathbf{Adv}^{\text{CCA}}_{\mathcal{A}}(\mathsf{MC}') \leq \frac{6\sigma^2}{N} + \mathbf{Insec}^{\text{CCA}}_{E}(2\sigma).$$

5 Conclusion

In this paper we analyze known online ciphers namely HCBC1 and HCBC2 and propose two new online ciphers MHCBC and MCBC which have several advantages over the previous ones. In particular, MHCBC is more efficient than HCBC2 and still has CCA-security. MCBC online cipher does not need any universal hash function and hence it has better performance as well as smaller key size. Our security analysis is somewhat different from the usual game based security analysis. We believe that our proof technique would be useful in many areas where indistinguishability is concerned. One of the research goal we can think is to provide online ciphers for incomplete plaintext blocks. One can analyze the hardware performance of all these online ciphers.

Acknowledgement. We would like to acknowledge Professor Palash Sarkar who had inspired us to write this paper. We also would like to thank anonymous reviewers whose comments helped us to modify our earlier draft.

References

1. Bellare, M., Boldyreva, A., Knudsen, L., Namprempre, C.: On-Line Ciphers and the Hash-CBC constructions. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 292–309. Springer, Heidelberg (2001)

2. Bellare, M., Boldyreva, A., Knudsen, L., Namprempre, C.: On-Line Ciphers and the Hash-CBC Constructions. Cryptology eprint archive, http://eprint.iacr.org/2007/197
3. Bellare, M., Killan, J., Rogaway, P.: The security of the cipher block chanining Message Authentication Code. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 341–358. Springer, Heidelberg (1994)
4. Black, J., Rogaway, P.: CBC MACs for arbitrary length messages. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 197–215. Springer, Heidelberg (2000)
5. Daemen, J., Rijmen, V.: Resistance Against Implementation Attacks. A Comparative Study of the AES Proposals. In: Proceedings of the Second AES Candidate Conference (AES2), Rome, Italy (March 1999), http://csrc.nist.gov/encryption/aes/aes_home.htm
6. Knudsen, L.: Block chaining modes of operation. In: Symmetric Key Block Cipher Modes of Operation Workshop (October 2000), http://csrc.nist.gov/encryption/modes/workshop1/
7. Krawczyk, H.: LFSR-based hashing and authenticating. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 129–139. Springer, Heidelberg (1994)
8. Luby, M., Rackoff, C.: How to construct pseudo-random permutations from pseudo-random functions. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, p. 447. Springer, Heidelberg (1986)
9. Nandi, M.: Two New Efficient CCA-Secure Online Ciphers: MHCBC and MCBC. eprint archive, http://eprint.iacr.org/2008/401
10. Nevelsteen, W., Preneel, B.: Software performance of universal hash functions. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 24–41. Springer, Heidelberg (1999)
11. Stinson, D.R.: On the connections between universal hashing, combinatorial designs and error-correcting codes. Congressus Numerantium 114, 7–27 (1996)
12. Vaudenay, S.: Decorrelation: A Theory for Block Cipher Security. Journal of Cryptology 16(4), 249–286 (2003)

Chai-Tea, Cryptographic Hardware Implementations of xTEA

Jens-Peter Kaps

Volgenau School of IT&E, George Mason University, Fairfax, VA, USA
jkaps@gmu.edu

Abstract. The tiny encryption algorithm (TEA) was developed by Wheeler and Needham as a simple computer program for encryption. This paper is the first design-space exploration for hardware implementations of the extended tiny encryption algorithm. It presents efficient implementations of XTEA on FPGAs and ASICs for ultra-low power applications such as RFID tags and wireless sensor nodes as well as fully pipelined designs for high speed applications. A novel ultra-low power implementation is introduced which consumes less area and energy than a comparable AES implementation. Furthermore, XTEA is compared with stream ciphers from the eSTREAM portfolio and lightweight ciphers. The high speed implementations of XTEA operate at 20.6 Gbps (FPGA) or 36.6 Gbps (ASIC).

Keywords: Efficient implementation, symmetric key algorithms, TEA, XTEA, FPGA, ASIC.

1 Introduction

The Tiny Encryption Algorithm (TEA) was introduced by David Wheeler and Roger Needham [1,2] in '94. Their main design goal was to produce a cipher that is simple, short and does not rely on large tables or pre-computations. Shortly after TEA was published, a few minor weaknesses were found [3]. The original authors eliminated those weaknesses in a new version of TEA called XTEA for extended TEA [4]. A positive side effect of the new version, also called *tean*, is that the main routine requires two fewer addition operations which results in a faster algorithm. Also in [4] the authors described a variant of TEA called *Block TEA* to cater for larger block sizes than the original algorithm's 64-bit. An attack on Block TEA was found and subsequently corrected in [5]. Other recent attacks [6,7] target a reduced round version of TEA. The recommended 32 round version is still considered to be secure.

TEA uses only simple addition, XOR and shifts, and has a very small code size. This makes TEA an ideal candidate to provide data security services for wireless sensor network (WSN) nodes which have limited memory and computational power. Many software implementations of TEA for these nodes have been reported [8,9,10]. Kanamori compared software implementations of the Advanced Encryption Standard (AES) to implementations of TEA on sensor nodes [11] and

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 363–375, 2008.

concluded that the memory requirements for TEA are one quarter the requirements of AES.

Even though TEA was proposed mainly for software implementations, its simple design makes it also very suitable for hardware implementations. The implementation designers can take advantage of the inherent parallelism of TEA to boost performance or, on the other hand, reduce its area and power consumption. This makes it possible to implement TEA for severe power constraint applications such as passive radio frequency identification (RFID) tags and the next generation WSN nodes. Customized hardware is currently the only option for inexpensive passive RFID tags as they do not have a processor. The same might become true for next generation sensor nodes which are predicted to be powered by energy scavenged from the environment [12].

For environments in which the power consumption is not the most important design criteria, field programable gate arrays (FPGAs) are an interesting option. FPGAs can be re-programmed in the field which makes it possible to update the cryptographic algorithm after deployment. This might be necessary if new cryptanalytic results lead to changes in the TEA algorithm as we have seen in the past.

Hardware implementations of the original TEA were first published in [13,14] targetting RFID tags. To the knowledge of the authors, this is the first design space exploration of hardware implementations of XTEA. This paper covers implementations for application specific integrated circuits (ASIC) as well as for FPGAs, small, power and energy efficient implementations for ubiquitous computing devices with stringent power constraints and high speed implementations for server environments.

2 Extended Tiny Encryption Algorithm

The Extended Tiny Encryption Algorithm (XTEA) is a block cipher that uses a cryptographic key of 128 bits to encrypt or decrypt data in blocks of 64 bits. Each input block is split into two halves y and z which are then applied to a routine similar to a Feistel network for N rounds where N is typically 32. Most Feistel networks apply the result of a mixing function to one half of the data using XOR as a reversible function. XTEA uses for the same purpose integer addition during encryption and subtraction during decryption.

Figure 1 shows the C source code for XTEA as it was introduced in [4]. Additional parenthesis were added to clarify the precedence of the operators. The main variables y, z, and *sum*, which assists with the subkey generation, have a length of 32 bits. All additions and subtractions within XTEA are modulo 2^{32}. Logical left shifts of z by 4 bits are denoted as $z \ll 4$ and logical right shift by 5 bits as $z \gg 5$. The bitwise XOR function is denoted as "^" in the source code and $\oplus$ in this paper.

The first part of the algorithm is the encryption routine and the second part is the decryption routine. The *while*-loop constitutes the round function. The formulae that compute the new values for y and z can be split into a permutation function $f(z) = (z \ll 4 \oplus z \gg 5) + z$ and a subkey generation function

```
int tean(long * v, long * k, int N)
{
  unsigned long y=v[0], z=v[1], DELTA=0x9e3779b9;
  if (N>0) {    /* encryption */
    unsigned long limit=DELTA*N, sum=0;
    while (sum != limit) {
      y += ((z<<4 ^ z>>5) + z) ^ (sum + k[sum&3]);
      sum += DELTA;
      z += ((y<<4 ^ y>>5) + y) ^ (sum + k[sum>>11 &3]);
    }
  } else {    /* decryption */
    unsigned long sum=DELTA*(-N);
    while (sum) {
      z -= ((y<<4 ^ y>>5) + y) ^ (sum + k[sum>>11 &3]);
      sum -= DELTA;
      y -= ((z<<4 ^ z>>5) + z) ^ (sum + k[sum&3]);
    }
  }
  v[0]=y, v[1]=z;
  return;}
```

Fig. 1. Source code of XTEA

$sum + k(sum)$. The function $k(sum)$ selects one block out of the four 32-bit blocks that comprise the key, depending on either bits 1 and 0 or bits 12 and 11 of sum. The results of the permutation function and the subkey generation function are XORed and then applied to y and z respectively, by addition in the case of encryption or subtraction in the case of decryption.

This leads to the simplified block diagram shown in Fig. 2. For encryption, z is applied to the left side, y to the right side, and all adder/subtracters are

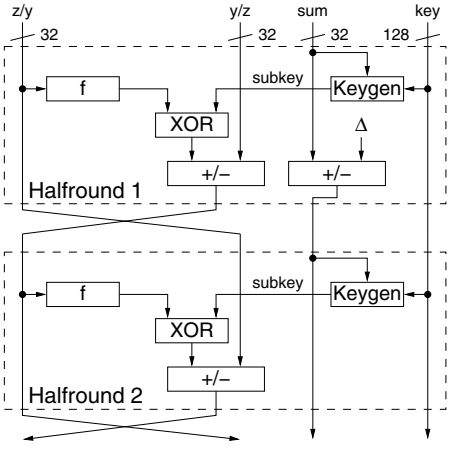


Fig. 2. Top level block diagram of XTEA

in addition mode. For decryption, the opposite is applied. The permutation function is shown as f and the subkey generation as Keygen. One round of xtea computes a new value for y and z. Therefore, we can view the computation of one value as a *halfround*. A new value for *sum* is computed between the first and the second halfround. It is incremented by a constant Δ during encryption and decremented during decryption. We included this computation into the first halfround as it can be performed concurrently with the final addition/subtraction of the data in this halfround.

3 Speed XTEA

Speed XTEA investigates how XTEA scales for fast hardware implementations. We implemented a fully loop-unrolled, pipelined architecture of XTEA on FPGA and ASIC. Each block of data can be associated with a new key and the mode can be switched between encryption and decryption without loss of throughput. A natural location for the pipeline cuts is between the half rounds, so called outer round pipelining. However, due to the long delay of one halfround additional pipeline cuts within the halfround, called inner round pipelining, are necessary. Figure 3 indicates through thick dashed lines the location of three inner round cuts that we want to investigate.

3.1 FPGA Implementation

FPGAs are composed of configurable logic blocks (CLB) and a programmable interconnection network. We targeted the low cost Xilinx FPGA series Spartan 3 to compare our results with other reported block cipher implementations on the same series. In addition, we implemented XTEA on the Xilinx Virtex 5 devices which are a natural choice for high speed applications. Xilinx divides each CLB into slices. Each slice on a Spartan-3 contains two sets of a 4-input look-up table (LUT) followed by a flip-flop. The Virtex 5 slice contains four sets of a 6-input

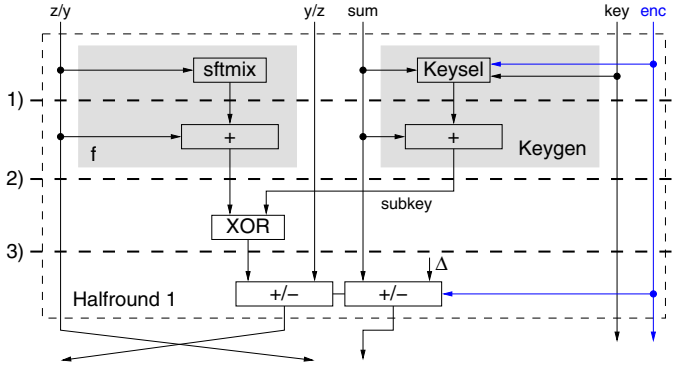


Fig. 3. Pipeline cuts for half rounds

LUT followed by a flip-flop. Slices in both families have dedicated circuitry and connections for fast carry propagation. Therefore, the result of additions can be stored within the same slices avoiding any additional wire delay. This makes this location ideal for a pipeline cut which is indicated by line number 2 in Fig. 3. Furthermore the LUT contained in these slices can accommodate the function sftmix and XOR.

Unfortunately the ideal clock speed of 3.18 ns on Spartan 3 (speed grade -5) and 1.00 ns on Virtex 5 (speed grade -3) cannot be reached due to wire delays at the input of the adder. Even though pipeline location 2 seems to be sufficient at first glance, the key select function Keysel $k(sum)$ can not be implemented with a single LUT per bit. Each output bit depends on 9 input bits: The encryption / decryption flag, four bits from sum, and four bits from the key. In addition, we noticed through timing analysis that the routing delay from the input of the half round to the inner round pipeline cut number 2 was much larger than the delay from the inner cut to the following outer round pipeline cut. Hence, we placed an additional pipeline cut, indicated as line number 1 in Fig. 3, before the first stage of adders.

Pipeline cut three is mainly of interest for ASIC implementation. The results for FPGA show that it has no effect on the critical path delay because the XOR can be performed in the LUTs of the slices occupied by the following adder.

We implemented variations of these possible cuts in a not loop-unrolled version of XTEA to analyze their effect on the delay and to estimate the throughput-area ratio of a fully unrolled implementation. Table 1 shows the results including an approximation of the ratio of delay due to logic cells "Logic Delay" versus routing delay. The throughput and the throughput area ratio are estimated for a fully unrolled design with the same delay and 32 times the area.

Placing pipeline cuts in either location 2 or locations 1 & 2 yields the smallest critical path delay and the best estimated throughput area ratio. We implemented two loop-unrolled pipelined version of XTEA called SpeedXTEA-1 with pipeline cut 2 and SpeedXTEA-2 with pipeline cuts 1 and 2. It is interesting to note that the 6-input LUTs of the Virtex 5 have no significant advantage over the 4-input LUTs on Spartan 3 in terms of critical path delay. The large speed improvement of Virtex 5 is mainly due to its newer CMOS technology.

Table 1. Results for pipeline cuts in Halfround 1 & 2 Implementations

FPGA	xc3s2000-5			xc5vlx100-3			
Pipeline Cuts	1	2	1&2	1	2	1&2	1&2&3
Critical Path Delay (ns)	8.200	7.372	5.471	3.097	2.999	2.193	2.199
Logic Delay	67%	53%	71%	62%	51%	64%	65%
Routing Delay (ns)	2.71	3.46	1.59	1.18	1.47	0.79	0.77
Clock Cycles for one round	4	4	6	4	4	6	8
Area including overhead (Slices)	597	577	701	410	383	445	522
Estimated Troughput (Mbps)	7,805	8,681	11,698	20,665	31,340	29,184	29,104
Estimated Ratio (Mbps/Slice)	0.409	0.470	0.521	1.575	1.741	2.049	1.742

3.2 ASIC Implementation

Our ASIC implementations of XTEA are based on the SpeedXTEA-1 and SpeedXTEA-2 FPGA designs and use the same pipeline cuts. SpeedXTEA-3 makes use of the pipeline cuts 1, 2, and 3. This takes the XOR as well as sftmix and keysel out of the critical path which is now determined solely by the delay of the adders. We used Brent-Kung [15] adders which are more than three times faster than ripple-carry adders (RCA) even though they are only 32-bit wide. All pipeline cuts are implemented by using positive edge triggered flip-flops.

4 Tiny XTEA

The objective of Tiny XTEA was to develop an ultra-low power implementation of XTEA. We assume that the 128-bit key and 64 bits of data are stored in memory and can be accessed via an 8-bit data bus. This bus width selection and the fact that we store a copy of the key and one data block in registers enables us to compare our design with an ultra-low power AES design reported in [16]. Our implementation acts as a co-processor and has commands for loading a key from memory and encrypting or decrypting one block of data.

4.1 ASIC Implementation

Power consumption in CMOS devices is the sum of the leakage power P_{Leak} (also called static power) and dynamic power P_{Dyn}. P_{Leak} is caused by the leakage current of each gate and therefore proportional to the circuit size. P_{Dyn} is caused by gate output changes from '0' to '1' and vice versa and hence proportional to the switching activity and to the clock frequency of the circuit. Ultra low-power applications such as RFID tags and energy scavenger powered sensor nodes [12] operate at frequencies of 100 kHz to 500 kHz where the leakage power is dominant. Therefore, its reduction must be the primary design goal. This can be achieved by reducing the circuit size.

From Fig. 2 we can see that it is sufficient to implement halfround 1 as it can perform the same function as halfround 2. We chose to use a 32-bit datapath as this is the native width of all operations used in XTEA. The main area consuming parts, as far as ASIC design is concerned, are the registers required to store the data (x and y), the key, and the variable *sum*. Using a smaller datapath, for example 8 bits, would reduce the size of the adders and XORs by one fourth. However, it would require the use of additional multiplexers to select between four bytes of the the 32-bit words[1] and to select 17 bits from z in order to facilitate the $f(z) = (z \ll 4 \oplus z \gg 5) + z$ function. It would also increase the number of clock cycles needed for one encryption, and require a more complex control logic. The number of registers will not be affected. Hence, a smaller datapath will not necessarily lead to smaller circuit with a reduced power or energy consumption.

[1] The current design employs only a single such multiplexer.

Table 2. Results for Halfround 1 Implementations

Number of Adders in Architecture	1	2	3	4	7[a]
Critical Path Delay (ns)	9.91	9.84	10.08	9.97	11.70
Clock Cycles for one operation	224	128	96	64	32
Area (NAND Equiv.)	1220	1330	1170	1351	2521
Dynamic Power (at 100 kHz) (μW)	0.49	0.54	0.67	0.86	1.54
Static Power (μW)	6.34	7.21	6.57	7.56	13.84
Total Power (at 100 kHz) (μW)	6.83	7.75	7.25	8.42	15.38
Energy per bit (pJ)	239.0	154.9	108.8	84.24	76.89

[a] Estimated, includes halfround1 and halfround 2.

The halfround function performs four integer additions if it is used as halfround 1 and three integer additions as halfround 2. We can use the same adder for additions and subtractions by including a small circuit to compute the 2's-complement of the subtrahend. The design of the halfround function can be scaled to use only one adder or up to four adders. In order to determine the ideal solution for an ultra-low power implementation on ASIC, we implemented all four versions. From those results, we estimated the results of an implementation that combines halfround 1 and halfround 2 and therefore incorporates seven adders. We used simple ripple carry adders (RCA) which consume less power than faster adders of this width and at the targeted clock frequency [16].

Table 2 shows that the critical path delay of all implementations is almost equivalent. The main contributors to the delay are the RCAs. The surprising result is, that the area of the circuit does not grow linearly with the number of adders. In fact, the halfround 1 architecture with three adders is smaller than the one with two adders. The architectures *adder1* and *adder2* need a 32-bit wide register to store temporary results which is not necessary in the other architectures. Another reason for the nonlinearity is that with each additional adder, the need for multiplexers to switch their inputs is reduced. The static power consumption also has nonlinearities. This is due to the fact that not all gates have the same leakage power. RCAs have a very high switching activity due to glitches which is emphasized when multiple adders are in series as in the *adder3* and *adder4* architectures. This leads to a higher dynamic power consumption. Another factor of the dynamic power consumption is the utilization of the adders in each state of the computation, e.g. the state machine of the *adder3* architecture utilizes only one adder in every third state and three in all other states. This reduces the average dynamic power consumption.

We implemented TinyXTEA-1 using the *adder1* architecture which has the lowest power consumption and TinyXTEA-3 using *adder3* which is a good compromise between power consumption and speed.

4.2 FPGA Implementation

The FPGA implementation is based on the design of TinyXTEA-1 and TinyXTEA-3. The main modification was the replacement of ripple carry adders with fast carry propagate adders offered by the Xilinx FPGAs.

5 Results

All our designs were described in VHDL. The FPGA designs were implemented using Xilinx ISE 9.1 tools and verified through post-place and route simulation with ModelSim SE 6.3a and test vectors generated from the C-code given in [4]. All results reported in the FPGA section are from post-place and route analysis. The ASIC implementations were synthesized using Synopsys power_compiler and a 0.13 μm, $V_{DD} = 1.2V$ ASIC library from TSMC which is characterized for power. All results were reported at the gate level. Results for power, energy and maximum delay from implementations using different CMOS processes, e.g.: 0.35 μm, 0.18 μm, can not be compared to ours as these results are depending on the technology used.

5.1 ASIC Implementations

The results for the high speed ASIC implementation of XTEA are shown in Table 3. It is interesting to note that SpeedXTEA-1 with one pipeline cut per halfround has the same maximum path delay as SpeedXTEA-2 with two cuts. However, the results are from gate level implementation, before placing and routing. Depending on the routing paths we will see a difference but it might not be as significant as for the FPGA implementations. The result of SpeedXTEA-3 shows that our assumption that XOR is in the critical path is correct. It yields the fastest speed of 36.6 Mbps. However, its throughput area ratio is less favorable than SpeedXTEA-1. Unfortunately, the comparison of SpeedXTEA with high speed AES implementations is rather difficult. Satoh's [17] AES implementation supports 128, 192, and 256-bit keys and operates in Galois Counter Mode. Hodjat's AES implementation in [18] has a throughput of 77 Gbps, however in later publications [19,20] no detailed results were shown. The numbers for those implementations in Table 3 were estimated from the published graphs.

Table 4 compares our tiny XTEA implementations with an ultra-low power AES implementation reported by Kaps in [16], the landmark AES implementations

Table 3. Results for Speed XTEA compared to fast AES implementations (ASIC)

Design	Maximum Delay (ns)	Clock Cycles	Clock Cycles Latency	Block Size (bits)	Area (Gate Equivalents (GE))	Throughput (Mbps)	Throughput/ Area (kbps/GE)
SpeedXTEA-1	2.87	1	128	64	307,190	22,300	72.6
SpeedXTEA-2	2.87	1	192	64	420,562	22,300	53.0
SpeedXTEA-3	1.75	1	256	64	529,987	36,571	69.0
AES (Satoh) [17]	3.00	1	11	128	297,542	42,667	143.4
AES (Hodjat) [18]	1.65	1	41	128	473,000	77,576	164.0
AES (composite) [19]	2.00	1	41	128	175,000	64,000	365.7
AES (LUT) [19]	1.91	1	21	128	275,000	67,000	143.4

Table 4. Results for Tiny XTEA compared to block ciphers and the eSTREAM ciphers (ASIC)

Design	Maximum Delay (ns)	Clock Cycles	Block Size (bits)	Key Size (bits)	Area (Gate Equivalents (GE))	Throughput at 100KHz (Kbps)	Throughput/ Area (Kbps/GE)	Power (μW)	Energy per bit (pJ)
TinyXTEA-1	11.28	240	64	128	3500	26.7	0.008	18.8	703
TinyXTEA-3	11.66	112	64	128	3490	57.1	0.016	19.5	341
AES 8-bit [16]	2.19	534	128	128	4070	24.0	0.006	23.8	994
AES 8-bit [21] [a]	–	1016	128	128	3595	12.6	0.004	26.9	2135
AES 8-bit [22] [a]	12.50	1032	128	128	3400	12.4	0.004	4.5	363
DESXL [24,25] [b]	–	144	64	184	2168	44.4	0.021	1.6	36
Camelia [26] [a]	27.67	21	128	128	11350	609.5	0.054	–	–
Camelia [27]	8.93	44	128	128	6511	290.1	0.045	–	–
Present-80 [28] [b]	–	32	64	80	1570	200.0	0.127	5.0	25
Present-128 [28]	–	32	64	128	1886 [c]	200.0	0.106	–	–
F-FCSR-H v2 [29]	2.55	1	8	80	4760	800	0.168	10.6	13
F-FCSR-16 [29]	3.15	1	16	128	8072	1,600	0.198	18.3	11
Grain v1 [29]	1.38	1	1	80	1294	100	0.077	3.3	33
Grain 128 [29]	1.08	1	1	128	1857	100	0.054	4.3	43
MICKEY v2 [29]	1.83	1	1	80	3188	100	0.031	7.1	71
MICKEY 128 [29]	2.42	1	1	128	5039	100	0.020	11.2	112
Trivium [29]	3.05	1	1	80	2580	100	0.039	5.5	55
Trivium (x64) [29]	2.87	1	64	80	4921	6,400	1.301	14.3	2

[a] Results are from 0.35 μm CMOS process.
[b] Results are from 0.18 μm CMOS process.
[c] Estimate, was not implemented in [28].

by Feldhofer [21,22] which both use 0.35 μm technology, as well as several lightweight crypto algorithm implementations. Just recently the eSTREAM portfolio [23] was published recommending four different stream ciphers for hardware implementations. Common to these stream ciphers is that they use an 80-bit key. Tables 5 and 4 list these ciphers including derivations of some of them that allow for a 128-bit key.

The power consumption of TinyXTEA-1 is only marginally smaller than the one of TinyXTEA-3. This confirms our results from the implementations of only one halfround (see Table 2). TinyXTEA and the AES implementations have a similar area consumption. Hence, the power consumption of both implementations should also be similar if the same CMOS technology were used. The AES implementation in [22] consumes only a fifth of the power of the implementation reported in [21], even though both use the same CMOS technology. This is due to low level optimizations and voltage scaling. These techniques could also be employed for TinyXTEA. The surprising result is that the AES from [21] uses nine times more clock cycles than TinyXTEA-3 to encrypt twice as much data.

Table 5. Results for Tiny XTEA compared to 8-bit AES and the eSTREAM Portfolio ciphers (FPGA)

Design	Maximum Delay (ns)	Clock Cycles	Block Size (bits)	Key Size (bits)	Area (slices)	Throughput (Mbps)	Throughput/ Area (Mbps/slice)	Device
TinyXTEA-1	13.87	240	64	128	266	19	0.07	xc3s50-5
TinyXTEA-3	15.97	112	64	128	254	36	0.14	xc3s50-5
AES 8-bit [30]	14.93	3900	128	128	264	2	0.01	xc2s15-6
AES [31]	16.67	112	128	128	522	69	0.13	xc2s30-6
F-FCSR-H v2 [32]	7.25	1	8	80	342	1,104	3.23	xc3s50-5
F-FCSR-16 [32]	7.46	1	16	128	473	2,144	4.53	xc3s50-5
Grain v1 [32]	5.10	1	1	80	44	196	4.45	xc3s50-5
Grain 128 [32]	5.10	1	1	128	50	196	3.92	xc3s50-5
MICKEY v2 [32]	4.29	1	1	80	115	233	2.03	xc3s50-5
MICKEY 128 [32]	4.48	1	1	128	176	223	1.27	xc3s50-5
Trivium [32]	4.17	1	1	80	50	240	4.80	xc3s50-5
Trivium (x64) [32]	4.74	1	64	80	344	13,504	39.26	xc3s400-5

This leads to a 4.5 times higher throughput for XTEA and could result in a 4.5 times lower energy consumption per bit.

The stream cipher implementation listed in Table 4 were published in [29] and have a much higher throughput than the XTEA or AES implementations at a similar power consumption. Therefore, they consume less energy per bit. Light weight ciphers like DESXL, Camelia and Present perform similarly well.

5.2 FPGA Implementations

The results of our XTEA implementations on FPGAs are summarized in Table 5 for tiny XTEA and Table 6 for speed XTEA. We expect from the ASIC analysis in Table 2 that TinyXTEA-1, using only one adder, consumes a slightly larger area than TinyXTEA-3 with three adders. Table 5 shows that this holds true for the FPGA implementation as well.

The throughput of TinyXTEA-3 is almost twice as fast as TinyXTEA-1 since it needs half as many clock cycles to encrypt one block of data. This is also reflected in the throughput to area ratio. The smallest AES implementation is the 8-bit AES by Good and Benaissa [30] which is similar in size to TinyXTEA. However, its throughput is almost 9 times lower than TinyXTEA-3. The AES by Chodowiec and Gaj [31] is twice as large as TinyXTEA-3 and its throughput is almost twice as fast leading to a similar throughput area ratio. The 128-bit key versions of the stream ciphers [32] MICKEY and Grain have a higher throughput area ratio than XTEA or AES and occupy less area. We would like to remark that our implementation does not involve block RAMs.

The high speed XTEA implementations summarized in Table 6 confirm our predictions from Table 1. SpeedXTEA-2, with 2 cuts per halfround, has a much

Table 6. Results for Speed XTEA compared to fast AES implementations (FPGA)

Design	Maximum Delay (ns)	Clock Cycles	Clock Cycles Latency	Block Size (bits)	Area (slices)	Throughput (Mbps)	Throughput/ Area (Mbps/slice)	Device
SpeedXTEA-1	8.00	1	128	64	14,574	8,004	0.55	xc3s2000-5
SpeedXTEA-2	5.79	1	192	64	18,515	11,050	0.60	xc3s2000-5
SpeedXTEA-1 (V)	3.50	1	128	64	8,655	18,286	2.11	xc5vlx85-3
SpeedXTEA-2 (V)	3.10	1	192	64	9,647	20,645	2.14	xc5vlx85-3
AES 128-bit (S) [30]	5.10	1	70	128	17,425	25,101	1.44	xc3s2000-5
AES 128-bit (V) [30]	5.41	1	70	128	16,693	23,654	1.42	xcv2000E-8

shorter critical path delay than SpeedXTEA-1, and even though it consumes more area, its throughput to area ratio is larger. Good and Benaissa report on two fully loop unrolled high speed AES implementations in [30], one on a Xilinx Spartan 3 (AES 128-bit (S)) and one on a Xilinx Virtex-E (AES 128-bit (V)). The AES implementation has a slightly shorter critical path delay than the SpeedXTEA-2 implementation on the same device and consume almost the same amount of area. However, due to its two times larger block size the throughput area ratio of AES is two times higher.

6 Conclusion

Our results on ASIC and FPGA show that XTEA is suitable for high-speed applications, however, it does not perform as fast as AES. Any speed improvement for XTEA would likely involve a significant increase in area and result in a lower throughput area ratio. Our ultra-low power implementation show that XTEA might be better suited for low resource environments than AES. Furthermore, XTEA's smaller block size of 64-bit is advantageous for applications where fewer than 128 bits of data have to be encrypted at a time. However, stream ciphers are a very interesting option as they have a higher throughput while consuming a smaller or similar sized area. The small code size of XTEA makes it an interesting choice in environments where some devices use software and other use hardware implementations.

References

1. Wheeler, D., Needham, R.: TEA, a tiny encryption algorithm. Technical report, Cambridge University, England (November 1994)
2. Wheeler, D., Needham, R.: TEA, a tiny encryption algorithm. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 363–366. Springer, Heidelberg (1995)
3. Kelsey, J., Schneier, B., Wagner, D.: Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, newDES, RC2, and TEA. In: Han, Y., Okamoto, T., Qing, S. (eds.) ICIS 1997. LNCS, vol. 1334, pp. 233–246. Springer, Heidelberg (1997)

4. Wheeler, D., Needham, R.: TEA extensions. Technical report, Cambridge University, England (October 1997)
5. Wheeler, D., Needham, R.: Correction to xtea. Technical report, Cambridge University (1998)
6. Castro, J.C.H., Viñuela, P.I.: New results on the genetic cryptanalysis of TEA and reduced-round versions of XTEA. In: Congress on Evolutionary Computation CEC 2004, vol. 2, pp. 2124–2129 (2004)
7. Moon, D., Hwang, K., Lee, W., Lee, S., Lim, J.: Impossible differential cryptanalysis of reduced round XTEA and TEA. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 49–60. Springer, Heidelberg (2002)
8. Liu, S., Gavrylyako, O.V., Bradford, P.G.: Implementing the TEA algorithm on sensors. In: ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference, pp. 64–69. ACM Press, New York (2004)
9. Pavlin, M.: Encryption using low cost microcontrollers. In: 42nd International Conference on Microelectronics, Devices and Materials and the Workshop on MEMS and NEMS, Society for Microelectronics Electronic, pp. 189–194 (2006)
10. Niati, R., Yazdani, N.: A more energy efficient network setup method for wireless sensor networks. In: Asia-Pacific Conference on Communications, pp. 640–643 (2005)
11. Kanamori, Y.: Reliability and security in a wireless body area network of intelligent sensors. Master's thesis, The University of Alabama in Huntsville (July 2002)
12. Amirtharajah, R., Chandrakasan, A.P.: Self-powered signal processing using vibration-based power generation. IEEE Journal of Solid-State Circuits 33(5), 687–695 (1998)
13. Israsena, P.: Securing ubiquitous and low-cost RFID using tiny encryption algorithm. In: Symp. on Wireless Pervasive Computing, 4 pp. IEEE, Los Alamitos (2006)
14. Israsena, P.: Design and implementation of low power hardware encryption for low cost secure RFID using TEA. In: Information, Communications and Signal Processing, pp. 1402–1406 (December 2005)
15. Brent, R.P., Kung, H.T.: A regular layout for parallel adders. IEEE Transactions on Computers C-31(3), 260–264 (1982)
16. Kaps, J.P., Sunar, B.: Energy comparison of AES and SHA-1 for ubiquitous computing. In: Zhou, X., Sokolsky, O., Yan, L., Jung, E.-S., Shao, Z., Mu, Y., Lee, D.C., Kim, D.Y., Jeong, Y.-S., Xu, C.-Z. (eds.) EUC Workshops 2006. LNCS, vol. 4097, pp. 372–381. Springer, Heidelberg (2006)
17. Satoh, A.: High-speed hardware architectures for authenticated encryption mode GCM. In: International Symposium on Circuits and Systems (ISCAS) 2006, pp. 4831–4834 (May 2006)
18. Hodjat, A., Verbauwhede, I.: Speed-area trade-off for 10 to 100 Gbits/s throughput AES processor. In: Asilomar Conference on Signals, Systems and Computers, vol. 2, pp. 2147–2150 (November 2003)
19. Hodjat, A., Verbauwhede, I.: Area-throughput trade-offs for fully pipelined 30 to 70 Gbits/s AES processors. IEEE Trans. Computers 55(4), 366–372 (2006)
20. Hodjat, A., Verbauwhede, I.: Minimum area cost for a 30 to 70 Gbits/s AES processor. In: IEEE Comp. Soc. Annual Symposium on VLSI, pp. 83–88 (February 2004)
21. 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)

22. Feldhofer, M., Wolkerstorfer, J., Rijmen, V.: AES implementation on a grain of sand. IEE Proceedings of Information Security 152(1), 13–20 (2005)
23. Babbage, S., Canniére, C.D., Canteaut, A., Cid, C., Gilbert, H., Johansson, T., Parker, M., Preneel, B., Rijmen, V., Robshaw, M.: The eSTREAM portfolio. Technical report, eSTREAM, ECRYPT Stream Cipher Project (April 2008)
24. Leander, G., Paar, C., Poschmann, A., Schramm, K.: New lightweight DES variants. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 196–210. Springer, Heidelberg (2007)
25. Poschmann, A., Leander, G., Schramm, K., Paar, C.: New light-weight crypto algorithms for RFID. In: International Symposium on Circuits and Systems (ISCAS), pp. 1843–1846 (May 2007)
26. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-bit block cipher suitable for multiple platforms – design and analysis. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012, pp. 39–56. Springer, Heidelberg (2001)
27. Satoh, A., Morioka, S.: Hardware-focused performance comparison for the standard block ciphers AES, Camellia, and Triple-DES. In: Boyd, C., Mao, W. (eds.) ISC 2003. LNCS, vol. 2851, pp. 252–266. Springer, Heidelberg (2003)
28. Bogdanov, A., Knudsen, L., 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)
29. Good, T., Benaissa, M.: Hardware performance of eStream phase-iii stream cipher candidates. In: State of the Art of Stream Ciphers Workshop (SASC 2008), pp. 163–173 (February 2008)
30. Good, T., Benaissa, M.: AES on FPGA from the fastest to the smallest. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 427–440. Springer, Heidelberg (2005)
31. Chodowiec, P., Gaj, K.: Very compact FPGA implementation of the AES algorithm. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 319–333. Springer, Heidelberg (2003)
32. Hwang, D., Chaney, M., Karanam, S., Ton, N., Gaj, K.: Comparison of FPGA-targeted hardware implementations of eSTREAM stream cipher candidates. In: State of the Art of Stream Ciphers Workshop (SASC 2008), pp. 151–162 (February 2008)

High Speed Compact Elliptic Curve Cryptoprocessor for FPGA Platforms

Chester Rebeiro[1] and Debdeep Mukhopadhyay[2]

[1] MS Student, Dept. of Computer Science and Engineering
Indian Institute of Technology Madras, India
rebeiro@cse.iitm.ernet.in
[2] Assistant Professor, Dept. of Computer Science and Engineering
Indian Institute of Technology Kharagpur, India
debdeep@cse.iitkgp.ernet.in

Abstract. This paper proposes an efficient high speed implementation of an elliptic curve crypto processor (ECCP) for an FPGA platform. The main optimization goal for the ECCP is efficient implementation of the important underlying finite field primitives namely multiplication and inverse. The techniques proposed maximize the utilization of FPGA resources. Additionally improved scheduling of elliptic curve point arithmetic results in lower number of register files thus reducing the area required and the critical delay of the circuit. Through several comparisons with existing work we demonstrate that the combination of the above techniques helps realize one of the fastest and compact elliptic curve processors.

1 Introduction

Elliptic Curve Cryptography (ECC) provides more security per key bit compared to other security standards. Although fast due to the shorter key size, software implementations of ECC do not meet the high speed required by some networking applications. These applications require ECC to be accelerated by dedicated hardware engines. The most common platform for such hardware accelerators are FPGAs. There are several advantages of using FPGAs for cryptographic applications [21]. Most important is the in-house programmability feature, reconfigurability, low non-recurring costs, simpler design cycles, faster time to market, and greater performance per unit area.

Implementation of ECC follows a layered hierarchical scheme. The performance of the top layers in the hierarchy is greatly influenced by the performance of the underlying layers. It is therefore important to have efficient implementations of finite field arithmetic which form the bottom layer of the hierarchy. Generally for ECC use of prime fields or binary extension fields is recommended. Binary extension fields have the advantage that they have an efficient representation on a computer.

The *elliptic curve crypto processor* (ECCP) proposed in this paper is designed for high speed applications using FPGA as the platform. There are several reported high performance FPGA processors for elliptic curve cryptography

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 376–388, 2008.

[1] [2] [3] [10]. Various acceleration techniques have been used although most implementations focus on the top layers of ECC. Pipelining and parallelism of the top layers is used to speed up point operations. High speed is also obtained by precomputations, efficient point representations, use of special curves, and efficient instruction scheduling techniques. The ECCP described here achieves high speed by implementing efficient finite field primitives. Additionally the primitives proposed in this paper are optimized for the FPGA platform thus resulting in good area×time product of the processor.

The most important arithmetic primitives for binary finite fields are multiplication and inversion as they occupy the most area and have the longest delay compared to other primitives. Efficient implementation of these primitives would therefore result in an efficient elliptic curve processor. There are several finite field multiplication algorithms that exist in literature. Of these the *Karatsuba* multiplier [7] is one that has sub-quadratic area complexity. It is also shown to be fastest if designed properly [17] [4]. The most common finite field inversion algorithms are based on the *extended Euclidean algorithm* (EEA) and the *Itoh-Tsujii algorithm* (ITA) [6]. Generally the EEA and its variants, the binary EEA and Montgomery algorithms, result in compact hardware implementations, while the ITA is faster. The large area required by the ITA is mainly due to the multiplication unit. ECC requires to have a multiplier present. This multiplier can be reused by the ITA for inverse computations. In this case the multiplier need not be considered in the area required by the ITA. The resulting ITA without the multiplier is as compact as the EEA making it the ideal choice for multiplicative inverse hardware [17].

The elliptic curve processor presented is built with novel multiplication and inversion algorithms. It efficiently implements the elliptic curve operations on the processor so that the scalar multiplication completes in minimum number of clock cycles (given the single finite field multiplier present in the design) and requires the minimum number of register files. The proposed implementation requires two register files, this is lesser compared to [14], which required three register files. The smaller number of register files results in lesser area required. The resulting ECCP is one of the fastest reported and has best area utilization.

The paper is organized as follows: Section 2 has the required background for ECC. Section 3 presents the implementation of the finite field primitives. This section proposes the use of a *hybrid Karatsuba multiplier* and a *quad-Itoh Tsujii inversion algorithm* for FPGA platforms. The 4^{th} section describes the construction of the ECCP, and the 5^{th} section has the comparison of the ECCP with reported works. The final section has the conclusion.

2 Background

The elliptic curve can be represented in *affine coordinates* (2 point system) or *projective coordinates* (3 point system). The projective coordinate representation of an elliptic curve over the field $GF(2^m)$ is given by

$$Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^4 \tag{1}$$

where a and $b \in GF(2^m)$ and $b \neq 0$. The points on the elliptic curve together with the point at infinity ($\mathcal{O}$) form an abelian group under addition.

The operations that are performed on the group are point addition and point doubling. The equation for point addition in *López-Dahab* (LD) projective coordinates [9] for the projective point $P = (X_1, Y_1, Z_1)$ and the affine point $Q = (x_2, y_2)$ is shown in Equation 2. The result is the point on the curve $(P + Q) = (X_3, Y_3, Z_3)$.

$$\begin{gathered} A = y_2 \cdot Z_1^2 + Y_1 \ ; \ B = x_2 \cdot Z_1 + X_1 \ ; \ C = Z_1 \cdot B \\ D = B^2 \cdot (C + a \cdot Z_1^2) \ ; \ Z_3 = C^2 \ ; \ E = A \cdot C \ ; \ X_3 = A^2 + D + E \\ F = X_3 + x_2 \cdot Z_3 \ ; \ G = (x_2 + y_2) \cdot Z_3^2 \ ; \ Y_3 = (E + Z_3) \cdot F + G \end{gathered} \tag{2}$$

The LD projective equation of doubling a point $P = (X_1, Y_1, Z_1)$ is given in Equation 3. The result is the point on the curve $2P = (X_3, Y_3, Z_3)$.

$$\begin{gathered} Z_3 = X_1^2 \cdot Z_1^2 \ ; X_3 = X_1^4 + b \cdot Z_1^4 \\ Y_3 = b \cdot Z_1^4 \cdot Z_3 + X_3 \cdot (a \cdot Z_3 + Y_1^2 + b \cdot Z_1^4) \end{gathered} \tag{3}$$

The cryptographic operation in the ECCP is *scalar multiplication* : Given a basepoint P on the curve and a scalar k, the scalar multiplication determines the product kP. This computation is done using a *double and add algorithm* which traverses the bits of the scalar k and does a point doubling for every bit in k. For every bit set to 1 the point doubling is followed by a point addition.

3 Implementing Finite Field Primitives on an FPGA

Maximizing the performance of the finite field primitives requires the design to be customized for the target hardware. The Xilinx FPGA [22] is made up of *configurable logic blocks* (CLBs). Each CLB on a Xilinx Virtex 4 FPGA contains two slices. Each slice contains two *lookup tables* (LUTs). The LUT is the smallest programmable element in the FPGA. A LUT has four inputs and can be configured for any logic function having a maximum of four inputs. The LUT can also be used to implement logic functions having less than four inputs, two for example. In this case only half the LUT is utilized the remaining part is not utilized. Such a LUT having less than four inputs is an *under utilized LUT*. Most compact implementations are obtained when the utilization of each LUT is maximized. The percentage of under utilized LUTs in a design is determined using Equation 4. LUT_k signifies that k ($1 \leq k \leq 4$) inputs out of 4 are used by the design block realized by the LUT. So, LUT_2 and LUT_3 are under utilized LUTs, while LUT_4 is fully utilized.

$$\%UnderUtilizedLUTs = \frac{LUT_2 + LUT_3}{LUT_2 + LUT_3 + LUT_4} * 100 \tag{4}$$

3.1 Finite Field Multiplication

Finite field multiplication of two elements in the field $GF(2^m)$ is defined as $C(x) = A(x)B(x)modP(x)$, where $A(x)$, $B(x)$, and $C(x) \in GF(2^m)$ and $P(x)$ is the irreducible polynomial of degree m which generates the field $GF(2^m)$. Implementing the multiplication requires two steps. First, the polynomial product $C'(x) = A(x)B(x)$ is determined, then the modular operation is done on $C'(x)$. The Karatsuba algorithm is used for the polynomial multiplication. The Karatsuba algorithm achieves its efficiency by splitting the m bit multiplicands into two 2-term polynomials : $A(x) = A_h x^{m/2} + A_l$ and $B(x) = B_h x^{m/2} + B_l$. The multiplication is then done using three $m/2$ bit multiplications as shown in Equation 5. The three $m/2$ bit multiplications are implemented recursively using the Karatsuba algorithm.

$$\begin{aligned} C'(x) &= (A_h x^{m/2} + A_l)(B_h x^{m/2} + B_l) \\ &= A_h B_h x^m + (A_h B_l + A_l B_h) x^{m/2} + A_l B_l \\ &= A_h B_h x^m + ((A_h + A_l)(B_h + B_l) + A_h B_h + A_l B_l) x^{m/2} + A_l B_l \end{aligned} \tag{5}$$

The basic recursive Karatsuba multiplier cannot be applied directly to ECC because the binary extension fields used in standards such as [19] have a degree which is prime. There have been several published variants of the Karatsuba multiplier for ECC such as the *binary Karatsuba multiplier* [15], the *simple Karatsuba multiplier* [20], and the *general Karatsuba multiplier* [20]. The simple Karatsuba multiplier is the basic recursive Karatsuba multiplier with a small modification. If an m bit multiplication is needed to be done, m being any integer, it is split into two polynomials as in Equation 5. The A_l and B_l terms have $\lceil m/2 \rceil$ bits and the A_h and B_h terms have $\lfloor m/2 \rfloor$ bits. The Karatsuba multiplication can then be done with two $\lceil m/2 \rceil$ bit multiplications and one $\lfloor m/2 \rfloor$ bit multiplication. In the general Karatsuba multiplier, the multiplicands are split into more than two terms. For example an m term multiplier is split into m different terms.

Hybrid Karatsuba Multiplier : The design of the proposed hybrid Karatsuba multiplier is based on observations from Table 1. The table compares the general and simple Karatsuba multipliers for gate counts, LUTs, and percentage

Table 1. Multiplication Comparison on Xilinx Virtex 4 FPGA

n	General			Simple		
	Gates	LUTs	LUTs Under Utilized	Gates	LUTs	LUTs Under Utilized
2	7	3	66.6%	7	3	66.6%
4	37	11	45.5%	33	16	68.7%
8	169	53	20.7%	127	63	66.6%
16	721	188	17.0%	441	220	65.0%
29	2437	670	10.7%	1339	669	65.4%
32	2977	799	11.3%	1447	723	63.9%

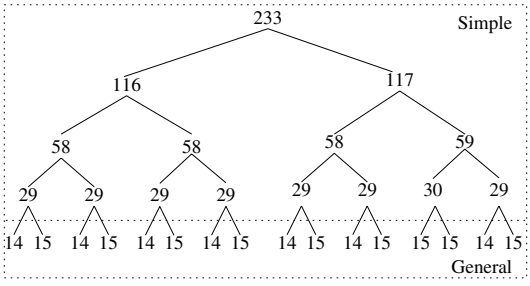


Fig. 1. 233 Bit Hybrid Karatsuba Multiplier

of under utilized LUTs on a Xilinx Virtex 4 FPGA. For the simple Karatsuba multiplier, the percentage of under utilized LUTs is high resulting in bloated area requirements. In the case of the general Karatsuba multiplier, the percentage of under utilized LUTs is low therefore there is better LUT utilization even though the gate count is higher. For $n > 29$, the number of gates in the general Karatsuba multiplier exceeds the benefits obtained by fully utilizing the LUTs resulting in bigger area requirements.

In the hybrid Karatsuba multiplier, all recursions are done using the simple Karatsuba multiplier except the final recursion. The final recursion is done using a general Karatsuba multiplier when the multiplicands have a size less than 29 bits. The initial recursions using the simple Karatsuba multiplier result in low gate count, while the final recursion using the general Karatsuba multiplier result in low LUT requirements. For a 233-bit hybrid Karatsuba multiplier shown in Figure 1, the four initial recursions are done using the simple Karatsuba multiplier, while the final recursion is done with 14-bit and 15-bit general Karatsuba multipliers.

3.2 Finite Field Inversion

The *multiplicative inverse* of an element $a \in GF(2^m)$ is the element $a^{-1} \in GF(2^m)$ such that $a^{-1} \cdot a \equiv 1 \bmod(m)$. From Fermat's little theorem, the multiplicative inverse can be written as $a^{-1} = a^{2^m-2} = (a^{2^{m-1}-1})^2$. The naive technique of computing a^{-1} requires $(m-2)$ multiplications and $(m-1)$ squarings. Itoh and Tsujii in [6] reduced the number of multiplications required by an efficient use of addition chains. An *addition chain* for $n \in \mathbb{N}$ is a sequence of integers of the form $U = (\ u_0\ \ u_1\ \ u_2\ \ \cdots u_r\ )$ satisfying the properties $u_0 = 1$, $u_r = n$ and $u_i = u_j + u_k$, for some $k \leq j < i$. An addition chain for 232 is $U = (\ 1\ \ 2\ \ 3\ \ 6\ \ 7\ \ 14\ \ 28\ \ 29\ \ 58\ \ 116\ \ 232\ )$.

Let $\beta_k(a) = a^{2^k-1} \in GF(2^m)$ and $\beta_{k+j}(a) = (\beta_j)^{2^k}\beta_k = (\beta_k)^{2^j}\beta_j$ [16], then $a^{-1} = (\beta_{m-1}(a))^2$. Using the addition chain shown above, the inverse of an element $a \in GF(2^{233})$ can be determined with 232 squarings and 10 multiplications as shown in Table 2.

Table 2. Inverse of $a \in GF(2^{233})$ using generic ITA

	$\beta_{\mathbf{u_i}}(\mathbf{a})$	$\beta_{\mathbf{u_j}+\mathbf{u_k}}(\mathbf{a})$	Exponentiation
1	$\beta_1(a)$		a
2	$\beta_2(a)$	$\beta_{1+1}(a)$	$(\beta_1)^{2^1}\beta_1 = a^{2^2-1}$
3	$\beta_3(a)$	$\beta_{2+1}(a)$	$(\beta_2)^{2^1}\beta_1 = a^{2^3-1}$
4	$\beta_6(a)$	$\beta_{3+3}(a)$	$(\beta_3)^{2^3}\beta_3 = a^{2^6-1}$
5	$\beta_7(a)$	$\beta_{6+1}(a)$	$(\beta_6)^{2^1}\beta_1 = a^{2^7-1}$
6	$\beta_{14}(a)$	$\beta_{7+7}(a)$	$(\beta_7)^{2^7}\beta_7 = a^{2^{14}-1}$
7	$\beta_{28}(a)$	$\beta_{14+14}(a)$	$(\beta_{14})^{2^{14}}\beta_{14} = a^{2^{28}-1}$
8	$\beta_{29}(a)$	$\beta_{28+1}(a)$	$(\beta_{28})^{2^1}\beta_1 = a^{2^{29}-1}$
9	$\beta_{58}(a)$	$\beta_{29+29}(a)$	$(\beta_{29})^{2^{29}}\beta_{29} = a^{2^{58}-1}$
10	$\beta_{116}(a)$	$\beta_{58+58}(a)$	$(\beta_{58})^{2^{58}}\beta_{58} = a^{2^{116}-1}$
11	$\beta_{232}(a)$	$\beta_{116+116}(a)$	$(\beta_{116})^{2^{116}}\beta_{116} = a^{2^{232}-1}$

Generalizing the Itoh-Tsujii Algorithm: The equation for the square of an element $a \in GF(2^m)$ is given by $a(x)^2 = \sum_{i=0}^{m-1} a_i x^{2i} \bmod p(x)$, where $p(x)$ is the irreducible polynomial. This is a linear equation and hence can be represented in the form of a matrix (T) as shown : $a^2 = T \cdot a$. The matrix depends on the finite field $GF(2^m)$ and the irreducible polynomial of the field. The exponentiation in the ITA is done with squarer circuits. We extend the ITA so that the exponentiation can be done with any 2^n circuit and not just squarers. Raising a to the power of 2^n is also linear and can be represented in the form of a matrix: $a^{2^n} = T^n(a) = T'a$.

For any $a \in GF(2^m)$ and $k \in \mathbb{N}$, define $\alpha_k(a) = a^{2^{nk}-1}$. Using the theorems shown below, we can conclude that any 2^n circuit can be used to implement the Itoh-Tsujii algorithm thus generalizing the algorithm.

Theorem 1. *If* $a \in GF(2^m)$, $\alpha_{k_1}(a) = a^{2^{nk_1}-1}$ *and* $\alpha_{k_2}(a) = a^{2^{nk_2}-1}$ *then* $\alpha_{k_1+k_2}(a) = (\alpha_{k_1}(a))^{2^{nk_2}}\alpha_{k_2}(a)$, *where* k_1, k_2 *and* $n \in \mathbb{N}$.

Theorem 2. *The inverse of an element* $a \in GF(2^m)$ *is given by* $a^{-1} = \left[\alpha_{\frac{m-1}{n}}(a)\right]^2$ *when* $n \mid (m-1)$ *and* $a^{-1} = \left[(\alpha_q(a))^{2^r}\beta_r(a)\right]^2$ *when* $n \nmid (m-1)$, *where* $nq + r = m - 1$ *and* n, q *and* $r \in \mathbb{N}$.

Quad Itoh Tsujii Inversion Algorithm: Consider the case when $n = 2$ such that $\alpha_k(a) = a^{4^k-1}$. To implement this requires quad circuits instead of the conventional squarers. On FPGA platforms, using quad circuits has advantages over squarers. An example of this advantage is shown in Table 3. The table shows the equation of each output bit for an element $b \in GF(2^9)$ for a squarer and a quad circuit and the number of LUTs each circuit takes.

We would expect the LUTs required by the quad circuit be twice that of the squarer. However this is not the case. The quad circuit's LUT requirement is only 1.5 times that of the squarer. This is because the quad circuit has a lower percentage of *under utilized LUTs* (Equation 4).For example, from Table 3 we note

Table 3. Comparison of LUTs required for a Squarer and Quad circuit for $GF(2^9)$

Output bit	Squarer Circuit		Quad Circuit	
	$b(x)^2$	#LUTs	$b(x)^4$	#LUTs
0	b_0	0	b_0	0
1	b_5	0	b_7	0
2	$b_1 + b_5$	1	$b_5 + b_7$	1
3	b_6	0	$b_3 + b_7$	1
4	$b_2 + b_6$	1	$b_1 + b_3 + b_5 + b_7$	1
5	b_7	0	b_8	0
6	$b_3 + b_8$	1	$b_6 + b_8$	1
7	b_8	0	$b_4 + b_8$	1
8	$b_4 + b_8$	1	$b_2 + b_4 + b_6 + b_8$	1
Total LUTs		4		6

Table 4. Comparison of Squarer and Quad Circuits on Xilinx Virtex 4 FPGA

Field	Squarer Circuit		Quad Circuit		Size ratio
	$\#LUT_s$	Delay (ns)	$\#LUT_q$	Delay (ns)	$\frac{\#LUT_q}{2(\#LUT_s)}$
$GF(2^{193})$	96	1.48	145	1.48	0.75
$GF(2^{233})$	153	1.48	230	1.48	0.75

Table 5. Inverse of $a \in GF(2^{233})$ using Quad-ITA

	$\alpha_{\mathbf{u_i}}(\mathbf{a})$	$\alpha_{\mathbf{u_j}+\mathbf{u_k}}(\mathbf{a})$	Exponentiation
1	$\alpha_1(a)$		a^3
2	$\alpha_2(a)$	$\alpha_{1+1}(a)$	$(\alpha_1)^{4^1}\alpha_1 = a^{4^2-1}$
3	$\alpha_3(a)$	$\alpha_{2+1}(a)$	$(\alpha_2)^{4^1}\alpha_1 = a^{4^3-1}$
4	$\alpha_6(a)$	$\alpha_{3+3}(a)$	$(\alpha_3)^{4^3}\alpha_3 = a^{4^6-1}$
5	$\alpha_7(a)$	$\alpha_{6+1}(a)$	$(\alpha_6)^{4^1}\alpha_1 = a^{4^7-1}$
6	$\alpha_{14}(a)$	$\alpha_{7+7}(a)$	$(\alpha_7)^{4^7}\alpha_7 = a^{4^{14}-1}$
7	$\alpha_{28}(a)$	$\alpha_{14+14}(a)$	$(\alpha_{14})^{4^{14}}\alpha_{14} = a^{4^{28}-1}$
8	$\alpha_{29}(a)$	$\alpha_{28+1}(a)$	$(\alpha_{28})^{4^1}\alpha_1 = a^{4^{29}-1}$
9	$\alpha_{58}(a)$	$\alpha_{29+29}(a)$	$(\alpha_{29})^{4^{29}}\alpha_{29} = a^{4^{58}-1}$
10	$\alpha_{116}(a)$	$\alpha_{58+58}(a)$	$(\alpha_{58})^{4^{58}}\alpha_{58} = a^{4^{116}-1}$

that output bit 4 requires three XOR gates in the quad circuit and only one in the squarer. However both circuits require only 1 LUT. These observations are scalable to large fields as is shown in Table 4.

Based on these observations we propose a *quad-ITA* which uses quad exponentiation circuits instead of squarers. The steps involved in obtaining the inverse of an element $a \in GF(2^{233})$ (Table 5) now requires 10 multiplications and 115 quads (compared to 232 quads when squarers are used).

4 Elliptic Curve Crypto Processor

The elliptic curve crypto processor (ECCP) (Figure 2) takes as input a scalar k and produces the product kP, where $P = (P_x, P_y)$ is the basepoint of the curve. The basepoint along with the curve constant is stored in the ROM and

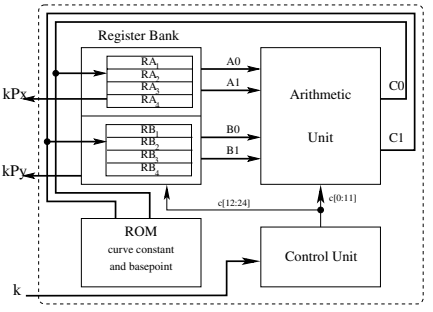


Fig. 2. Block Diagram of the Elliptic Curve Crypto Processor

loaded into registers during initialization. The ECCP implements the scalar multiplication algorithm using the elliptic curve double and add formulae. To be implemented, these formulae require arithmetic operations such as additions, squarings, and multiplications (Figure 3). Of these the hardware for multiplication is the biggest therefore the ECCP can afford to have only one finite field multiplier. Several adders and squarers can be used as they contribute marginally to the latency and area of the processor.

The equation for point addition (Equation 2) has 8 multiplications (assuming a=1) therefore with one multiplier (which is capable of doing one multiplication per clock cycle) it would require a minimum of 8 clock cycles. Similarly point doubling (Equation 3) would require a minimum of 4 clock cycles. The design of the arithmetic unit is optimized so that the addition and doubling takes the minimum required clock cycles.

The arithmetic unit in the ECCP has two outputs. At every clock cycle, at least one of the outputs is derived from the multiplier. This ensures that the multiplier is used in every clock cycle. In order to generate two outputs the

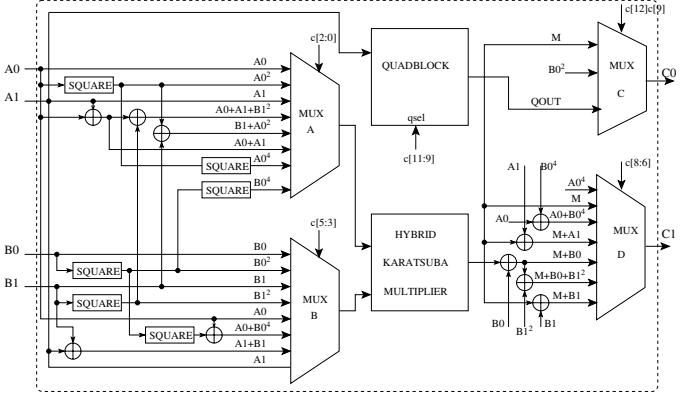


Fig. 3. Arithmetic Unit

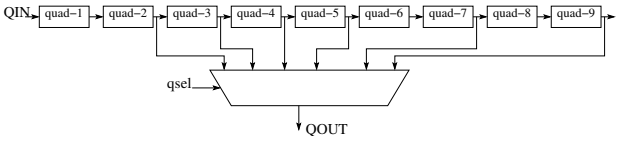


Fig. 4. Quadblock: Raises the Input to the Power of 4^q

arithmetic unit requires at least four input lines. The data on the four input lines is read from registers. The registers are implemented using the FPGA's *dual ported distributed RAM*. Each dual ported RAM has two address lines, two output data lines, and one input data line, therefore to feed the four input arithmetic unit at least two dual ported RAMs (RA and RB) are required. Each dual ported RAM, called a *register file*, implements four 233 bit registers and the two register files are collectively known as a *register bank*.

The *control unit* generates a 25 bit control word every clock cycle. This control word selects the four registers whose data would be read, selects the inputs to the multiplier using multiplexers $MUXA$ and $MUXB$ and selects the output of the arithmetic unit through $MUXC$ and $MUXD$. The control word also determines the register where the result gets written into.

Using LD projective coordinates has the overhead that the result has to be converted from projective to affine coordinates. This requires an inverse to be computed followed by two multiplications. The inverse is computed using the proposed quad-ITA. Obtaining the inverse requires steps in Table 5 to be computed. Each step has a computation of a power followed by a multiplication. The computation of the power has the form α^{4^q}, where q is as large as 58. Computing $\alpha^{4^{58}}$ would require 58 cascaded quad circuits. However this would result in a large latency. An alternate solution is to have $s(< 58)$ cascaded quad circuits. Computing the power α^{4^s} can be done in one clock cycle. Computing α^{4^q} with $q < s$ is done by tapping out the interim result using a multiplexer (Figure 4). Computing α^{4^q} with $q > s$ is done by recycling the result in the quadblock. This would require $\lceil q/s \rceil$ clock cycles. The number of cascades s is the largest number of quads such that the overall delay of the quadblock is less than the longest

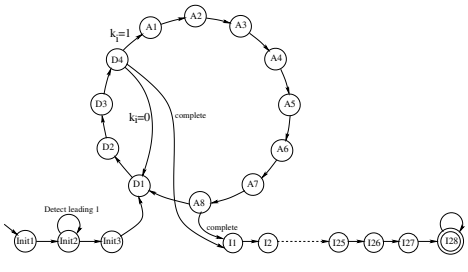


Fig. 5. Finite State Machine

Table 6. Scheduling Point Doubling on the ECCP

Cycle	Operation	AU Inputs				AU Output	
		$A0$	$A1$	$B0$	$B1$	$C0$	$C1$
1	$RA_1 = RB_4^2 \cdot RA_1^2$; $RB_3 = RA_1^4$	RA_1	-	-	RB_4	RA_1	RB_3
2	$RB_3 = RA_2 = RB_3 \cdot RA_3$	-	RA_3	RB_3	-	RA_2	RB_3
3	$RA_3 = (RB_4^4 + RA_2)(RA_1 + RB_1^2 + RA_2)$	RA_2	RA_1	RB_4	RB_1	RA_2	RB_4
	$RB_4 = (RB_4^4 + RA_2)$						
4	$RB_1 = RB_3 \cdot RA_1 + RA_2$	RA_1	RA_2	-	RB_3	-	RB_1

delay in the ECCP, which is through the multiplier. This ensures that the quad-block does not alter the maximum frequency of operation. For $GF(2^{233})$, nine cascades of quad produce the required result.

The finite state machine (FSM) for the ECCP is shown in Figure 5. There are three initialization states ($Init1$ to $Init3$), four states ($D1$ to $D4$) are required for the point doubling, eight states ($A1$ to $A8$) are required for the point addition. At the state $D4$ a decision is made depending on the value of the key bit k_i. If $k_i = 1$, the addition states are entered. If $k_i = 0$ the doubling corresponding to the next bit in k is considered. The final conversion of the result from projective to affine requires 28 states ($I1$ to $I28$). The number of clock cycles required is given by $\#ClockCycles = 3 + 12(h-1) + 4(l-h) + 28$, where l is the length of the scalar k and h its hamming weight.

4.1 Point Equations on the ECCP

The projective double and add equations on the ECCP (Equations 3 and 2) require to be scheduled efficiently in order to minimize the number of clock cycles required. The scheduling ensures that every clock cycle has a multiplication. Table 6 and Table 7 shows the scheduling for point doubling and point addition. They show the operations that are performed at every clock cycle, the data that is driven into the arithmetic unit, and the registers used to store the output of the arithmetic unit. The constant b is assumed to be present in register RA_3. The point $P = (X_1, Y_1, Z_1)$ is present in registers (RB_4, RB_1, RA_1) and another affine point $Q = (x_2, y_2)$ is present in registers (RA_4, RB_2). The point doubling operation computes $P = 2P$ while the point addition operation computes $P = P + Q$. Point doubling taking 4 clock cycles while addition takes 8 clock cycles.

Table 7. Scheduling Point Addition on the ECCP

Cycle	Operation	AU Inputs				AU Output	
		$A0$	$A1$	$B0$	$B1$	$C0$	$C1$
1	$RB_1 = RB_2 \cdot RA_1^2 + RB_1$	RA_1	-	RB_1	RB_2	-	RB_1
2	$RB_4 = RA_4 \cdot RA_1 + RB_4$	RA_1	RA_4	RB_4	-	-	RB_4
3	$RA_2 = RB_3 = RA_1 \cdot RB_4$	-	RA_1	-	RB_4	RA_2	RB_3
4	$RB_4 = RB_4^2(RB_3 + RA_1^2)$	RA_1	-	RB_4	RB_3	-	RB_4
5	$RA_2 = RB_1 \cdot RB_2$	RA_2	-	RB_4	RB_1	RA_2	RB_4
	$RB_4 = RB_1^2 + RB_4 + RB_1 \cdot RA_2$						
6	$RA_1 = RB_3^2$; $RB_3 = RB_4 + RA_4 \cdot RB_3^2$	-	RA_4	RB_3	RB_4	RA_1	RB_3
7	$RB_1 = (RA_4 + RB_2)RA_1^2$	RA_1	RA_4	-	RB_2	-	RB_1
8	$RB_1 = (RA_1 + RA_2)RB_3 + RB_1$	RA_1	RA_2	RB_1	RB_3	-	RB_1

5 Performance Evaluation

In this section we compare our work with reported $GF(2^m)$ elliptic curve crypto processors implemented on FPGA platforms (Table 8). Our ECCP was synthesized using *Xilinx's ISE* for *Virtex 4* and *Virtex E* platforms. Since the published works are done on different field sizes, we use the measure *latency/bit* for evaluation. Here latency is the time required to compute kP. Latency is computed by assuming the scalar k has half the number of bits 1. The only faster implementations are [18] and [3]. However [18] does not perform the final inverse computation required for converting from LD to affine coordinates. Also, as shown in Table 9, our implementation has better area time product compared to [3], while the latency is almost equal. To compare the two designs we scaled the area of [3] by a factor of $(233/m)^2$ since area of the elliptic curve processors is mostly influenced by the multiplier, which has an area of $O(m^2)$. The time is scaled by a factor $(233/m)$ since it is linear.

Table 8. Comparison of the Proposed $GF(2^m)$ ECCP with FPGA based Published Results

Work	Platform	Field m	Slices	LUTs	Gate Count	Freq (MHz)	Latency (ms)	Latency /bit (ns)
Orlando [12]	XCV400E	163	-	3002	-	76.7	0.21	1288
Bednara [2]	XCV1000	191	-	48300	-	36	0.27	1413
Kerins [8]	XCV2000	239	-	-	74103	30	12.8	53556
Gura [5]	XCV2000E	163	-	19508	-	66.5	0.14	858
Mentens [11]	XCV800	160	-	-	150678	47	3.810	23812
Lutz [10]	XCV2000E	163	-	10017	-	66	0.075	460
Saqib [18]	XCV3200	191	18314	-	-	10	0.056	293
Pu [13]	XC2V1000	193	-	3601	-	115	0.167	865
Ansari [1]	XC2V2000	163	-	8300	-	100	0.042	257
Chelton [3]	XCV2600E	163	15368	26390	238145	91	0.033	202
	XC4V200	163	16209	26364	264197	153.9	0.019	116
This Work	XCV3200E	233	18705	36802	306516	28.91	0.065	279
	XC4V140	233	19674	37073	314818	60.05	0.031	124

Table 9. Comparing Area×Time Requirements with [3]

Work	Field (m)	Platform	Slices (S)	Scaled Slices $SS = S(\frac{233}{m})^2$	Latency (ms) (T)	Scaled Latency (ms) $TS = T(\frac{233}{m})$	Area ×Time ($SS \times TS$)
Chelton [3]	163	XC4V200	16209	33120	0.019	0.027	894
This Work	233	XC4V140	19674	19674	0.031	0.031	609

6 Conclusion

In this paper we proposed an elliptic curve crypto processor for binary finite fields. It is compact and one of the fastest implementations reported. High speed of operation is obtained by having a combinational finite field multiplier using the proposed quad Itoh Tsujii algorithm to find the inverse, duplicating hardware units, and efficient implementation of point arithmetic. The ECCP is also compact and has better area×time product compared to the fastest ECCP. Compactness is achieved by the proposed hybrid Karatsuba multiplier and by minimizing the register file requirements in the ECCP.

References

1. Ansari, B., Hasan, M.A.: High Performance Architecture of Elliptic Curve Scalar Multiplication. Technical report, Department of Electrical and Computer Engineering, University of Waterloo (2006)
2. Bednara, M., Daldrup, M., von zur Gathen, J., Shokrollahi, J., Teich, J.: Reconfigurable Implementation of Elliptic Curve Crypto Algorithms. In: Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM, pp. 157–164 (2002)
3. Chelton, W.N., Benaissa, M.: Fast Elliptic Curve Cryptography on FPGA. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 16(2), 198–205 (2008)
4. Grabbe, C., Bednara, M., Shokrollahi, J., Teich, J., von zur Gathen, J.: FPGA Designs of Parallel High Performance $GF(2^{233})$ Multipliers. In: Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS 2003), Bangkok, Thailand, vol. II, pp. 268–271 (May 2003)
5. Gura, N., Shantz, S.C., Eberle, H., Gupta, S., Gupta, V., Finchelstein, D., Goupy, E., Stebila, D.: An End-to-End Systems Approach to Elliptic Curve Cryptography. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 349–365. Springer, Heidelberg (2003)
6. Itoh, T., Tsujii, S.: A Fast Algorithm For Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases. Inf. Comput. 78(3), 171–177 (1988)
7. Karatsuba, A.A., Ofman, Y.: Multiplication of Multidigit Numbers on Automata. Soviet Physics Doklady 7, 595–596 (1963)
8. Kerins, T., Popovici, E., Marnane, W.P., Fitzpatrick, P.: Fully Parameterizable Elliptic Curve Cryptography Processor over $GF(2)$. In: Glesner, M., Zipf, P., Renovell, M. (eds.) FPL 2002. LNCS, vol. 2438, pp. 750–759. Springer, Heidelberg (2002)
9. López, J., Dahab, R.: Improved Algorithms for Elliptic Curve Arithmetic in $GF(2^n)$. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 201–212. Springer, Heidelberg (1999)
10. Lutz, J., Hasan, A.: High Performance FPGA based Elliptic Curve Cryptographic Co-Processor. In: ITCC 2004: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC 2004), Washington, DC, USA, vol. 2, p. 486. IEEE Computer Society, Los Alamitos (2004)
11. Mentens, N., Ors, S.B., Preneel, B.: An FPGA Implementation of an Elliptic Curve Processor $GF(2^m)$. In: GLSVLSI 2004: Proceedings of the 14th ACM Great Lakes symposium on VLSI, pp. 454–457. ACM, New York (2004)
12. Orlando, G., Paar, C.: A High Performance Reconfigurable Elliptic Curve Processor for $GF(2^m)$. In: Paar, C., Koç, Ç.K. (eds.) CHES 2000. LNCS, vol. 1965, pp. 41–56. Springer, Heidelberg (2000)
13. Pu, Q., Huang, J.: A Microcoded Elliptic Curve Processor for $GF(2^m)$ Using FPGA Technology. In: Proceedings of 2006 International Conference on Communications, Circuits and Systems, vol. 4, pp. 2771–2775 (June 2006)
14. Rodríguez, S.M.H., Rodríguez-Henríquez, F.: An FPGA Arithmetic Logic Unit for Computing Scalar Multiplication using the Half-and-Add Method. In: ReConFig 2005: International Conference on Reconfigurable Computing and FPGAs, Washington, DC, USA. IEEE Computer Society, Los Alamitos (2005)
15. Rodríguez-Henríquez, F., Koç, Ç.K.: On Fully Parallel Karatsuba Multipliers for $GF(2^m)$. In: Proc. of the International Conference on Computer Science and Technology (CST), pp. 405–410

16. Rodríguez-Henríquez, F., Morales-Luna, G., Saqib, N.A., Cruz-Cortés, N.: Parallel Itoh-Tsujii Multiplicative Inversion Algorithm for a Special Class of Trinomials. Des. Codes Cryptography 45(1), 19–37 (2007)
17. Rodríguez-Henríquez, F., Saqib, N.A., Díaz-Pèrez, A., Ķoc, Ç.K.: Cryptographic Algorithms on Reconfigurable Hardware (Signals and Communication Technology). Springer-Verlag New York, Inc., Secaucus (2006)
18. Saqib, N.A., Rodríiguez-Henríquez, F., Diaz-Perez, A.: A Parallel Architecture for Fast Computation of Elliptic Curve Scalar Multiplication Over $GF(2^m)$. In: Proceedings of 18th International Parallel and Distributed Processing Symposium (April 2004)
19. U.S. Department of Commerce, National Institute of Standards and Technology. Digital signature standard (DSS) (2000)
20. Weimerskirch, A., Paar, C.: Generalizations of the Karatsuba Algorithm for Efficient Implementations. Cryptology ePrint Archive, Report 2006/224 (2006)
21. Wollinger, T., Guajardo, J., Paar, C.: Security on FPGAs: State-of-the-art Implementations and Attacks. Trans. on Embedded Computing Sys. 3(3), 534–574 (2004)
22. Xilinx. Virtex-4 User Guide (2007)

More Discriminants with the Brezing-Weng Method

Gaetan Bisson[1,2] and Takakazu Satoh[3]

[1] LORIA, 54506 Vandoeuvre-lès-Nancy, France
[2] Technische Universiteit Eindhoven, 5600 Eindhoven, The Netherlands
[3] Tokyo Institute of Technology, 152-8551 Tokyo, Japan

Abstract. The Brezing-Weng method is a general framework to generate families of pairing-friendly elliptic curves. Here, we introduce an improvement which can be used to generate more curves with larger discriminants. Apart from the number of curves this yields, it provides an easy way to avoid endomorphism rings with small class number.

Keywords: Pairing-friendly curve generation, Brezing-Weng method.

1 Introduction

Since its birth in 2000, pairing-based cryptography has solved famous open problems in public key cryptography: the identity-based key-exchange [1], the one-round tripartite key-exchange [2] and the practical identity-based encryption scheme [3]. Pairings are now considered not only as tools for attacking the discrete logarithm problem in elliptic curves [4] but as building blocks for cryptographic protocols.

However, for these cryptosystems to be practical, elliptic curves with an efficiently computable pairing and whose discrete logarithm problem is intractable are required.

There are essentially two general methods for the generation of such curves: the Cocks-Pinch method [5], which generates individual curves, and the Brezing-Weng method [6], which generates families of curves while achieving better ρ-values.

Our improvement extends constructions based on these methods by providing more curves with discriminants larger than what the constructions would normally provide (by a factor typically up to 10^9 given current complexity of algorithms for computing Hilbert class polynomials). In the Cocks-Pinch method the discriminant can be freely chosen so our improvement is of little interest in this case; however, the Cocks-Pinch method is limited to $\rho \approx 2$. To achieve smaller ρ-values, one has to use the Brezing-Weng method where known efficient constructions mostly deal with small (one digit) discriminants; our improvement then provides an easy and efficient way to generate several curves with a wide range of discriminants, extending known constructions while preserving their efficiency (in particular, the ρ-value).

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 389–399, 2008.

The curves we generate, having a larger discriminant, are possibly more secure than curves whose endomorphism ring has small class number —even though, at the time of this writing, no attack taking advantage of a small class number is known. To say the least, our improvement brings a bit of diversity to families of curves as generated by the Brezing-Weng method.

In Section 2, we recall the general framework for pairing-friendly elliptic curve generation. Then, in Section 3, we present the Brezing-Weng algorithm and our improvement. Eventually, in Section 4, we study practical constructions and their efficiency; we also present a few examples.

2 Framework

2.1 Security Parameters

Let $\mathcal{E}$ be an elliptic curve defined over a prime finite field $\mathbb{F}_p$. We consider the discrete logarithm problem in some subgroup $\mathcal{H}$ of $\mathcal{E}$ of large prime order r. In addition, we assume that r is different from p.

For security reasons, the size of r should be large enough to avoid generic discrete logarithm attacks. For efficiency reasons, it should also not be too small when compared to the size of the ground field; indeed, it would be impractical to use the arithmetic of a very large field to provide the security level that could be achieved with a much smaller one. Therefore, the so-called ρ-value

$$\rho := \frac{\log p}{\log r}$$

must be as small as possible. Note that, for practical applications, it is desirable that the parameters of a cryptosystem (here, p) be of reasonable size relatively to the security provided by this cryptosystem (here, r), which is precisely what a small ρ-value asserts.

We wish to generate such an elliptic curve and ensure that it has an efficiently computable pairing, that is a non-degenerate bilinear map from $\mathcal{H}^2$ to some cyclic group.

Known pairings on elliptic curves, i.e. the Weil and Tate pairings, map to the multiplicative group of an extension of the ground field. By linearity, the non-degeneracy of the pairing (on the subgroup of order r) forces the extension to contain primitive r^{th} roots of unity. Let $\mathbb{F}_{p^k}$ be the minimal such extension; the integer k is called the embedding degree with respect to r. It can also be defined elementarily as

$$k = \min\{i \in \mathbb{N} : r \mid p^i - 1\}.$$

There are different ways of evaluating pairings, each featuring specific implementation optimizations. However, all known efficient methods are based on Miller's algorithm which relies on the arithmetic of $\mathbb{F}_{p^k}$. Therefore, the evaluation of a pairing can only be carried out when k is reasonably small.

In addition, the discrete logarithm problem must be practically intractable in both the subgroup of the curve and the multiplicative group of the embedding field. At the time of this writing, minimal security can be provided by the bounds

$$\log_2 r \geq 160 \text{ and } k \log_2 p \geq 1024.$$

However, these are to evolve and, as the bound on $k \log_2 p$ is expected to grow faster than that on $\log_2 r$ (because the complexity of the index-calculus attack on finite fields is subexponential whereas those of elliptic curve discrete logarithm algorithms are exponential), we have to consider larger embedding degrees in order to preserve small ρ-values.

2.2 Curve Generation

In order to generate an ordinary elliptic curve with a large prime order subgroup and an efficiently computable pairing, we look for suitable values of the parameters:

- p, the cardinality of the ground field;
- t, the trace of the Frobenius endomorphism of the curve (such that the curve has $p + 1 - t$ rational points);
- r, the order of the subgroup;
- k, its embedding degree.

Here, "suitable" means that there exists a curve achieving those values. This consistency of the parameters can be written as the following list of conditions:

1. p is prime.
2. t is an integer relatively prime to p.
3. $|t| \leq 2\sqrt{p}$.
4. r is a prime factor of $p + 1 - t$.
5. k is the smallest integer such that $r \mid p^k - 1$.

By a theorem of Deuring [7], Conditions 1–3 ensure that there exists an ordinary elliptic curve over $\mathbb{F}_p$ with trace t. The last conditions then imply that its subgroup of order r has embedding degree k.

When r does not divide k —which is always the case in cryptographic applications as we want k to be small (for the pairing to be computable) and r to be large (to avoid generic discrete logarithm attacks)— Condition 5 is equivalent to $r \mid \Phi_k(p)$, which is a much more handy equation; therefore, assuming Condition 4, it is also equivalent to

$$r \mid \Phi_k(t-1).$$

To retrieve the Weierstrass equation of a curve with such parameters using the complex multiplication method, we need to look at $-D$, the discriminant (which need not be squarefree) of the quadratic order in which the curve has complex multiplication. Indeed, the complex multiplication method is only effective when

this order has reasonably small class number. Due to a result of Heilbronn [8], in practice we ask for D to be a small positive integer.

Writing the Frobenius endomorphism as an element of the complex multiplication order leads to the very simple condition

$$\exists y \in \mathbb{N}, 4p = t^2 + Dy^2$$

which ensures that $-D$ is a possible discriminant. It is referred to as the complex multiplication equation. Note that, instead of being added to the list, this condition may supersede Condition 3 as it is, in fact, stronger.

Using the cofactor of r, namely the integer h such that $p + 1 - t = hr$, we can also write the complex multiplication equation as

$$Dy^2 = 4p - t^2 = 4hr - (t-2)^2 .$$

Note that if both the above equation considered modulo r and the "original" complex multiplication equation hold, we recover the equation that states that the curve has a subgroup of order r.

Assuming $p > 5$, the third condition implies that p divides t if and only if $t = 0$; therefore, as p is expected to be large, we only have to check whether $t \neq 0$. This condition is omitted from the list below as it (mostly) always holds in practical constructions; bear in mind that it is required, though.

Finally, we can summarize the requirements to generate a pairing-friendly elliptic curve; we are looking for

$$\begin{cases} p, r \text{ primes} \\ t, y \text{ integers} \\ D, k \text{ positive integers} \end{cases} \text{ such that } \begin{cases} r \mid Dy^2 + (t-2)^2 \\ r \mid \Phi_k(t-1) \\ t^2 + Dy^2 = 4p \end{cases} .$$

In practical computations, r may not necessarily be given as a prime. However, if r is a prime times a small cofactor, replacing it by that prime leads to the generation of a pairing-friendly elliptic curve without affecting much the ρ-value. Therefore, this slightly weaker condition is acceptable.

3 Algorithms

Let us fix D and k as small positive integers. The Cocks-Pinch method consists in solving the above equations to retrieve values of p, r, t and y; it proceeds in the following way:

1. Choose a prime r such that the finite field $\mathbb{F}_r$ contains $\sqrt{-D}$ and z, some primitive k^{th} root of unity.
2. Put $t = 1 + z$ and $y = \frac{t-2}{\sqrt{-D}} \mod r$.
3. Take lifts of t and y in $\mathbb{Z}$ and put $p = \frac{1}{4}\left(t^2 + Dy^2\right)$.

This algorithm has to be run for different parameters r and z until the output p is a prime integer; then, the complex multiplication method can be used to generate an elliptic curve over $\mathbb{F}_p$ with $p + 1 - t$ points, a subgroup of order r and embedding degree k.

Asymptotically, pairing-friendly elliptic curves generated by this algorithm have ρ-value 2.

3.1 The Brezing-Weng Method

The Brezing-Weng method starts similarly by fixing small positive integers D and k. Then, it looks for solutions to these equations as polynomials p, r, t and y in $\mathbb{Q}[x]$. Once a solution is found, for any integer x, an elliptic curve with parameters $(p(x), r(x), t(x), y(x), D, k)$ can be generated provided that $p(x)$ and $r(x)$ are prime and that $t(x)$ and $y(x)$ are integers.

To enable this, we expect polynomials p and r to have infinitely many simultaneous prime values. There is actually a very precise conjecture on the density of prime values of a family of polynomials:

Conjecture 1 (Bateman and Horn [9]). *Let $f_1, \ldots, f_s$ be s distinct (non-constant) irreducible integer polynomials in one variable with positive leading coefficient. The cardinality of R_N, the set of positive integers x less than N such that the $f_i(x)$'s are all prime, has the following asymptotic behavior:*

$$\operatorname{card} R_N \sim \frac{C(f_1, \ldots, f_s)}{\prod_i \deg f_i} \int_2^N \frac{du}{(\log u)^s} \qquad \text{when } N \to \infty,$$

the constant $C(f_1, \ldots, f_s)$ being defined as

$$\prod_{p \in \mathcal{P}} \left(1 - \frac{1}{p}\right)^{-s} \left(1 - \frac{1}{p} \operatorname{card} \left\{x \in \mathbb{F}_p : \prod_i f_i(x) = 0\right\}\right)$$

where $\mathcal{P}$ denotes the set of prime numbers.

The latter constant quantifies how much the f_i's differ from independent random number generators, based on their behavior over finite fields; it can, of course, be estimated using partial products.

However, if we only need a quick computational way of checking polynomials p and r, we may use a weaker corollary, earlier conjectured by Schinzel [10] and known as *hypothesis H*, which just consists in assuming that the constant $C(f_i)$ is non-zero. Consider two polynomials, p and r; in that case, the corollary states that, provided that

$$\gcd\{p(x) r(x) : x \in \mathbb{Z}\} = 1,$$

the polynomials p and r have infinitely many simultaneous prime values.

Actually, there is a subtle difference with the polynomials we are dealing with here: they might have rational coefficients. However, we believe that the assumption of the above conjecture can be slightly weakened as

$$\gcd\{p(x) r(x) : x \in \mathbb{Z} \text{ such that } p(x) \in \mathbb{Z} \text{ and } r(x) \in \mathbb{Z}\} = 1$$

so to work with families of rational polynomials. Of course, we use the convention $\gcd \emptyset = 0$ (in case there is no x such that both $p(x)$ and $r(x)$ are integers).

Given small positive integers D and k, the Brezing-Weng method works as follows:

1. Choose a polynomial r with positive leading coefficient such that $\mathbb{Q}[x]/(r)$ is a field containing $\sqrt{-D}$ and z, some primitive k^{th} root of unity.
2. Put $t = 1 + z$ and $y = \frac{t-2}{\sqrt{-D}}$ (represented as polynomials modulo r).
3. Take lifts of t and y in $\mathbb{Q}[x]$ and put $p = \frac{1}{4}\left(t^2 + Dy^2\right)$.

This algorithm has to be run for different parameters r and z until the polynomials p and r satisfy the conditions of the above conjecture. Then, we might be able to find values of x at which the instantiation of the polynomials yields a suitable set of parameters and thus generate an elliptic curve.

To heuristically check whether p and r satisfy the above conjecture, we compute the gcd of the product $p(x)\,r(x)$ for those $x \in \left\{1, \ldots, 10^2\right\}$ such that $p(x)$ and $r(x)$ are both integers. If this gcd is 1, the hypothesis of the conjecture is satisfied; otherwise, we assume it is not.

The main feature of this algorithm is that the ρ-value of the generated curves is asymptotically equal to $\frac{\deg p}{\deg r}$; therefore, a good ρ-value will be achieved if the parameters (D, k, r, z) can be chosen so that the polynomial p is of degree close to that of r. Because of the way p is defined, the larger the degree of r is, the more unlikely this is to happen.

Such wise choices are rare and mainly concerned with small discriminants; indeed, when D is a small positive integer, $\sqrt{-D}$ is contained in a cyclotomic extension of small degree which can therefore be taken as $\mathbb{Q}[x]/(r)$, thus providing a r-polynomial with small degree.

There exist a few wise choices for large D (cf. Paragraph 6.4 of [11]) but those are restricted to a small number of polynomials (p, r, t, y) and do not provide as many families as we would like.

3.2 Our Improvement

The key observation is that, if there exists an elliptic curve with parameters (p, r, t, y, D, k), then for every divisor n of y there also exists an elliptic curve with parameters $\left(p, r, t, \frac{1}{n}y, Dn^2, k\right)$. Note that this transformation preserves the ground field and the number of points of the curve, and therefore its ρ-value.

For one-shot Cocks-Pinch-like methods, this is of little interest since we could have set the discriminant to be $-Dn^2$ in the first place. However, for the Brezing-Weng method where good choices of the parameters (D, k, r, z) are not easily found, it provides a way to generate curves with a wider range of discriminants with the same machinery that we already have.

This improvement works as follows:

1. Generate a family (p, r, t, y, D, k) using the Brezing-Weng method.
2. Choose an integer x such that $p(x)$ and $r(x)$ are prime, and $t(x)$ and $y(x)$ are integers.
3. Compute the factorization of $y(x)$.
4. Choose some divisor n of $y(x)$ and generate a curve with parameters $\left(p(x), r(x), t(x), \frac{1}{n}y(x), Dn^2, k\right)$ using the complex multiplication method.

In Step 3, we do not actually have to compute the complete factorization of $y(x)$. Indeed, n cannot be too large in order for the complex multiplication

method with discriminant $-Dn^2$ to be practical. So, we only have to deal with the smooth part of $y(x)$.

However, to avoid efficiently computable isogenies between the original curve (with $n = 1$, as generated by the standard Brezing-Weng method) and our curve, n must have a sufficiently large prime factor [12]. Indeed, such an isogeny would reduce the discrete logarithm problem from our curve to the original curve.

These constraints are best satisfied when n is a prime in some interval. Specifically, let D be fixed and consider prime values for the variable n; the complexity of computing the Hilbert class polynomial (with discriminant $-Dn^2$) is $\Theta\left(n^2\right)$ [13] and that of computing the above-mentioned isogeny is $\Theta\left(n^3\right)$ [12].

Therefore, we recommend to choose a prime factor n of $y(x)$ as large as possible among those n such that the complex multiplication method with discriminant $-Dn^2$ is practical, that is, the Hilbert class polynomial is computable in reasonable time. Given current computing power, $n \approx 10^5$ seems to be a good choice; however, to choose the size of the parameter n more carefully, we refer to a detailed analysis of the complexity [13].

By a theorem of Siegel [14], when D is fixed, the class number of the quadratic field with discriminant $-Dn^2$ grows essentially linearly in n. Therefore, with n chosen as described above, the class number is expected to be reasonably large. This helps avoiding potential (though not yet known) attacks on curves with principal or nearly-principal endomorphism ring.

A toy example. Let $D = 8$, $k = 48$ and $r = \Phi_k$ (the cyclotomic polynomial of order k).

As x is a primitive k^{th} root of unity in $\mathbb{Q}[x]/(r)$, put

$$t(x) = 1 + x \text{ and } \sqrt{-D} = 2\left(x^6 + x^{18}\right).$$

The Brezing-Weng method outputs polynomials

$$y(x) = \frac{1}{4}\left(-x^{11} + x^{10} - x^7 + x^6 + x^3 - x^2\right) \text{ and } p = \frac{1}{4}\left(t^2 + Dy^2\right),$$

and the degree of p is such that this family has ρ-value 1.375.

For example, if $x = 137$ then

$$p(x) = 125429351059163205052743035650972214424622957 13$$

which is a prime number and $r(x)$ is a prime number as well. The next step is to factor $y(x)$ as

$$y(x) = -1 \cdot 2 \cdot 17 \cdot 137^2 \cdot 229 \cdot 9109 \cdot 84191 \cdot 706631$$

and n can possibly be any product of these factors.

Take for instance $n = 17$, which results in discriminant -2312 with class number 16 (as opposed to class number one which would be provided by the standard Brezing-Weng method, i.e. with $n = 1$). The Weierstrass equation

of a curve with parameters $\left(p\left(x\right), r\left(x\right), t\left(x\right), \frac{1}{n}y\left(x\right), Dn^2, k\right)$ is given by the complex multiplication method as

$$\begin{aligned} Y^2 = X^3 &+ 935824186433623028047894899424144532036848777X \\ &+ 898583952823329568888146564301424398299942966 0; \end{aligned}$$

this being, of course, an equation over $\mathbb{F}_{p(x)}$.

4 Constructions

We already mentioned that n should have a large prime factor. To increase chances for y to have such factors, we seek constructions where y is a nearly-irreducible polynomial, i.e. of degree close to that of its biggest (in terms of degree) irreducible factor.

Many constructions based on the Brezing-Weng method can be found in Section 6 of the survey article [11]. However, only few involve a nearly-irreducible y (most of those y are divisible by a power of x). Here, we describe a generic construction that is likely to provide nearly-irreducible y's.

4.1 Generic Construction

Fix an odd prime D and a positive integer k.

The extension $\mathbb{Q}\left[x\right]/\left(r\right)$ has to contain primitive k^{th} roots of unity; the simplest choice is therefore to consider a cyclotomic extension.

So, let us put $r = \Phi_{ke}$ for some integer e to be determined. Let ζ_D be a primitive D^{th} root of unity; the Gauss sum (for details, see, e.g., [15, Theorem 1.2.4])

$$\sqrt{\left(\frac{-1}{D}\right) D} = \sum_{i=1}^{D-1} \left(\frac{i}{D}\right) \zeta_D^i$$

shows that, for $\sqrt{-D}$ to be in $\mathbb{Q}\left[x\right]/\left(r\right)$, the product ke may be any multiple of εD where $\varepsilon = 4$ if -1 is a square modulo D, $\varepsilon = 1$ otherwise.

Therefore, we can use the following setting for the Brezing-Weng method:

1. Choose an odd prime D and a positive integer k.
2. Put $\varepsilon = 4$ if -1 is a square modulo D, $\varepsilon = 1$ otherwise.
3. Choose a positive integer e such that $\varepsilon D \mid ke$.
4. Choose a positive integer f relatively prime to k.
5. Put $r = \Phi_{ke}$, $z = x^{ef}$.
6. Use the expression

$$\sqrt{-D} = x^{\frac{ke}{\varepsilon}} \sum_{i=1}^{D-1} \left(\frac{i}{D}\right) x^{i\frac{ke}{D}} \mod r$$

for the computation of y in the Brezing-Weng method.

As the latter polynomial is of large degree, it can be expected to be quite random once reduced modulo r. Therefore, it is likely to be nearly-irreducible and so the polynomial y given by the Brezing-Weng method might also be nearly-irreducible.

To support this expectation, we have computed $\delta := \frac{\deg m}{\deg y}$ where m is the biggest irreducible factor of $y = \frac{-1}{D}(z-1)\sqrt{-D}$, the polynomials for z and $\sqrt{-D}$ being given by the above algorithm. There are 4670 valid quadruplets $(D,k,e,f) \in \{1,\ldots,20\}^4$ (i.e. for which D is an odd prime and $\varepsilon D \mid ke$); the following table gives the number of valid quadruplets in this range leading to values of δ with prescribed first decimal.

δ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
	79	27	51	72	26	309	388	320	807	1127	1464

We see that, in this range, more than 70% of valid quadruplets lead to a y-polynomial whose largest irreducible factor is of degree at least $0.8 \deg y$.

4.2 Examples

Generic Construction. Let $D = 3$, $k = 9$, $e = 1$ and $f = 4$.

The Brezing-Weng method outputs the polynomials

$$p(x) = \tfrac{1}{3}\left(x^8 + x^7 + x^6 + x^5 + 4x^4 + x^3 + x^2 + x + 1\right)$$
$$y(x) = \tfrac{1}{3}\left(x^4 + 2x^3 + 2x + 1\right)$$

which represent a family of elliptic curves with ρ-value 1.333.

To generate a cryptographically useful curve from this family, we look for an integer x such that $p(x)$ is a prime, $r(x)$ is nearly-prime and $y(x)$ is an integer; we also have to make sure that $p(x)^k$ and $r(x)$ are of appropriate size for both security and efficiency.

Many such x's are easily found by successive trials; for instance, in the integer interval $\left[2^{27}; 2^{28}\right]$, there are 58812 of them, which is only 6 times less than what a pair of independent random number generators would be expected to achieve (calculated as $\int_{2^{27}}^{2^{28}} \log^{-2}$); for slightly more than a fifth of these, $y(x)$ has a prime factor in the integer interval $\left[10^4; 10^6\right]$, which can therefore be used as n in our algorithm.

For example, let us put $x = 134499652$; we have:

$$\begin{aligned} p(x) &= 35698341005790839038787210375794\backslash \\ &\quad 98567395936309418834417714720 7303 \\ r(x) &= 3 \cdot 19733572211579266804451632197669472566760550628 91 \\ y(x) &= 419 \cdot 153733 \cdot 16934885676704545717544 77 \end{aligned}$$

If we choose $n = 153733$, the discriminant is $-3 \cdot 153733^2$ and has class number 51244; computations give a Weierstrass equation for the curve:

$$\begin{aligned} Y^2 = X^3 &+ 18380344310754022726680092877438\backslash \\ &\quad 21739421574060526966589831576899 7X \\ &+ 35411587190573547152432512636 04\backslash \\ &\quad 83038157372705450329206494776897 \end{aligned}$$

Sporadic Families. Our improvement requires families with nearly-irreducible y's which is why we described a generic construction that is able to generate such families for various parameters (D, k). However, for a few specific parameters, there are sporadic constructions with good ρ-values that also feature nearly-irreducible y's, and our improvement produces curves with larger discriminants without changing ρ-values.

To illustrate this, let us consider the Barreto-Naehrig family [16] which features the optimal ρ-value of 1 for parameters $D = 3$, $k = 12$ and is parametrized by the following polynomials:

$$\begin{aligned} p(x) &= 6^2x^4 + 6^2x^3 + 4 \cdot 6x^2 + 6x + 1 \\ r(x) &= 6^2x^4 + 6^2x^3 + 3 \cdot 6x^2 + 6x + 1 \\ y(x) &= 6x^2 + 4x + 1 \end{aligned}$$

For instance, if $x = 549755862066$, we have:

$$\begin{aligned} p(x) &= 32883798367124994775048315314962202487571011972 93 \\ r(x) &= 13 \cdot 61 \cdot 41467589365857496563743123809674312650342931 49 \\ y(x) &= 151579 \cdot 11963326366170669619 \end{aligned}$$

If we choose $n = 151579$, the discriminant is $-3 \cdot 151579^2$ and has class number 50526; computations give a Weierstrass equation for the curve:

$$\begin{aligned} Y^2 = X^3 &+ 98384233147804093223276013838047008541927121229 6X \\ &+ 28481481121270269398250611132511268894509149397 26 \end{aligned}$$

Acknowledgements

The authors would like to thank Pierrick Gaudry for helpful discussions and Andreas Enge for computing the explicit curve equations found in Section 4.2. Our gratitude also goes to Tanja Lange for her comments and suggestions on a draft version of this paper.

References

1. Sakai, R., Ohgishi, K., Kasahara, M.: Cryptosystems based on pairing. In: Proceedings of the Symposium on Cryptography and Information Security (2000); ref. C20
2. Joux, A.: A one round protocol for tripartite Diffie-Hellman. In: Bosma, W. (ed.) ANTS 2000. LNCS, vol. 1838, pp. 385–394. Springer, Heidelberg (2000)
3. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. SIAM Journal of Computing 32(3), 586–615 (2003)
4. Menezes, A., Okamoto, T., Vanstone, S.: Reducing elliptic curve logarithms in a finite field. IEEE Transactions on Information Theory 39(5), 1639–1646 (1993)
5. Cocks, C., Pinch, R.: Identity-based cryptosystems based on the Weil pairing (Unpublished manuscript, 2001)

6. Brezing, F., Weng, A.: Elliptic curves suitable for pairing based cryptography. Design, Codes and Cryptography 37(1), 133–141 (2005)
7. Deuring, M.: Die Typen der Multiplikatorenringe elliptischer Funktionenkörper. Abhandlungen aus dem mathematischen Seminar der hamburgischen Universität 14, 197–272 (1941)
8. Heilbronn, H.: On the class-number in imaginary quadratic fields. Quarterly Journal of Mathematics 5, 150–160 (1934)
9. Bateman, P., Horn, R.: Primes represented by irreducible polynomials in one variable. In: Proceedings of Symposia in Pure Mathematics, vol. 3, pp. 119–132. American Mathematical Society (1965)
10. Schinzel, A., Sierpinski, W.: Sur certaines hypothèses concernant les nombres premiers. Acta Arithmetica 4, 185–208 (1958)
11. Freeman, D., Scott, M., Teske, E.: A taxonomy of pairing-friendly elliptic curves. Cryptology ePrint Archive, Report 2006/372 (2006)
12. Galbraith, S.: Constructing isogenies between elliptic curves over finite fields. The London Mathematical Society Journal of Computation and Mathematics 2, 118–138 (1999)
13. Enge, A.: The complexity of class polynomial computation via floating point approximations. ArXiv preprint, cs.CC/0601104 (2006)
14. Siegel, C.: Über die Classenzahl quadratischer Zahlkörper. Acta Arithmetica 1, 83–86 (1935)
15. Berndt, B., Evans, R., Williams, K.: Gauss and Jacobi sums. John Wiley & Sons, Chichester (1998)
16. Barreto, P., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006)

Another Approach to Pairing Computation in Edwards Coordinates

Sorina Ionica[2] and Antoine Joux[1,2]

[1] DGA
[2] Université de Versailles Saint-Quentin-en-Yvelines, 45 avenue des États-Unis, 78035 Versailles CEDEX, France
{sorina.ionica,antoine.joux}@m4x.org

Abstract. The recent introduction of Edwards curves has significantly reduced the cost of addition on elliptic curves. This paper presents new explicit formulae for pairing implementation in Edwards coordinates. We prove our method gives performances similar to those of Miller's algorithm in Jacobian coordinates and is thus of cryptographic interest when one chooses Edwards curve implementations of protocols in elliptic curve cryptography. The method is faster than the recent proposal of Das and Sarkar for computing pairings on supersingular curves using Edwards coordinates.

Keywords: Tate pairing, Miller's algorithm, Edwards coordinates.

1 Introduction

Pairings on elliptic curves are currently of great interest due to their applications in a number of cryptographic protocols such as the tripartite Diffie-Hellman protocol [19], identity-based encryption [5], short signatures [6] and group signatures [7]. In this paper we propose to reassess the computational cost of pairings in the light of the introduction by Edwards [12] of a new representation of the addition law on elliptic curves. Recently, a method for computing pairings in Edwards coordinates for supersingular curves was proposed in [11]. The approach proposed in the present paper is very different from [11].

Our starting point concerning Edwards curves is a generalized result of Bernstein and Lange [3]. They showed that an elliptic curve defined over a field K of characteristic different from 2 is birationally equivalent over some extension of K to an Edwards curve, i.e. a curve of the form $x^2 + y^2 = 1 + dx^2y^2$ with $d \notin \{0, 1\}$. A simple and symmetric addition law can be defined on such a curve:

$$(x_1, y_1), (x_2, y_2) \to \left(\frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2} \right). \quad (1)$$

Bernstein and Lange showed that this addition law is, in fact, the standard addition law on the corresponding elliptic curve and gave explicit formulae for additions and doublings, which are faster than all previously known formulae.

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 400–413, 2008.

The basic algorithm used in pairing computation was first described by Miller and is an extension of the double-and-add method for finding a point multiple. Our goal in this paper is to extend Miller's algorithm to allow computation of pairings on curves given in Edwards coordinates. The difficulty when trying to express Miller's algorithm in Edwards coordinates is that it is hard to find the equations of rational functions that need to be evaluated at each addition step. On a curve in Weierstrass form, these equations correspond to straight lines. For curves in Edwards form matters are more complex.

Our main idea is to describe a map of degree 4 from the Edwards curve to a curve $E_{s,p} : s^2p = (1 + dp)^2 - 4p$. This curve has an equation of total degree 3 and as in the Weierstrass case, we can easily compute the equations of the two lines that appear naturally when adding two points P_1 and P_2, i.e. the line l passing through P_1 and P_2 and the vertical line v that passes through $P_1 + P_2$. We then pullback l and v to the Edwards curve. The output of our algorithm is essentially the desired pairing. More precisely, we obtain the 4-th power of the usual pairing.

The remainder of this paper is organised as follows: Section 2 recalls basic properties of Edwards curves and of the Edwards addition law. It also presents Miller's algorithm on an elliptic curve given by a Weierstrass equation. Section 3 introduces the curve $E_{s,p}$ and explains how to compute pairings on Edwards curves by using this representation. Finally, in section 4 we give estimates of the computational cost of the Tate pairing in Edwards coordinates and compare this cost to that of a pairing implementation in Jacobian coordinates (for a Weierstrass equation). We only treat the case of curves with even embedding degree k, which is prefered in most of the cryptographic applications. For benchmark purposes, we compare the efficiency of our suggestion to the use of Jacobian coordinates, which is, to the best of our knowledge, the fastest existing method for computing pairings. A proposal for the operation count in Jacobian coordinates using recent formulas from [2] is presented in an extended version of this paper [18]. In the case of supersingular curves, we also compare our method to results obtained for supersingular curves in [11].

2 Preliminaries

2.1 Edwards Coordinates

Edwards showed in [12] that every elliptic curve E defined over an algebraic number field K is birationally equivalent over some extension of K to a curve given by the equation:

$$x^2 + y^2 = c^2(1 + x^2y^2). \tag{2}$$

In this paper, we make use of the results concerning elliptic curves over finite fields obtained by Bernstein et al. [1]:

Theorem 1. Fix a finite field $\mathbb{F}_q$ with $\mathrm{char}(\mathbb{F}_q) \neq 2$. Let E be an elliptic curve over $\mathbb{F}_q$. E is birationally equivalent over $\mathbb{F}_q$ to a curve $x^2 + y^2 = 1 + dx^2y^2$ if and only if the group $E(\mathbb{F}_q)$ has an element of order 4.

In the sequel, we call the curve $x^2 + y^2 = 1 + dx^2y^2$ an Edwards curve. It was shown in [3] that an Edwards curve E is birationally equivalent to the elliptic curve $E_d : (1/(1-d))v^2 = u^3 + 2((1+d)/(1-d))u^2 + u$ via the rational map:

$$\begin{aligned} \psi : \quad E_d &\to E \\ (u, v) &\to \left(\frac{2u}{v}, \frac{(u-1)}{(u+1)} \right). \end{aligned} \tag{3}$$

On an Edwards curve, we consider the following addition law:

$$(x_1, y_1), (x_2, y_2) \to \left(\frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2} \right). \tag{4}$$

In [3], it was shown that this addition law corresponds to the standard addition law on the birationally equivalent elliptic curve and that the Edwards addition law is *complete* when d is not a square. This means it is defined for all pairs of input points on the Edwards curve with no exceptions for doublings, neutral element, ...

The neutral element of this addition law is $O = (0, 1)$. For every point $P = (x, y)$ the opposite element is $-P = (-x, y)$. The curve has a 4-torsion subgroup defined over $\mathbb{F}_q$. We note $T_2 = (0, -1)$ the point of order 2 and $T_4 = (1, 0)$, $-T_4 = (-1, 0)$ the two points of order 4.

In the following sections we use projective coordinates. A projective point (X, Y, Z) satisfying $(X^2 + Y^2)Z^2 = Z^4 + dX^2Y^2$ and $Z \neq 0$ corresponds to the affine point $(X/Z, Y/Z)$ on the curve $x^2 + y^2 = 1 + dx^2y^2$. The Edwards curve has two points at infinity $(0 : 1 : 0)$ and $(1 : 0 : 0)$. These points are actually singularities of the curve and, as stated in [3], resolving them produces four points defined over $\mathbb{F}_q(\sqrt{d})$. If d is not a square in $\mathbb{F}_q$ then this is a quadratic extension of $\mathbb{F}_q$.

Edwards curves became interesting for elliptic curve cryptography when it was proven by Bernstein and Lange in [3] that they provide addition and doubling formulae faster than all addition formulae known at that time. Table 1 below gives a cost comparison between operations of addition, doubling and mixed addition (i.e. the Z-coordinate of one of the two points is 1) on the Edwards curve and on the Weierstrass form in Jacobian coordinates. These results are taken from [2]. We remind the reader that a point (X, Y, Z) in Jacobian coordinates corresponds to the affine point (x, y) with $x = X/Z^2$ and $y = Y/Z^3$. We note

Table 1. Performance evaluation: Edwards versus Jacobian

	Edwards coordinates	Jacobian coordinates
addition	10**M**+1**S**	11**M**+5**S** (plus **S**-**M** tradeoff)
doubling	3**M**+4**S**	1**M**+8**S** (plus 2 **S**-**M** tradeoffs) or 3**M**+5**S** for $a = -3$
mixed addition	9**M**+1**S**	7**M**+4**S** (plus **S**-**M** tradeoff)

by **M** the cost of a field multiplication and by **S** the cost of a field squaring. We assume that the cost of addition and that of multiplication by d are negligible (we choose d a small constant).

2.2 Background on Pairings

In this section we give a brief overview of the definition of the Tate pairing and of Miller's algorithm [21] used in pairing computations. This algorithm heavily relies on the double and add method for finding a point multiple. Let E be an elliptic curve given by a Weierstrass equation:

$$y^2 = x^3 + ax + b, \tag{5}$$

defined over a finite field $\mathbb{F}_q$. Consider r a large prime dividing $\#E(\mathbb{F}_q)$ and k the corresponding embedding degree, i.e. the smallest positive integer such that r divides $q^k - 1$.

Let P be a r-torsion point and for every integer i, denote by $f_{i,P}$ the function with divisor $\mathrm{div}\,(f_{i,P}) = i(P) - (iP) - (i-1)(O)$ (see [22] for an introduction to divisors). Note $f_{r,P}$ is such that $\mathrm{div}\,(f_{r,P}) = r(P) - r(O)$.

In order to define the Tate pairing we take Q an element of $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$. Let T be a point on the curve such that the support of the divisor $D = (Q + T) - (T)$ is disjoint from the one of $f_{r,P}$. We then define the Tate pairing as:

$$t_r(P, Q) = f_{r,P}(D). \tag{6}$$

This value is a representative of an element of $\mathbb{F}^*_{q^k}/(\mathbb{F}^*_{q^k})^r$. However for cryptographic protocols it is essential to have a unique representative so we will raise it to the $((q^k - 1)/r)$-th power, obtaining an r-root of unity. We call the resulting value the *reduced* Tate pairing:

$$T_r(P, Q) = t_r(P, Q)^{\frac{q^k-1}{r}}.$$

As stated in [14] if the function $f_{r,P}$ is normalized, i.e. $(u_0^r f_{r,P})(O) = 1$ for some $\mathbb{F}_q$-rational uniformizer u_O at O, then one can ignore the point T and compute the pairing as:

$$T_r(P, Q) = f_{r,P}(Q)^{(q^k-1)/r}.$$

In the sequel of this paper we only consider normalized functions. Before going into the details of Miller's algorithm, we recall the standard addition law on an elliptic curve of Weierstrass equation. Suppose we want to compute the sum of iP and jP for $i, j \geq 1$. Let l be the line through iP and jP. Then l intersects the cubic curve E at one further point R according to Bezout's theorem (see [16]). We take v the line between R and O (which is a vertical line when R is not O). Then v intersects E at one more point which is defined to be the sum of iP and jP, that is $(i + j)P$.

The lines l and v are functions on the curve and the corresponding divisors are:

$$\mathrm{div}\,(l) = (iP) + (jP) + (R) - 3(O),$$
$$\mathrm{div}\,(v) = (R) + ((i + j)P) - 2(O).$$

One can then easily check the following relation:

$$f_{i+j,P} = f_{i,P} f_{j,P} \frac{l}{v}. \tag{7}$$

In the sequel, we will call this relation *Miller's equation*. Turning back to Miller's algorithm, suppose we want to compute $f_{r,P}(D)$. We compute at each step of the algorithm on one side $[m]P$, where m is the integer with binary expansion given by the i topmost bits of the binary expansion of r, and on the other side $f_{m,P}$ evaluated at D, by exploiting the formula above. We call the set of operations executed for each bit i of r a *Miller operation*.

Algorithm 1. Miller's algorithm

INPUT: An elliptic curve E defined over a finite field $\mathbb{F}_q$, P an r-torsion point on the curve and $Q \in E(\mathbb{F}_{q^k})$.
OUTPUT: the Tate pairing $t_r(P, Q)$.
Let $i = [\log_2(r)]$, $K \leftarrow P$,$f \leftarrow 1$.
while $i \geq 1$ **do**
  Compute equations of l and v arising in the doubling of K.
  $K \leftarrow 2K$ and $f \leftarrow f^2 l(Q)/v(Q)$.
  if the i-th bit of r is 1 **then**
    Compute equations of l and v arising in the addition of K and P.
    $K \leftarrow P + K$ and $f \leftarrow f l(Q)/v(Q)$.
  end if
  Let $i \leftarrow i - 1$.
end while

The advantage of dealing with the Weierstrass form when running the algorithm is that the equations of l and v are easy to find as they already appear in the addition process. This is obviously not the case with the Edwards curve, whose equation has degree 4. It is difficult to describe the equation of a function with divisor equal to $\mathrm{div}(f_{i+j,P}/f_{i,P}f_{j,P})$ and to establish a relation of type (7). An idea would be to consider Miller's equation on the birationally equivalent Weierstrass curve and then transport this equation on the Edwards curve. However this yields an unefficient pairing computation. Our proposal is to map the Edwards curve to another genus 1 curve with an equation of degree 3, get l and v as straight lines and then pull them back to the Edwards curve.

3 Pairings on Edwards Curves

In this section, E denotes an Edwards curve defined over some finite field $\mathbb{F}_{q^k}$ of odd characteristic. Let us take a look at the action of the 4-torsion subgroup defined over $\mathbb{F}_{q^k}$ on a fixed point on the Edwards curve $P = (x, y)$, with $xy \neq 0$. A simple computation shows that $P + T_4 = (y, -x)$, $P + T_2 = (-x, -y)$ and $P - T_4 = (-y, x)$. We notice then that by letting $p = (xy)^2$ and $s = x/y - y/x$ we characterize the point P up to an addition with a 4-torsion point. This leads

us to consider the following morphism from the Edwards curve to a curve $E_{s,p}$ given by a $s^2p = (1+dp)^2 - 4p$ equation:

$$\begin{aligned} \phi : E \quad &\to E_{s,p} \\ (x,y) &\to ((xy)^2, \frac{x}{y} - \frac{y}{x}). \end{aligned}$$

In this section we study the arithmetic of the curve $E_{s,p}$, establish Miller's equation on this curve and then take its pullback, getting Miller's equation, this time on the Edwards curve. This yields all the tools needed to apply Miller's algorithm on the Edwards curve.

3.1 Arithmetic of the Curve $s^2p = (1+dp)^2 - 4p$

In this section we study the arithmetic of the curve:

$$E_{s,p} : s^2p = (1+dp)^2 - 4p.$$

The equation of $E_{s,p}$ in homogeneous coordinates (P,S,Z) is given by $S^2P = (Z+dP)^2Z - 4PZ^2$. If we dehomogenize this equation by putting $P = 1$ we get the Weierstrass equation of an elliptic curve:

$$s^2 = z^3 + (2d-4)z^2 + d^2z. \tag{8}$$

We note $O_{s,p} = (0,1,0)$ the point at infinity and $T_{2,s,p} = (1,0,0)$ which is a two torsion point. The following definition is simply another way to write the addition law on an elliptic curve in (p,s) coordinates.

Definition 1. *Let $P_1, P_2 \in E_{s,p}$, L the line connecting P_1 and P_2 (tangent line to $E_{s,p}$ if $P_1 = P_2$), and R the third point of intersection of L with E. Let L' be the vertical line through R (of equation $p = p_R$). Then $P_1 + P_2$ is the point such that L' intersects $E_{s,p}$ at R and $P_1 + P_2$ (the point symmetric to R with respect to the p-axis).*

We now show that this addition law corresponds to the addition law induced by the Edwards addition law via the map ϕ.

Theorem 2. *Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on the Edwards curve and P_3 their sum. Then $\phi(P_3)$ is the sum of $\phi(P_1)$ and $\phi(P_2)$ in the addition law of Definition 1.*

Proof. Consider $\psi : E_d \to E$ the map defined in equation (3). By using proposition 2.1 in [22] one can easily see that $\phi \circ \psi$ is a morphism from E_d to the elliptic curve $E_{s,p}$. As $\phi \circ \psi(O_{E_d}) = O_{s,p}$ (where O_{E_d} is the point at infinity of E_d), we deduce that $\phi \circ \psi$ is an isogeny. Moreover it was shown in Theorem 3.2 of [3] that the Edwards addition law on E is the same as the addition law induced by ψ. It follows that the addition law induced by ϕ is the same as the standard addition law on the elliptic curve, so it corresponds to the addition law described at Definition 1. □

As in the sequel we need to compute the pullback of certain functions on the curve $E_{s,p}$ we now compute the degree of this map.

Proposition 1. *The map $\phi : E \to E_{s,p}$ is separable of degree 4.*

Proof. Let $P = (x, y)$ be a point on the Edwards curve. The doubling formula gives:

$$2P = \left(\frac{2xy}{1 + d(xy)^2}, \frac{y^2 - x^2}{1 - d(xy)^2}\right) = \left(\frac{2xy}{x^2 + y^2}, \frac{y^2 - x^2}{2 - (x^2 + y^2)}\right).$$

If $xy \neq 0$ then by letting $p = (xy)^2$ and $s = x/y - y/x$ we can write:

$$4P = \left(\frac{4ps(1 - d^2p^2)}{(1 - d^2p^2)^2 - 4dp^2s^2}, \frac{4p(1 + dp)^2 - ps^2}{(1 - d^2p^2)^2 + 4dp^2s^2}\right).$$

This means that by defining:

$$\begin{aligned} \psi : \quad & E_{s,p} \to E \\ & (p, s) \to \left(\frac{4ps(1 - d^2p^2)}{(1 - d^2p^2)^2 - 4dp^2s^2}, \frac{4p(1 + dp)^2 - ps^2}{(1 - d^2p^2)^2 + 4dp^2s^2}\right), \end{aligned}$$

we get a rational map ψ such that $\phi \circ \psi = [4]$ on E. It follows that $\deg \phi$ divides 16. As the inseparable degree $\deg_i \phi$ is a power of the characteristic of $\mathbb{F}_q$, we deduce that ϕ is a separable map (we have supposed that $\mathrm{char}(\mathbb{F}_q) \neq 2$). By putting $\phi(P) = Q$ we easily get $\phi^{-1}(Q) = \{P, P + T_2, P + T_4, P - T_4\}$. We conclude that $\deg \phi = 4$. □

3.2 Miller's Algorithm on the Edwards Curve

Let P be a r-torsion point on the Edwards curve. We consider slightly modified functions $f_{i,P}^{(4)}$:

$$\begin{aligned} f_{i,P}^{(4)} = i((P) + (P + T_4) + (P + T_2) + (P - T_4)) - ((iP) + (iP + T_4) \\ +(iP + T_2) + (iP - T_4)) - (i - 1)((O) + (T_4) + (T_2) + (-T_4)). \end{aligned}$$

Then $f_{r,P}^{(4)} = r((P)+(P+T_4)+(P+T_2)+(P-T_4))-r((O)+(T_4)+(T_2)+(-T_4))$, which means that we can compute the Tate pairing up to a 4-th power:

$$T_r(P, Q)^4 = f_{r,P}^{(4)}(Q)^{\frac{q^k-1}{r}}.$$

We also get the following Miller equation:

$$f_{i+j,P}^{(4)} = f_{i,P}^{(4)} f_{j,P}^{(4)} \frac{l}{v}, \tag{9}$$

where l/v is the function of divisor:

$$\begin{aligned} \mathrm{div}\,(l/v) = & ((iP) + (iP + T_4) + (iP + T_2) + (iP - T_4)) \\ & +((jP) + (jP + T_4) + (jP + T_2) + (jP - T_4)) \\ & -(((i + j)P) + ((i + j)P + T_4) + ((i + j)P + T_2) + ((i + j)P - T_4))) \\ & -((O) + (T_4) + (T_2) + (-T_4)). \end{aligned}$$

Let $P' = \phi(P)$ and let $l_{s,p}$ and $v_{s,p}$ be functions on the $E_{s,p}$ curve such that div $(l_{s,p}) = (iP') + (jP') + (-(i+j)P') - 2(T_{2,s,p}) - (O_{s,p})$ and div $(v_{s,p}) = ((i+j)P') + (-(i+j)P') - 2(T_{2,s,p})$.

We observe that we have $l/v = \phi^*(l_{s,p}/v_{s,p})$ up to constants in $\mathbb{F}_q$. It is easy to find the equations of $l_{s,p}$ and $v_{s,p}$ as they appear naturally in the definition of the sum $iP' + jP'$, namely $l_{s,p}$ is the line connecting iP' and jP', and $v_{s,p}$ is the vertical line through $(i+j)P'$. As we will see in the next section, we can compute their pullback via the map ϕ without any significant computational cost.

4 Pairing Computation in Edwards Coordinates

In this section, we take a look into the details of the computation of pairings in Edwards coordinates and give estimates of the computational costs of the Miller operation. We start by estimating the cost of evaluating the function $f^{(4)}_{r,P}(Q)$ in terms of the cost of the doubling part of a Miller operation, which is executed for every bit of r. This seems reasonable, as it gives an evaluation which is independent from any fast exponentiation techniques that might be used in the implementation of the algorithm, such as the sliding window method or the use of a signed Hamming weight representation for r. We recall that a signed representation $(m_{n-1}...m_0)_s$ is said to be in *non-adjacent form*, or NAF for short, if $m_i m_{i+1} = 0$, with $m_i \in \{-1, 0, 1\}$. The advantage of using such a representation is that on average the number of non-zero terms in a NAF expansion of length n is $n/3$ (see [8] for a precise analysis of the NAF density).

Moreover, in many cryptographic applications it is possible to choose r with low Hamming weight. The construction of Cocks and Pinch as described in [4, p. 210] allows for r to be chosen arbitrarily, so a prime of low Hamming weight can be chosen. Further examples are provided by a construction of Brezing and Weng [9] for prime embeddings degrees k, extended in [13] for all odd $k < 200$. Note that if the loop length parameter r does not have low Hamming weight, it is sufficient to have some multiple of r whose Hamming weight is small (usually the elliptic curve group order $\sharp E(F_q)$).

Example 1. The following example is given in [11]. Consider $E : y^2 = x^3 + x$ over $\mathbb{F}_q$, with $q \equiv 3 \bmod 4$. This curve is supersingular and its corresponding Edwards form is $x^2 + y^2 = 1 - (xy)^2$, so $d = -1$. One may choose for instance $q = 2^{520} + 2^{363} - 2^{360} - 1$, $r = 2^{160} + 2^3 - 1$ or $q = 2^{1582} + 2^{1551} - 2^{1326} - 1, r = 2^{256} + 2^{225} - 1$.

During the past few years, research in pairing-based cryptography focussed on the reduction of the loop length in Miller's algorithm. It was proven that for curves with a small Frobenius trace t, the parameter r giving the length of the loop can be replaced by t. The interested reader should refer to [17] for a discussion on the choice of the loop length parameter.

Therefore, in order to give a complete evaluation of the complexity, we also count the number of operations in the mixed addition step of the Miller operation and compare it to the mixed addition step in Jacobian coordinates. The

reader may refer to [10] for global estimates of pairing computation in Jacobian coordinates for some families of curves with $k = 2$ (in particular those in Example 1). While estimates for the doubling part for the Weiestrass form can be found in [20] and in [15], we were not able to find in the literature estimates of the cost of the mixed addition step for Jacobian coordinates in a general context. We propose in [18] a detailed computation of a full Miller operation (doubling and mixed addition), which makes use of recent formulas for Jacobian coordinates [2].

The remaining of this paper presents efficient computation of the Tate pairing in Edwards coordinates for curves with even embedding degree. These curves are preferred in cryptographic applications because a major part of the computations is performed in a proper subfield of $\mathbb{F}_{q^k}$. However, the results obtained in section 3 are independent of the embedding degree, so similar computations are to be done in a general case.

4.1 The Case of an Even Embedding Degree

Koblitz and Menezes showed in [20] that if q and k are chosen such as $p \equiv 1 \pmod{12}$ and $k = 2^i 3^j$, then the arithmetic of the extension field $\mathbb{F}_{q^k}$ can be implemented very efficiently as this field can be built up as a tower of extension fields:

$$\mathbb{F}_q \subset \mathbb{F}_{q^{d_1}} \subset \mathbb{F}_{q^{d_2}} ... \subset \mathbb{F}_{q^k},$$

where the ith field $F_{q^{d_i}}$ is obtained by adjoining a root of some irreducible polynomial $X^{d_i/d_{i-1}} - \beta_i$ and $d_i/d_{i-1} \in \{2, 3\}$.

We note by $\mathbf{m}, \mathbf{M}$ (respectively $\mathbf{s}, \mathbf{S}$) the costs of multiplications (respectively squarings) in the field $\mathbb{F}_q$ and in the extension $\mathbb{F}_{q^k}$. Then according to [20] we get

$$\mathbf{M} \approx v(k)\mathbf{m} \text{ and } \mathbf{S} \approx v(k)\mathbf{s},$$

where $v(k) = 3^i 5^j$. Moreover, a multiplication of an element in $\mathbb{F}_{q^k}$ by an element in $\mathbb{F}_q$ costs $k\mathbf{m}$ operations.

In most cryptographic protocols there is some flexibility in the choice of the order r subgroups generated by P and Q. P can be chosen such that $< P >$ is the unique subgroup of order r in $E(\mathbb{F}_q)$. Moreover, if the embedding degree is even, it was shown that the subgroup $< Q > \subset E(\mathbb{F}_{q^k})$ can be taken so that the x-coordinates of all its points lie in $\mathbb{F}_{q^{k/2}}$ and the y-coordinates are products of elements of $\mathbb{F}_{q^{k/2}}$ with $\sqrt{\beta}$, where β is a nonsquare in $\mathbb{F}_{q^{k/2}}$ and $\sqrt{\beta}$ is a fixed squareroot in $\mathbb{F}_{q^k}$ (see [20] for details). The same kind of considerations apply to Edwards curves. To do this we need to take a look at the birational map that transforms a curve given by a Weierstrass equation into a curve given by an Edwards equation. As stated in Section 2.1 the curve $x^2 + y^2 = 1 + dx^2y^2$ is birationally equivalent to the curve E_d, via the rational map $\psi : E_d \to E$. By looking at the shape of this map, it follows that in the case of an even embedding degree, the coordinates of elements of $< P >$ can be chosen in $\mathbb{F}_q$. The subgroup $< Q > \in F_{q^k}$ can be chosen such that its elements have y-coordinates in the

quadratic subextension $F_{q^{k/2}}$ and x-coordinates that can be written as products of elements of $F_{q^{k/2}}$ with some squareroot of a nonsquare element β of $F_{q^{k/2}}$.

We now take a look into the details of the computation of a Miller iteration. We note $K = (X_1, Y_1, Z_1)$. Following [3] the doubling formulas for $2K = (X_3, Y_3, Z_3)$ are:

$$\begin{aligned} X_3 &= 2X_1Y_1(2Z_1^2 - (X_1^2 + Y_1^2)), \\ Y_3 &= (X_1^2 + Y_1^2)(Y_1^2 - X_1^2), \\ Z_3 &= (X_1^2 + Y_1^2)(2Z_1^2 - (X_1^2 + Y_1^2)). \end{aligned}$$

On the curve $E_{s,p}$ we consider $l_{s,p}$ the tangent line to the curve at $\phi(K) = (p_1, s_1)$ and $v_{s,p}$ the vertical line passing through $\phi(2K) = (p_3, s_3)$. These lines have the following equations:

$$\begin{aligned} l_{s,p}(s,p) &= 2p_1^2 s_1(s - s_1) - p_1(2d(1 + dp_1) - (s_1^2 + 4))(p - p_1), \\ v_{s,p}(s,p) &= p - p_3. \end{aligned}$$

Consequently we get the following evaluations of l and v at point $Q = (x, y)$ on the Edwards curve:

$$\begin{aligned} l(x,y) = l_1(x,y)/l_2 = ((X_1^2 + Y_1^2 - Z_1^2)(X_1^2 - Y_1^2)((2X_1Y_1(x/y - y/x) \\ -2(X_1^2 - Y_1^2)) - Z_3(dZ_1^2(xy)^2 - (X_1^2 + Y_1^2 - Z_1^2)))/ \\ (2X_1Y_1(X_1^2 + Y_1^2 - Z_1^2)(X_1^2 - Y_1^2)), \\ v(x,y) = v_1(x,y)/v_2 = (dZ_3^2(xy)^2 - (X_3^2 + Y_3^2 - Z_3^2))/(X_3^2 + Y_3^2 - Z_3^2). \end{aligned}$$

We now show that the computational cost of the doubling part in Miller's algorithm is significantly lower because we can ignore terms that lie in a proper subfield of $\mathbb{F}_{q^k}$. These terms can be ignored because k is the multiplicative order of q modulo r, so $(q^k - 1)/r$ is a multiple of $q^{k'} - 1$ for some proper divisor k' of k. So we ignore l_2 and v_2 because they depend only of the coordinates of P, so they lie in $\mathbb{F}_q$. Since $(xy)^2 \in \mathbb{F}_{q^{k/2}}$ and hence $v_1(Q) \in \mathbb{F}_{q^{k/2}}$, it follows that we can also ignore $v_1(Q)$. Hence the function evaluation step in the doubling part of Miller's algorithm becomes:

$$f_1 \leftarrow f_1^2 l_1(Q). \tag{10}$$

Note that multiplications by $(xy)^2$ and $x/y - y/x$ cost $k/2\mathbf{m}$ ($x/y - y/x$ is the product of some element in $\mathbb{F}_{q^{k/2}}$ with $\sqrt{\beta}$). Also note that computing $x/y - y/x$ costs one inversion in $\mathbb{F}_{q^{k/2}}$. In some protocols Q is a fixed point, so we can precompute $x/y - y/x$.

If $k = 2$, we actually have $(xy)^2 \in \mathbb{F}_q$, so we compute:

$$\begin{aligned} l_1(x,y) = ((X_1^2 + Y_1^2 - Z_1^2)(X_1^2 - Y_1^2)) \cdot 2XY(x/y - y/x) - ((X_1^2 + Y_1^2 - Z_1^2) \\ \cdot(X_1^2 - Y_1^2)) \cdot 2(X_1^2 - Y_1^2) - Z_3 \cdot (dZ_1^2 \cdot (xy)^2 - (X_1^2 + Y_1^2 - Z_1^2)), \end{aligned}$$

Table 2. Operations of the doubling part of the Miller operation for $k > 2$

$A \leftarrow X_1^2,\ B \leftarrow Y_1^2,\ C \leftarrow (X_1 + Y_1)^2,\ D \leftarrow A + B,$	$(3\mathbf{s})$
$E \leftarrow C - D,\ F \leftarrow B - A,\ G \leftarrow Z_1^2,\ H \leftarrow 2G - D,$	$(1\mathbf{s})$
$X_3 \leftarrow E \cdot H,\ Y_3 \leftarrow D \cdot F,\ Z_3 \leftarrow D \cdot H,\ I \leftarrow G \cdot F,\ J \leftarrow (I - Y_3) \cdot C$	$(5\mathbf{m})$
$K \leftarrow J \cdot (x/y - y/x),\ L \leftarrow (I - Y_3) \cdot 2F,\ M \leftarrow Z_3 \cdot dG$	$((2 + \frac{k}{2})\mathbf{m})$
$N \leftarrow M \cdot (xy)^2,\ P \leftarrow Z_3 \cdot (A + B - G),\ l_1 \leftarrow K + L - N + P$	$((1\mathbf{m} + \frac{k}{2})\mathbf{m})$
$f_1 \leftarrow f_1^2 l_1$	$(1\mathbf{M}{+}1\mathbf{S})$

Table 3. Comparison of costs for the doubling step of the Miller operation in the case of k even

	$k = 2$	$k \geq 4$
Jacobian coordinates	$10\mathbf{s} + 3\mathbf{m} + \mathbf{S} + \mathbf{M}$	$11\mathbf{s} + (k+1)\mathbf{m} + \mathbf{S} + \mathbf{M}$
Jacobian coordinates for $a = -3$	$4\mathbf{s} + 8\mathbf{m} + \mathbf{S} + \mathbf{M}$	$4\mathbf{s} + (k+7)\mathbf{m} + \mathbf{S} + \mathbf{M}$
Das/Sarkar Edwards coordinates (supersingular curves)	$6\mathbf{s} + 9\mathbf{m} + \mathbf{S} + \mathbf{M}$	-
Edwards coordinates	$4\mathbf{s} + 9\mathbf{m} + \mathbf{S} + \mathbf{M}$	$4\mathbf{s} + (k+8)\mathbf{m} + \mathbf{S} + \mathbf{M}$

For $k > 2$ some operations are done in $\mathbb{F}_{q^k}$ and others in $\mathbb{F}_q$, so we compute l_1 as it follows:

$$l_1(x,y) = ((X_1^2 + Y_1^2 - Z_1^2)(X_1^2 - Y_1^2)) \cdot 2X_1Y_1(x/y - y/x) - ((X_1^2 + Y_1^2 - Z_1^2) \cdot (X_1^2 - Y_1^2)) \cdot 2(X_1^2 - Y_1^2) - Z_3 \cdot dZ_1^2 \cdot (xy)^2 + Z_3 \cdot (X_1^2 + Y_1^2 - Z_1^2),$$

As an example, we detail the computation of the doubling step for $k > 2$ in Table 2. Results are summarized in Table 3. The computations in the general case for Jacobian coordinates are detailed in [18], while the '$a = -3$' case is taken directly from [20], although some further $\mathbf{s} - \mathbf{m}$ tradeoffs might be possible.

Next, we take a look at the mixed addition step in a Miller iteration. We first count the number of operations that must be executed when adding $K = (X_1, Y_1, Z_1)$ and $P = (X_0, Y_0, 1)$. The result is $K + P = (X_3, Y_3, Z_3)$ with:

$$\begin{aligned} X_3 &= Z_1(X_0Y_1 + Y_0X_1)(Z_1^2 + dX_0X_1Y_0Y_1), \\ Y_3 &= Z_1(Y_0Y_1 - X_0X_1)(Z_1^2 - dX_0X_1Y_0Y_1), \\ Z_3 &= (Z_1^2 + dX_0X_1Y_0Y_1)(Z_1^2 - dX_0X_1Y_0Y_1). \end{aligned}$$

On the $E_{s,p}$, we consider $l_{s,p}$ the straight line passing through $\phi(K) = (p_1, s_1)$ and $\phi(P) = (p_0, s_0)$ and $v_{s,p}$ the vertical line passing through the point $\phi(K) + \phi(P) = (p_3, s_3)$. We get:

$$\begin{aligned} l_{s,p}(s,p) &= (p_0 - p_1)(s - s_1) - (s_0 - s_1)(p - p_1); \\ v_{s,p}(s,p) &= p - p_3. \end{aligned}$$

Table 4. Operations of the mixed addition step of a Miller operation for $k > 2$

$A \leftarrow X_1^2, B \leftarrow Y_1^2, C \leftarrow (X_1 + Y_1)^2 - A - B, D \leftarrow C \cdot (dX_0Y_0)$	(1**m**+3**s**)
$E \leftarrow 2(X_1 + X_0) \cdot (Y_0 + Y_1) - C - 2X_0Y_0,$	(1**m**)
$F \leftarrow 2(X_1 + Y_0) \cdot (Y_1 - X_0) - C + 2X_0Y_0, G \leftarrow Z_1^2$	(1**m**+1**s**)
$X_3 \leftarrow Z_1 \cdot E \cdot (2G + D), Y_3 \leftarrow Z_1 \cdot F \cdot (2G - D), Z_3 \leftarrow (2G - D) \cdot (2G + D)$	(5**m**)
$H \leftarrow dG \cdot (X_0Y_0)^2, I \leftarrow (A + B - G - H) \cdot C,$	(2**m**)
$J \leftarrow I \cdot (x/y - y/x), K \leftarrow 2(A + B - G - H) \cdot (A - B)$	$((1 + \frac{k}{2})$**m**)
$L \leftarrow C \cdot (X_0/Y_0 - Y_0/X_0), M \leftarrow dG \cdot (2A - 2B - L), N \leftarrow M \cdot (xy)^2$	$((2 + \frac{k}{2})$**m**)
$P \leftarrow (2A - 2B - L) \cdot (A + B - G), l_1 \leftarrow J - K - N + P$	(1**m**)
$f_1 \leftarrow f_1 \cdot l_1$	(1**M**)

Consequently, we have the following equations for pullbacks:

$$\begin{aligned} l(x,y) = l_1(x,y)/l_2 = (X_1^2 + Y_1^2 - Z_1^2 - dZ_1^2(X_0Y_0)^2))(X_1Y_1(\frac{x}{y} - \frac{y}{x}) - \\ (X_1^2 - Y_1^2)) - \left(X_1^2 - Y_1^2 - X_1Y_1(\frac{X_0}{Y_0} - \frac{Y_0}{X_0})\right) \\ \cdot(dZ_1^2(xy)^2 - (X_1^2 + Y_1^2 - Z_1^2)) \\ /(X_1Y_1(X_1^2 + Y_1^2 - Z_1^2 - dZ_1^2(X_0Y_0)^2)); \\ v(x,y) = v_1(x,y)/v_2 = (dZ_3^2(xy)^2 - (X_3^2 + Y_3^2 - Z_3^2))/(X_3^2 + Y_3^2 - Z_3^2). \end{aligned}$$

For the same reasons as above, the mixed addition step in the case of even k becomes:

$$f_1 \leftarrow f_1 l_1(Q). \tag{11}$$

Detailed computations of the mixed addition step for $k > 2$ are presented in Table 4. We suppose having computed expressions like X_0Y_0, $(X_0Y_0)^2$, $X_0/Y_0 - Y_0/X_0$ once and for all in the very beginning. As computing $X_0/Y_0 - Y_0/X_0$ costs one inversion in $\mathbb{F}_q$, in some cases it will be less expensive to work with $l_1' = (X_0Y_0)l_1$ instead of l_1.

Results and performance comparison are presented in Table 5.

By looking at tables 3 and 5 one can see that in the case of an even embedding degree the cost of an implementation of Miller's algorithm in Edwards coordinates will be comparable to the cost of an implementation in Jacobian coordinates. We find it important to state that, no matter the representation

Table 5. Comparison of costs for the mixed addition step of the Miller operation in the case of k even

	$k = 2$	$k \geq 4$
Jacobian coordinates	$3\mathbf{s} + 14\mathbf{m} + \mathbf{M}$	$3\mathbf{s} + (k + 13)\mathbf{m} + 1\mathbf{M}$
Das/Sarkar Edwards coordinates (supersingular curves)	$1\mathbf{s} + 18\mathbf{m} + \mathbf{M}$	-
Edwards coordinates	$4\mathbf{s} + 15\mathbf{m} + \mathbf{M}$	$4\mathbf{s} + (k + 14)\mathbf{m} + 1\mathbf{M}$

one might choose to implement Miller's algorithm in high embedding degrees, it would be impossible to avoid the costly computation of $1\mathbf{M}+1\mathbf{S}$ in equation (10) or the $1\mathbf{M}$ in equation (11), as these are the updates in the Miller loop.

5 Conclusion

In this paper, we have given a new algorithm to compute pairings on Edwards curves and compared its performance to that of an implementation of Miller's algorithm in Jacobian coordinates and to the method for Edwards coordinates from [11]. We showed that this algorithm is competitive and that the presented approach is faster than previously known methods for computing pairings on Edwards curves.

References

1. Bernstein, D.J., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted Edwards curves. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 389–405. Springer, Heidelberg (2008)
2. Bernstein, D.J., Lange, T.: Explicit-formulas database (2007), http://hyperelliptic.org/EFD/
3. Bernstein, D.J., Lange, T.: Faster addition and doubling on elliptic curves. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 29–50. Springer, Heidelberg (2007)
4. Blake, I.F., Seroussi, G., Smart, N.P.: Advances in Elliptic Curve Cryptography. London Mathematical Society Lecture Note Series, vol. 317. Cambridge University Press, Cambridge (2005)
5. Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
6. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001)
7. Boneh, D., Shacham, H.: Group signatures with verifier-local revocation. In: Atluri, V., Pfitzmann, B., McDaniel, P. (eds.) ACM CCS 2004: 11th Conference on Computer and Communications Security, pp. 168–177. ACM Press, New York (2004)
8. Bosma, W.: Signed bits and fast exponentiation. J. de théorie des nombres de Bordeaux 13(1), 27–41 (2001)
9. Brezing, F., Weng, A.: Elliptic curves suitable for pairing based cryptography. Des. Codes Cryptography 37(1), 133–141 (2005)
10. Chatterjee, S., Sarkar, P., Barua, R.: Efficient computation of Tate pairing in projective coordinate over general characteristic fields (2004)
11. Das, M.P.L., Sarkar, P.: Pairing computation on twisted Edwards form elliptic curves. In: Galbraith, S.D., Paterson, K.G. (eds.) Pairing 2008. LNCS, vol. 5209. Springer, Heidelberg (2008)
12. Edwards, H.M.: A normal form for elliptic curves. Bull. AMS 44, 393–422 (2007)
13. Freeman, D., Scott, M., Teske, E.: A taxonomy of pairing-friendly elliptic curves. Cryptology ePrint Archive, Report 2006/372 (2006), http://eprint.iacr.org/

14. Granger, R., Hess, F., Oyono, R., Thériault, N., Vercauteren, F.: Ate pairing on hyperelliptic curves (2007)
15. Granger, R., Page, D., Smart, N.P.: High security pairing-based cryptography revisited. In: Hess, F., Pauli, S., Pohst, M. (eds.) ANTS 2006. LNCS, vol. 4076, pp. 480–494. Springer, Heidelberg (2006)
16. Hartshorne, R.: Algebraic Geometry. Graduate texts in Mathematics, vol. 52. Springer, Heidelberg (1977)
17. Hess, F., Smart, N.P., Vercauteren, F.: The Eta Pairing Revisited. IEEE Transactions on Information Theory 52, 4595–4602 (2006)
18. Ionica, S., Joux, A.: Another approach on pairing computation in Edwards coordinates. Cryptology ePrint Archive, Report 2008/292 (2008), http://eprint.iacr.org/
19. Joux, A.: A one round protocol for tripartite Diffie-Hellman. Journal of Cryptology 17(4), 263–276 (2004)
20. Koblitz, N., Menezes, A.: Pairing-based cryptography at high security levels. In: Smart, N.P. (ed.) Cryptography and Coding 2005. LNCS, vol. 3796, pp. 13–36. Springer, Heidelberg (2005)
21. Miller, V.S.: The Weil pairing, and its efficient calculation. Journal of Cryptology 17(4), 235–261 (2004)
22. Silverman, J.H.: The Arithmetic of Elliptic Curves. Graduate texts in Mathematics, vol. 106. Springer, Heidelberg (1986)

A Verifiable Secret Sharing Scheme Based on the Chinese Remainder Theorem

Kamer Kaya* and Ali Aydın Selçuk

Department of Computer Engineering
Bilkent University
Ankara, 06800, Turkey
{kamer,selcuk}@cs.bilkent.edu.tr

Abstract. In this paper, we investigate how to achieve verifiable secret sharing (VSS) schemes by using the Chinese Remainder Theorem (CRT). We first show that two schemes proposed earlier are not secure by an attack where the dealer is able to distribute inconsistent shares to the users. Then we propose a new VSS scheme based on the CRT and prove its security. Using the proposed VSS scheme, we develop a joint random secret sharing (JRSS) protocol, which, to the best of our knowledge, is the first JRSS protocol based on the CRT.

Keywords: Verifiability, joint random secret sharing, Chinese Remainder Theorem, Asmuth-Bloom secret sharing scheme.

1 Introduction

Threshold cryptography deals with the problem of sharing a highly sensitive secret among a group of users so that only when a sufficient number of them come together can the secret be reconstructed. Well-known secret sharing schemes (SSS) in the literature include Shamir [18] based on polynomial interpolation, Blakley [2] based on hyperplane geometry, and Asmuth-Bloom [1] based on the Chinese Remainder Theorem (CRT).

A t-out-of-n secret sharing scheme contains two phases: In the *dealer phase*, the dealer shares a secret among n users. In the *combiner phase*, a coalition of size greater than or equal to t constructs the secret. We call a SSS *verifiable* if each user can verify the correctness of his share in the dealer phase and no user can lie about his share in the combiner phase. Hence, neither the dealer nor the users can cheat in a VSS scheme. Verifiable secret sharing schemes based on Shamir's SSS have been proposed in the literature [6,15]. These schemes have been extensively studied and used in threshold cryptography and secure multiparty computation [9,14,15].

* Supported by the Turkish Scientific and Technological Research Agency (TÜBİTAK) Ph.D. scholarship.

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 414–425, 2008.

There have been just two CRT-based VSS schemes by Iftene [10] and Qiong et al. [16]. In this paper, we show that these schemes are vulnerable to attacks where a corrupted dealer can distribute *inconsistent* shares without detection such that different coalitions will obtain different values for the secret. To the best of our knowledge, these are the only VSS schemes that have been proposed so far based on the CRT.

A typical application of a VSS scheme is the joint random secret sharing (JRSS) primitive frequently used in threshold cryptography [9,11,14,15]. In a JRSS scheme, all players act as a dealer and jointly generate and share a random secret. So far, there have been no JRSS protocols proposed based on the CRT.

In this paper, we first show why existing attempts for a CRT-based verifiable secret sharing scheme fail by attacks on the existing schemes. We then propose a VSS scheme based on the Asmuth-Bloom secret sharing [1] and using this VSS scheme, we propose a JRSS scheme. To the best of our knowledge the VSS and JRSS schemes we propose are the first secure CRT-based schemes of their kind in the literature.

The rest of the paper is organized as follows: In Section 2, we describe the Asmuth-Bloom SSS in detail and introduce the notation we followed in the paper. The VSS schemes proposed in [10,16] are described Section 3 and their flaws are analyzed. After presenting our VSS scheme in Section 4, we propose the joint random scheme in Section 5. Section 6 concludes the paper.

2 Asmuth-Bloom Secret Sharing Scheme

The Asmuth-Bloom SSS [1] shares a secret d among n parties by modular arithmetic such that any t users can reconstruct the secret by the CRT. The scheme presented in Figure 1 is a slightly modified version by Kaya and Selcuk [12] in order to obtain better security properties.

According to the Chinese Remainder Theorem, y can be determined uniquely in $\mathbb{Z}_{M_S}$ since $y < M \leq M_S$ for any coalition S of size t.

Kaya and Selcuk [12] showed that the Asmuth-Bloom version presented here is *perfect* in the sense that no coalition of size smaller than t can obtain any information about the secret.

Quisquater et al. [17] showed that when m_is are chosen as consecutive primes, the scheme has better security properties. In this paper, we will also assume that all m_is are prime and we will choose them such that $p_i = 2m_i + 1$ is also a prime for $1 \leq i \leq n$. The notation used in the paper is summarized in Table 1.

For the protocols in this paper, we assume that private channels exist between the dealer and users. The share of each user is sent via these private channels; hence no one except the user himself knows the share. Besides, we assume that a broadcast channel exists and if some data is broadcast each user will read the same value. Hence an adversary cannot send two different values to two different users for a broadcast data.

– *Dealer Phase:* To share a secret d among a group of n users, the dealer does the following:
  - A set of relatively prime integers $m_0 < m_1 < \ldots < m_n$ are chosen where m_0 is a prime and
$$\prod_{i=1}^{t} m_i > {m_0}^2 \prod_{i=1}^{t-1} m_{n-i+1}. \quad (1)$$
  - Let M denote $\prod_{i=1}^{t} m_i$. The dealer computes $y = d + Am_0$ where A is a positive integer generated randomly subject to the condition that $0 \leq y < M$.
  - The share of the ith user, $1 \leq i \leq n$, is $y_i = y \bmod m_i$.
– *Combiner Phase:* Let S be a coalition of t users gathered to construct the secret. Let M_S denote $\prod_{i \in S} m_i$.
  - Let $M_{S\setminus\{i\}}$ denote $\prod_{j \in S, j \neq i} m_j$ and $M'_{S,i}$ be the multiplicative inverse of $M_{S\setminus\{i\}}$ in $\mathbb{Z}_{m_i}$, i.e., $M_{S\setminus\{i\}} M'_{S,i} \equiv 1 \pmod{m_i}$. First, the ith user computes
$$u_i = y_i M'_{S,i} M_{S\setminus\{i\}} \bmod M_S.$$
  - The users first compute
$$y = \left(\sum_{i \in S} u_i \right) \bmod M_S$$
and then obtain the secret d by computing
$$d = y \bmod m_0.$$

Fig. 1. Asmuth-Bloom secret sharing scheme

Table 1. Notations

Notation	Explanation
n	The number of users.
t	The threshold, the minimum number of users required to construct the secret.
d	The secret to be shared.
m_0	A prime; specifies the domain of $d \in \mathbb{Z}_{m_0}$.
$m_i : 1 \leq i \leq n$	The prime modulus for user i.
$p_i : 1 \leq i \leq n$	A safe prime, $2m_i + 1$.
P	$\prod_{i=1}^{n} p_i$.
y	$d + Am_0$, where A is a random number.
M	The domain of $y \in \mathbb{Z}_M$.
$y_i : 1 \leq i \leq n$	$y \bmod m_i$, the share of user i.
$E(y)$	The commitment value of an integer y.
S	A coalition of users.
M_S	The modulus of coalition S, $\prod_{i \in S} m_i$.

3 Analysis of the Existing CRT-Based VSS Schemes

There have been two different approaches to achieve VSS by a CRT-based secret sharing scheme. The first one, proposed by Iftene [10], obtains a VSS scheme from Mignotte's SSS [13] which is another CRT-based SSS similar to Asmuth-Bloom. Here, we adapt Iftene's approach to the Asmuth-Bloom SSS. The scheme is given in Figure 2.

If the dealer is honest and the discrete logarithm problem is hard, the scheme in Figure 2 is secure against a dishonest user because the verification data, $g_i{}^y \bmod p_i$, can be used to detect an invalid share from a corrupted user in the first step of the combiner phase.

However, if the dealer is dishonest, he can mount an attack despite the additional verification data above: Let y be an integer and $y_i = y \bmod m_i$ for $1 \leq i \leq n$. In the combiner phase of Asmuth-Bloom SSS, the minimum number of users required to obtain the secret is t; hence, $y = d + Am_0$ must be smaller than $M = \prod_{i=1}^{t} m_i$. Note that, to reconstruct the secret d, each coalition S must first compute $y \bmod M_S$ where $M_S \geq M$. If the dealer distributes the shares for some $y > M$, then y will be greater than M_S for some coalition S of size t. Hence, S may not compute the correct y value and the correct secret d even though $y_i = y \bmod m_i$ for all i. Therefore, the given VSS scheme cannot detect this kind of inconsistent shares from the dealer where different coalitions end up with different d values. The same problem also arises in Iftene's original VSS scheme [10].

- *Dealer Phase:* To share a secret $d \in \mathbb{Z}_{m_0}$ among a group of n users with verifiable shares, the dealer does the following:
  1. Use the dealing procedure of the Asmuth-Bloom SSS to obtain the shares $y_i = y \bmod m_i$ for each $1 \leq i \leq n$ where $y = d + Am_0 < M$. Choose m_is such that each $p_i = 2m_i + 1$ is also a prime.
  2. Let $g_i \in \mathbb{Z}_{p_i}^*$ be an element of order m_i. The dealer sends y_i to the ith user privately and makes the values p_i, g_i and $z_i = g_i^y \bmod p_i$ public for $1 \leq i \leq n$. The ith user can find whether his share is valid or not by checking
$$z_i \stackrel{?}{\equiv} g_i{}^{y_i} \bmod p_i. \tag{2}$$
- *Combiner Phase:* Let S be a coalition gathered to construct the secret.
  1. The share y_i of user $i \in S$ can be verified by the other users in S by the verification equation $z_i \stackrel{?}{\equiv} g_i{}^{y_i} \bmod p_i$.
  2. If all shares are valid then the coalition S can obtain the secret d: First, the ith user computes
$$u_i = y_i M'_{S,i} M_{S\setminus\{i\}} \bmod M_S.$$
  3. Then the users compute $y = \left(\sum_{i \in S} u_i\right) \bmod M_S$ and obtain the secret d by computing $d = y \bmod m_0$.

Fig. 2. Iftene's CRT-based VSS extension

– *Dealer Phase:* To share a secret $d \in \mathbb{Z}_{m_0}$ among a group of n users with verifiable shares, the dealer does the following:
  1. Use the dealing procedure of the Asmuth-Bloom SSS to obtain the shares $y_i = y \bmod m_i$ for all $1 \leq i \leq n$ where $y = d + Am_0 < M$.
  2. Let p, q be primes such that $q|(p-1)$. Construct the unique polynomial $f(x) \in \mathbb{Z}_q[x]$ where $deg(f(x)) = n-1$ and $f(m_i) = y_i$. Construct a random polynomial $f'(x) \in \mathbb{Z}_q[x]$ where $deg(f'(x)) = n-1$. Let $z_i = f'(m_i)$ for all $1 \leq i \leq n$.
  3. Let $g \in \mathbb{Z}_p$ with order q, h be a random integer in the group generated by g and $E(a,b) = g^a h^b \bmod p$ for inputs $a, b \in \mathbb{Z}_q^*$. Compute
  $$E_i = E(f_i, f_i') = g^{f_i} h^{f_i'} \bmod p,$$
  where f_i and f_i' are the $(i-1)$th coefficients of $f(x)$ and $f'(x)$, respectively, for all $1 \leq i \leq n$. Broadcast E_is to all users.
  4. Send (y_i, z_i) secretly to the ith user for all $1 \leq i \leq n$.
  5. To verify the validity of his share, each user checks
  $$E(y_i, z_i) \stackrel{?}{\equiv} \prod_{j=1}^{n} {E_j}^{{m_i}^{j-1}} \equiv \prod_{j=1}^{n} g^{f_j {m_i}^{j-1}} \prod_{j=1}^{n} h^{f_j' {m_i}^{j-1}} \equiv g^{y_i} h^{z_i} \bmod p. \quad (3)$$

– *Combiner Phase:* Let S be a coalition gathered to construct the secret.
  1. The share (y_i, z_i) of user $i \in S$ can be verified by the other users in S with the verification equality $E(y_i, z_i) \stackrel{?}{\equiv} \prod_{j=1}^{n} {E_j}^{{m_i}^{j-1}} \bmod p$.
  2. If all shares are valid; the coalition S can obtain the secret d by using the reconstruction procedure described in Section 2.

Fig. 3. Qiong et al.'s CRT-based VSS extension

Another VSS scheme based on Asmuth-Bloom secret sharing was proposed by Qiong et al. [16]. Their approach is similar to the VSS of Pedersen [15] based on Shamir's SSS. Their scheme is given in Figure 3.

As the scheme shows, Qiong et al. treated the shares of Asmuth-Bloom SSS as points on a degree-$(n-1)$ polynomial and adopted the approach of Pedersen by evaluating the polynomial in the exponent to verify the shares. If the dealer is honest, the scheme in Figure 3 is secure because the verification data can be used to detect an invalid share from a corrupted user in the first step of the combiner phase.

However, similar to the attack on Iftene's VSS scheme, if the dealer uses some $y > M$ and computes the verification data by using the shares $y_i = y \bmod m_i$, $1 \leq i \leq n$, the verification equation (3) holds for each user. But, for a coalition S where $y > M_S$, the coalition S cannot compute the correct y value and the secret d.

Note that Iftene's VSS scheme uses a separate verification data for each user; hence even if all the verification equations hold, the secret can still be inconsistent for different coalitions. Quiong et al.'s VSS scheme generates a polynomial $f(x)$ from the shares as in Feldman's and Pedersen's VSS schemes. This polynomial

is used to check all verification equations. But Asmuth-Bloom SSS depends on the CRT and unlike Shamir's SSS, here f is not inherently related to the shares. Hence, even if all the equations hold, the shares can still be inconsistent as we have shown.

4 Verifiable Secret Sharing with Asmuth-Bloom SSS

As discussed in Section 3, existing CRT-based VSS schemes in the literature cannot prevent a dealer from cheating. To solve this problem, we will use a range proof technique originally proposed by Boudot [4] and modified by Cao et al. [5].

4.1 Range Proof Techniques

Boudot [4] proposed an efficient and non-interactive technique to prove that a committed number lies within an interval. He used the Fujisaki-Okamoto commitment scheme [8], where the commitment of a number y with bases (g, h) is computed as

$$E = E(y, r) = g^y h^r \bmod N$$

where g is an element in $\mathbb{Z}_N^*$, h is an element of the group generated by g, and r is a random integer. As proved in [4,8], this commitment scheme is statistically secure assuming the factorization of N is not known.

After Boudot, Cao et al. [5] applied the same proof technique with a different commitment scheme

$$E = E(y) = g^y \bmod N$$

to obtain shorter range proofs. Here, we will use Cao et al.'s non-interactive range-proof scheme as a black box. For further details, we refer the user to [4,5]. For our needs, we modified the commitment scheme as

$$E = E(y) = g^y \bmod PN$$

where $P = \prod_{i=1}^{n} p_i$ and N is an RSA composite whose factorization is secret. Note that even if $\phi(P)$ is known, $\phi(PN)$ cannot be computed since $\phi(N)$ is secret. Throughout the section, we will use $\mathrm{RngPrf}(E(y), M)$ to denote the range proof that a secret integer y committed with $E(y)$ is in the interval $[0, M)$.

4.2 A CRT-Based VSS Scheme

In our VSS scheme, the RSA composite N is an integer generated jointly by the users and the dealer where its prime factorization is not known. Such an integer satisfying these constraints can be generated by using the protocols proposed for shared RSA key generation [3,7] at the beginning of the protocol. Note that we do not need the private and the public RSA exponents in our VSS scheme as in the original protocols [3,7]; hence those parts of the protocols can be omitted.

Let $g_i \in \mathbb{Z}_{p_i}^*$ be an element of order m_i. Let $P = \prod_{i=1}^{n} p_i$ and

$$g = \left(\sum_{i=1}^{n} g_i \frac{P}{p_i} P_i' \right) \bmod P \tag{4}$$

– *Dealer Phase:* To share a secret $d \in \mathbb{Z}_{m_0}$ among a group of n users with verifiable shares, the dealer does the following:
  1. Use the dealing procedure of the Asmuth-Bloom secret sharing scheme described in Section 2 to obtain the shares
  $$y_i = y \bmod m_i$$
  for each $1 \leq i \leq n$ where $y = d + Am_0 < M = \prod_{i=1}^{t} m_i$. Note that the m_is are large primes where $p_i = 2m_i + 1$ is also a prime for $1 \leq i \leq n$.
  2. Let N be an integer whose prime factorization is not known by the users and the dealer. Compute $E(y) = g^y \bmod PN$. Send y_i to the ith user secretly for all $1 \leq i \leq n$ and broadcast $(E(y), \text{RngPrf}(E(y), M))$.
  3. The ith user checks
  $$g_i{}^{y_i} \stackrel{?}{\equiv} E(y) \bmod p_i \tag{5}$$
  to verify $y_i = y \bmod m_i$. Then he checks the validity of the range proof to verify $y < M$.
– *Combiner Phase:* Let S be a coalition gathered to construct the secret.
  1. The share y_i of user $i \in S$ can be verified by the other users in S with the verification equality $g_i{}^{y_i} \stackrel{?}{\equiv} E(y) \bmod p_i$.
  2. If all shares are valid, the participants can obtain the secret d by using the reconstruction procedure described in Section 2. Otherwise, the corrupted users are disqualified.

Fig. 4. CRT-based verifiable secret sharing scheme

where $P_i' = \left(\frac{P}{p_i}\right)^{-1} \bmod p_i$ for all $1 \leq i \leq n$, i.e., g is the unique integer in $\mathbb{Z}_P$ satisfying $g \equiv g_i \bmod p_i$ for all i. Our VSS scheme is described in Figure 4.

4.3 Analysis of the Proposed VSS Scheme

We analyze the correctness of the scheme and its security against passive and active attackers below:

Correctness. Aside from the verification equation, the scheme uses the original Asmuth-Bloom scheme. Hence, for correctness, we only need to show that when the dealer and the users are honest, the verification equations in the dealer and combiner phases hold. Note that, the condition $y < M$ is checked in Step 3 of the dealer phase by using $\text{RngPrf}(E(y), M))$. Furthermore, for a valid share y_i,

$$\begin{aligned} E(y) \bmod p_i &= g^y \bmod PN \bmod p_i = g^y \bmod p_i \\ &= g_i{}^y \bmod p_i = g_i{}^{y_i} \bmod p_i. \end{aligned}$$

Hence if the dealer and the users behave honestly, the verification equation holds and the ith user verifies that his share is a residue modulo m_i of the integer $y < M$ committed with $E(y)$.

Security. For the security analysis, we will first show that the underlying SSS is perfect as proved by Kaya et al. [12], i.e., no coalition of size smaller than t can obtain any information about the secret.

Theorem 1 (Kaya and Selcuk [12]). *For a passive adversary with $t-1$ shares in the VSS scheme, every candidate for the secret is equally likely, i.e., the probabilities $\Pr(d = d')$ and $\Pr(d = d'')$ are approximately equal for all $d', d'' \in \mathbb{Z}_{m_0}$.*

Proof. Suppose the adversary corrupts $t-1$ users and just observes the inputs and outputs of the corrupted users without controlling their actions, i.e., the adversary is honest in user actions but curious about the secret. Let S' be the adversarial coalition of size $t-1$, and let y' be the unique solution for y in $Z_{M_{S'}}$. According to (1), $M/M_{S'} > m_0$, hence $y' + jM_{S'}$ is smaller than M for $j < m_0$. Since $\gcd(m_0, M_{S'}) = 1$, all $(y' + jM_{S'}) \bmod m_0$ are distinct for $0 \leq j < m_0$, and there are m_0 of them. That is, d can be any integer from $\mathbb{Z}_{m_0}$. For each value of d, there are either $\lfloor M/(M_{S'}m_0) \rfloor$ or $\lfloor M/(M_{S'}m_0) \rfloor + 1$ possible values of y consistent with d, depending on the value of d. Hence, for two different integers in $\mathbb{Z}_{m_0}$, the probabilities of d equals these integers are almost equal. Note that $M/(M_{S'}m_0) > m_0$ and given that $m_0 \gg 1$, all d values are approximately equally likely.

Besides the shares, the only additional information a corrupted user can obtain is $E(y)$ and $\mathrm{RngPrf}(E(y), M)$. Given that the discrete logarithm problem is hard and Cao et al.'s range proof technique is computationally secure, the proposed VSS scheme is also computationally secure. □

The shares distributed by a dealer are said to be inconsistent if different coalitions of size at least t obtain different values for the secret. The following theorem proves that the dealer cannot distribute shares inconsistent with the secret.

Theorem 2. *A corrupted dealer cannot cheat in the VSS scheme without being detected. I.e., if the shares are inconsistent with the secret d then at least one verification equation does not hold.*

Proof. Let $U = \{1, \ldots, n\}$ be the set of all users. If the shares are inconsistent, for two coalitions S and S' with $|S|, |S'| \geq t$,

$$\left(\sum_{i \in S} y_i M'_{S,i} M_{S \setminus \{i\}} \right) \bmod M_S \neq \left(\sum_{i \in S'} y_i M'_{S',i} M_{S' \setminus \{i\}} \right) \bmod M_{S'}.$$

hence,

$$y = \left(\sum_{i=1}^{n} y_i M'_{U,i} M_{U \setminus \{i\}} \right) \bmod M_U > M,$$

because we need at least $t+1$ congruences to hold. If this is true then the dealer cannot provide a valid range proof $\mathrm{RngPrf}(E(y), M)$. So, when a user tries to verify that $y < M$, the range proof will not be verified.

If the dealer tries to use a different $y' \neq y$ value in the commitment $E(y')$ and generates a valid proof $\mathrm{RngPrf}(E(y'), M)$, the verification equation (5) will not hold for some user i. Hence, the VSS scheme guarantees that the n distributed shares are consistent and they are residues of some number $y < M$. □

Theorem 3. *A user cannot cheat in the VSS scheme without being detected; i.e., if a share given in the combiner phase is inconsistent with the secret, then the verification equation does not hold.*

Proof. When a user i sends an incorrect share $y_i' \neq y_i = y \bmod m_i$ in the combiner phase, the verification equation

$$E(y) \stackrel{?}{\equiv} {g_i}^{y_i} \bmod p_i$$

will not hold because $E(y) = g^y \bmod PN$, $p_i | P$ and since the order of $g_i \in \mathbb{Z}_{p_i}$ is m_i, the only value satisfying the verification equation is y_i. □

5 Joint Random Secret Sharing

Joint random secret sharing (JRSS) protocols enable a group of users to jointly generate and share a secret where a trusted dealer is not available. Although there have been JRSS schemes based on Shamir's SSS, so far no JRSS scheme has been proposed based on CRT. Here we describe a JRSS scheme based on the VSS scheme in Section 4. We first modify (1) used in the Asmuth-Bloom secret sharing scheme in Section 2 as

$$\prod_{i=1}^{t} m_i > n {m_0}^2 \prod_{i=1}^{t-1} m_{n-i+1}. \tag{6}$$

We also change the definition of M as $M = \left\lfloor (\prod_{i=1}^{t} m_i)/n \right\rfloor$. The proposed JRSS scheme is given in Figure 5.

5.1 Analysis of the Proposed JRSS Scheme

Correctness. Observe that when all users behave honestly, the JRSS scheme works correctly. Let $y = \sum_{i \in \mathcal{B}} y^{(i)}$. It is easy to see that $y < \prod_{i=1}^{t} m_i$ since $y^{(i)} < M$ for all $i \in \mathcal{B}$, where $|\mathcal{B}| \leq n$ and $M = \left\lfloor (\prod_{i=1}^{t} m_i)/n \right\rfloor$. One can see that $y_j = y \bmod m_j$ for all $j \in \mathcal{B}$ by checking

$$y \bmod m_j = \left(\sum_{i \in \mathcal{B}} y_j^{(i)} \right) \bmod m_j = y_j \bmod m_j = y_j.$$

Hence, each y_i satisfies $y_i = y \bmod m_i$ and $y < \prod_{i=1}^{t} m_i$; so, y can be constructed with t shares.

For correctness of the verification procedure in (7), one can observe that

$$\left(\prod_{i \in \mathcal{B}} E(y^{(i)}) \right) \equiv g^{\sum_{i \in \mathcal{B}} y^{(i)}} \equiv g^y \equiv {g_i}^{y_i} \pmod{p_i}.$$

– *Dealing Phase:* To jointly share a secret $d \in \mathbb{Z}_{m_0}$ the users do the following:
  1. Each user chooses a secret $d_i \in \mathbb{Z}_{m_0}$ and shares it by using the VSS scheme as follows: He first computes
  $$y^{(i)} = d_i + A_i m_0$$
  where $y^{(i)} < M = \lfloor (\prod_{i=1}^{t} m_i)/n \rfloor$. Then the secret for the jth user is computed as
  $$y_j^{(i)} = y^{(i)} \bmod m_j.$$
  He sends $y_j^{(i)}$ to user j secretly for all $1 \leq i \leq n$ and broadcasts $(E(y^{(i)}), \mathrm{RngPrf}(E(y^{(i)}), M))$.
  2. After receiving shares the jth user verifies them by using the verification procedure in (5). Let $\mathcal{B}$ be the set of users whose shares are verified correctly. The jth user computes his overall share
  $$y_j = \left(\sum_{i \in \mathcal{B}} y_j^{(i)} \right) \bmod m_j$$
  by using the verified shares.
– *Combiner Phase:* Let S be a coalition of t users gathered to construct the secret.
  1. The share y_i of user $i \in S$ can be verified by the other users in S with the verification equation,
  $$g^{y_i} \stackrel{?}{\equiv} \left(\prod_{j \in \mathcal{B}} E(y^{(j)}) \right) \bmod p_i. \tag{7}$$
  2. If all shares are valid, the participants obtain the secret $d = \left(\sum_{i \in \mathcal{B}} d_i \right) \bmod m_0$ by using the reconstruction procedure described in Section 2.

Fig. 5. CRT-based joint random secret sharing scheme.

Security. We will show that no coalition of size smaller than t can obtain any information about the secret.

Theorem 4. *For a passive adversary with $t-1$ shares in the JRSS scheme, every candidate for the secret is equally likely. I.e., the probabilities* $\Pr(d = d')$ *and* $\Pr(d = d'')$ *are approximately equal for all* $d', d'' \in \mathbb{Z}_{m_0}$.

Proof. Suppose the adversary corrupts $t-1$ users and just observes the inputs and outputs of the corrupted users without controlling their actions, i.e., the adversary is honest in user actions but curious about the secret. Let S' be the coalition of the users corrupted by the adversary. The shares are obtained when each user shares his partial secret d_i, i.e., the adversary will obtain $t-1$ share for each d_i. We will prove that the probabilities that $d_i = d_i'$ and $d = d_i''$ are almost equal for two secret candidates $d_i', d_i'' \in \mathbb{Z}_{m_0}$.

We already proved that the Asmuth-Bloom SSS described in Section 2 is perfect with equation (1). By using the shares of S', the adversary can compute $y'^{(i)} = y^{(i)} \bmod M_{S'}$. But even with these shares, there are $\frac{M}{M_{S'}}$ consistent $y^{(i)}$s

which are smaller than M and congruent to $y'^{(i)}$ modulo $M_{S'}$. By replacing (1) with (6) and changing the definition of M to $\left\lfloor (\prod_{i=1}^{t} m_i)/n \right\rfloor$, the value of the ratio

$$\frac{M}{M_{S'}} > \frac{M}{\prod_{i=1}^{t-1} m_{n-i+1}} \approx \frac{\prod_{i=1}^{t} m_i}{n \prod_{i=1}^{t-1} m_{n-i+1}}$$

is greater than ${m_0}^2$. Hence, even with $t-1$ shares, there are still ${m_0}^2$ candidates for each $y^{(i)}$ which is used to share the secret d_i. Since $\gcd(m_0, M_{S'}) = 1$, there are approximately m_0 $y^{(i)}$s, consistent with a secret candidate d'_i. Hence, for a secret candidate d'_i the probability that $d_i = d'_i$ is approximately equal to $\frac{1}{m_0}$ and the perfectness of the scheme is preserved.

Besides the shares, the only other information the adversary can observe is the commitments and range proofs. Given that the discrete logarithm problem is hard and Cao et al.'s range proof scheme is secure, the proposed JRSS scheme is also computationally secure. □

A corrupted user cannot cheat in the JRSS scheme without being detected. Since we are using a VSS scheme, while user i is sharing his partial secret d_i, the conditions of the Asmuth-Bloom SSS must be satisfied as proved in Theorem 2. Furthermore, if user i sends an incorrect share in the combiner phase, the verification equation (7) will not hold. As a result, we can say that the JRSS scheme is secure for up to $t-1$ corrupted users and no user can cheat in any phase of the scheme.

6 Conclusion

In this paper, a CRT-based verifiable secret sharing scheme is proposed. We showed that previous solutions for this problem did not guarantee the consistency of the shares. A secure JRSS scheme based on Asmuth-Bloom scheme is also proposed as a practical application of a VSS scheme. To the best of our knowledge, the proposed schemes are the first CRT-based secure VSS and JRSS schemes in the literature.

References

1. Asmuth, C., Bloom, J.: A modular approach to key safeguarding. IEEE Trans. Information Theory 29(2), 208–210 (1983)
2. Blakley, G.: Safeguarding cryptographic keys. In: AFIPS 1979, pp. 313–317 (1979)
3. Boneh, D., Franklin, M.: Efficient generation of shared RSA keys. J. ACM 48(4), 702–722 (2001)
4. Boudot, F.: Efficient proofs that a committed number lies in an interval. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 431–444. Springer, Heidelberg (2000)
5. Cao, Z., Liu, L.: Boudot's range-bounded commitment scheme revisited. In: Qing, S., Imai, H., Wang, G. (eds.) ICICS 2007. LNCS, vol. 4861, pp. 230–238. Springer, Heidelberg (2007)

6. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: FOCS 1987: IEEE Symposium on Foundations of Computer Science, pp. 427–437 (1987)
7. Frankel, Y., MacKenzie, P.D., Yung, M.: Robust and efficient distributed RSA-Key generation. In: STOC 1998: ACM Symposium on Theory of Computing, pp. 663–672. ACM Press, New York (1998)
8. Fujisaki, E., Okamoto, T.: Statistical zero knowledge protocols to prove modular polynomial relations. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 16–30. Springer, Heidelberg (1997)
9. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust threshold DSS signatures. Information and Computation 164(1), 54–84 (2001)
10. Iftene, S.: Secret sharing schemes with applications in security protocols. Technical report, University Alexandru Ioan Cuza of Iaşi, Faculty of Computer Science (2007)
11. Ingemarsson, I., Simmons, G.J.: A protocol to set up shared secret schemes without the assistance of a mutually trusted party. In: EUROCRYPT 1991, pp. 266–282. Springer, Heidelberg (1990)
12. Kaya, K., Selçuk, A.A.: Threshold cryptography based on Asmuth-Bloom secret sharing. Information Sciences 177(19), 4148–4160 (2007)
13. Mignotte, M.: How to share a secret? In: Proc. of the Workshop on Cryptography, pp. 371–375. Springer, Heidelberg (1983)
14. Pedersen, T.P.: Distributed provers with applications to undeniable signatures. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 221–242. Springer, Heidelberg (1991)
15. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)
16. Qiong, L., Zhifang, W., Xiamu, N., Shenghe, S.: A non-interactive modular verifiable secret sharing scheme. In: ICCCAS 2005: International Conference on Communications, Circuits and Systems, pp. 84–87. IEEE, Los Alamitos (2005)
17. Quisquater, M., Preneel, B., Vandewalle, J.: On the security of the threshold scheme based on the Chinese Remainder Theorem. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 199–210. Springer, Heidelberg (2002)
18. Shamir, A.: How to share a secret? Comm. ACM 22(11), 612–613 (1979)

Secure Threshold Multi Authority Attribute Based Encryption without a Central Authority

Huang Lin, Zhenfu Cao*, Xiaohui Liang, and Jun Shao

Department of Computer Science and Engineering, Shanghai Jiao Tong University
faustlin@sjtu.edu.cn, zfcao@cs.sjtu.edu.cn

Abstract. An attribute based encryption scheme (ABE) is a cryptographic primitive in which every user is identified by a set of attributes, and some function of these attributes is used to determine the ability to decrypt each ciphertext. Chase proposed the first multi authority ABE scheme in TCC 2007 as an answer to an open problem presented by Sahai and Waters in EUROCRYPT 2005. However, her scheme needs a fully trusted central authority which can decrypt every ciphertext in the system. This central authority would endanger the whole system if it's corrupted.

This paper presents a threshold multi authority fuzzy identity based encryption(MA-FIBE) scheme without a central authority for the first time. An encrypter can encrypt a message such that a user could only decrypt if he has at least d_k of the given attributes about the message for at least $t+1, t \leq n/2$ honest authorities of all the n attribute authorities in the proposed scheme. The security proof is based on the secrecy of the underlying joint random secret sharing protocol and joint zero secret sharing protocol and the standard decisional bilinear Diffie-Hellman assumption. The proposed MA-FIBE could be extended to the threshold multi authority attribute based encryption (MA-ABE) scheme and be further extended to a proactive MA-ABE scheme.

Keywords: Threshold Multi Authority ABE, Without a central authority.

1 Introduction

The primitive idea of identity based encryption(IBE) was first proposed by Shamir[12] in 1984. In IBE system, a user is identified by a unique string, the sender has to know the identity of the receiver in order to generate a ciphertext for the receiver. However, this wouldn't always be a realistic scenario since the sender might only know the "attribute" of the receivers rather than the exact identity of the receiver sometimes. For instance[3], in a secure database of an intelligence agency, one may specify a certain document can be accessed by agents in the counterspy program. In this situation, it's much more natural to encrypt to the single attribute "counterspy", instead of an enumerative list of all agents

* Corresponding author.

D.R. Chowdhury, V. Rijmen, and A. Das (Eds.): INDOCRYPT 2008, LNCS 5365, pp. 426–436, 2008.

in the program. Fuzzy identity based encryption, a generalization of IBE, was first proposed by Sahai and Waters[11] (denoted as SW05) in 2005 to deal with this circumstance. In the FIBE system, the user is identified by a certain set of attributes, and the ciphertext is encrypted under another set of attributes. The user is able to decrypt the ciphertext when the intersection of the above two attribute sets is larger than a certain preset threshold. For example, a sender might encrypt a message which could be decrypted by those who are in any two of the following three programs: "anti-drug" program, "anti-terror" program, and "counterspy" program. In the FIBE system proposed by SW05, there would be an authority, monitoring these three attributes, which is responsible for distributing secret keys to the users after verifying whether the users are indeed in the claimed program.

However, there are commonly three departments each of which is responsible for one program and thus monitors the corresponding attribute in reality. The benefit of three independent departments is noticeable: first, the other authorities could still be trusted even when certain authorities in the system are corrupted. In other words, the trust on one single authority is reduced and distributed through all the existing authorities. Second, it would reduce the burden of the authority since the task of monitoring all these attributes and distributing secret keys corresponding to these attributes is now shared by the other authorities. Therefore, the following open problem is presented in SW05: is it possible to construct an attribute encryption scheme in which many different authorities operate simultaneously, each handing out secret keys for a different set of attributes.

1.1 Previous Work

To our best knowledge, there is only one existing multi authority attribute based encryption scheme proposed by Chase [2]. Her scheme allows any polynomial number of independent authorities to monitor attributes and distribute secret keys. An encryption chooses, for each authority, a number d_k and a set of attributes. He can then encrypt a message such that a user could only decrypt if he has at least d_k of the given attributes for each authority k. Chase's scheme mainly employs the following *two techniques*: The first is to require that every user has a global identifier (GID), and the second is to use a fully trusted central authority.

The global GID is used to prevent a collusion attack between different users. More specifically, a user who only has enough secret keys from a certain set of authorities might collude with another user who only has enough keys from the rest authorities to decrypt a ciphertext. The global GID could be considered as a randomness embedding to each user's secret keys. This technique is also adopted in this paper.

The secret keys for each user could be considered as an evaluation of a function on GID in Chase's scheme. This evaluation is ephemeral and decoupled from the master secret key y_0 but the ability to decrypt is required to depend on the users' attribute rather than their individual GID (This is what distinguishes

ABE from traditional IBE). The central authority is the second tool used in Chase's scheme in order to guarantee the above property. The central authority is responsible to provide a final secret key to integrate the secret keys from the other attribute authorities such that the decryption process would be independent of GID. The central authority would be able to decrypt all the ciphertext in the Chase's system because it masters the system secret key. In other words, now that the central authority has to be fully trusted, the trust is not totally distributed through all the authorities. The security of the whole system would be totally broken if the central authority is corrupted. Furthermore, it would also increase the computation and communication cost to run and maintain such a fully trusted authority in the system.

There is a possible attack, other than the collusion attack, which is ignored in Chase's paper (as shown footnote 2). Any qualified user couldn't be able to decrypt the ciphertext in her system even if there exists only one authority who doesn't distribute the correct secret keys. This attack is possible because all the attribute authorities could be corrupted in any way in the system.

There are some other ABE schemes proposed recently, such as [10], [9], [6], [1], [3]. The first key policy attribute based encryption scheme(KP-ABE) is proposed in [6]. Both ABE schemes described in this paper are KP-ABE.

1.2 Our Contribution

This paper focuses on removing the central authority from multi authority ABE scheme. It's difficult to remove the central authority while preventing the collusion attack and keeping the decryption process independent of the identifier of each user. As indicated in the above, it's the central authority which is responsible to integrate the secret keys from the other attribute authorities in Chase's scheme, and thus to integrate these secret keys without the central authority would be an obstacle in our system. Another difficulty is that the integration must be accompanied with the last decryption step as Chase's scheme did. The integration aims to emancipate the users from the restriction of individual identifier, which means this integration shouldn't be completed before the final decryption step because the collusion attack might be possible if the users are free from the bondage of GID.

In this paper, we replace the pseudo random function used in Chase's scheme by a polynomial. After that, we adopt the key distribution technique and the joint zero secret sharing technique [5] to construct the first secure threshold multi authority fuzzy identity based encryption scheme without a central authority.

In the proposed scheme, an encrypter can encrypt a message such that a user could only decrypt if he has at least d_k of the given attributes about the message for at least $t+1, t \leq n/2$ honest authorities in the proposed scheme. All the authorities only need to communicate with each other without revealing any private information to the others during the initialization (or the periodical update step when it comes to the proactive multi authority attribute scheme), and they could operate independently and keep autonomous in the left steps of the scheme. In other words, the whole relatively costly process is transparent

to the users, and won't cause any additional inconvenience to them. The trust is truly distributed between each authority in the proposed system, and the expense due to the central authority also disappears since the central authority is removed.

Our adversary is allowed to distribute incorrect secret keys deliberately. Our proposed system always contains at least $t + 1$ honest authorities who would distribute correct secret keys (due to the restriction for the $(t - 1, n)$-threshold adversary, here $t \leq n/2$), and those who obtain enough keys from these honest authorities could still decrypt the respective ciphertext. In other words, the faulty behavior of authorities wouldn't disrupt our proposed system[1].

The proposed threshold MA-FIBE scheme could be extended to multi authority attribute based encryption scheme. The proposed scheme also could be extended a proactive scheme which results in a more convenient and secure system for the users.

1.3 Organization

At first, some preliminaries will be introduced in Section 2. Then, the security model would be given in Section 3. In Section 4, the construction of threshold MA-FIBE scheme without a central authority is provided. In the last section, we'll discuss about the extensions.

2 Preliminaries

2.1 Decisional BDH Assumption

The bilinear maps [9] is crucial to our construction, some basic facts related to bilinear maps are introduced here.

Let G_1 and G_2 be two multiplicative cyclic groups of prime order q. Let g be a generator of G and e be a bilinear map, $e : G_1 \times G_1 \rightarrow G_2$. The bilinear map e has the following properties:

1. Bilinearity: for all $u, v \in G_1$ and $a, b \in Z_q$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non degeneracy: $e(g, g) \neq 1$.

G_1 is a bilinear group if the group operation in G_1 and the bilinear map $e : G_1 \times G_1 \rightarrow G_2$ are both efficiently computable. Note that the map e is symmetric since $e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a)$.

The security proof of the proposed scheme relies on Decisional Bilinear Diffie-Hellman assumption, its definition is shown as follow [2]:

Definition 1. *Decisional Bilinear Diffie-Hellman Assumption. Let* $a, b, c, z \leftarrow Z_q$ *be chosen randomly,* G_1 *be a group of prime order* q *and generator* g*. The Decisional BDH assumption is that no probabilistic polynomial time algorithm*

[1] Chase's scheme wouldn't guarantee this, and even qualified users don't have the ability to decrypt if even one authority distributes incorrect secret keys.

$\mathcal{B}$ can distinguish the tuple $(A = g^a, B = g^b, C = g^c, e(g,g)^{abc})$ from the tuple $(A = g^a, B = g^b, C = g^c, e(g,g)^z)$ with more than a negligible advantage. The advantage of $\mathcal{B}$ is

$$|Pr[\mathcal{B}(A, B, C, e(g,g)^{abc})] - Pr[\mathcal{B}(A, B, C, e(g,g)^z)] = 0|$$

where the probability is taken over the random choice of the generator g, the random choice of a, b, c, z in Z_q, and the random bits consumed by $\mathcal{B}$.

2.2 Communication Model

The communication model of this paper is basically the same to that of [5]. Our computation model is composed of a set of n attribute authorities which could be modeled by a PPT machine. Related definitions can be found in [8]. They are connected by a complete network of private (i.e. untappable) point-to-point channels. In addition, all the authorities have access to a dedicated broadcast channel. A partially synchronous communication model is assumed in this paper. In other words, messages sent on either a point-to-point or the broadcast channel are received by their recipients within some fixed time bound. For simplicity of discussion, we assume that the authorities are equipped with synchronized clocks.

2.3 Distributed Key Generation Protocol (DKG) and Joint Zero Secret Sharing Protocol (JZSS)

Distributed key generation protocol and joint zero secret sharing protocol are two vital components of the proposed construction. A more concrete introduction to these two techniques could be found in Section 4.4 and 4.5 of [4]. Note that Distributed key generation protocol is denoted as Joint-Exp-RSS and Joint zero secret sharing scheme is denoted as Joint-Uncond-Secure-ZSS for short in this paper. The original construction of DKG protocol could be found in [5].

In the DKG protocol, the players collectively choose shares corresponding to a (t, n)-secret sharing of a random value σ. At the end of such a protocol each player P_i has a share σ_i, where $(\sigma_1, \cdots, \sigma_n) \xrightarrow{(t,n)} \sigma$, and σ is uniformly distributed over the interpolation field, and the protocol also outputs g^σ as a public key. There's *no trusted dealer*, who masters the secret σ in the regular secret sharing scheme, in the DKG protocol. This secret can only be reconstructed through the cooperation of at least $t+1$ honest players. This property plays an important role during the proposed construction since it's the reason why the central authority could be *removed*.

The secrecy [5] of DKG protocol could be defined as: no information on the secret σ can be learned by a (t, n)-threshold static adversary which corrupts at most t players except for what's implied by the value $y = g^\sigma \mod q$. The simulation of the DKG protocol could be found in Fig.4 of [4].

The JZSS protocol is similar to the joint random secret sharing protocols but instead the players collectively choose shares corresponding to a known value zero *without a trusted dealer*. The JZSS protocol can be deduced from the joint

unconditionally secure random secret sharing protocol with a slight modification as shown in [4]. The players would obtain "zero-shares" from the JZSS protocol since all these shares form a (t, n)-secret sharing of 0. By adding such "zero-shares" to existing shares of some secret σ, one obtains a randomization of the shares of σ without changing the secret. This is the typical way to use the JZSS protocol.

However, JZSS protocol is used in a different manner from the typical way during our basic constructions. m JZSS protocols along with two DKG protocols are executed independently during the initialization to hide non-constant terms of the secret polynomial of each authority from the rest authorities while ensuring that all these terms would combine to 0. Informally, the JZSS protocol and DKG protocol are traditionally used in a *serial* way, while they run *parallel* in our basic construction, and a polynomial is used to tie up these independent protocols. As a result, we consider a $(t-1, n)$-threshold static adversary attacking the JZSS protocol, then the secrecy of the JZSS protocol could be stated as: the share σ_i of the honest players are information theoretically secure to the $(t-1, n)$-threshold static adversary. Note that the adversary shouldn't be allowed to corrupt t players as in the DKG protocol because the adversary would be able to compute the shares of the honest players since the shared secret 0 is known to the adversary. The proof of the secrecy of the JZSS protocol is straightforward.

3 Adversary and Security Model

The proposed multi authority system consists of n attribute authorities. Each attribute authority monitors a set of n_k attributes in the universe of attributes $\mathcal{U}$. The universe $\mathcal{U}$ consists of N attributes where $N = \sum_{k=1}^{n} n_k$. The proposed multi authority scheme consists of the following algorithms:

Setup: A randomized algorithm takes as input the security parameter κ and outputs all the system public parameters PK and the secret key SK for attribute authorities.

Secret Key Distribution(SKD)(SK, GID, $\mathfrak{A}_u$)**:** A randomized algorithm takes as input the authorities's secret key SK, a user u's GID, and a set of attributes $\mathfrak{A}_u^k$ in the authority AA_k's domain (We will assume that the user's claim of these attributes has been verified before this algorithm is run, $\mathfrak{A}_u = \{\mathfrak{A}_u^k, k = 1, \cdots, n\}$). Output a secret key D_u for the user u.

Encryption(ENC)($\mathfrak{A}_\mathrm{C}, M, PK$)**:** A randomized algorithm takes as input an attribute set $\mathfrak{A}_\mathrm{C}$ of a message M, the system public parameters PK and outputs the ciphertext C.

Decryption(DNC)(C, D_u)**:** A deterministic algorithm takes as input a ciphertext C, which was encrypted under an attribute set $\mathfrak{A}_\mathrm{C}$ and decryption key D_u. Output a message m if $|\mathfrak{A}_\mathrm{C}^k \bigcap \mathfrak{A}_u^k| \geq d_k$ for at least $t+1$ honest [2] attribute authorities.

As in [6] and [2], our scheme is proved secure in the selective attribute set model (SAS), in which the adversary must select the challenge attribute set he

[2] Honest means the authority would distribute correct secret keys to the users.

wishes to attack before receiving the parameters of the system. Our adversary is static, i.e., chooses up to $t-1$, $t \leq n/2$ corrupted authorities, and it's denoted as $(t-1,n)$ threshold adversary. At last, the authority controlled by the adversary here is allowed to distribute incorrect secret keys to the users.

Consider the following game:

Setup: The $(t-1,n)$-threshold adversary sends a list of attribute sets $\mathfrak{A}_{C^*} = \mathfrak{A}^1_{C^*}, \cdots, \mathfrak{A}^n_{C^*}$, one for each authority. He must also provide a list of corrupted authorities (up to $t-1$). The simulator provides the adversary with the following information after the generation of system parameters: the public keys for all the honest authorities, and the secret keys for all corrupted authorities.

Secret Key Queries (SKD) (the total number of these queries are m for each authority) The requirements for the queries are: for each GID and any $t+1$ authorities the adversary query, there must be at least one honest authority k from which the adversary requests fewer than d_k of the attributes given in $\mathfrak{A}^k_{C^*}$. The adversary never queries the same authority twice with the same GID.

Challenge: The adversary sends two messages M_0 and M_1. The challenger chooses a bit x, computes the encryption of M_x for the attribute sets list $\mathfrak{A}_{C^*}$, and sends this ciphertext C^* to the adversary.

More Secret Key Queries: The adversary may make more secret key queries subject to the requirements described in the above.

Guess: The adversary outputs a guess x'. The adversary succeed if $x = x'$.

Definition 2. *A multi authority attribute scheme is selective attribute sets (SAS) CPA secure if there exists a negligible function ν such that, in the above game any $(t-1,n)$-threshold adversary will succeed with probability at most $\frac{1}{2} + \nu(\kappa)$ (κ denotes the system security parameter).*

4 Threshold MA-FIBE Scheme without a Central Authority

4.1 Primitive Idea

A large universe construction of FIBE is given by Sahai and Waters [11], and the FIBE construction adopted in this section could be considered as the respective variant construction with large size of public parameters. Different from Chase's scheme, each authority AA_k selects a polynomial $a_{k,0} + a_{k,1}x + \cdots + a_{k,m}x^m$ rather than a pseudo random function to evaluate on a user's GID. Only in this way could we manage to use DKG and JZSS protocol to enable all authorities to share a secret without letting any authority to master the system master key. The system secret key a_0 is generated by executing a DKG protocol and thus a_0 isn't known to any authority, and each authority $AA_k, k = 1, \cdots, n$ would have the share $a_{k,0}$ about a_0. In order to generate the polynomial $a_{k,0} + a_{k,1}x + \cdots + a_{k,m}x^m$, the authority AA_k needs to decide how to pick up the other coefficients $a_{k,j}, j = 1, \cdots, m$. However, the system has to ensure that the ability to decrypt independent of GID. In consequence, JZSS protocol is adopted to generate the other coefficients $a_{k,j}, k = 1, \cdots, n, j = 1, \cdots, m$ since

$a_{1,j}, \cdots, a_{n,j}, j = 1, \cdots, m$ forms a (t, n) threshold polynomial secret sharing of 0 due to JZSS protocol, and thus each user's GID would have nothing to do with the final decryption.

If the users obtain enough secret keys from honest authorities AA_k, then they would be able to compute the respective share $e(g, g_2)^{p_k(0)s} = e(g, g_2)^{(a_{k,0}+a_{k,1}GID+\cdots+a_{k,m}GID^m)s}$ in the proposed scheme. As stated in the above, $(a_{1,0}, \cdots, a_{n,0})$ form a random (t, n) threshold polynomial secret sharing of a_0 due to DKG protocol, and $(a_{1,k}, \cdots, a_{n,k})_{k=1,\cdots,m}$ form m random (t, n) threshold polynomial secret sharing of 0. When the shares combine to the corresponding secrets, GID would disappear in the final combining polynomial, and they'll have $e(g, g_2)^{a_0 s} = e(g_1, g_2)^s$ to decrypt the ciphertext. In other words, the users are only liberated from GID in the final decryption step.

4.2 The Proposed Scheme

Fix prime order groups G_1, G_2, bilinear map $e : G_1 \times G_1 \to G_2$, and generator $g \in G_1$. Define the universe of attributes $\mathcal{U} = \{1, 2, \cdots, N\}$. Each attribute authority AA_k monitors a set of n_k attributes in this universe, where $N = \sum_{k=1}^{n} n_k$. For simplicity of discussion and without lost of generality, the first $t+1$ authorities are assumed to be honest and distribute correct secret keys and the decryptor has enough secret keys from all these honest authorities in the following discussion.

1. **Setup**
   - (a) Execute DKG protocol twice and JZSS protocol independently m times, the shares of attribute authority $AA_i, i = 1, \cdots, n$ obtained from the execution of these two protocols constitute the set of partial secret keys for each authority. In consequence, there are totally $m + 2$ secret keys for each authority, as shown in the following table:

     The secret keys $Sk_i, i = 1, \cdots, n$ for the respective AA_i are $a_{i,0}$, $\cdots$, $a_{i,m}$, $b_{i,m+1}$. The respective public keys of two DKG protocols are g^{a_0}, g^{b_0}, which would be treated as a part of system public keys.

Table 1. Authorities secret keys

	DKG	JZSS	$\cdots$	JZSS	DKG
Sk_1	$a_{1,0}$	$a_{1,1}$	$\cdots$	$a_{1,m}$	$b_{1,m+1}$
Sk_2	$a_{2,0}$	$a_{2,1}$	$\cdots$	$a_{2,m}$	$b_{2,m+1}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
Sk_n	$a_{n,0}$	$a_{n,1}$	$\cdots$	$a_{n,m}$	$b_{n,m+1}$
Public keys	$g_1 = g^{a_0}$				$g_2 = g^{b_0}$

According to the DKG protocol, we have $a_0 = \sum_{l=1}^{t+1} a_{k_l,0}\gamma_{k_l}$, $b_0 = \sum_{l=1}^{t+1} b_{k_l,m+1}\gamma_{k_l}$. We also have $0 = \sum_{l=1}^{t+1} a_{k_l,j}\gamma_{k_l}, j = 1, 2, \cdots, m$ according to the JZSS protocol. [3]

[3] $\gamma_j = \prod_{k\in\{1,2,\cdots,t+1\}, j\neq k} \frac{0-k}{j-k}$ denotes the Lagrange interpolation coefficients for the set $\{1, 2, \cdots, t+1\}$.

– (b) Each authority $AA_k, k = 1, \cdots, n$ also needs to randomly choose another set of secret keys $t_{k,1}, \cdots, t_{k,n_k}$ from Z_q, each of which corresponds to the j-th attribute mastered by the authority AA_k in the universe $\mathcal{U}$. The corresponding public keys are $T_{k,1} = g^{t_{k,1}}, \cdots, T_{k,n_k} = g^{t_{k,n_k}}$.
– (c) the secret key SK_k for each authority $AA_k, k = 1, \cdots, n$ are

$$a_{k,0}, a_{k,1}, a_{k,2}, \cdots, a_{k,m}, b_{k,m+1}, t_{k,1}, \cdots, t_{k,n_k}$$

The secret keys of all authorities form the set of secret keys $SK = \{SK_1, \cdots, SK_n\}$.

The published public parameters PK are $g, g_1 = g^{a_0}, g_2 = g^{b_0}$, $\{T_{k,1}, \cdots, T_{k,n_k}\}_{k=1,\cdots,n}$.

2. **SKD**(SK, GID, $\mathfrak{A}_u$)
The user randomly selects a GID from Z_q and presents it to the authority, and the authority checks whether it's valid (according to the method mentioned in footnote 4), if it's valid then continue, else reject.

For $AA_k, k = 1, \cdots, n$, the SKD process is shown as follows:

For each GID, the authority would first randomly select $p_k(x)$ (where $p_k(x)$ is a $d_k - 1$ degree polynomial) satisfied with $p_k(0) = a_{k,0} + a_{k,1}GID + \cdots + a_{k,m}GID^m$. The secret keys D_u for u are $D_u = \{D_{k,j}\}_{j\in\mathfrak{A}_u^k, k=1,\cdots,n}$, where $D_{k,j} = \{g_2^{p_k(j)/t_{k,j}}\}_{j\in\mathfrak{A}_u^k}$.

3. **ENC**($\mathfrak{A}_\mathrm{C}, M, PK$)
Choose a random value $s \in Z_q$. For the attribute set $\mathfrak{A}_\mathrm{C} = \{\mathfrak{A}_\mathrm{C}^k\}, k \in \{1, \cdots, n\}$, generate the ciphertext C=$\{\mathfrak{A}_\mathrm{C}, E = e(g_1, g_2)^s \cdot M||0^{l'}, \{E_{k,j} = T_{k,j}^s\}_{j\in\mathfrak{A}_\mathrm{C}^k, \forall k}\}$ (k corresponds to the authority AA_k, $\mathfrak{A}_\mathrm{C}^k$ denotes the attribute set of the ciphertext monitored by authority AA_k. In order to check whether a decryption is valid, prior to encryption, we append M trailing 0s denoted $0^{l'}$ to message $M \in \{0,1\}^l$).

4. **DEC**(C, D_u)
(1) For $AA_k, k = 1, 2, \cdots, t+1$ run the following two steps.
– (a) For d_k attributes $i \in \mathfrak{A}_\mathrm{C}^k \bigcap \mathfrak{A}_u$, compute $e(E_{k,i}, D_{k,i})$ $= e(g^{t_{k,j}s}, g_2^{p_k(j)/t_{k,j}}) = e(g, g_2)^{p_k(j)s}$.
– (b) Interpolate to find

$$e(g, g_2)^{p_k(0)s} = e(g, g_2)^{(a_{k,0}+a_{k,1}GID+\cdots+a_{k,m}GID^m)s}$$

(2) $\prod_{k=1}^{t+1} e(g, g_2)^{p_k(0)s\gamma_k}$
$= e(g, g_2)^{\sum_{k=1}^{t+1}(\gamma_k a_{k,0}+\gamma_k a_{k,1}GID+\cdots+\gamma_k a_{k,m}GID^m)s}$
$= e(g, g_2)^{a_0 s}$
$= e(g_1, g_2)^s$
(3) $E/e(g_1, g_2)^s = M||0^{l'}$

A possible collusion attack and the restriction of m***:***
The adoption of polynomial $a_{k,0} + a_{k,1}GID + a_{k,2}GID^2 + \cdots + a_{k,m}GID^m$ as the master key while generating the secret key for a user might cause a collusion

attack. Assume $m+1$ users who only have qualified secret keys for an attribute set $\mathfrak{A}_{\mathrm{C}}^{k}$ of a ciphertext E, which means each user can calculate

$$e(g,g_2)^{s\cdot(a_{k,0}+a_{k,1}GID+a_{k,2}GID^2+\cdots+a_{k,m}GID^m)}.$$

Consequently, $e(g,g_2)^{s\cdot(a_{k,0}+a_{k,1}x+a_{k,2}x^2+\cdots+a_{k,m}x^m)}$ could be interpolated since there would be $m+1$ evaluations of $e(g,g_2)^{s\cdot(a_{k,0}+a_{k,1}x+a_{k,2}x^2+\cdots+a_{k,m}x^m)}$ on different GIDs. Consider another user with identifier GID' which only has eligible secret keys for the other t attribute sets $\mathfrak{A}_{\mathrm{C}}^{k_l}, l=1,\cdots,t$ of the same ciphertext, then $e(g,g_2)^{s\cdot(a_{k_l,0}+a_{k_l,1}GID'+a_{k_l,2}GID'^2+\cdots+a_{k_l,m}GID'^m)}$, $l=1,\cdots,t$ can be obtained by them. If this user colludes with the above $m+1$ users with satisfied keys from $\mathcal{A}_k$, then they'll have the ability to interpolate $e(g_1,g_2)^s$. It remains to evaluate $e(g,g_2)^{s\cdot(a_{k,0}+a_{k,1}x+a_{k,2}x^2+\cdots+a_{k,m}x^m)}$ on GID' and do the rest interpolation, which means all of them would be able to open the message even though none of them is qualified to do this. That's how the possible collusion attack works.

In order to prevent this attack in practice, we have to restrict the number of GIDs no more than m in the basic constructions. That's why the number of secret key queries is restricted to be equal to m in the security model. As we can see here, m is a crucial parameter for the security of the basic constructions, and there is a tradeoff between the security property and the efficiency of the initialization step. Although the system could accommodate more GIDs as m grows, more JZSS protocols need to be executed during the initialization while m is bigger, and thus the computation and communication cost increase. The security of the proposed construction can be stated as the following theorem, the proof of which is given in the full version[7].

Theorem 1. *The proposed threshold MA-FIBE scheme is SAS CPA secure in the standard model and $(t-1,n)$-threshold adversary model if Decisional BDH assumption holds in (G_1,G_2) and the underling DKG protocol and JZSS protocol are secure.*

5 Extensions

We can employ the methods introduced in [6] to convert the proposed MA-FIBE scheme into MA-ABE schemes without a central authority, and the concrete constructions and the proofs could be found in the full version[7]. Also, the primitive idea of constructing a proactive secret sharing could be borrowed to construct a proactive MA-ABE scheme in the large universe, and this construction would also be shown in the full version[7]. Most extensions mentioned in Chase's scheme such as **Changing d_k**, **Leaving out authorities**, could be realized by applying the method used in Chase's scheme directly to our proposed schemes. Our proposed scheme could also be considered as a non trivial realization of an extension named **More complicated functions of the authorities** in Chase's paper since the central authority is removed in the proposed constructions. However,

there seems no convenient way to realize another extension **Adding attribute authorities** in our proposed constructions. It seems that all the authorities, no matter old or newly-jointed authorities, need to get together to renew the whole system. This kind of trivial re-initialization implies certain inconvenience for the users as mentioned in Section 4.2. Thus, how to add new authorities to the proposed constructions without causing too much trouble for the users would be left as an open problem.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under Grant Nos. 60773086, 60673079 and 60572155 .

References

1. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: IEEE Symposium on Security and Privacy, pp. 321–334 (2007)
2. Chase, M.: Multi-authority attribute based encryption. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 515–534. Springer, Heidelberg (2007)
3. Cheung, L., Newport, C.: Provably secure ciphertext policy abe. In: ACM Conference on Computer and Communications Security, pp. 456–465 (2007)
4. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust threshold dss signatures. Inf. Comput. 164(1), 54–84 (2001)
5. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. J. Cryptology 20(1), 51–83 (2007)
6. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: ACM Conference on Computer and Communications Security, pp. 89–98 (2006)
7. Lin, H., Cao, Z., Liang, X., Shao, J.: Secure threshold multi authority attribute based encryption without a central authority (2008), http://eprint.iacr.org/2008/
8. Cerecedo, M., Matsumoto, T., Imai, H.: Efficient and secure multiparty generation of digital signatures based on discrete logarithms. IEICE Transactions on Fundamentals, 532–545 (1993)
9. Ostrovsky, R., Sahai, A., Waters, B.: Attribute-based encryption with non-monotonic access structures. In: ACM Conference on Computer and Communications Security, pp. 195–203 (2007)
10. Pirretti, M., Traynor, P., McDaniel, P., Waters, B.: Secure attribute-based systems. In: ACM Conference on Computer and Communications Security, pp. 99–112 (2006)
11. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005)
12. 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)

Author Index