Tetsu Iwata
Masakatsu Nishigaki (Eds.)

LNCS 7038

Advances in Information and Computer Security

6th International Workshop, IWSEC 2011
Tokyo, Japan, November 2011
Proceedings

Springer

Lecture Notes in Computer Science 7038

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

Editorial Board

Tetsu Iwata Masakatsu Nishigaki (Eds.)

Advances in Information and Computer Security

6th International Workshop, IWSEC 2011
Tokyo, Japan, November 8-10, 2011
Proceedings

Springer

Volume Editors

Tetsu Iwata
Nagoya University
Dept. of Computational Science and Engineering
Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan
E-mail: iwata@cse.nagoya-u.ac.jp

Masakatsu Nishigaki
Shizuoka University
Graduate School of Science and Technology
3-5-1 Johoku, Naka-ku, Hamamatsu 432-8011, Japan
E-mail: nisigaki@inf.shizuoka.ac.jp

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-25140-5 e-ISBN 978-3-642-25141-2
DOI 10.1007/978-3-642-25141-2
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: Applied for

CR Subject Classification (1998): E.3, G.2.1, D.4.6, K.6.5, K.4.4, F.2.1, C.2

LNCS Sublibrary: SL 4 – Security and Cryptology

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

Printed on acid-free paper

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

Preface

The 6th International Workshop on Security (IWSEC 2011) was held at the Institute of Industrial Science, the University of Tokyo, Japan, during November 8–10, 2011. The workshop was co-organized by ISEC in ESS of the IEICE (The Technical Group on Information Security in the Engineering Sciences Society of the Institute of Electronics, Information and Communication Engineers) and CSEC of the IPSJ (The Special Interest Group on Computer Security of the Information Processing Society of Japan).

This year, the workshop received 45 submissions, of which 14 were accepted for presentation. Each submission was anonymously reviewed by at least three reviewers, and these proceedings contain the revised versions of the accepted papers. In addition to the presentations of the papers, the workshop also featured a poster session and two invited talks. The invited talks were given by Mitsuru Matsui on "Linear Cryptanalysis: History and Open Problems" and by Takashi Shinzaki on "Palm Vein Authentication Technology and Its Application Systems."

The best paper award was given to "REASSURE: A Self-contained Mechanism for Healing Software Using Rescue Points" by Georgios Portokalidis and Angelos D. Keromytis, and the best student paper award was given to "Identity-Based Deterministic Signature Scheme without Forking-Lemma" by S. Sharmila Deva Selvi, S. Sree Vivek, and C. Pandu Rangan.

A number of people contributed to the success of IWSEC 2011. We would like to thank the authors for submitting their papers to the workshop. The selection of the papers was a challenging and delicate task, and we are deeply grateful to the members of the Program Committee and the external reviewers for their in-depth reviews and detailed discussions. We are also grateful to Thomas Baignères and Matthieu Finiasz for developing iChair, which was used for the paper submission, reviews, and discussions, and to Andrei Voronkov for developing EasyChair, which was used to prepare these proceedings.

Last but not least, we would like to thank the General Co-chairs, Kanta Matsuura and Naoya Torii, for leading the Local Organizing Committee, and we would also like to thank the members of the Local Organizing Committee for their efforts to ensure the smooth running of the workshop.

August 2011

Tetsu Iwata
Masakatsu Nishigaki

IWSEC 2011
6th International Workshop on Security

Tokyo, Japan, November 8–10, 2011

Co-organized by

ISEC in ESS of the IEICE
(The Technical Group on Information Security in the Engineering Sciences Society of the Institute of Electronics, Information and Communication Engineers)

and

CSEC of the IPSJ
(The Special Interest Group on Computer Security of the Information Processing Society of Japan)

General Co-chairs

Kanta Matsuura	The University of Tokyo, Japan
Naoya Torii	Fujitsu Laboratories Ltd., Japan

Advisory Committee

Hideki Imai	Chuo University, Japan
Kwangjo Kim	Korea Advanced Institute of Science and Technology, Korea
Günter Müller	University of Freiburg, Germany
Yuko Murayama	Iwate Prefectural University, Japan
Koji Nakao	National Institute of Information and Communications Technology, Japan
Eiji Okamoto	University of Tsukuba, Japan
C. Pandu Rangan	Indian Institute of Technology, Madras, India

Program Co-chairs

Tetsu Iwata	Nagoya University, Japan
Masakatsu Nishigaki	Shizuoka University, Japan

Local Organizing Committee

Takuro Hosoi	The University of Tokyo, Japan
Mitsugu Iwamoto	The University of Electro-Communications, Japan
Shin'ichiro Matsuo	National Institute of Information and Communications Technology, Japan
Koji Nuida	National Institute of Advanced Industrial Science and Technology, Japan
Katsuyuki Takashima	Mitsubishi Electric Corporation, Japan
Satoru Tezuka	Tokyo University of Technology, Japan
Katsunari Yoshioka	Yokohama National University, Japan

Program Committee

Rafael Accorsi	University of Freiburg, Germany
Claudio Ardagna	Università degli Studi di Milano, Italy
Andrey Bogdanov	Katholieke Universiteit Leuven, Belgium
Kevin Butler	University of Oregon, USA
Pau-Chen Cheng	IBM Thomas J. Watson Research Center, USA
Sabrina De Capitani di Vimercati	Università degli Studi di Milano, Italy
Bart De Decker	Katholieke Universiteit Leuven, Belgium
Isao Echizen	National Institute of Informatics, Japan
William Enck	North Carolina State University, USA
Eiichiro Fujisaki	NTT, Japan
Steven Furnell	Plymouth University, UK
Dieter Gollmann	Hamburg University of Technology, Germany
Goichiro Hanaoka	AIST, Japan
Swee-Huay Heng	Multimedia University, Malaysia
Naofumi Homma	Tohoku University, Japan
Jin Hong	Seoul National University, Korea
Seokhie Hong	CIST, Korea University, Korea
Yoshiaki Hori	Kyushu University, Japan
Koray Karabina	University of Waterloo, Canada
Angelos D. Keromytis	Columbia University, USA
Seungjoo Kim	Korea University, Korea
Tetsutaro Kobayashi	NTT, Japan
Noboru Kunihiro	The University of Tokyo, Japan
Kwok-Yan Lam	National University of Singapore, Singapore
Jigang Liu	Metropolitan State University, USA
Javier Lopez	University of Malaga, Spain
Stephen Marsh	Communications Research Centre, Canada
Keith Martin	Royal Holloway, University of London, UK
Wakaha Ogata	Tokyo Institute of Technology, Japan

Raphael Phan	Loughborough University, UK
Hartmut Pohl	University of Applied Sciences Bonn-Rhein-Sieg, Germany
Axel Poschmann	Nanyang Technological University, Singapore
Kai Rannenberg	Goethe University Frankfurt, Germany
Christian Rechberger	ENS Paris, France
Palash Sarkar	Indian Statistical Institute, India
Ryoichi Sasaki	Tokyo Denki University, Japan
Francesco Sica	
Ron Steinfeld	Macquarie University, Australia
Reima Suomi	Turku School of Economics, Finland
Willy Susilo	University of Wollongong, Australia
Keisuke Takemori	KDDI Corporation, Japan
Mikiya Tani	NEC, Japan
Ryuya Uda	Tokyo University of Technology, Japan
Guilin Wang	University of Wollongong, Australia
Sven Wohlgemuth	National Institute of Informatics, Japan
Toshihiro Yamauchi	Okayama University, Japan
Sung-Ming Yen	National Central University, Taiwan
Hiroshi Yoshiura	University of Electro-Communications, Japan
Ilsun You	Korean Bible University, Korea

External Reviewers

Andreas Albers
Elena Andreeva
Jean-Philippe Aumasson
Sambuddho Chakravarty
Donghoon Chang
Ji-Jian Chin
Kuo-Zhe Chiou
Wei Gao
Fuchun Guo
Yuichi Hayashi
Fumitaka Hoshino
David Jao
Jérémy Jean
Ik Rae Jeong
Kitae Jeong
Markus Kasper
Yutaka Kawai
Kaoru Kurosawa
Jorn Lapon
Wei Lei
Hsi-Chung Lin
Amir Moradi
M. Prem Laxman Das
Mohammad Reza Reyhanitabar
Ahmad Sabouri
Somitra K. Sanadhya
Tsunekazu Saito
Martin Salfer
Katherine Stange
Thomas Stocker
Koutarou Suzuki
Jheng-Hong Tu
Go Yamamoto
Kan Yasuda

Table of Contents

Mathematical and Symmetric Cryptography

Public Key Encryption

A New Soft Decision Tracing Algorithm for Binary Fingerprinting Codes

Minoru Kuribayashi

Graduate School of Engineering, Kobe University
1-1 Rokkodai-cho, Nada-ku, Kobe, Hyogo, 657-8501 Japan
kminoru@kobe-u.ac.jp

Abstract. The performance of fingerprinting codes has been studied under the well-known marking assumption. In a realistic environment, however, a pirated copy will be distorted by an additional attack. Under the assumption that the distortion is modeled as AWGN, a soft decision method for a tracing algorithm has been proposed and the traceability has been experimentally evaluated. However, the previous soft decision method works directly with a received signal without considering the communication theory. In this study, we calculate the likelihood of received signal considering a posterior probability, and propose a soft decision tracing algorithm considering the characteristic of Gaussian channel. For the estimation of channel, we employ the expectation-maximization algorithm by giving constraints under the possible collusion strategies. We also propose an equalizer to give a proper weighting parameter for calculating a correlation score.

1 Introduction

Digital fingerprinting [14] is used to trace illegal users, where a unique ID known as a digital fingerprint is embedded into a content before distribution. When a suspicious copy is found, the owner can identify illegal users by extracting the fingerprint. Since each user purchases a content involving his own fingerprint, the fingerprinted copy slightly differs with each other. Therefore, a coalition of users will combine their differently marked copies of the same content for the purpose of removing/changing the original fingerprint. To counter this threat, coding theory has produced a number of collusion resistant codes under the well-known principle referred to as the marking assumption.

Tardos [13] has proposed a probabilistic fingerprinting code which has a length of theoretically minimal order with respect to the number of colluders. Theoretical analysis about the Tardos code yields more efficient probabilistic fingerprinting codes improving the traceability, code length, and so on. Among the variants of the Tardos code, Nuida et al. [10] studied the parameters to generate the codewords of the Tardos code which are expressed by continuous distribution, and presented a discrete version in an attempt to reduce the code length and the required memory amount without degrading the traceability.

T. Iwata and M. Nishigaki (Eds.): IWSEC 2011, LNCS 7038, pp. 1–15, 2011.

It is reported in [2] that a correlation sum calculated in a tracing algorithm is expected to be Gaussian distribution based on the Central Limit Theorem (CLT). Using the Gaussian approximation, the code length is further shortened under a given false-positive probability. The results are supported and further analyzed by Furon et al. [3], and the validity is experimentally evaluated in [8]. In [12], it is shown that the tails of the distribution follow a power law which depends on the collusion strategy. Independent of the strategy, the right tail falls off faster than the left tail.

Recently, the relaxation of the marking assumption has been employed in the analysis of the Tardos code and its variants [5],[6],[7],[9]. In [7], a pirated copy is produced by collusion attack and it is further distorted by additive white Gaussian noise (AWGN). Considering the distortion, two kinds of tracing algorithms are proposed; one rounds each element of codeword into binary digit before calculating a correlation score, and the other directly calculates the score from the distorted codeword. The former is called a hard decision method, and the latter, a soft decision method. In [6], it is reported that the probability of false-positive for the Tardos code is considerably increased in the amount of noise while that for the Nuida code is not sensitive against the noise. However, the soft decision method does not utilize the analog signals to maximize the performance of a detector. It merely calculates a correlation score directly from the received signal without the consideration of a posterior probability.

In this paper, we propose a soft decision tracing algorithm considering a posterior probability of codeword extracted from a pirated copy. We assume that a codeword is produced by a certain collusion strategy based on the marking assumption and is distorted by additive white Gaussian noise. Depending on the collusion strategy, the probability that an i-th bit becomes 1 is slightly/greatly changed from the original probability, namely 0.5. In order to estimate the probability as well as the variance of the Gaussian noise, the Expectation-Maximzation(EM) algorithm is used in this paper. Generally, the EM algorithm is not assured to find a global optimum whose estimated values are well-matched with actual ones. By giving some constraints on the parameters estimated by the EM algorithm, we improve the accuracy to find the global optimum. Using the estimated parameters, we calculate a new correlation sum based on the posterior probability. If the sum exceeds a specific threshold, the corresponding candidate is judged guilty. Based on the CLT, the variance of the sum is derived from a Monte Carlo simulator and the threshold for judgment is calculated by a given false-positive probability. The validity of the threshold is also evaluated by the rare event simulation method proposed in [5]. We further study the bias in the calculation of the correlation score, and propose an equalizer to cancel the bias by giving a weight on each score.

The experimental results reveal the following properties. 1: When the EM algorithm fails to estimate the conditions of Gaussian channel, the performance of the proposed method without the equalizer is degraded with the increase of SNR. 2: The proposed method with the equalizer outperforms the method without it. Especially for the cryptographic collusion strategy [3], we get a drastic

improvement from the conventional methods. 3. The total false-positive probability is almost stable against the changes of SNR, and is slightly affected by a collusion strategy if the threshold is designed under the Gaussian assumption.

2 Preliminaries

In this section, probabilistic fingerprinting codes are reviewed, and the related works are briefly introduced.

2.1 Probabilistic Fingerprinting Code

Tardos [13] has proposed a probabilistic c-secure code which has a length of theoretically minimal order with respect to the number of colluders. The binary codewords of length L are arranged as an $N \times L$ matrix $\boldsymbol{X}$, where N is the number of users and each element $X_{j,i} \in \{0, 1\}$ in the matrix is the i-th element of j-th user's codeword. The element $X_{j,i}$ is generated from an independently and identically distributed random number with a probability p_i such that $\Pr[X_{j,i} = 1] = p_i$ and $\Pr[X_{j,i} = 0] = 1 - p_i$. This probability p_i referred to as the *bias distribution* follows a certain continuous distribution represented by $f(p)$:

$$f(p) = \frac{1}{\pi\sqrt{p(1-p)}}. \tag{1}$$

Assuming that the number of colluders is at most c, the minimum length L for a constant and tiny error probability is theoretically derived. The maximum allowed probability of accusing a fixed innocent user is denoted by ϵ_1, and the total false positive probability by $\eta = 1 - (1 - \epsilon_1)^{N-c} \approx N\epsilon_1$. The false negative probability denoted by ϵ_2 is coupled to ϵ_1 according to $\epsilon_2 = \epsilon_1^{c/4}$.

Nuida et al. [10] proposed a specific discrete distribution introduced by a discrete variant [11] of Tardos code that can be tuned for a given number c of colluders. The bias distribution is called "Gauss-Legendre distribution" due to the deep relation to Gauss-Legendre quadrature in numerical approximation theory (see [10] for detail). Except for the bias distribution, the Nuida code employs the same encoding mechanism as the Tardos code.

Let L be a code length of a fingerprinting code. Suppose that $\tilde{c}(\leq c)$ malicious users out of N users are colluded, and they produce a pirated codeword $\boldsymbol{y} = (y_1, \ldots, y_L)$, $y_i \in \{0, 1\}$. A tracing algorithm first calculates a score $S_i^{(j)}$ for i-th bit of j-th user using a real-valued function $U_{j,i}$, and then sums them up as the total score $S^{(j)} = \sum_{i=0}^{L} S_i^{(j)}$ of j-th user.

$$S^{(j)} = \sum_{i=1}^{L} S_i^{(j)} = \sum_{i=1}^{L} y_i U_{j,i}, \tag{2}$$

where

$$U_{j,i} = \begin{cases} \sqrt{\frac{1-p_i}{p_i}} & (X_{j,i} = 1) \\ -\sqrt{\frac{p_i}{1-p_i}} & (X_{j,i} = 0). \end{cases} \tag{3}$$

Because the above correlation sum adds the score $S_i^{(j)}$ only when $y_i = 1$, half of the elements in a pirated codeword is discarded. Considering the symmetry, Škorić et al. [2] proposed a symmetric version of the correlation score by substituting $\hat{y}_i = 2y_i - 1 \in \{-1, 1\}$ for y_i in Eq.(2).

For the Tardos code, if the sum $S^{(j)}$ exceeds a threshold Z, the j-th user is determined as guilty. Such a tracing algorithm is called "catch-many" type explained in [14]. By decoupling ϵ_1 from ϵ_2, the tracing algorithm can detect more colluders under a constant ϵ_1 and L. For the Nuida code [10], its original tracing algorithm outputs only one guilty user whose score becomes maximum, which type is called "catch-one". Due to the similarity with the Tardos code, the catch-many tracing algorithm of the Tardos code can be applied to the Nuida code. The report in [6] stated that the performance of the Nuida code is better than that of the Tardos code when the catch-many tracing algorithm is used. Under a same code length and a same number of colluders, it is experimentally measured that the correlation sum of the Nuida code is higher than that of the Tardos code. It is remarkable that the false-positive probability of the Nuida code is stable no matter how many colluders get involved in to generate a pirated copy and no matter how much amount of noise is added to the copy if a threshold is calculated under the Gaussian approximation for the correlation score. In this paper, the validity of the previous tracing algorithms is discussed from the Nuida code point of view, which does not limit the use of proposed method for the Tardos code.

2.2 Attack Model

Under the marking assumption, colluders can select an arbitrary bit for such elements that a bit embedded into the segments of their copies is different. Based on an attack strategy, various collusion strategies under the marking assumption could be selected by colluders. Among them, there are 5 major types:

- majority(maj): If the sum of i-th bit exceeds $\tilde{c}/2$, $y_i = 1$; otherwise, $y_i = 0$.
- minority(min): If the sum of i-th bit exceeds $\tilde{c}/2$, $y_i = 0$; otherwise, $y_i = 1$.
- random(ran): $y_i \in_R \{0, 1\}$
- all-0: $y_i = 0$
- all-1: $y_i = 1$

In [5], the collusion attack is described by the parameter vector: $\boldsymbol{\theta} = (\theta_0, \cdots, \theta_{\tilde{c}})$ with $\theta_\rho = \Pr_y[1|\Phi = \rho]$, where the random variable $\Phi \in \{0, \cdots, \tilde{c}\}$ denotes the number of symbol "1" in the colluders' copies at a given index. Furthermore, the Worst Case Attack(WCA) is defined as the collusion attack minimizing the rate of the code, or equivalently, the asymptotic positive error exponent. For example, when $\tilde{c} = 5$, the parameter vector of WCA is given by $\theta^\star = (0, 0.594, 0.000, 1.00, 0.406, 1)$.

On the other hand, the attack strategies are not limited to the above types in a realistic situation such that a codeword is binary and each bit is embedded

into one of segments of a digital content without overlapping using a robust watermarking scheme. It is reasonable to assume that each bit is embedded into a segment using an antipodal signal: $\hat{X}_{j,i} = 2X_{j,i} - 1$, namely it is binary phase shift keying(BPSK) modulation. In this case, colluders can apply the other attack strategy at the detectable positions. Since each bit of codeword of $\hat{\boldsymbol{y}}$ is one of $\{-1, 1\}$ after the BPSK modulation, it is possible for colluders to alter the signal amplitude of each element from the signal processing point of view. One simple example is averaging attack that $\hat{y}_i = \sum \hat{X}_{j,i}/c$, we call this attack "average(ave)". Considering the removal of fingerprint signal, a worst case may be $\hat{y}_i = 0$. At the detectable position, it is sufficient to average only two segments whose $\hat{X}_{j,i}$ are different with each other, which attack is denoted by "average2(ave2)".

Even if a robust watermarking method is used to embed the binary fingerprinting code into digital contents, it must be degraded by attacks. For convenience, the distortion is modeled as AWGN in this study. So, we assume that a pirated copy is produced by one of the above collusion strategies and is further distorted by the Gaussian noise.

2.3 Conventional Tracing Algorithm

Assuming that the pirated codeword $\hat{\boldsymbol{y}}$ is transmitted over AWGN channel. Then, the codeword extracted from a pirated copy is represented by analog value:

$$\boldsymbol{y'} = \hat{\boldsymbol{y}} + \boldsymbol{e} = (\hat{y}_1 + e_1, \ldots, \hat{y}_L + e_L), \tag{4}$$

because of the addition of noise $\boldsymbol{e}$ that follows $N(0, \sigma_e^2)$. If a tracing algorithm strictly follows the definition, each extracted symbol of the pirated codeword should be rounded into a bit $\{-1, 1\}$ when the symmetric version of the tracing algorithm is used. Because of the rounding operation, this procedure is called a hard decision (HD) method in [7] and [6]. On the other hand, it is possible to directly calculate the correlation sum $S^{(j)}$ from the distorted pirated codeword $\boldsymbol{y'}$, which procedure is called a soft decision (SD) method. A soft decoding method is very beneficial in error correcting code, so it is worthy to try for fingerprinting. However, in the SD method, the likelihood of the received signal is not considered to maximize the traceability. It is strongly required for the soft decision method to calculate the correlation score based on the information theoretic analysis.

3 Proposed Tracing Algorithm

The proposed tracing algorithm first estimates the amount of noise involved in a pirated copy and then measures the likelihood of each symbol of pirated copy. Using the likelihood, the correlation score is calculated and guilty users are identified with a constant false probability ϵ_1.

3.1 Channel Estimation

The accurate estimation of the Gaussian channel can maximize the performance of tracing algorithm. The estimator proposed in [7] does not make use of all the available samples, but only half samples in average. In addition, it only estimates the variance σ_e^2 of Gaussian noise. In this paper, we estimate the probability distribution function that is regarded as a Gaussian mixture model.

If a collusion strategy is based on the marking assumption, each symbol of a pirated codeword is $\hat{y}_i \in \{-1, 1\}$. Here, the probability $\Pr[\hat{y}_i = 1]$ is not always equal to $\Pr[\hat{y}_i = -1]$. So, the probability distribution function $pdf(y'_i)$ is represented by

$$pdf(y'_i) = aN(y'_i; 1, \sigma_e^2) + (1 - a)N(y'_i; -1, \sigma_e^2), \tag{5}$$

where $a \geq 0$ and

$$N(y'_i; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(- \frac{(y'_i - \mu)^2}{2\sigma^2} \right). \tag{6}$$

Under the relaxed version of the marking assumption, the value of $\hat{y}_i$ is not limited to these two symbols. Hence, the probability distribution function can be a mixture of several Gaussian components, and in general, it is denoted by

$$pdf(y'_i) = \sum_{k=1}^{m} a_k N(y'_i; \mu_k, \sigma_k^2), \tag{7}$$

where m is the number of Gaussian components, and $\sum_{k=1}^{m} a_k = 1$ and $a_k \geq 0$.

Thanks to the EM algorithm [1], we can derive unknown parameters a_k, μ_k, and σ_k^2 from $\boldsymbol{y'}$ and $pdf(y'_i)$. The EM algorithm is a well-established maximum likelihood algorithm for fitting a mixture model to a set of training data. The algorithm is an iterative method which alternates between performing an expectation(E)-step and a maximization(M)-step. The E-step computes the expectation of the log-likelihood evaluated from the current estimate for the latent variables, and the M-step computes parameters maximizing the expected log-likelihood found on the E-step. Because it is very popular to estimate the parameters of Gaussian mixture model using the EM algorithm, we only describe the procedure to estimate the unknown parameters in this paper (see [1] for detail).

Let $\boldsymbol{\Theta}$ be a vector of unknown parameters a_k, μ_k, and σ_k^2. The log-likelihood function $L(\boldsymbol{y'}, \boldsymbol{\Theta})$ with respect to $\boldsymbol{y'}$ is represented by

$$L(\boldsymbol{y'}, \boldsymbol{\Theta}) = \log \Pr[\boldsymbol{y'}, \boldsymbol{\Theta}] = \sum_{i=1}^{L} \log \left(\sum_{k=1}^{m} a_k N(y'_i; \mu_k, \sigma_k^2) \right). \tag{8}$$

The goal is to maximize the posterior probability of the parameters $\boldsymbol{\Theta}$ from $\boldsymbol{y'}$ in the presense of hidden parameters $\boldsymbol{\xi}$. The EM algorithm seeks to find the maximum likelihood estimate of $L(\boldsymbol{y'}, \boldsymbol{\Theta})$ by iteratively applying the following two steps:

– E-step: Calculate the conditional distribution of $\xi_{k,i}$ under the current estimate of the parameters $\boldsymbol{\Theta}^{(t)}$:

$$\xi_{k,i} = \frac{a_k N(y'_i; \mu_k, \sigma_k^2)}{\sum_{h=1}^{m} a_h N(y'_i; \mu_h, \sigma_h^2)} \tag{9}$$

– M-step: Calculate the estimated parameters $\boldsymbol{\Theta}^{(t+1)}$ that maximize the expected value of $L(\boldsymbol{y'}, \boldsymbol{\Theta}^{(t+1)})$ using $\boldsymbol{\xi}$:

$$a_k = \frac{1}{N} \sum_{i=1}^{L} \xi_{k,i}, \tag{10}$$

$$\mu_k = \frac{\sum_{i=1}^{L} \xi_{k,i} y'_i}{\sum_{i=1}^{L} \xi_{k,i}}, \tag{11}$$

and

$$\sigma_k^2 = \frac{\sum_{i=1}^{L} \xi_{k,i} (y'_i - \mu_k)^2}{\sum_{i=1}^{L} \xi_{k,i}}. \tag{12}$$

The above E-step and M-step are iteratively performed until $|L(\boldsymbol{y'}, \boldsymbol{\Theta}^{(t+1)}) - L(\boldsymbol{y'}, \boldsymbol{\Theta}^{(t)})| < T_L$ for an appropriately designed threshold T_L. The EM algorithm is known to converge in finite iterations for an arbitrary T_L.

An important property of the EM algorithm is that it is not guaranteed to converge to the global optimum. Instead, it stops at some local optimums, which can be much worse than the global optimum. In our model, the following constraints on the above parameters improve the accuracy of the performance. At least, we have two values $\hat{y}_i = \pm 1$ under the our attack model, and hence, we fix

$$\mu_1 = 1, \tag{13}$$

$$\mu_2 = -1. \tag{14}$$

All variances σ_k^2 are equal because $\hat{y}_i$ is distorted only by Gaussian noise.

If the "average" or "average2" attack is performed, the number of Gaussian components is at most $m = 3$; otherwise, $m = 2$ for collusion strategies under the marking assumption. When $m = 3$, the EM algorithm must estimate the following five parameters: a_1, a_2, a_3, μ_3 and $\sigma_e^2 (= \sigma_1^2 = \sigma_2^2 = \sigma_3^2)$. On the other hand, among these five parameters, a_3 and μ_3 are omitted when $m = 2$. Hence, the accuracy of the estimation at $m = 2$ is much better because the number of unknown parameters is reduced. Thus, the accurate estimation of m

will further improve the performance of EM algorithm when the number m is properly estimated.

For the estimation of m, we need to find the collusion strategy selected for producing a pirated copy. In [4], the EM algorithm is applied for the estimation of the collusion strategy. However, the experimental results indicate that the accuracy of the estimation is getting worse for more colluders and/or more harmful process. In our case, even if we wrongly estimate $m = 3$, the estimated parameters are not always bad. For example, when $a_3 = 0$ or $\mu_3 = 0$ in the case $m = 3$, the other parameters will be coincident with the case $m = 2$. So, we roughly determine m as follows:

$$m = \begin{cases} 2 & \text{if } \lambda(\boldsymbol{y'}) \geq L/2 \\ 3 & \text{otherwise,} \end{cases} \tag{15}$$

where $\lambda(\boldsymbol{y'})$ is the number of elements satisfying $|y'_i| \geq 1$.

3.2 Correlation Score

Suppose that we transmit over a Gaussian channel with input $\hat{\boldsymbol{y}}$ and output $\boldsymbol{y'}$. Now, the probability distribution function is given by Eq.(5). Here, we start with the case $m = 2$. Then,

$$\Pr[\hat{y}_i = 1|y'_i] = \frac{a_1 N(y'_i; \mu_1, \sigma_e^2)}{a_1 N(y'_i; \mu_1, \sigma_e^2) + a_2 N(y'_i; \mu_2, \sigma_e^2)}, \tag{16}$$

and

$$\Pr[\hat{y}_i = -1|y'_i] = \frac{a_2 N(y'_i; \mu_2, \sigma_e^2)}{a_1 N(y'_i; \mu_1, \sigma_e^2) + a_2 N(y'_i; \mu_2, \sigma_e^2)}. \tag{17}$$

In a noiseless case, we get $y'_i = \hat{y}_i$, and the correlation score $S_i^{(j)}$ is calculated by Eq.(2). Considering the above probabilities in a noisy case, Eq.(2) is rewritten by

$$S_i^{(j)} = 1 \cdot \Pr[\hat{y}_i = 1|y'_i] U_{j,i} + (-1) \cdot \Pr[\hat{y}_i = -1|y'_i] U_{j,i}, \tag{18}$$

$$= \frac{a_1 N(y'_i; \mu_1, \sigma_e^2) - a_2 N(y'_i; \mu_2, \sigma_e^2)}{a_1 N(y'_i; \mu_1, \sigma_e^2) + a_2 N(y'_i; \mu_2, \sigma_e^2)} U_{j,i}. \tag{19}$$

Next, we generalize the above discussion. Now, we get the following probabilities:

$$\Pr[\hat{y}_i = 1|y'_i] = \frac{a_1 N(y'_i; \mu_1, \sigma_e^2)}{\sum_{k=1}^{m} a_k N(y'_i; \mu_k, \sigma_e^2)}, \tag{20}$$

and

$$\Pr[\hat{y}_i = -1|y'_i] = \frac{a_2 N(y'_i; \mu_2, \sigma_e^2)}{\sum_{k=1}^{m} a_k N(y'_i; \mu_k, \sigma_e^2)}. \tag{21}$$

Therefore, the correlation score $S_i^{(j)}$ is generally represented by

$$S_i^{(j)} = \frac{a_1 N(y_i'; \mu_1, \sigma_e^2) - a_2 N(y_i'; \mu_2, \sigma_e^2)}{\sum_{k=1}^{m} a_k N(y_i'; \mu_k, \sigma_e^2)} U_{j,i}. \tag{22}$$

3.3 Threshold

A simple approach to estimate the false-positive probability is to perform the Monte Carlo simulation. Indeed, it is not easy in general because of the heavy computational costs for estimating a tiny probability. Furon et al. proposed an efficient method estimating the probability of rare events [5]. The method can estimate the false-positive probability ϵ_1 for a given threshold Z, which means that the method calculates the map $\epsilon_1 = F(Z)$. Once the relations are obtained, it is sufficient to store them as a reference table. In other word, this method must be iteratively performed to obtain an objective threshold for a given ϵ_1.

In [7], an easy method to obtain a threshold for a given ϵ_1 has been proposed. The method is based on the CLT. At first, it calculates the variance of the correlation sum $S^{(\tilde{j})}$ such that an $\tilde{j}$-th codeword is randomly generated one and is not assigned to any user in a fingerprinting system. For a sufficient number of $\tilde{j}$, the variance σ_S^2 of $S^{(\tilde{j})}$ is calculated by $\sum (S^{(\tilde{j})} - E[S^{(\tilde{j})}])^2$, where $E[x]$ is the expectation of x. Because of the Gaussian approximation based on the CLT, the threshold Z for a given ϵ_1 can be calculated as follows:

$$Z = \sqrt{2\sigma_S^2} \cdot \mathrm{erfc}^{-1}(2\epsilon_1). \tag{23}$$

The disadvantage of this method is the uncertainty-based approximation because there is an argument about the validity of CLT applying for the estimation of ϵ_1.

Our main interest in this paper is to evaluate the traceability of the proposed detector compared with the conventional one. So, we roughly calculate the threshold Z by Eq.(23) for a given ϵ_1, and then, derive $F(Z)$ as the actual false-positive probability.

4 Equalization of Probability

Because of the symmetry of the bias distribution $f(p)$, it is expected to be $\Pr[\hat{y}_i = 1] = \Pr[\hat{y}_i = -1]$ unless the colluders do not know the actual values $X_{j,i}$ of their codewords. However, when they happen to get the values contained in segments, they can perform more active collusion strategies such as "all-0" and "all-1". Such a scenario is defined in [3] as the cryptographic colluders. Then, $\Pr[\hat{y}_i = 1]$ is not always equal to $\Pr[\hat{y}_i = -1]$. Under this condition, we reconsider the optimality of the proposed detector.

If the parameters a_1 and a_2 are accurately estimated by the EM algorithm,

$$\Pr[\hat{y}_i = 1] = a_1, \tag{24}$$

and

$$\Pr[\hat{y}_i = -1] = a_2. \tag{25}$$

Because of the imbalance between $\Pr[\hat{y}_i = 1]$ and $\Pr[\hat{y}_i = -1]$, it occurs the bias between the first term $\Pr[\hat{y}_i = 1|y'_i]U_{j,i}$ and the second term $\Pr[\hat{y}_i = -1|y'_i]U_{j,i}$ in Eq.(22). In order to equalize the bias of these probabilities, the correlation score $S_i^{(j)}$ is modified as follows:

$$\begin{aligned} S_i^{(j)} &= 1 \cdot \frac{\Pr[\hat{y}_i = 1|y'_i]}{\Pr[\hat{y}_i = 1]} U_{j,i} + (-1)\frac{\Pr[\hat{y}_i = -1|y'_i]}{\Pr[\hat{y}_i = -1]} U_{j,i}, \\ &= \frac{N(y'_i; \mu_1, \sigma_e^2) - N(y'_i; \mu_2, \sigma_e^2)}{\sum_{k=1}^{m} a_k N(y'_i; \mu_k, \sigma_e^2)} U_{j,i}. \end{aligned} \tag{26}$$

This modification also changes the distribution of the correlation sum S_j, and hence, the corresponding threshold must be accommodated. Thanks to the method in Sect.3.3, it is easy to derive the threshold Z under the above conversion of $S_i^{(j)}$.

5 Experimental Results

For the comparison of the performance of proposed methods, the number of detected colluders and the false-positive probability are evaluated for the Nuida code under the following conditions. The length is $L = 5000$, the number of users is $N = 10^4$ and the false-positive probability is $\epsilon_1 = 10^{-8}$. Under this condition, the total false-positive probability η is approximated to be 10^{-4}. In our attack model, a pirated codeword is produced by collusion attack using randomly selected 10^5 combinations of $\tilde{c} = 8$ colluders and it is distorted by additive white Gaussian noise. The performance of the tracing algorithms is evaluated by changing SNR. Using a threshold Z calculated by Eq.(23), η is evaluated by $F(Z)$ as well as the Monte Carlo simulation. We denote the detector proposed in Sect.3 and Sect.4 by "method Ⅰ" and "method Ⅱ", respectively. The threshold for the EM algorithm is set to be $T_L = 0.01$. In order to reduce the computational costs required for each trial of a Monte Carlo simulation, the number of iterations for the EM algorithm is limited to be 100 at most.

The number of detectable colluders under the "majority" attack is plotted in Fig.1. It is observed that both of the proposed methods approach to that of SD method in the decrease of SNR, and that the method Ⅱ outperforms the other methods. The reason why the traceability of method Ⅰ is dropping with the increase of SNR comes from the wrong estimation of parameters in the EM algorithm. Such a wrong estimation is occurred in the case that the estimator judges $m = 3$ when in fact $m = 2$. By intensively measuring the estimated

values, we found that μ_3 is very close to one of μ_1 and μ_2 in many cases. It means that the EM algorithm finds only two distribution in spite of the wrong judgment of $m = 3$. In case $\mu_3 \approx 1 (= \mu_1)$, we see $\Pr[\hat{y}_i = 1] = a_1 + a_3$, but it is judged $\Pr[\hat{y}_i = 1] = a_1$ by mistake in the proposed method I, which affects on the probability $\Pr[\hat{y}_i = 1 | y'_i]$. As the result, the score $S_i^{(j)}$ given by Eq.(22) is affected by the miscalculation in the method I. By contrast, the score $S_i^{(j)}$ in Eq.(26) in the method II is stable for the miscalculation. Assuming an ideal case that the EM algorithm can estimate the parameters with no error, the performance of the proposed methods is evaluated under a same condition. For the comparison, we plot the results of ideal case by solid lines and the actual values by dotted lines in Fig.2. We can see that the traceability of method I is very close to, but is slightly lower than that of method II in an ideal case. For further comparison, we check the performance in the ideal case under the other collusion attacks for 10^3 trials of Monte Carlo simulation, which results are described in Fig.3. Notice that the results of method II under "all-0" and "all-1" collusion strategies are much higher than that of method I. It comes from the effect of equalization explained in Sect.4. From this result, we can say that colluders can not get any benefit from the information of symbols embedded in a copy. Under the "WCA", we also evaluate the performance for 10^5 trials of Monte Carlo simulation, which results are plotted in Fig.4. The results are almost equal to those of the "majority" attack.

Even if the score of innocent users can be approximated by a Gaussian distribution, the probability of false-positive cannot be simply expressed by Gauss error function. The total false-positive probabilities under the "majority" attack and "WCA" are plotted in Fig.5. In these figures, the solid and dotted lines are the results derived from the experiment and $F(Z)$, respectively. Although the experimental results are slightly dispersed because the number of Monte Carlo simulation is only 10^5, they are almost equal to $F(Z)$ and are less than a given probability $\eta = 10^{-4}$. It means that the Gaussian approximation based on the CLT for calculating the threshold Z is not bad under this condition.

In order to numerically compare the performance against collusion strategies, the number of detected colluders and the total false-positive probability are summarized in Table 1 and Table 2, respectively. As a whole, it is observed that the traceability of the method II is better than that of the method I, and the method II outperforms the conventional methods. It is remarkable that the total false-positive probability of "minority" attack is the worst one among 8 collusion strategies under this experimental condition. Since our scope in this paper is not to evaluate the validity of Gaussian assumption, but to calculate a proper correlation score $S_i^{(j)}$ under the noisy environment, the design of appropriate threshold Z is not deeply discussed and we merely employ the Gaussian assumption to calculate Z for a given ϵ_1 for its simplicity. Indeed, the use of rare event simulator $F(Z)$ can be a better method for designing the threshold though it requires an iterative search for obtaining an objective threshold for a given ϵ_1.

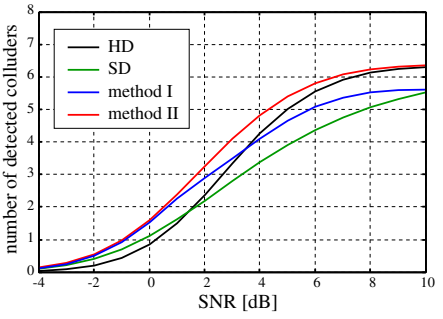


Fig. 1. Comparison of the traceability under the majority attack for $L = 5000$, $\tilde{c} = 8$, and $\epsilon_1 = 10^{-8}$

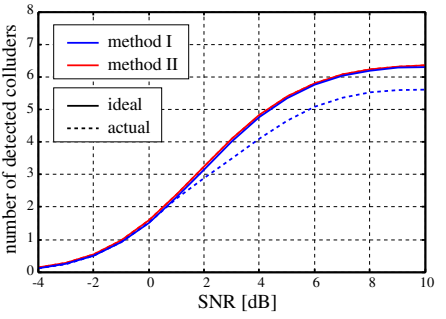


Fig. 2. Comparison of the traceability of ideal case under the majority attack for $L = 5000$, $\tilde{c} = 8$, and $\epsilon_1 = 10^{-8}$

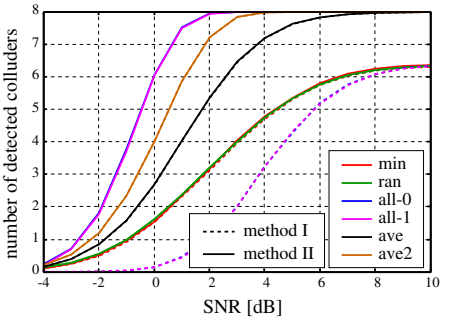


Fig. 3. Comparison of the traceability under various collusion strategies for $L = 5000$, $\tilde{c} = 8$, and $\epsilon_1 = 10^{-8}$

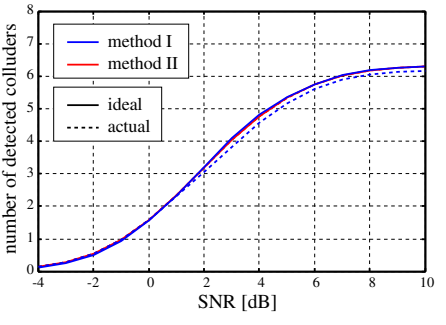
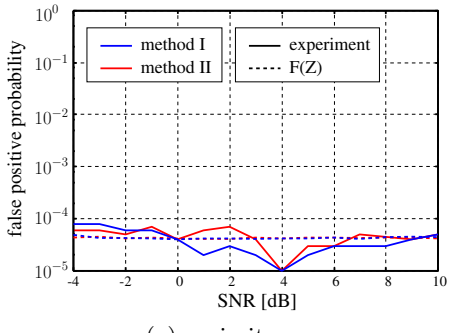
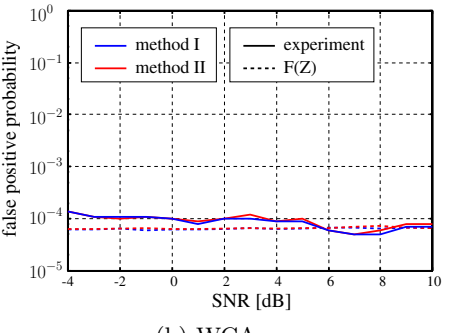


Fig. 4. Comparison of the traceability under the WCA for $L = 5000$, $\tilde{c} = 8$, and $\epsilon_1 = 10^{-8}$



(a) majority



(b) WCA

Fig. 5. Comparison of the total false-positive probability η for $L = 5000$, $\tilde{c} = 8$, and $\epsilon_1 = 10^{-8}$

Table 1. Number of detected colluders for 8 collusion strategies when $L = 5000$ and $\tilde{c} = 8$

SNR [dB]	detector	collusion strategy							
		maj	min	ran	all-0	all-1	ave	ave2	WCA
−4	HD	0.037	0.042	0.040	0.040	0.039	0.038	0.040	0.039
	SD	0.109	0.123	0.117	0.118	0.116	0.187	0.243	0.113
	method I	0.120	0.132	0.137	0.001	0.001	0.179	0.202	0.134
	method II	0.132	0.145	0.137	0.262	0.260	0.179	0.202	0.135
0	HD	0.860	0.881	0.874	0.878	0.865	0.864	0.872	0.868
	SD	1.125	1.132	1.132	1.157	1.140	2.779	4.058	1.128
	method I	1.532	1.545	1.579	0.172	0.165	2.586	3.872	1.574
	method II	1.596	1.608	1.579	6.079	6.069	2.780	4.060	1.574
4	HD	4.270	4.242	4.257	4.268	4.241	4.258	4.250	4.255
	SD	3.425	3.391	3.410	3.471	3.440	7.240	7.969	3.413
	method I	4.109	4.107	4.547	5.418	5.394	7.132	7.984	4.562
	method II	4.822	4.817	4.768	8.000	8.000	7.138	7.985	4.762
8	HD	6.144	6.131	6.143	6.148	6.131	6.139	6.135	6.137
	SD	5.097	5.078	5.098	5.160	5.127	7.972	8.000	5.097
	method I	5.604	5.531	6.041	7.016	7.006	7.547	8.000	6.054
	method II	6.228	6.232	6.183	8.000	8.000	7.966	8.000	6.174

Table 2. False-positive probability $\eta[\times 10^{-4}]$ experimentally derived for 8 collusion strategies when $L = 5000$ and $\tilde{c} = 8$, where the values in parenthesis are $F(Z)$

SNR [dB]	detector	collusion strategy							
		maj	min	ran	all-0	all-1	ave	ave2	WCA
−4	method I	0.8	1.5	0.3	0.7	0.9	0.1	0.3	1.4
		(0.432)	(1.183)	(0.734)	(0.227)	(0.233)	(0.504)	(0.720)	(0.620)
	method II	0.6	1.3	0.3	0.4	0.3	0.1	0.3	1.4
		(0.432)	(1.183)	(0.734)	(0.227)	(0.233)	(0.504)	(0.720)	(0.620)
0	method I	0.4	0.8	0.4	0.7	1.4	0.2	0.3	1.0
		(0.411)	(1.202)	(0.693)	(0.073)	(0.077)	(0.252)	(0.347)	(0.628)
	method II	0.4	1.0	0.5	0.1	0.0	0.2	0.3	1.0
		(0.403)	(1.202)	(0.693)	(0.073)	(0.077)	(0.252)	(0.347)	(0.628)
4	method I	0.1	1.6	0.6	0.6	1.2	0.2	0.1	0.9
		(0.414)	(1.215)	(0.716)	(0.039)	(0.038)	(0.245)	(0.148)	(0.636)
	method II	0.1	1.2	1.0	0.0	0.3	0.1	0.1	0.9
		(0.414)	(1.212)	(7.416)	(0.038)	(0.038)	(0.245)	(0.148)	(0.636)
8	method I	0.3	1.2	0.7	0.1	1.0	0.0	0.0	0.5
		(0.431)	(1.221)	(0.699)	(0.029)	(0.032)	(0.035)	(0.079)	(0.644)
	method II	0.4	1.1	0.7	0.0	0.2	0.0	0.0	0.6
		(0.431)	(1.221)	(0.699)	(0.029)	(0.032)	(0.035)	(0.079)	(0.644)

6 Conclusion

In this paper, we proposed a soft decision tracing algorithm to catch more colluder even if a pirated codeword is distorted by Gaussian noise. We first estimate the parameters of Gaussian channel using the EM algorithm by giving some constrains. Then, the correlation score is calculated using the posterior probability of each symbol of received codeword. Considering the bias between the probability of symbols, we give a weight on the posterior probability. The experimental results show that the proposed method without the weighting requires an accurate estimation of the number of Gaussian mixture model to get a best performance, and the method with the weighting is not so sensitive for such an estimation. For the specific collusion strategies such as "all-0" and "all-1", it is confirmed from our experiment that the weighting effectively enhances the performance of tracing algorithm.

Although the proposed method is specified for AWGN channel, it can be extended for further complicated attack channels by tuning the EM algorithm. For example, if additive *colored* Gaussian noise is injected to a pirated codeword, we must estimate the mean values μ_1 and μ_2, while they are fixed under the AWGN channel. Furthermore, when the distribution of additive noise is modeled by a certain distribution such as Laplace and Rayleigh distributions, it is sufficient to replace the Gaussian term $N(y'_i; \mu, \sigma^2)$ appeared in this paper with the modeled one.

References

1. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, Heidelberg (2006)
2. Škorić, B., Vladimirova, T.U., Celik, M., Talstra, J.C.: Tardos fingerprinting is better than we thought. IEEE Trans. Inform. Theory 54(8), 3663–3676 (2008)
3. Furon, T., Guyader, A., Cérou, F.: On the design and optimization of Tardos probabilistic fingerprinting codes. In: Solanki, K., Sullivan, K., Madhow, U. (eds.) IH 2008. LNCS, vol. 5284, pp. 341–356. Springer, Heidelberg (2008)
4. Furon, T., Preire, L.P.: EM decoding of Tardos traitor tracing codes. ACM Multimedia and Security, 99–106 (2009)
5. Furon, T., Preire, L. P., Guyader, A., Cérou, F.: Estimating the minimal length of Tardos code. In: Katzenbeisser, S., Sadeghi, A.-R. (eds.) IH 2009. LNCS, vol. 5806, pp. 176–190. Springer, Heidelberg (2009)
6. Kuribayashi, M.: Experimental assessment of probabilistic fingerprinting codes over AWGN channel. In: Echizen, I., Kunihiro, N., Sasaki, R. (eds.) IWSEC 2010. LNCS, vol. 6434, pp. 117–132. Springer, Heidelberg (2010)
7. Kuribayashi, M.: Tardos's fingerprinting code over AWGN channel. In: Böhme, R., Fong, P.W.L., Safavi-Naini, R. (eds.) IH 2010. LNCS, vol. 6387, pp. 103–117. Springer, Heidelberg (2010)
8. Kuribayashi, M., Morii, M.: Systematic generation of Tardos's fingerprinting codes. IEICE Trans. Fundamentals E93-A(2), 508–515 (2009)
9. Nuida, K.: Making collusion-secure codes (more) robust against bit erasure. eprint 549, 2009 (2009)

10. Nuida, K., Fujitu, S., Hagiwara, M., Kitagawa, T., Watanabe, H., Ogawa, K., Imai, H.: An improvement of discrete Tardos fingerprinting codes. Designs, Codes and Cryptography 52(3), 339–362 (2009)
11. Nuida, K., Hagiwara, M., Watanabe, H., Imai, H.: Optimization of Tardos's fingerprinting codes in a viewpoint of memory amount. In: Furon, T., Cayre, F., Doërr, G., Bas, P. (eds.) IH 2007. LNCS, vol. 4567, pp. 279–293. Springer, Heidelberg (2008)
12. Simone, A., Skoric, B.: Accusation probabilities in Tardos codes: the Gaussian approximation is better than we thought. Cryptology ePrint Archive, Report 2010/472 (2010), http://eprint.iacr.org/
13. Tardos, G.: Optimal probabilistic fingerprint codes. J. ACM 55(2), 1–24 (2008)
14. Wu, M., Trappe, W., Wang, Z.J., Liu, K.J.R.: Collusion resistant fingerprinting for multimedia. IEEE Signal Processing Mag., 15–27 (2004)

REASSURE: A Self-contained Mechanism for Healing Software Using Rescue Points

Georgios Portokalidis and Angelos D. Keromytis

Network Security Lab, Department of Computer Science,
Columbia University, New York, NY, USA
{porto,angelos}@cs.columbia.edu

Abstract. Software errors are frequently responsible for the limited availability of Internet Services, loss of data, and many security compromises. Self-healing using rescue points (RPs) is a mechanism that can be used to recover software from unforeseen errors until a more permanent remedy, like a patch or update, is available. We present REASSURE, a self-contained mechanism for recovering from such errors using RPs. Essentially, RPs are existing code locations that handle certain anticipated errors in the target application, usually by returning an error code. REASSURE enables the use of these locations to also handle unexpected faults. This is achieved by rolling back execution to a RP when a fault occurs, returning a valid error code, and enabling the application to gracefully handle the unexpected error itself. REASSURE can be applied on already running applications, while disabling and removing it is equally facile. We tested REASSURE with various applications, including the MySQL and Apache servers, and show that it allows them to successfully recover from errors, while incurring moderate overhead between 1% and 115%. We also show that even under very adverse conditions, like their continuous bombardment with errors, REASSURE protected applications remain operational.

1 Introduction

Program errors or bugs are ever-present in software, and specially in large and highly complex code bases [20]. They manifest as application crashes or unexpected behavior and can cause significant problems, like limited availability of Internet services [22], loss of user data [11], or lead to system compromise [24]. Many attempts have been made to increase the quality of software and reduce the number of bugs. Companies enforce strict development strategies and educate their developers in proper development practices, while static and dynamic analysis tools are used to assist in bug discovery [2,5]. However, it has been established that it is extremely difficult to produce completely error-free software.

To alleviate some of the dangers that bugs like buffer overflows and dangling pointers entail, various containment and runtime protection techniques have been proposed [8,1,7,12,18]. These techniques can offer assurances that certain types of program vulnerabilities cannot be exploited to compromise security,

T. Iwata and M. Nishigaki (Eds.): IWSEC 2011, LNCS 7038, pp. 16–32, 2011.

but they do not also offer high availability and reliability, as they frequently terminate the compromised program to prevent the attacker from performing any useful action.

In response, researchers have devised novel mechanisms for recovering execution in the presence of errors [13]. ASSURE [26], in particular, presents a powerful system that enables applications to automatically self-heal. Its operation revolves around the understanding that programs usually include code for handling certain anticipated errors, and it introduces the concept of rescue points (RPs), which are locations of error handling code that can be reused to gracefully recover from unexpected errors. In ASSURE, RPs are the product of offline analysis that is triggered when a new and unknown error occurs, but they can also be the result of manual analysis. For example, RPs can by identified by examining the memory dump produced when a program abnormally terminates. Also, they serve a dual role, first they are the point where execution can be rolled back after an error occurs, and second they are responsible for returning a valid and meaningful error to the application (*i.e.,* one that will allow it to resume normal operation).

Regrettably, deploying RPs using ASSURE is not straightforward, but it demands that various complex systems are present. For instance, to support execution rollback, applications are placed inside the Zap [19,15] virtual execution environment, while RP code is injected using Dyninst [4]. Zap is a considerably complex component that is tightly coupled with the Linux kernel, and requires maintenance along with the operating system (OS). In practice, RPs are a useful but temporary solution for running critical software until a proper solution, in the form of a dynamic patch or update, is available. It is our opinion that RPs have not been widely used mainly because of the numerous requirements, in terms of additional software and setup, of previous solutions like ASSURE.

We propose REASSURE, a self-contained mechanism for healing software using RPs. REASSURE assumes that a RP has already been identified, and needs to be deployed quickly and in a straightforward manner. It builds on Intel's PIN dynamic binary instrumentation (DBI) framework to install the RP and provide the virtual execution environment for rolling back execution. As Pin itself is simply an application, installation is simple and very little maintenance (or none at all) is necessary. Furthermore, REASSURE does not need to be continuously operating or even present, but it can be easily installed and attached only when needed. Disabling it and removing it from a system is equally uncomplicated, since it can be detached from a running application without interrupting its operation. Combined with a dynamic patching mechanism [4,9,17], applications protected with REASSURE can be run and eventually patched without any interruption.

We have implemented REASSURE as a Pin tool for Linux[1]. Our evaluation with popular servers, like Apache and MySQL, that suffer from well known vulnerabilities shows that REASSURE successfully prevents the protected applications from terminating. When no faults occur, the performance overhead

[1] Interested readers can contact the authors for a copy.

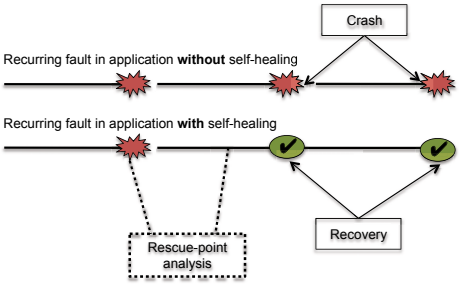


Fig. 1. Software self-healing overview. A faulty application will crash and need to be restarted every time a fault occurs. With self-healing, an analysis of the fault when it first occurs, results in the definition of a *rescue point* for the application, which allows it to gracefully recover from future occurrences of the same fault.

imposed by REASSURE varies between 1% and 115% depending on the application, while in the presence of errors there is little effect on the protected application until the frequency of faults surpasses five faults per second. We should also note that Pin supports multiple platforms (*e.g.,* Windows and Mac OS), and REASSURE can be extended to support them with little effort.

This paper is organized as follows: Section 2 presents an overview of software healing using RPs. We describe REASSURE in Sect. 3, and evaluate its effectiveness and performance in Sect. 4. Section 5 discusses limitations and future work, while related work is discussed in Sect. 6. We conclude in Sect. 7.

2 Software Self-healing Using Rescue Points

Software self-healing using RPs was first proposed in ASSURE [26], where the authors describe an architecture that enables unmodified applications to automatically heal themselves in the presence of unanticipated faults. An overview of the idea behind this scheme is presented in Fig. 1. The architecture can be decomposed into two parts. The first, is responsible for generating a RP when an unexpected error occurs, while the second is in charge of applying the produced RP on the application and recovering from future errors.

2.1 What Is a Rescue Point?

We define a rescue point as a function, preceding and encapsulating code suffering from an fault (*i.e.,* the fault it aims to mend) that contains error handling code, which can be reused to gracefully handle the unexpected error. For instance, consider the function shown in Fig. 2. It calls three other functions, namely *f1()*, *f2()*, and *f3()*. Let's assume that *f3()* contains a bug, which if triggered will terminate the application. We observe that *f3()* does not return any value, which means that it either always succeeds or simply does not handle certain conditions, such as the one causing the fault. On the other hand, the

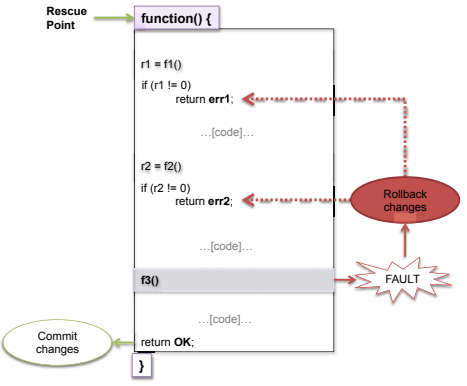


Fig. 2. Rescue point example. The function shown contains error handling code which can be used to handle errors occurring in the faulty *f3()* function.

function encompassing it contains code that handles erroneous conditions, like *f1()* and *f2()* returning an error. Therefore, we can use this function as a RP that will enable the application to self-heal from an error in *f3()*.

2.2 Rescue Point Discovery

ASSURE described a mechanism to automatically discover possible RPs and select the best fit to deploy in terms of survivability after an error occurs. Briefly, the procedure starts by profiling the application before it is deployed to discover all possible RPs. This is achieved by monitoring the values returned by the application's functions, as it is provided with fuzzed and faulty inputs. Later, when it is deployed and running normally, ASSURE takes periodic checkpoints of the application state and maintains an execution log that includes network traffic by running the application within Zap.

Concurrently, it monitors the application to detect failures and misbehavior. The simplest way to achieve this is to intercept signals such as a segmentation fault that indicates improper memory handling. Other approaches that detect memory errors can also be employed [18,8,1,21]. When an error is detected ASSURE initiates offline rescue point analysis (see Fig. 1) in a replica, which returns the application to the last checkpoint before the fault and attempts to reproduce the fault by replaying the execution log. The aim of this analysis is to detect the location of the error, thus enabling the selection of an appropriate RP. Interested readers are referred to [26] for detailed information.

Alternatively, RPs can be discovered manually. For instance, an application terminating due to a segmentation fault can be configured to dump core, a file that describes the state of the application at the time of the fault. Processing the dumped core can reveal the function containing the fault, which can be frequently used as a RP itself or assist the user to find a nearby RP fit to handle the error.

2.3 Rescue Point Deployment

In ASSURE, RPs are deployed using two systems. First, Dyninst [4] is employed to inject special code in the beginning of the corresponding function that checkpoints the application, and in case of an error returns a valid error code. Second, the Zap-based virtual environment is used to actually perform the checkpoint, as well as rollback the application when an error occurs. In the latter case, execution returns in the RP, which returns an error.

Using Zap enabled ASSURE to keep overhead low and achieve fast recovery times. Unfortunately, deploying RPs in this fashion is not very practical. Zap requires extensive modifications to the OS and cannot be dynamically installed and removed. Software self-healing targets systems that require temporary protection against known bugs until an official patch is available that properly addresses the error. As such, users are reluctant to install and maintain the additional software required to deploy ASSURE.

We offer an attractive alternative that simplifies RP deployment in the form of a self-contained mechanism built using Intel's Pin dynamic instrumentation framework. Our tool, REASSURE, only requires the Pin framework which operates on stock software and hardware. It can be dynamically applied for as long as it is required. For example, until the application is updated, or until an ingress filtering mechanism is used to block the inputs causing the fault. Afterward, it can detach itself from the application and be removed from the system.

3 REASSURE Implementation

3.1 The Pin DBI Framework

Pin [16] enables the development of tools that can augment, modify, or simply monitor a binary's execution at the instruction level. It provides a rich API that can be used by developers of tools (Pintools) to install callbacks to inspect a program's instructions and routines, as well as intercept system calls and signals. In Pin's terms, it allows the *instrumentation* of the application. Additionally, instrumentation routines can modify original code by removing instructions or by more frequently adding new code, referred to as *analysis* code. The instrumented application executes on top of Pin's virtual machine (VM) runtime, which essentially consists of a just-in-time (JIT) compiler that combines the original and analysis instructions, and places the produced code blocks into a code cache, where the application executes from.

The same block of application code can be instrumented in different ways through *versioning*. Every application thread initially executes in version zero, which corresponds to the default code cache. Instrumentation code can change the version of a running thread by adding analysis code that will change the version of the thread executing a particular instruction or block of code. When a thread switches to a new version, execution continues from the code cache of that version. If a block of code has not been instrumented for a certain version,

the instrumentation routine is called again and can install different analysis code based on the version.

Pin is actively developed and supports multiple hardware architectures and OSs. Pintools can be applied on any supported binary by either launching the binary through Pin or by attaching on an already running binary. The latter behavior is highly desirable for REASSURE, as it allows us to deploy RPs without interrupting an already executing application. We implemented REASSURE as a Pintool on Linux, but it is by no means limited to the Linux OS.

3.2 Installing Rescue Points

RPs can be installed on any callable application function. Such a function can be identified by its name or its address. The latter can be useful in cases where a binary has been entirely stripped of symbol information, and as such its functions are only identifiable by their address. In systems where the targeted binary is stripped and address space layout randomization (ASLR) [21] is used, specifying a RP's function may require additional analysis. That is because the function cannot be located by name, and its address may change due to the executable or library containing it being mapped to a different location because of ASLR. In such cases, the application can be launched without REASSURE, so we can first obtain the address where the object containing the RP's function was actually loaded. For instance, *libfoo.so* may be loaded at address *0xb6e7a000*. In Linux, such information can be obtained through the */proc* pseudo-file system. Additionally, we can statically determine the offset of the RP's function within the object. For example, function *foo()* may be defined at offset *0x800* in *libfoo.so*. By combining this information, we can calculate the address *foo()*, which in this example would be *0xb6e7a800*, and attach REASSURE on the process using the calculated RP address.

Assuming we have the means to identify RP functions, installing them is straightforward. If the function is defined by name, REASSURE first determines the address it resides in. This is accomplished by scanning the application and all its shared libraries as they are loaded. Concurrently, we scan each RP function we encounter to find at least one exit point (*i.e.,* a *ret* instruction) that will be used to return a valid error when a fault occurs. Finally, we install an instrumentation callback, which causes Pin to notify our tool whenever a new block of code is encountered. The instrumentation routine performs the following operations:

1. If a RP's entry point is encountered, analysis code is inserted to switch the thread that enters the RP to *checkpointing mode*. Primarily, this causes the thread entering the RP to switch to a different code cache version (discussed in Sect. 3.1) and saves the thread's CPU state. The *checkpointing* version of the instrumentation inserts analysis code that logs all the writes being performed by the application required for rolling back when an error occurs.
2. If a RP's exit point is encountered, analysis code is inserted to switch the thread returning from the function out of *checkpoint mode* and to normal execution. Besides switching to the original code cache that does not log program writes, the analysis code also discards the log of writes (*i.e.,* commits the changes).

Table 1. Signals intercepted by REASSURE to identify and recover from program errors

Signal	Description
SIGSEGV	Invalid memory reference/segmentation fault
SIGILL	Illegal instruction (*e.g.,* because of an invalid control-flow transfer)
SIGABRT	Abort signal sent by the *abort* system call
SIGFPE	Floating point exception (*e.g.,* divide by zero)

3.3 Memory Writes Logging

A RP's code, as well as all code called from it, is instrumented so as to log all the writes being performed. This *write log* serves the purpose of keeping track of all the modifications performed within a RP, so that it can be rolled back when an error occurs (*i.e.,* usually the same error that necessitated the introduction of the RP). This is achieved by augmenting every memory write instruction within a RP with analysis code that appends an entry in a dynamically expanding array, which holds the address being written and the value being overwritten. Because we are using Pin's instrumentation versioning, only the instructions being reached from within a RP are actually instrumented this way.

The analysis functions responsible for writes logging need to be carefully written to avoid certain erroneous conditions. For instance, consider a program performing an illegal memory write that causes a page fault within a RP. This memory write is also instrumented, so that the value being overwritten is saved in the log. Unfortunately, since the target address is invalid, the logging code executing before the actual write will cause the page fault instead. We have written these analysis routines in such a way that such a fault will not leave the writes log in a corrupted state (*e.g.,* with an erroneous number of entries).

3.4 Recovery from Faults

When terminal faults occur in Linux, the OS issues a synchronous signal, which if not handled will cause a process to terminate. For instance, an invalid memory reference will cause a *SIGSEGV* signal to be delivered by the OS. REASSURE intercepts such signals to identify errors occurring within RPs and initiate recovery. Table 1 lists all the signals intercepted by REASSURE to recover from program faults. Note that other OSs have similar mechanisms to synchronously notify applications of such errors. For example, Windows uses exceptions.

When REASSURE receives one of the signals in Table 1, we first check that the thread that received the signal is actually within a RP. If that is the case, we proceed to restore the values that have been overwritten since the entry to the RP and restore the saved CPU state. These actions effectively rollback the CPU and memory modifications in single-threaded applications, and applications where the function the RP was applied on does not access shared data or interact with other threads. We discuss concurrency issues in multithreaded applications separately in Sect. 3.5. We proceed by updating the program counter to point to

the *ret* instruction found during the RP's installation and use Pin's API to set the function's return value to the one specified by the RP as a valid error return value. In x86 architectures the return value is simply placed in the *eax* register. Recovery is completed by suppressing the delivery of the signal to the application and resuming execution from the updated program counter. In opposition, if one of these signals is received while the thread is not in a RP, we deliver it to the application for processing.

3.5 Concurrency

Restoring the CPU state and undoing memory writes is sufficient for recovering from faults in single-threaded applications, but this may not be the case in multithreaded applications. In general, threads share a common address space and, as such, updates made by one thread are immediately visible to all of them. Let's consider a multithreaded application with a buggy function that makes updates that affect multiple threads. It is possible that memory updates made by thread A within a RP are used by thread B to make further updates. Consequently, if an error occurs in thread A, the recovery process may leave residual data because of thread B having propagated the updates of thread A.

We address such concurrency issues by introducing *blocking* RPs that block other threads for their duration. REASSURE provides two modes of operation to accommodate blocking RPs. The first caters to applications that expect a very high rate of faults, while the second offers faster operation as long as the rate of faults is reasonable (evaluated in Sect. 4.3).

Always-on blocking mode operates by conditionally instrumenting every block of instructions with an analysis routine that blocks the executing thread when a certain flag, which is asserted by the blocking-RP upon entry, is set. Because this mode introduces frequent checks of the "block" flag, it incurs high overheads, but has low latency (*i.e.,* we can quickly activate/deactivate blocking) and is thus more appropriate for applications where faults occur very frequently.

On-demand blocking mode utilizes OS facilities to achieve better performance. In particular, we use signals (*i.e.,* the *SIGUSR2* signal) to asynchronously interrupt the remaining threads whenever a blocking-RP is entered. Similarly, to fault-related signals, REASSURE intercepts the delivery of *SIGUSR2* to install temporary blocks in receiving threads. Since the code that the thread was executing may have already been instrumented, we first remove the code currently executing from the code cache. After suppressing the delivery of the signal, Pin attempts to resume execution and since the block of code is no longer present in the code cache, our instrumentation routine is invoked again. This allows us to install an analysis routine that will block the thread. When a RP exits, we remove the blocking analysis code by once again removing the corresponding instructions from the code cache. This method has the advantage of the application generally executing faster, since "blocking" code is *not* installed *de facto* for every block of code. On the other hand, since it relies on the OS to issue and deliver signals, it takes longer to block threads which may lead to decreased performance when a high rate of errors is observed.

Table 2. Applications and benchmarks used for the evaluation of REASSURE. All of applications contain exploitable bugs as described by their *common vulnerability and exposure* (CVE) id.

Application		Bug type		Benchmark
MySQL	v5.0.67	Input validation	CVE-2009-4019	MySQL's *test-insert* and *test-select*
Apache	v1.3.24	Memory corruption	CVE-2002-0392	Apache's *ab* utility
CoreHTTP	v0.5.3a	Stack overflow	CVE-2007-4060	Apache's *ab* utility
Samba	v3.0.21	Heap overflow	CVE-2007-2446	Linux's *dd* utility

4 Evaluation

We evaluated REASSURE along two axes. First, we show that it is able to correctly *heal* various applications that contain bugs that can cause them to abnormally terminate. Second, we evaluate the performance overhead imposed by REASSURE on these applications. In both cases, we employed existing benchmarks and tools to generate workloads. Table 2 lists the applications and benchmarks used during the evaluation. We conducted the experiments presented in this section on a DELL Precision T5500 workstation with dual 4-core Xeon CPUs (with HyperThreading disabled) and 24GB of RAM running Linux 2.6.

4.1 Recovery from Errors

We tested REASSURE's ability to heal software by triggering known bugs in the applications listed in Table 2, while concurrently running the corresponding benchmarks. When REASSURE is not employed, the applications terminate and the benchmarks are interrupted in all cases. In contrast, when using REASSURE to apply a RP that engulfs the function that causes the crash, the applications recover from the error and the benchmarks conclude successfully.

Table 3 shows the RPs applied on the applications. All applications except MySQL do not use multiple threads, but instead consist of either a single event-driven process or multiple processes. For this reason, we used non-blocking RPs for all applications besides MySQL. For the latter, even though its RP does not access shared data and consequently does not require blocking, we tested it with both RP types to demonstrate REASSURE's correctness.

4.2 Performance in the Absence of Errors

For each application in Table 2, we performed the corresponding benchmark, first with the application executing natively, then running under the Pin DBI framework, and last under REASSURE with the corresponding RP installed. This allows us to quantify the overhead imposed by REASSURE compared with native execution, as well as the relative overhead compared with the baseline, which in our case is Pin. In the tests described in this section, we did not inject

Table 3. The rescue points applied to recover from the bugs listed in Table 2

Application		Function name	Return value	Type
MySQL	v5.0.67	Item_func_set_user_var::update()	1	Non-blocking Blocking
Apache	v1.3.24	ap_bread()	-1	Non-blocking
CoreHTTP	v0.5.3a	HttpSprockMake()	0	Non-blocking
Samba	v3.0.21	switch_message()	-1	Non-blocking

any requests that would trigger the bugs each application suffers from, nevertheless the RPs listed in Table 3 were installed.

Figure 3 shows the results obtained after running 10 iterations of MySQL's *test-insert* and *test-select* benchmark tests over an 1Gb/s network link. The y-axis lists the various server configurations tested, which from top to bottom are: native execution, execution over Pin, REASSURE using a non-blocking RP, and REASSURE using a blocking RP both in on-demand and always-on blocking mode. The x-axis shows the average time (in seconds) needed to complete each benchmark, while the errors bars represent standard deviation. Note that the figure also includes standard deviation for *test-select*, but it is insignificant and thus not visible. We observe that the *test-insert* and *test-select* benchmarks take on average 24% and 53% more time to complete when running the server over REASSURE and no blocking RPs, while a significant part of the overhead is because of Pin (under Pin the tests take 18% and 46% more time). Using on-demand blocking has little effect on performance, while using always-on blocking increases the overhead to 42% and 115% respectively.

Figures 4(a) and 4(b) depict the results obtained after running 10 iterations of Apache's *ab* benchmark utility over an 1Gb/s network link for the Apache and corehttp web servers respectively. The y-axis displays the average throughput in

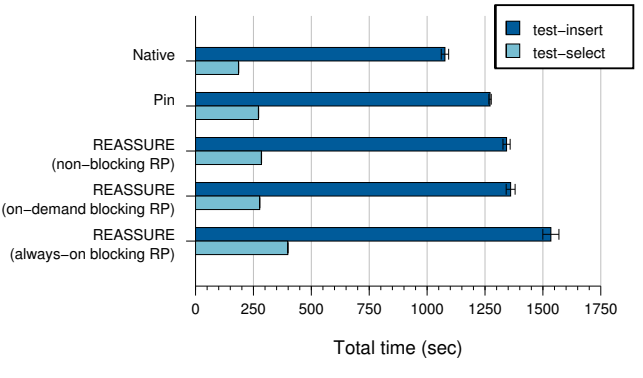
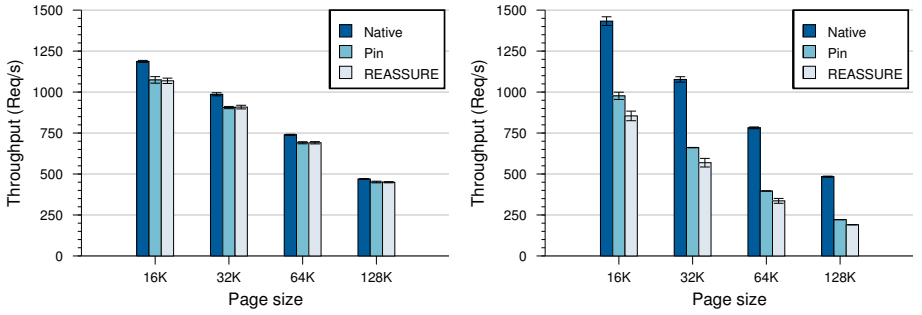


Fig. 3. MySQL performance. Time needed to complete MySQL's test-insert benchmark over an 1Gb/s network link. We apply the rescue point in three different ways: as a non-blocking RP, a blocking RP with on-demand thread blocking, and a blocking RP with always-on blocking.



(a) Apache (b) Corehttp

Fig. 4. Web server performance. We used Apache's *ab* benchmark utility to measure the throughput of the Apache and corehttp web servers when requesting files of different size over an 1Gb/s network link.

requests per second as reported by *ab*, and the error bars represent standard deviation. We performed the experiments requesting files of different size from the web servers (listed in the x-axis), while we repeated each test with the corresponding server running: natively, over Pin, and with REASSURE (the RPs used are non-blocking). Corehttp is a single-process server and Apache was configured to only spawn a single process for serving requests to obtain comparable results.

In Fig. 4(a), we see that Apache performs approximately 4%-10% slower when run with REASSURE and the greater part of the overhead is because of Pin. We also notice that the overhead drops as the size of the requested file increases. This is due to the workload becoming more I/O intensive (*i.e.,* more data need to be transferred per request) and the number of requests arriving at the server shrinks. On the other hand, Fig. 4(b) shows that corehttp performs significantly worse than Apache. When running under REASSURE its throughput is reduced by approximately 40%-60%, while even when running under Pin we observe a 31%-54% reduction in throughput. There are two reasons corehttp performs so poorly. First, it is the only application where the RP is actually in the critical path of execution and it is entered for every performed request. Second, corehttp consists of many and short lived function calls that require additional processing by Pin, which be design receives control before performing any indirect control transfer like a function return. Note that the performance of code running within a RP greatly depends on parameters like the initial size of the writes log described in Sect. 3.3. If the RP is in the critical path, as in the case of corehttp, and contains many memory writes, the log will have to be frequently enlarged to accommodate the application. In the experiments described in this section, the initial size of the writes log, as well as the step used to enlarge it, is 50000 entries.

Finally, Fig. 5 shows the results of copying an 100MB file to a directory shared through samba over an 1Gb/s network link. The y-axis shows the average transfer rate (in MB/s) achieved by the *dd* utility. Once again, we performed 10 iterations of each test and we display standard deviation using error bars. We observe that

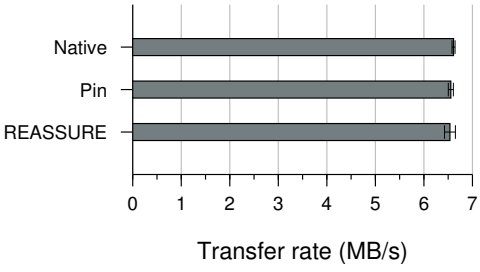


Fig. 5. Samba performance. We used the *dd* utility to copy an 100MB file containing randomly generated data to a directory shared using samba. The shared directory was mounted on a remote host over an 1Gb/s network link.

when running the samba server over REASSURE there is a negligible drop in the transfer rate (approximately 1%), even though the installed RP is entered on every file transfer request.

4.3 Performance in the Presence of Errors

We complemented the experiments in the previous section by performing a set of tests against the Apache web server and the MySQL DB server running over REASSURE and in the presence of errors. For Apache, we measured its throughput (in requests per second) using the *ab* utility to request a 16KB file, while concurrently we issued requests with varying frequency that triggered the server's fault, which was protected by a non-blocking RP. Figure 6 shows the results of this experiment. The x-axis is in logarithmic scale and corresponds to the time interval (in seconds) used to submit a faulty request to the server (*i.e.,* we attempted to crash the server every x seconds). When there is an one second or longer interval between the attacks to the server, it performs as well as when no errors occur, while at the same time it "heals" from the occurring errors. As the frequency of the attacks increases the attainable throughput drops. Finally, if errors occur continuously (zero seconds injection interval) the server still survives, even though throughput is greatly reduced.

In Fig. 7, we show the results obtained from running MySQL's *test-select*, while faults were injected as in the experiment described above. The y-axis shows the time needed to complete each test and the x-axis corresponds to the time interval between fault injections. Both axes are in logarithmic scale. We utilized a blocking RP to recover from the faults, both in on-demand and always-on blocking mode, and in both cases we observe that if the time between faults is one second or longer, there is only a minor decrease in performance. As the frequency of the faults increases, so does the overhead in both blocking modes. Predominantly, on-demand blocking outperforms always-on blocking, but in high fault frequencies (approximately one fault per 0.1s or less) the situation is reversed. Users of REASSURE that are able to anticipate the rate of faults, can use this knowledge to select the better performing blocking mode. Alternatively,

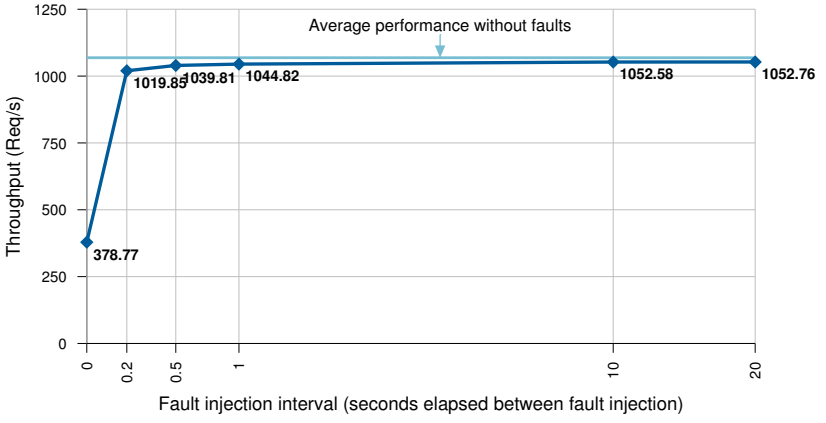


Fig. 6. Throughput of Apache web server as the number of faulty requests changes. Measured using Apache's *ab* benchmark utility to request a 16KB file containing randomly generated data. At the same time a non-blocking RP was employed to recover from the injected errors.

REASSURE could also monitor the frequency of faults to automatically switch from one mode to the other (discussed in Sect. 5).

5 Limitations and Future Work

There are various issues that should be considered before deploying a blocking-RP. For instance, if the RP function attempts to acquire a lock that is already held by another thread, the application may be deadlocked as REASSURE blocks all other threads. One should also be aware of certain library calls that may obtain locks (*e.g., printf()*). Therefore, blocking-RPs should be used with prudence and only when necessary, specially considering their overhead. Some of the issues with blocking-RPs can be addressed by extending REASSURE, so that RPs can be installed just on certain parts of a function (fine-grained RPs). We could then install a rescue point within the critical region of a function, after a lock has already been obtained.

Some caution is also needed when setting up RPs in functions that perform certain system calls, as the current implementation of REASSURE does not rollback the effects of system calls. Frequently, this fact does not have any adverse effect on the application. For instance, retrieving the system's time, reading a random number from */dev/urandom*, and requiring more heap size do not affect the application in case of a rollback. In fact, even operations such as writing an entry in a log file or a network socket can be allowed from within a RP. While the data are still written to the destination, a rollback will leave the system in a valid state. However, this may not be the case for all applications, as RPs containing writes on critical files (*e.g.,* a DB's data file) may lead to a corrupted state. In practice, the cases where a RP should not be installed can be even more uncommon, as the error a RP is guarding against may occur before any such

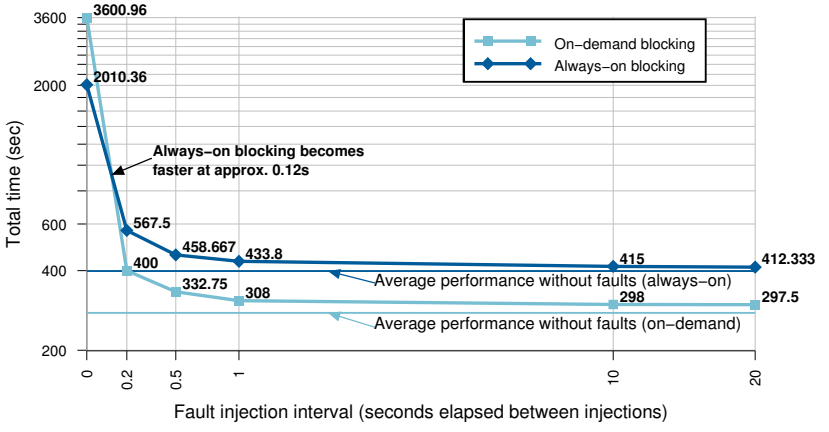


Fig. 7. Performance of MySQL DB server as the number of faulty requests changes. We measured the time needed to complete MySQL's *test-select* test in the presence of faults, and used a blocking RP to recover from the errors in always-on and on-demand blocking mode.

critical system calls, and as such the latter are inconsequential (*i.e.,* they are only executed when the error does not occur). In the future, we plan to extend REASSURE to also rollback certain system calls, like *mmap* and *munmap*. Some support for file and socket writes may also be incorporated by delaying/buffering the writes until the RP has concluded.

Finally, RPs should be applied with caution, when the targeted function updates data shared with other processes through *shared memory*. Since the updates are immediately visible to other processes, a rollback in case of error cannot guarantee that the new memory values have not been already read by another process. A possible-yet-costly solution could involve committing the updates into shadow memory, private to the thread in a RP, and upon completion copying the updates from shadow to application memory. Fine-grained RPs could also provide a workaround for applications with such issues.

In the future, we also plan to include certain optimizations that will improve performance. For instance, the write log (Sect. 3.3) is dynamically expanding and its expansion can be costly if it occurs frequently. We can "remember" the write log size required by a RP to reduce this cost. We also saw in Sect. 4.3 that depending on the frequency of errors when using blocking-RPs, it may be preferable to use always-on instead of on-demand blocking and *vice versa*. Instead, REASSURE can automatically switch between versions based on the observed fault rate.

6 Related Work

Software self-healing using RPs was first proposed in ASSURE [26]. The authors described a mechanism that can automatically analyze an application

error to identify and select the appropriate RP. The deployment of the RPs was performed using a modified OS featuring the Zap [19] virtual execution environment. REASSURE does not require any modifications to the OS, and can be easily enabled and disabled. However, the RP identification component of ASSURE can be used in combination with our work.

Selective transactional emulation (STEM) [27] is a speculative recovery technique that also identifies the function where an error occurs, and it could be also used to assist in identifying RPs. Unlike REASSURE, STEM requires source code to perform the error analysis, and does not work with multiprocess and multithreaded applications. Failure-oblivious computing [25] is another speculative recovery technique that is based on the compiler inserting code to handle invalid memory writes by virtually extending the target buffer. This approach offers more robust fault response than simply crashing, but at significant performance overhead, ranging from 80% up to 500% for a variety of different applications, while it also requires recompilation of the target application.

Rebooting techniques [28,10,6] attempt to restore a system to a clean state before or after a fault. Program restart takes significantly longer time, resulting in substantial application down-time, while data loss may also occur. Micro-rebooting can be faster by only restarting parts of the system, but requires a complete rewrite of applications to compartmentalize failures. None of these techniques effectively deal with deterministic bugs, since these may recur post-restart.

Checkpoint-restart techniques [3,14] can be used in a similar fashion to rebooting, but achieve better restart times since the application can start from a checkpoint. While down time is reduced, compared with rebooting, these techniques still do not handle deterministic bugs, or bugs maliciously triggered by an attacker (*e.g.,* a DoS attack). Checkpoint-restart has been also combined with running N-versions of a program [3]. This method assumes that failures occur independently in the various versions, but introduces prohibitive costs for most applications, as multiple versions need to be maintained and run concurrently.

Automatically generating and applying patches has also been proposed, as a way to heal software [23,17,29]. Unfortunately, automatically applying patches has not been generally adopted, due to the possibility that additional errors are introduced during the patching, or that the patched application stops behaving as expected.

7 Conclusions

We presented REASSURE, a self-contained mechanism for healing software using rescue points. REASSURE is easy to use and does not require modifications to the OS, making RPs an attractive and practical solution for temporary healing software until a patch or update is made available. It enables the reuse of existing error handling code to also handle unanticipated failures, such as the ones that can lead to the abnormal termination of an application. We have tested REASSURE with various applications including the Apache and MySQL servers, and

show that it successfully allows them to recover from otherwise terminal errors. In the absence of errors REASSURE incurs an overhead between 1% and 115% depending on whether a RP is encountered frequently, and whether the application is I/O or CPU bound. We also show that when errors occur frequently, REASSURE protected applications survive, even under very adverse conditions like their continuous bombardment with errors.

Acknowledgements. This work was supported by the US Air Force and the National Science Foundation through Contracts AFRL-FA8650-10-C-7024 and AFOSR-MURI-FA9550-07-1-0527, and Grant CNS-09-14845, respectively. Any opinions, findings, conclusions or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government, the Air Force, or the NSF.

References

1. Akritidis, P., Cadar, C., Raiciu, C., Costa, M., Castro, M.: Preventing memory error exploits with WIT. In: Proc. of the Symposium on Security and Privacy, pp. 263–277 (May 2008)
2. Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Henri-Gros, C., Kamsky, A., McPeak, S., Engler, D.: A few billion lines of code later: using static analysis to find bugs in the real world. Commun. ACM 53, 66–75 (2010)
3. Bressoud, T.C., Schneider, F.B.: Hypervisor-based fault tolerance. In: Proc. of the 15th ACM symposium on Operating systems principles (SOSP), pp. 1–11 (1995)
4. Buck, B., Hollingsworth, J.K.: An api for runtime code patching. Int. J. High Perform. Comput. Appl. 14, 317–329 (2000)
5. Cadar, C., Dunbar, D., Engler, D.: KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proc. of the 8th OSDI, pp. 209–224 (2008)
6. Candea, G., Fox, A.: Crash-only software. In: Proc. of the 9th Workshop on Hot Topics in Operating Systems, HotOS IX (May 2003)
7. Cowan, C., Barringer, M., Beattie, S., Kroah-Hartman, G.: FormatGuard: Automatic Protection From printf Format String Vulnerabilities. In: Proc. of the 10th USENIX Security Symposium, pp. 191–199 (August 2001)
8. Etoh, J.: GCC extension for protecting applications from stack-smashing attacks, `http://www.trl.ibm.com/projects/security/ssp/`
9. Hicks, M., Nettles, S.: Dynamic software updating. ACM Trans. Program. Lang. Syst. 27, 1049–1096 (2005)
10. Huang, Y., Kintala, C., Kolettis, N., Fulton, N.: Software rejuvenation: Analysis, module and applications. In: Proc. of the 25th International Symposium on Fault-Tolerant Computing (FTCS), p. 381 (1995)
11. Information Week: Windows home server bug could lead to data loss, `http://informationweek.com/news/205205974` (December 2007)
12. Kc, G.S., Keromytis, A.D., Prevelakis, V.: Countering code-injection attacks with instruction-set randomization. In: Proc. of the 10th CCS, pp. 272–280 (October 2003)
13. Keromytis, A.D.: Characterizing self-healing software systems. In: Proc. of the 4th MMM-ACNS (September 2007)

14. King, S.T., Dunlap, G.W., Chen, P.M.: Debugging operating systems with time-traveling virtual machines. In: Proc. of the USENIX Annual Technical Conference (2005)
15. Laadan, O., Nieh, J.: Transparent checkpoint-restart of multiple processes on commodity operating systems. In: Proc. of the 2007 USENIX ATC, pp. 323–336 (2007)
16. Luk, C.K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: Building customized program analysis tools with dynamic instrumentation. In: Proc. of the 2005 PLDI, pp. 190–200 (June 2005)
17. Makris, K., Ryu, K.D.: Dynamic and adaptive updates of non-quiescent subsystems in commodity operating system kernels. In: Proc. of the 2nd EuroSys, pp. 327–340 (March 2007)
18. Newsome, J., Song, D.: Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: Proc. of the 12th NDSS (February 2005)
19. Osman, S., Subhraveti, D., Su, G., Nieh, J.: The design and implementation of Zap: a system for migrating computing environments. In: Proc. of the 5th OSDI, pp. 361–376 (December 2002)
20. Ostrand, T.J., Weyuker, E.J.: The distribution of faults in a large industrial software system. In: Proc. of the 2002 ACM SIGSOFT ISSTA, pp. 55–64 (2002)
21. PaX Project: Address space layout randomization (March 2003), `http://pageexec.virtualave.net/docs/aslr.txt`
22. PCWorld: Amazon EC2 outage shows risks of cloud (April 2011), `http://www.pcworld.com/businesscenter/article/226199/amazon_ec2_outage_shows_risks_of_cloud.html`
23. Perkins, J.H., Kim, S., Larsen, S., Amarasinghe, S., Bachrach, J., Carbin, M., Pacheco, C., Sherwood, F., Sidiroglou, S., Sullivan, G., Wong, W.F., Zibin, Y., Ernst, M.D., Rinard, M.: Automatically patching errors in deployed software. In: Proc. of the ACM SIGOPS 22nd symposium on Operating systems principles, pp. 87–102 (2009)
24. Porras, P., Saidi, H., Yegneswaran, V.: Conficker C analysis. Tech. rep., SRI International (2009)
25. Rinard, M., Cadar, C., Dumitran, D., Roy, D., Leu, T.W., Beebee, J.: Enhancing server availability and security through failure-oblivious computing. In: Proc. of the 6th OSDI (December 2004)
26. Sidiroglou, S., Laadan, O., Perez, C., Viennot, N., Nieh, J., Keromytis, A.D.: ASSURE: automatic software self-healing using rescue points. In: Proc. of the 14th ASPLOS, pp. 37–48 (2009)
27. Sidiroglou, S., Locasto, M.E., Boyd, S.W., Keromytis, A.D.: Building a reactive immune system for software services. In: Proc. of the 2005 USENIX ATC (April 2005)
28. Sullivan, M., Chillarege, R.: Software defects and their impact on system availability - A study of field failures in operating systems. Digest of Papers., 21st International Symposium on Fault Tolerant Computing (FTCS-21), pp. 2–9 (1991)
29. Susskraut, M., Fetzer, C.: Automatically finding and patching bad error handling. In: Proc. of the Sixth European Dependable Computing Conference, pp. 13–22 (2006)

Characterization of Strongly Secure Authenticated Key Exchanges without NAXOS Technique

Atsushi Fujioka

NTT Information Sharing Platform Laboratories
3-9-11 Midori-cho, Musashino-shi, Tokyo 180-8585 Japan
fujioka.atsushi@lab.ntt.co.jp

Abstract. This paper examines two-pass authenticated key exchange (AKE) protocols that do not use the NAXOS technique and that are secure under the gap Diffie-Hellman assumption in the random oracle model. Their internal structures are also discussed. We introduce an imaginary protocol, however insecure, to analyze the protocols and show the relations between these protocols from the viewpoint of how they overcome the insecurity of the introduced protocol.

In addition, this paper provides ways to characterize the AKE protocols and defines two parameters: one consists of the number of static keys, the number of ephemeral keys, and the number of shared values, and the other is defined as the total sum of these numbers. When an AKE protocol is constructed based on some group, these two parameters indicate the number of elements in the group, i.e., they are related to the sizes of the storage and communication data.

Keywords: Two-pass authenticated key exchange, extended Canetti-Krawczyk security, gap Diffie-Hellman assumption, NAXOS technique.

1 Introduction

The authenticated key exchange (AKE) [1,3,4,14] is one of the most important cryptographic protocols. In key exchange protocols, two parties exchange ephemeral information and then they generate a common secret, called a *session key*, from their private information, the exchanged information, and other information. The AKE assures that the session key is derived only by their intended peers.

Most of the AKEs are constructed utilizing the Diffie-Hellman (DH) protocol [9] and the security relies on the computational Diffie-Hellman problem or its related problem such as the gap Diffie-Hellman problem [19].

Roughly speaking, in the computational Diffie-Hellman (CDH) problem, the computational Diffie-Hellman value, g^{xy}, must be computed from X $(= g^x)$ and Y $(= g^y)$ in a cyclic group where the group is generated by primitive element g. In the decisional Diffie-Hellman (DDH) problem, it is required to decide that Z is the computational Diffie-Hellman value of X and Y or a random one given X,

T. Iwata and M. Nishigaki (Eds.): IWSEC 2011, LNCS 7038, pp. 33–50, 2011.

Y, and Z. In the gap Diffie-Hellman (GDH) problem, the computational Diffie-Hellman value of X and Y must be computed with the help of the DDH oracle. The GDH (CDH and DDH, respectively) assumption is that the GDH (CDH and DDH, respectively) problem is assumed to be hard for any polynomial-time algorithm to solve.

The first formal security definition for AKEs was given by Bellare and Rogaway [1].[1] After much discussion, a framework, named the CK model, was established by Canetti and Krawczyk [4], and later, LaMacchia, Lauter, and Mityagin proposed another model, the extended Canetti-Krawczyk (eCK) model [14].

In the eCK model [14], the adversary activates all parties and this activation is done in two ways: One is to force the party to send a message and the other is to let the party receive a message. Both are done while all communications between the parties are controlled by the adversary. When a party generates the session key in a session, the session is referred to as being complete. In two-pass protocols, the party would have both outgoing and incoming messages in the session for the completed session. If an owner has a completed session and its peer has the same session ID as the completed session, the session of the peer is called a *matching session*. The adversary can adaptively access session keys, static private keys, and ephemeral private keys. At some point, the adversary chooses a session as the *test session* and it is given a value, which is the session key of the session or a random value with probability $\frac{1}{2}$. The adversary continues the actions and at the end, a bit is output regarding whether or not the given value is the session key with a better probability than $\frac{1}{2}$. This is called the *indistinguishability test* and the aim of the adversary is to pass the indistinguishability test. The adversary is not allowed to access both the static private key and the ephemeral private key of the owner or of its peer (if one exists). If the winning probability of this game is negligible for any adversary, then the AKE protocol is eCK-secure.

The eCK security is strong in the sense that the adversary in the eCK model obtains much information regarding the test session since the adversary is allowed to access either the static private key or the ephemeral private key of each party establishing the test session.[2]

After the eCK model was proposed, several eCK-secure authenticated key exchange protocols were invented. NAXOS [14], NETS [15], and CMQV [22] are typical examples of the eCK-secure protocols under the GDH assumption in the random oracle model (ROM) [2], and they utilize the NAXOS technique. Here, the NAXOS technique is a technique in which the exponent of the ephemeral public key is generated as the output of a hash function from the static private key and the ephemeral private key, i.e., $X = g^{H(x,a)}$ where X is the ephemeral public key, x is the ephemeral private key, a is the static private key, and H is the hash function (regarded as a random oracle). Even when the ephemeral key is revealed, the discrete log of the ephemeral public key is unknown to the

[1] Bellare and Rogaway provided the model in shared-key setting. The model in public-key setting was defined by Blake-Wilson, Johnson and Menezes [3].

[2] Note that the eCK security is not stronger in all senses than the CK security. Cremers pointed out that they are incompatible [7,8].

adversary without the static private key and this allows for a simple security proof in the eCK model.

Recently, disadvantages in the NAXOS technique have been independently pointed out by Cheng, Ma, and Hu [6] and Wu and Ustaoğlu [24]. The AKE protocols that do not rely on the NAXOS technique but utilize the DH protocol decrease the risk of leaking the static private key in comparison to protocols that utilize the NAXOS technique since the derivation of the ephemeral public key is independent from the static private key. The DH-like AKE protocols that do not utilize the NAXOS technique are secure even when the discrete logarithm of the ephemeral public key is revealed. Several two-pass eCK-secure AKE protocols that do not use the NAXOS technique have been proposed [24,21,12,17,23,10] and there are subtle differences in their efficiency levels.

1.1 Background

Many eCK-secure AKE protocols have been proposed based on the DH protocol. These AKE protocols are roughly classified from the following viewpoints: (1) whether or not the NAXOS technique is used, (2) security is proven in the ROM or in the standard model, and (3) the security is based on the CDH assumption, the DDH assumption, and the GDH assumption.

NAXOS [14], NETS [15], and CMQV [22] are eCK-secure AKE protocols under the GDH assumption in the ROM and all of them adopt the NAXOS technique.

SMEN$^-$ [24] is an eCK-secure protocol under the GDH assumption in the ROM, and it does not use the NAXOS technique. In [24], a protocol that is eCK-secure under the GDH assumption and uses the NAXOS technique (SMEN) was also proposed. In [12], Kim, Fujioka, and Ustaoğlu proposed an eCK-secure protocol under the GDH assumption in the ROM (Protocol 1, denoted as KFU1 in this paper) and an eCK-secure protocol under the CDH assumption in the ROM (Protocol 2). UP proposed in [23] and FHMQV proposed in [21] are eCK-secure protocols under the GDH assumption in the ROM also.

NAXOS+ [16] and Huang-Cao [11] are eCK-secure AKE protocols under a weaker assumption, the CDH assumption, in the ROM, and Okamoto [17,18] and Moriyama-Okamoto [17] are eCK-secure protocols under the DDH assumption in the standard model. NAXOS+, Huang-Cao, and Okamoto utilize the NAXOS technique but the Moriyama-Okamoto protocol does not rely on the NAXOS technique.

As indicated above, many AKE protocols have been proposed and the respective security proofs are given in the corresponding references. Recently, Fujioka and Suzuki have provided a sufficient condition to construct a eCK-secure AKE protocol under the GDH assumption in the ROM [10], and the condition is that the exponents of the shared value in an AKE protocol are expressed by *admissible* polynomials.

This paper concentrates on the structure most commonly used in AKEs without the NAXOS technique. In the structure, peers have static keys, they exchange ephemeral information, compute the shared values, and then generate the session keys with a hash function. We call this kind of AKE protocol *regular*.

1.2 Contributions

In this paper, we first examine the two-pass AKE protocols that do not use the NAXOS technique and that are eCK-secure under the GDH assumption in the ROM, and discuss their relations. We introduce an imaginary but insecure protocol, called the *multiplied biclique DH protocol*, to analyze the AKE protocols that utilize the imaginary protocol as an internal protocol. Although this protocol is insecure, its security discussion reveals the technical difficulty in designing a secure AKE protocol. This paper analyzes the relations among SMEN$^-$, FHMQV, KFU1, and UP which are two-pass AKE protocols that do not use the NAXOS technique and that are eCK-secure under the GDH assumption in the ROM. We clarify the internal structures of SMEN$^-$, FHMQV, KFU1, and UP, and show that they utilize the multiplied biclique DH protocol as an internal protocol. By analyzing how to avoid insecurity in the protocol, we show the relations between the AKEs. KFU1 runs two multiplied biclique DH protocols in parallel with the same ephemeral key on two static keys. SMEN$^-$ can be thought of as a two ephemeral key variant of KFU1 and this reduces the number of shared values to one. UP is a version of KFU1 in which one of the static public keys is generated with a random oracle. In the security proof, FHMQV virtually runs two multiplied biclique DH protocols in sequence with the same ephemeral key on two randomized static keys. These analyses could be useful since they do not only give intuitive security proofs of the subject protocols but also lead to a design principle of eCK-secure AKE protocols. Actually, a design principle was provided as a sufficient condition to construct eCK-secure protocols under the GDH assumption in the ROM [10]. An analysis that leads to this condition can be viewed as an extension of these analyses on how the above protocols overcome the insecurity of the multiplied biclique DH protocol.

To characterize these two-pass eCK-secure AKE protocols, we define two parameters: One consists of the number of static keys, the number of ephemeral keys, and the number of shared values, and the other is defined as the total sum of these numbers. When an AKE protocol is constructed based on some group, the numbers of keys and shared values are counted as the number of elements in the group so that these parameters give efficiency indices for the protocol since the number of static keys and the number of ephemeral keys are related to the sizes of the storage and communication data, respectively.

1.3 Organization

Section 2 provides the definitions to characterize the AKE protocols in addition to the security definitions, introduces an imaginary protocol, the *multiplied biclique DH protocol*, and discusses the insecurity of the protocol. In **Section 3**, the relations among SMEN$^-$, FHMQV, KFU1, and UP are presented, and **Section 4** discusses the characterization of the AKE protocols, and describes open problems.

2 Preliminaries

2.1 Security Definitions

For further eCK details and explanations see [14].

In the eCK model, each party is a probabilistic polynomial-time Turing machine and is assigned a static public and private key pair together with a certificate that binds the identity of the party to its public key. We denote the identity of a party as $\hat{A}, \hat{B}, \hat{C}, \ldots$[3]. We assume that, the certificate authority (CA) does not require proof of possession of the corresponding private key included in a certificate. However, the CA verifies that the public key is in $\mathbb{G}^{\times} = \mathbb{G} - \{\mathrm{id}_{\mathbb{G}}\}$, where $\mathrm{id}_{\mathbb{G}}$ is the identity element of group $\mathbb{G}$.

We outline the eCK model for two-pass Diffie-Hellman protocols where two parties, $\hat{A}$ and $\hat{B}$, exchange static and ephemeral public keys and thereafter compute a session key that depends on the exchanged public keys and identities of the parties.

Session. An invocation of a protocol is called a *session*. A session is activated by an incoming message in the form of $(\mathcal{I}, \hat{A}, \hat{B})$ or $(\mathcal{R}, \hat{A}, \hat{B}, Y)$. If $\hat{A}$ is activated with $(\mathcal{I}, \hat{A}, \hat{B})$, then $\hat{A}$ is called the session *initiator*; otherwise, it is called the session *responder*. After activation, session initiator $\hat{A}$ creates ephemeral public key X and sends $(\mathcal{R}, \hat{B}, \hat{A}, X)$ to session responder $\hat{B}$. Responder $\hat{B}$ then prepares ephemeral public key Y, computes the session key, and sends $(\mathcal{I}, \hat{A}, \hat{B}, X, Y)$ to $\hat{A}$. Upon receiving $(\mathcal{I}, \hat{A}, \hat{B}, X, Y)$, $\hat{A}$ also computes a session key for its own session. We say that a session is *completed* if its owner computes a session key.

If $\hat{A}$ is the initiator of a session, the session is identified via $(\mathcal{I}, \hat{A}, \hat{B}, X, \times)$ or $(\mathcal{I}, \hat{A}, \hat{B}, X, Y)$. For responder $\hat{A}$ the session is identified via $(\mathcal{R}, \hat{A}, \hat{B}, Y, X)$. The *matching session* of $(\mathcal{I}, \hat{A}, \hat{B}, X, Y)$ is a session with identifier $(\mathcal{R}, \hat{B}, \hat{A}, X, Y)$ and vice versa. In the remainder of the paper we will omit $\mathcal{I}$ and $\mathcal{R}$ since these "role markers" are implicitly defined from the order of the ephemeral public keys.

Adversary. Adversary $\mathcal{A}$ is modeled as a probabilistic Turing machine that controls all communications including session activation and is performed via the $\mathsf{Send}(\mathtt{message})$ query. The $\mathtt{message}$ has one of the following forms, $(\mathtt{pid}, \overline{\mathtt{pid}})$, $(\mathtt{pid}, \overline{\mathtt{pid}}, X)$, or $(\mathtt{pid}, \overline{\mathtt{pid}}, X, Y)$, where $\mathtt{pid}$ and $\overline{\mathtt{pid}}$ are identities. Each party submits its responses to the adversary, who decides the global delivery order.

The adversary does not have immediate access to the private information of a party. However, leakage of private information is captured via the adversary queries given below.

- $\mathsf{EphemeralKeyReveal}(\mathtt{sid})$: The adversary obtains the ephemeral private key associated with session $\mathtt{sid}$.
- $\mathsf{SessionKeyReveal}(\mathtt{sid})$: The adversary obtains the session key for session $\mathtt{sid}$ provided that the session holds a session key.

[3] In the eCK model, the adversary selects these identifier strings.

– StaticKeyReveal(pid): The adversary learns the static private key of party pid.

To define eCK security we need the following definition.

Definition 1 (Freshness). *Let* $\mathtt{sid}^*$ *be the session identifier of a completed session owned by honest party* $\hat{A}$ *with peer* $\hat{B}$*, who is also honest. If a matching session exists, then let* $\overline{\mathtt{sid}^*}$ *be the session identifier of the matching session of* $\mathtt{sid}^*$*. Define* $\mathtt{sid}^*$ *to be fresh if none of the following conditions hold.*

1. *An adversary issues a* SessionKeyReveal($\mathtt{sid}^*$) *or* SessionKeyReveal($\overline{\mathtt{sid}^*}$) *query (if* $\overline{\mathtt{sid}^*}$ *exists).*
2. $\overline{\mathtt{sid}^*}$ *exists and the adversary makes either of the following queries*
   – *both* StaticKeyReveal($\hat{A}$) *and* EphemeralKeyReveal($\mathtt{sid}^*$)*, or*
   – *both* StaticKeyReveal($\hat{B}$) *and* EphemeralKeyReveal($\overline{\mathtt{sid}^*}$)*.*
3. $\overline{\mathtt{sid}^*}$ *does not exist and the adversary makes either of the following queries.*
   – *both* StaticKeyReveal($\hat{A}$) *and* EphemeralKeyReveal($\mathtt{sid}^*$)*, or*
   – StaticKeyReveal($\hat{B}$)*.*

Security Experiment. Initially, adversary $\mathcal{A}$ is given a set of honest parties, for whom $\mathcal{A}$ selects identifiers. Then the adversary makes any sequence of the queries described above. During the experiment, $\mathcal{A}$ makes special query Test($\mathtt{sid}^*$), where $\mathtt{sid}^*$ is a fresh session. $\mathcal{A}$ is given either a random key or the session key held by $\mathtt{sid}^*$ with equal probability. The test query does not terminate the experiment, and the experiment continues until $\mathcal{A}$ makes a guess whether or not the key is random. The adversary *wins* the game if test session $\mathtt{sid}^*$ is still fresh and if the guess by $\mathcal{A}$ was correct.

Definition 2 (eCK security). *The advantage of adversary* $\mathcal{A}$ *in the AKE experiment with AKE protocol* Π *is defined as*

$$\mathrm{Adv}_{\Pi}^{\mathrm{AKE}}(\mathcal{A}) = \Pr[\mathcal{A}\ wins] - \frac{1}{2}.$$

We say that AKE protocol Π *is secure in the eCK model if the following conditions hold.*

1. *If two honest parties complete matching sessions, then except with negligible probability, they both compute the same session key.*
2. *For any probabilistic polynomial-time bounded adversary* $\mathcal{A}$*,* $\mathrm{Adv}_{\Pi}^{\mathrm{AKE}}(\mathcal{A})$ *is negligible.*

2.2 Efficiency Definitions

We call an AKE protocol *regular* when it has the following structure. Assume party $\hat{A}$ wants to share a key with another party, $\hat{B}$, where $\hat{A}$ has α static private keys $a_1, \ldots, a_\alpha$ and their static public keys $A_1\ (= g^{a_1}), \ldots, A_\alpha\ (= g^{a_\alpha})$, and $\hat{B}$ has α static private keys $b_1, \ldots, b_\alpha$ and their static public keys B_1 (=

$g^{b_1}), \ldots, B_\alpha\ (= g^{b_\alpha})$. Here α is a natural number ($\in \mathbb{N}$), g is a primitive element whose order is a prime, q, in group $\mathbb{G}$, and $a_1, \ldots, a_\alpha$, $b_1, \ldots, b_\alpha$ are randomly selected in $\mathbb{Z}_q$. Then, $\hat{A}$ generates β ephemeral private keys $x_1, \ldots, x_\beta \in \mathbb{Z}_q$ where β is a natural number ($\in \mathbb{N}$), computes ephemeral public keys $X_1\ (= g^{x_1}), \ldots, X_\beta\ (= g^{x_\beta})$, and sends $X_1, \ldots, X_\beta$ to $\hat{B}$. In response, $\hat{B}$ also generates β ephemeral private keys $y_1, \ldots, y_\beta\ (\in \mathbb{Z}_q)$, computes ephemeral public keys $Y_1\ (= g^{y_1}), \ldots, Y_\beta\ (= g^{y_\beta})$, and returns $Y_1, \ldots, Y_\beta$ to $\hat{A}$. $\hat{A}$ computes γ shared values $Z_1, \ldots, Z_\gamma$ from $a_1, \ldots, a_\alpha$, $x_1, \ldots, x_\beta$, $B_1, \ldots, B_\alpha$, and $Y_1, \ldots, Y_\beta$ while $\hat{B}$ computes shared values $Z_1, \ldots, Z_\gamma$ from $b_1, \ldots, b_\alpha$, $y_1, \ldots, y_\beta$, $A_1, \ldots, A_\alpha$, and $X_1, \ldots, X_\beta$ where γ is a natural number ($\in \mathbb{N}$). Both parties obtain session key K from the shared values and the session information using function H as $K = H(Z_1, \ldots, Z_\gamma, \hat{A}, \hat{B}, X_1, \ldots, X_\beta, Y_1, \ldots, Y_\beta)$. Hereafter, we call this H the *session key derivation function.*

We note that α, β, and γ may not be natural numbers, that is, these can be zero. We discuss this later.

Here, we give three definitions to evaluate the efficiency of the AKE protocol.

Definition 3. *A regular AKE protocol is* type (α-β-γ) *when it has α static keys, β ephemeral keys, and γ shared values where $\alpha, \beta, \gamma \in \mathbb{N}$.*

Definition 4. *The* count *of a regular AKE protocol, δ, is defined as the total sum of the keys and shared values.*

In other words, a type (α-β-γ) AKE protocol has count δ ($= \alpha + \beta + \gamma$).

When a type (α-β-γ) AKE protocol is constructed based on some group $\mathbb{G}$, the number of keys and the shared values are the number of elements in $\mathbb{G}$ and their sum total yields the efficiency index of the protocol.

Definition 5. *We denote a regular AKE protocol with its count δ as a* δ-count AKE protocol.

2.3 Optimality and Impossibility

Here, we extend regular AKE protocols to allow $\alpha, \beta, and \gamma$ to be zero and discuss the security of the protocols. In other words, the protocol has 0 as an entry in its type, i.e., it is a type (0-$*$-$*$), type ($*$-0-$*$), or type ($*$-$*$-0) protocol where $*$ is a non-zero value.

It is clear that the following lemmas and proposition are trivial.

Lemma 1. *There exists no eCK-secure type* (0-$*$-$*$) *AKE protocol.*

Proof. It is well known that the classic DH protocol is vulnerable against the person-in-the-middle attack.

Assume that the AKE is type (0-$*$-$*$). Then, in the protocol, since the peers have no static key and exchange ephemeral keys $X_1, \ldots, X_\beta$ and $Y_1, \ldots, Y_\beta$ where β is the number of ephemeral keys, the peers compute the session key only

from the ephemeral keys and public information, e.g., the session ID. When the adversary generates $Y_1, \ldots, Y_\beta$, the adversary can compute the same session key as the peer who is the owner of the test session. It is clear that there exists an adversary who can pass the indistinguishability test.

Lemma 2. *There exists no eCK-secure type* $(*\text{-}0\text{-}*)$ *AKE protocol.*

Proof. Assume that the AKE is type $(*\text{-}0\text{-}*)$. Then, all sessions are performed with static keys and public information, e.g., the session ID. When the adversary makes a query for static keys, the session keys in every session with the static keys are known and this means that forward secrecy cannot be assured. This concludes that any type $(*\text{-}0\text{-}*)$ AKE is insecure.

Lemma 3. *There exists no eCK-secure type* $(*\text{-}*\text{-}0)$ *AKE protocol.*

Proof. Assume that the AKE is type $(*\text{-}*\text{-}0)$. Then, the AKE without shared values cannot guarantee the secrecy of the session keys since the session key is generated only from the publicly available part. It is clear that there exists an adversary who can pass the indistinguishability test.

Proposition 1. *There exists no eCK-secure δ-count AKE protocol where δ is less than three.*

Proof. When $\delta < 3$, it is clear that some entry in its type must be 0. Then, the above three lemmas conclude this. □

2.4 Multiplied Biclique DH Protocol and Security

We introduce an imaginary AKE protocol, the *multiplied biclique DH protocol*, to analyze the internal structures of SMEN$^-$, FHMQV, KFU1, and UP. The shared value in the multiplied biclique DH protocol is computed by multiplying all DH values of possible pairs of static and ephemeral keys.

Assume party $\hat{A}$ wants to share a key with another party, $\hat{B}$. This means that $\hat{A}$ is the initiator and $\hat{B}$ is the responder of the session. $\hat{A}$ has a static private key, a, and its static public key A $(= g^a)$. $\hat{B}$ has a static private key, b, and its static public key B $(= g^b)$. Term g is a primitive element whose order is a prime, q, in group $\mathbb{G}$, and a and b are randomly selected in $\mathbb{Z}_q$. Then, $\hat{A}$ generates an ephemeral private key, $x \in \mathbb{Z}_q$, computes ephemeral public key X $(= g^x)$, and sends X to $\hat{B}$. In response, $\hat{B}$ generates an ephemeral private key, y $(\in \mathbb{Z}_q)$, computes ephemeral public key Y $(= g^y)$, and returns Y to $\hat{A}$. $\hat{A}$ computes one shared value, Z, as $Z = (YB)^{(x+a)}$ while $\hat{B}$ computes $Z = (XA)^{(y+b)}$. Both parties obtain session key K from the shared values and the session information using the session key derivation function, H.

In the multiplied biclique DH protocol, shared value Z is $g^{ya}g^{xb}g^{ab}g^{xy}$, and g^{ya}, g^{xb}, g^{ab}, and g^{xy} are DH values of all possible pairs of static and ephemeral keys. The multiplied biclique DH protocol is regular and a type (1-1-1) AKE protocol.

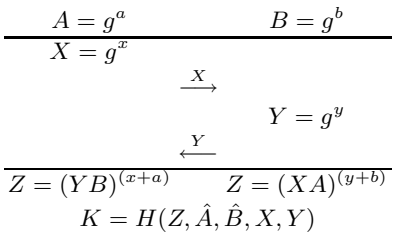


Fig. 1. Multiplied biclique DH protocol

We consider the security of the multiplied biclique DH protocol in the eCK model. We try to prove that the existence of an adversary breaks the GDH assumption. In other words, we construct a CDH solver with the DDH oracle from the eCK adversary of the protocol. The task of the CDH solver is to compute g^{uv} (with the help of the DDH oracle) given g, g^u $(= U)$, and g^v $(= V)$. To do so, the solver must simulate all responses of the queries asked by the adversary. We note that in the simulation, it is the most important point to keep consistency between the hash table and the session key table. The session key table contains a session key value with its identifier including $\hat{A}$, $\hat{B}$, X, and Y. The hash table contains a hash value of Z, $\hat{A}$, $\hat{B}$, X, and Y, and the value must coincide with a session key value where Z is the valid shared value of the session with respect to $(\hat{A}, \hat{B}, X, Y)$. Here, a shared value, Z is to be said *valid* on the session with respect to $(\hat{A}, \hat{B}, X, Y)$ when Z is equal to $g^{(ya+xb+ab+xy)}$ where x is the ephemeral private key of X, y is the ephemeral private key of Y, a is $\hat{A}$'s static private key of A, and b is $\hat{B}$'s static private key of B.

In the eCK model, the adversary is not allowed to access private information in either the static or ephemeral keys, which the owner and its peer (if one exists) of the test session possess [14]. The multiplied biclique DH protocol consists of four types of information: the static key of the initiator, the ephemeral key of the initiator, the static key of the responder, and the ephemeral key of the responder. The shared key contains all DH values of all meaningful combinations of the above keys, i.e., except the DH values of the static key and the ephemeral key owned by the same peer. So we may construct the solver by embedding U $(= g^u)$ and V $(= g^v)$ into the keys. We denote $u = \mathrm{dl}_g(U)$ when $U = g^u$.

The basic strategy to prove that an AKE is eCK-secure in the ROM is to guess the session that the eCK adversary chooses as the test session to embed an instance of a hard problem into the keys in which the adversary cannot access the private information, and to simulate all answers to the queries from the adversary. When the adversary can pass the indistinguishability test even in this situation, the adversary is forced to ask the hash query related to the test session since the session key derivation function is regarded as the random oracle. This means that we may extract the answer from the values in the query and can construct a solver for the hard problem.

Let the instance of the CDH problem be (g, U, V) where $U = g^u$ and $V = g^v$. Assume that the test session has the matching session. Then, the adversary is allowed to ask the following queries: (a) $\mathsf{EphemeralKeyReveal}(\mathtt{sid}^*)$ and

$\mathsf{EphemeralKeyReveal}(\overline{\mathtt{sid}^*})$, (b) $\mathsf{StaticKeyReveal}(\hat{A})$ and $\mathsf{StaticKeyReveal}(\hat{B})$. (c) $\mathsf{EphemeralKeyReveal}(\mathtt{sid}^*)$ and $\mathsf{StaticKeyReveal}(\hat{B})$, and (d) $\mathsf{StaticKeyReveal}(\hat{A})$ and $\mathsf{EphemeralKeyReveal}(\overline{\mathtt{sid}^*})$.

In case (a), the solver randomly generates x and y, and sets $A = U$ and $B = V$ where $\hat{A}$ is the owner of the test session and $\hat{B}$ is the peer. When the adversary makes a hash query including $\hat{A}$, $\hat{B}$, g^x, and g^y, the solver can extract the CDH answer, W, from Z in the query as $W = Z/U^y V^x g^{xy}$ with the help of the DDH oracle to ask $\mathrm{DDH}(U, V, W)$. If the adversary can correctly guess Z, then W is the CDH value of U and V since $W = Z/U^y V^x g^{xy} = g^{(yu+xv+uv+xy)}/g^{yu} g^{xv} g^{xy} = g^{uv}$.

The solver must simulate all other sessions; however, the solver can easily carry out these simulations. When the initiator and the responder of the session are neither $\hat{A}$ nor $\hat{B}$, the solver randomly generates the ephemeral and the static private keys, and then perfectly simulates the session.

Without loss of generality, assume that the initiator of the session is $\hat{A}$ and that the peer, $\hat{C}$, is controlled by the adversary. Let C be $\hat{C}$'s static public key. The solver can generate ephemeral private key x of the ephemeral public key which $\hat{A}$ sends to $\hat{C}$ but may not know both the static private key of C and the ephemeral private key of Y' which is generated by $\hat{C}$ (maybe the adversary). However, when the adversary makes a hash query including Z, $\hat{A}$, $\hat{C}$, g^x, and Y', the solver can check the validity of them with the help of the DDH oracle to ask $\mathrm{DDH}(Y'C, g^x U, Z)$ (note that $U = A$). When they are valid, the solver maintains the hash table and the session key table maintaining consistency between them.

Similar arguments can be applied to the other cases ((b), (c), and (d)). Thus, the security proof is accomplished when the test session has the matching session.

Next, assume that the test session does not have the matching session. Again, let the instance of the CDH problem be (g, U, V) where $U = g^u$ and $V = g^v$. In this situation, the message of the responder in the test session is modified by the adversary. Then, the solver must embed U or V of the instance into the static key of the responder. This is the most difficult case in the security proof. The validity check for the simulation can be easily performed with the help of the DDH oracle to ask $\mathrm{DDH}(YB, XA, Z)$ from publicly available values Y, B, X, and A, and the value Z in the hash query as $Z = (YB)^{(x+a)} = (XA)^{(y+b)}$.

It seems that the simulation can be performed perfectly. However, it is hard to extract the CDH value. The solver must set $B = V$ since ephemeral public key Y may be controlled by the adversary and the solver cannot embed V into Y. We assume that the solver embeds the other value, U, into A or X depending on the event. This relates to the third condition of the freshness definition. In other words, the adversary is not allowed to make either query: (i) both $\mathsf{StaticKeyReveal}(\hat{A})$ and $\mathsf{StaticKeyReveal}(\hat{B})$, or (ii) both $\mathsf{EphemeralKeyReveal}(\mathtt{sid}^*)$ and $\mathsf{StaticKeyReveal}(\hat{B})$.

Setting $A = U$ in the former case, the solver wants to extract g^{uv} from Z where $u = \mathrm{dl}_g(U)$ and $v = \mathrm{dl}_g(V)$. In this case, ephemeral key X can be set as $X = g^x$ where x is selected by the solver. Although the solver can obtain $g^{(uy+uv)} = Z/(YB)^x$ where $y = \mathrm{dl}_g(Y)$, it seems hard to extract g^{uv} without knowing u and y since the solver knows only x.

On the other hand, setting $X = U$ in the latter case, the solver wants to extract g^{uv} from Z. In this case, static key A can be set as $A = g^a$ where a is selected by the solver. Although the solver can obtain $g^{(uv+uy)} = Z/(YB)^a$, it seems hard to extract g^{uv} without knowing u and y since the solver knows only a.

The security proof seems to be difficult when the test session does not have the matching session. Indeed, the insecurity of this protocol is pointed out in the paper by Krawczyk that proposes HMQV [13]. The adversary chooses a value, $\tilde{x}$, computes $\tilde{X} = g^{\tilde{x}}/A$, and sends $\tilde{X}$ to $\hat{B}$ as the initial message from $\hat{A}$. $\hat{B}$ sends $Y = g^y$ and computes shared value $Z = (\tilde{X}A)^{(y+b)}$. Then, the adversary can also compute Z as $(BY)^{\tilde{x}}$.

At the end, let us observe a tuple, (g^{ya}, g^{xb}, g^{ab}, g^{xy}). It is worthy to note that when the value the solver wants to extract appears in a half of the tuple, then the value the solver cannot compute appears in the other half of the tuple. In the former case above, g^{ab} $(= g^{uv})$ appears in the second half of the tuple but the unknown value g^{ya} $(= g^{yu})$ appears in the first half. In the latter case above, g^{xb} $(= g^{uv})$ appears in the first half but the unknown value g^{xy} $(= g^{uy})$ appears in the second half.

3 Relations among SMEN$^-$, FHMQV, KFU1, and UP

In this section, we show that the previous four protocols (SMEN$^-$, FHMQV, KFU1, and UP) use the multiplied biclique DH protocol as an internal protocol and show how they overcome the above insecurity. Based on this analysis, we show the relations among the protocols.

3.1 KFU1 Protocol

KFU1 is a two-pass AKE protocol that does not use the NAXOS technique and is eCK-secure under the GDH assumption in the ROM (proposed as Protocol 1) [12]. In [12], they proposed an eCK-secure protocol (Protocol 2) also under the CDH assumption in the ROM using the twinning DH technique [5].

We describe KFU1 in Fig. 2. It is clear that KFU1 is a type (2-1-2) AKE protocol and that KFU1 performs two multiplied biclique DH protocols in parallel for two independent static keys with the same ephemeral key.

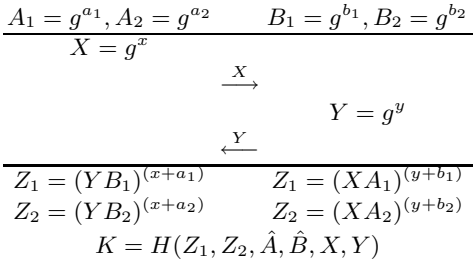


Fig. 2. KFU1 protocol

We consider a similar difficult case discussed in the security proof of the multiplied biclique DH protocol.

Setting $A_1 = U$, $A_2 = U^t$, $B_1 = V$, and $B_2 = V^r$ with randomly chosen t and r, the solver wants to extract $g^{a_1 b_1}$ from Z_1 and Z_2 where $a_1 = \mathrm{dl}_g(U)$ and $b_1 = \mathrm{dl}_g(V)$. Since the solver can set ephemeral key X $(= g^x)$, the solver computes $Z_1' = g^{(a_1 y + a_1 b_1)} = Z_1/(YB_1)^x$ and $Z_2' = g^{(a_2 y + a_2 b_2)} = Z_2/(YB_1)^x$ where $a_2 = \mathrm{dl}_g(U^t) = t a_1$ and $b_2 = \mathrm{dl}_g(V^r) = r b_1$. Then, the solver can extract $g^{a_1 b_1}$ from $Z_1'/(Z_2')^{1/t} = g^{(1-r)a_1 b_1}$.

On the other hand, setting $X = U$, $B_1 = V$, and $B_2 = V^r$ with randomly chosen r, the solver wants to extract $g^{x b_1}$ from Z_1 and Z_2 where $x = \mathrm{dl}_g(U)$ and $b_1 = \mathrm{dl}_g(V)$. Since the solver can set static keys A_1 $(= g^{a_1})$ and A_2 $(= g^{a_2})$, the solver computes $Z_1' = g^{(x b_1 + xy)} = Z_1/(YB_1)^{a_1}$ and $Z_2' = g^{(x b_2 + xy)} = Z_2/(YB_2)^{a_2}$ where $y = \mathrm{dl}_g(Y)$, $b_2 = \mathrm{dl}_g(V^r) = r b_1$. Then, the solver can extract $g^{x b_1}$ from $Z_1'/Z_2' = g^{(1-r)x b_1}$.

This shows that two shared values enable us to extract the CDH answer.

In the above protocol, the shared values are given as $Z_1 = (YB_1)^{(x+a_1)} = (XA_1)^{(y+b_1)}$ and $Z_2 = (YB_2)^{(x+a_2)} = (XA_2)^{(y+b_2)}$. However, even if we adopt $Z_1 = (YB_2)^{(x+a_1)} = (XA_1)^{(y+b_2)}$ and $Z_2 = (YB_1)^{(x+a_2)} = (XA_2)^{(y+b_1)}$ as the shared values, the modified protocol is still secure and is referred to as *cross-type KFU1* hereafter.

3.2 UP Protocol

UP is a two-pass AKE protocol that does not follow the NAXOS technique and is eCK-secure under the GDH assumption in the ROM [23].

We describe UP in Fig. 3. It is clear that UP is a type (1-1-2) AKE protocol.

UP can be explained using cross-type KFU1 as follows: $A_1 = A$, $A_2 = A^d$, $(a_1 = a,\ a_2 = da)$ and $B_1 = B$, $B_2 = B^e$, $(b_1 = b,\ b_2 = eb)$.

In other words, UP is a cross-type KFU1 protocol in which a static public key is generated from the other static public key and the ephemeral public key with additional function H', which must also be regarded as a random oracle.

$A = g^a$	$B = g^b$
$X = g^x$	
	$\xrightarrow{X}$
	$Y = g^y$
	$\xleftarrow{Y}$
$d' = H'(X)$	$d' = H'(X)$
$e' = H'(Y)$	$e' = H'(Y)$
$Z_1 = (YB^{e'})^{(x+a)}$	$Z_1 = (XA)^{(y+e'b)}$
$Z_2 = (YB)^{(x+d'a)}$	$Z_2 = (XA^{d'})^{(y+b)}$
$K = H(Z_1, Z_2, \hat{A}, \hat{B}, X, Y)$	

Fig. 3. UP protocol

3.3 SMEN$^-$ Protocol

SMEN$^-$ is a two-pass AKE protocol that does not use the NAXOS technique and is eCK-secure under the GDH assumption in the ROM [24]. In [24], they also proposed an eCK-secure protocol using the NAXOS technique, SMEN, under the GDH assumption in the ROM.

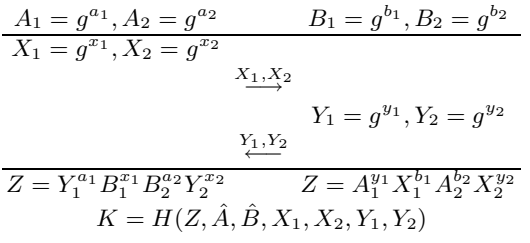


Fig. 4. SMEN$^-$ protocol

We describe SMEN$^-$ in Fig. 4. This shows that SMEN$^-$ is a type (2-2-1) AKE protocol. It is worthy to note that SMEN$^-$ achieves the smallest number of shared values. This will be discussed later.

We consider a similar difficult case discussed in the security proof of the multiplied biclique DH protocol.

Setting $A_2 = U$ and $B_2 = V$, the solver wants to extract $g^{a_2 b_2}$ from Z where $a_2 = \mathrm{dl}_g(U)$ and $b_2 = \mathrm{dl}_g(V)$. Since the solver can set the other static key, A_1 $(= g^{a_1})$, in addition to ephemeral keys X_1 $(= g^{x_1})$ and X_2 $(= g^{x_2})$, the solver can obtain $g^{a_2 b_2}$ by computing $Z/(B_1^{x_1} Y_1^{a_1} Y_2^{x_2})$.

On the other hand, setting $X_1 = U$ and $B_1 = V$, the solver wants to extract $g^{x_1 b_1}$ from Z where $x_1 = \mathrm{dl}_g(U)$ and $b_1 = \mathrm{dl}_g(V)$. Since the solver can set the other ephemeral key, X_2 $(= g^{x_2})$, in addition to static keys A_1 $(= g^{a_1})$ and A_2 $(= g^{a_2})$, the solver can obtain $g^{x_1 b_1}$ by computing $Z/(Y_1^{a_1} B_2^{a_2} Y_2^{x_2})$.

This enables us to extract the CDH answer from the single shared value.

SMEN$^-$ essentially utilizes two multiplied biclique DH protocols. Let us recall the discussion at the end of **Section 2.4**. The shared value of SMEN$^-$, Z, consists of $g^{y_1 a_1}$, $g^{x_1 b_1}$, $g^{a_2 b_2}$, and $g^{x_2 y_2}$, that is, they are the first half of a tuple, $(g^{y_1 a_1}, g^{x_1 b_1}, g^{a_1 b_1}, g^{x_1 y_1})$, and the second half of a tuple, $(g^{y_2 a_2}, g^{x_2 b_2}, g^{a_2 b_2}, g^{x_2 y_2})$. This trick enables us to extract the answer as the value we want to extract and the value we need to compute will appear in (imaginary) different protocols.

3.4 FHMQV Protocol

FHMQV is a two-pass AKE protocol that does not use the NAXOS technique and is eCK-secure under the GDH assumption in the ROM [21].

We describe FHMQV in Fig. 5. This shows that FHMQV is a type (1-1-1) AKE protocol. It is worthy to note that FHMQV achieves not only the smallest

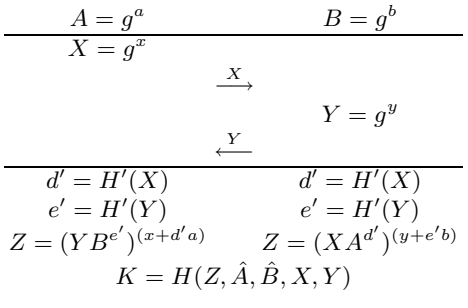


Fig. 5. FHMQV protocol

number of shared values but also the smallest number of all values. This will be discussed later.

We consider a similar difficult case discussed in the security proof of the multiplied biclique DH protocol.

Setting $A = U$ and $B = V$, the solver wants to extract g^{ab} from Z where $a = \mathrm{dl}_g(U)$ and $b = \mathrm{dl}_g(V)$. Since the solver can set ephemeral key X $(= g^x)$, the solver computes $\tilde{Z} = g^{(d'ay+d'e'ab)} = Z/(YB^{e'})^x$ with random oracle H' where $d' = H'(X)$ and $e' = H'(Y)$, and reiterates the protocol with different random oracle H''. Then the solver computes $\tilde{Z}' = g^{(d''ay+d''e''ab)} = Z'/(YB^{e''})^x$ where Z' is the shared value in the reiterated protocol, $d'' = H''(X)$, and $e'' = H'(Y)$. Then, the solver can extract g^{ab} from $(\tilde{Z})^{d''}/(\tilde{Z}')^{d'} = g^{(d'd''e'-d'd''e'')ab}$.

On the other hand, setting $X = U$ and $B = V$, the solver wants to extract g^{xb} from Z where $x = \mathrm{dl}_g(U)$ and $b = \mathrm{dl}_g(V)$. Since the solver can set static key A $(= g^a)$, the solver computes $\tilde{Z} = g^{(xy+e'xb)} = Z/(YB^{e'})^{d'a}$ with random oracle H' where $d' = H'(X)$ and $e' = H'(Y)$, and reiterates the protocol with different random oracle H''. Then, the solver computes $\tilde{Z}' = g^{(xy+e''xb)} = Z'/(YB^{e''})^{d''a}$ where Z' is the shared value in the reiterated protocol, $d'' = H''(X)$ and $e'' = H'(Y)$. Then, the solver can extract g^{xb} from $\tilde{Z}/\tilde{Z}' = g^{(e'-e'')xb}$.

This shows that reiterating the protocol with a different random oracle enables us to extract the CDH answer even from a single shared value. Therefore, the security proof of FHMQV requires the forking lemma [20]. Roughly speaking, the forking lemma implies that if an adversary wins a game with non-negligible probability, then there exist two random oracles where the adversary wins the game with non-negligible probability even when either random oracle is used. Thus, this proves the eCK security of FHMQV.

3.5 Reviewing SMEN$^-$, FHMQV, KFU1, and UP

KFU1 performs two multiplied biclique DH protocols in parallel for two independent static keys with the same ephemeral key. This enables us to extract the answer of the embedded problem by canceling the unknown part in each shared value by division. In other words, KFU1 is a two static key multiplied biclique DH protocol.

UP is a variant of the KFU1 protocol (cross-type KFU1). A static public key is generated from the other static public key and the ephemeral public key with additional function H' is regarded as a random oracle.

SMEN$^-$ can be regarded as a two ephemeral key variant of the KFU1 protocol and the additional ephemeral key enables us to extract the answer of the embedded problem from a single shared value by canceling the unknown part in the shared value since the solver knows the private key.

FHMQV can be regarded as a (imaginary) sequential variant of the KFU1 protocol and, in the security proof, reiterating the protocol enables us to extract the answer of the embedded problem by canceling the unknown part in each (single) shared value by division as KFU1.

These four protocols are two-pass AKE protocols that do not use the NAXOS technique and are eCK-secure under the GDH assumption in a ROM; however, their internal protocol, the multiplied biclique DH protocol, is insecure.

4 Efficiency Parameters

4.1 Comparison among Existing Protocols

We briefly summarize SMEN$^-$, FHMQV, KFU1, and UP.

SMEN$^-$, FHMQV, KFU1, and UP are type (2-2-1), type (1-1-1), type (2-1-2) AKEs, and type (1-1-2), respectively. In other words, SMEN$^-$, FHMQV, KFU1, and UP are 5-count, 3-count, 5-count, and 4-count AKEs, respectively. The multiplied biclique DH protocol is a type (1-1-1) AKE, i.e., a 3-count AKE, however it is insecure. This leads us to characterize the efficiency of an AKE protocol in terms of the count.

Table 1 shows a comparison of two-pass eCK-secure protocols under GDH in the ROM with an entry for the multiplied biclique DH protocol (MBC DH) from the viewpoint of data size. We denote the number of static keys, the number of ephemeral keys, and the number of shared values as #SK, #EK, and #SV, respectively. In the "NAXOS" entry, `o` means that the protocol uses the NAXOS technique, and `x` indicates that it does not.

Table 1. Comparison of two-pass eCK-secure protocols under GDH in ROM

Protocol	#SK	#EK	#SV	Count	NAXOS	Forking Lemma
(MBC DH)	1	1	1	3	x	—
FHMQV	1	1	1	3	x	Required
CMQV	1	1	1	3	o	Required
UP	1	1	2	4	x	Not required
NETS	1	1	2	4	o	Not required
SMEN	1	2	1	4	o	Not required
KFU1	2	1	2	5	x	Not required
SMEN$^-$	2	2	1	5	x	Not required
NAXOS	1	1	3	5	o	Not required

CMQV, NETS, SMEN, and NAXOS are two-pass eCK-secure AKE protocols that use the NAXOS technique under the GDH assumption in the ROM. On the contrary, SMEN$^-$, FHMQV, KFU1, and UP are two-pass eCK-secure protocols that do not follow the NAXOS technique under the GDH assumption in the ROM. NAXOS is the first eCK-secure protocol and the table shows that it uses many shared values. CMQV is the most efficient eCK-secure protocol among them in terms of the count. We should note that CMQV uses the forking lemma [20] to prove its security (as discussed later) so its security reduction is not as tight as those of others. NETS and SMEN have tighter security reductions than CMQV, and NETS uses the NAXOS technique but SMEN does not. Both SMEN and SMEN$^-$ achieve single shared key protocols with and without the NAXOS technique, respectively. UP is the most efficient eCK-secure protocol among the protocols that do not use the NAXOS technique in terms of the count. To the best of our knowledge, CMQV and FHMQV are the only protocols that achieve three in the count.

4.2 Open Problems

Note that the number of shared values in CMQV, SMEN$^-$, SMEN, and FHMQV is one so they achieve optimality in terms of the number of shared values.

The three-count AKE protocol is optimal in the sense of the number of static keys, ephemeral keys, and shared values. There exist a three-count eCK-secure protocol since CMQV and FHMQV satisfy the condition. It seems difficult to extract the CDH answer from one shared value in a type (1-1-1) AKE protocol as shown in the security discussion of the multiplied biclique DH protocol. In the security proof of CMQV and FHMQV, the CDH solver rewinds the adversary and runs it with a different random oracle, which is used to randomize the static keys. This means that the CDH solver obtains two substantially different shared values and then, the solver can extract the CDH value from them. However, this requires the forking lemma in the security proof of CMQV and FHMQV, and this implies that tightness of its security reduction is not as tight.

It is natural to raise the following questions.

1. Is it possible to devise a four-count eCK-secure protocol other than UP that does not utilize the NAXOS technique?
2. Is it possible to devise a three-count eCK-secure protocol that has a tighter security reduction than CMQV or FHMQV?

5 Conclusion

This paper proposed ways to characterize AKE protocols and defined two parameters: A regular AKE protocol is *type* (α-β-γ) when it has α static keys, β ephemeral keys, and γ shared values, and the *count* of a regular AKE protocol, δ, is defined as the total sum of the keys and the shared values.

Based on the characterization, we examined two-pass eCK-secure AKE protocols that do not use the NAXOS technique: SMEN$^-$, FHMQV, KFU1, and

UP. We showed relations among the protocols by introducing an imaginary protocol, the *multiplied biclique DH protocol*. KFU1 is type (2-1-2) and runs two multiplied biclique DH protocols in parallel with the same ephemeral key on two static keys. SMEN$^-$ is type (2-2-1) and a two ephemeral key variant of KFU1, and this reduces the number of shared values to one. UP is type (1-1-2) and a version of KFU1 in which one of the static public keys is generated with an additional function regarded as a random oracle. FHMQV is type (1-1-1) and a (imaginary) sequential variant of the KFU1 protocol, and it achieves optimality on each parameter.

References

1. Bellare, M., Rogaway, P.: Entity Authentication and Key Distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994)
2. Bellare, M., Rogaway, P.: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: 1st ACM Conference on Computer and Communications Security, pp. 62–73. ACM, New York (1993)
3. Blake-Wilson, S., Johnson, D., Menezes, A.: Key Agreement Protocols and Their Security Analysis. In: Darnell, M. (ed.) Cryptography and Coding 1997. LNCS, vol. 1355, pp. 30–45. Springer, Heidelberg (1997)
4. 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)
5. Cash, D., Kiltz, E., Shoup, V.: The Twin Diffie-Hellman Problem and Applications. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 127–145. Springer, Heidelberg (2008)
6. Cheng, Q., Ma, C., Hu, X.: A New Strongly Secure Authenticated Key Exchange Protocol. In: Park, J.H., Chen, H.-H., Atiquzzaman, M., Lee, C., Kim, T.-H., Yeo, S.-S. (eds.) ISA 2009. LNCS, vol. 5576, pp. 135–144. Springer, Heidelberg (2009)
7. Cremers, C.J.F.: Session-state Reveal Is Stronger Than Ephemeral Key Reveal: Attacking the NAXOS Authenticated Key Exchange Protocol. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 20–33. Springer, Heidelberg (2009)
8. Cremers, C.J.F.: Examining Indistinguishability-Based Security Models for Key Exchange Protocols: The Case of CK, CK-HMQV, and eCK. In: 6th ACM Symposium on Information, Computer and Communications Security, pp. 80–91. ACM, New York (2011)
9. Diffie, W., Hellman, M.E.: New Directions in Cryptography. IEEE Transactions of Information Theory 22(6), 644–654 (1976)
10. Fujioka, A., Suzuki, K.: Designing Efficient Authenticated Key Exchange Resilient to Leakage of Ephemeral Secret Keys. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 121–141. Springer, Heidelberg (2011)
11. Huang, H., Cao, Z.: Strongly Secure Authenticated Key Exchange Protocol Based on Computational Diffie-Hellman Problem. http://eprint.iacr.org/2008/500 (accepted as a short paper for Inscrypt 2008)
12. Kim, M., Fujioka, A., Ustaoğlu, B.: Strongly Secure Authenticated Key Exchange without NAXOS' Approach. In: Takagi, T., Mambo, M. (eds.) IWSEC 2009. LNCS, vol. 5824, pp. 174–191. Springer, Heidelberg (2009)

13. Krawczyk, H.: HMQV: A High-Performance Secure Diffie-Hellman Protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005)
14. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger Security of Authenticated Key Exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (2007)
15. Lee, J., Park, C.: An Efficient Key Exchange Protocol with a Tight Security Reduction, http://eprint.iacr.org/2008/345
16. Lee, J., Park, J.: Authenticated Key Exchange Secure under the Computational Diffie-Hellman Assumption, http://eprint.iacr.org/2008/344
17. Moriyama, D., Okamoto, T.: An eCK-Secure Authenticated Key Exchange Protocol without Random Oracles. In: Pieprzyk, J. P., Zhang, F. (eds.) ProvSec 2009. LNCS, vol. 5848, pp. 154–167. Springer, Heidelberg (2009)
18. Okamoto, T.: Authenticated Key Exchange and Key Encapsulation in the Standard Model. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 474–484. Springer, Heidelberg (2007)
19. Okamoto, T., Pointcheval, D.: The Gap-Problems: A New Class of Problems for the Security of Cryptographic Schemes. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 104–118. Springer, Heidelberg (2001)
20. Pointcheval, D., Stern, J.: Security Proofs for Signature Schemes. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (1996)
21. Sarr, A.P., Elbaz-Vincent, P., Bajard, J.-C.: A Secure and Efficient Authenticated Diffie-Hellman Protocol. In: Martinelli, F., Preneel, B. (eds.) EuroPKI 2009. LNCS, vol. 6391, pp. 83–98. Springer, Heidelberg (2010)
22. Ustaoğlu, B.: Obtaining a Secure and Efficient Key Agreement Protocol for (H)MQV and NAXOS. Designs, Codes and Cryptography 46(3), 329–342 (2008)
23. Ustaoğlu, B.: Comparing *SessionStateReveal* and *EphemeralKeyReveal* for Diffie-Hellman protocols. In: Pieprzyk, J. P., Zhang, F. (eds.) ProvSec 2009. LNCS, vol. 5848, pp. 183–197. Springer, Heidelberg (2009)
24. Wu, J., Ustaoğlu, B.: Efficient Key Exchange with Tight Security Reduction. Technical Report CACR 2009-23, http://eprint.iacr.org/2009/288

A Secure M + 1st Price Auction Protocol Based on Bit Slice Circuits

Takuho Mitsunaga[1], Yoshifumi Manabe[2], and Tatsuaki Okamoto[3]

[1] Graduate School of Informatics, Kyoto University, Sakyo-ku Kyoto city Japan
[2] NTT Communication Science Laboratories NTT Laboratory
2-4 Hikari-dai, Soraku-gun, Seikacho, Kyoto, Japan
[3] NTT Information Sharing Platform Laboratories
3-9-11 Midoricho, Musashino city, Tokyo, Japan

Abstract. This paper presents an efficient secure auction protocol for $M+1st$ price auction. In our proposed protocol, bidding prices are represented as binary numbers. Thus, when the bidding price is an integer up to p and the number of bidders is m, the complexity of our protocol is a polynomial of $\log p$ and m, while in previous secure $M+1st$ price auction protocols, the complexity is a polynomial of p and m. We apply the Boneh-Goh-Nissim encryption to the mix-and-match protocol to reduce the computation costs.

1 Introduction

1.1 Background

Recently, as the Internet has expanded, many researchers have become interested in secure auction protocols and various schemes have been proposed to ensure the safe transaction of sealed-bid auctions. A secure auction is a protocol in which each player can find only the highest bid and its bidder (called the first price auction) or the second highest bid and the first price bidder (called the second price auction). There is also a generalized auction protocol called $M+1st$ price acution. The $M+1st$ price auction is a type of sealed-bid auction for selling M units of a single kind of goods, and the $M+1st$ highest price is the winning price. M bidders who bid prices higher than the winning price are the winning bidders, and each winning bidder buys one unit of the goods at the winning price.

A simple solution is to assume a trusted auctioneer. Bidders encrypt their bids and send them to the auctioneer, and the auctioneer decrypts them to decide the winner. To remove the trusted auctioneer, some secure multi-party protocols have been proposed. The common essential idea is the use of threshold cryptosystems, where a private decryption key is shared by the players. Jakobsson and Juels proposed a secure MPC protocol to evaluate a function comprising a logical circuit, called mix-and-match [6]. As for a target function f and the circuit that calculates f, C_f, all players evaluate each gate in C_f based on their

T. Iwata and M. Nishigaki (Eds.): IWSEC 2011, LNCS 7038, pp. 51–64, 2011.

encrypted inputs and the evaluations of all the gates in turn lead to the evaluation of f. Based on the mix-and-match protocol, we can easily find a secure auction protocol by repeating the millionaires' problem for two players. Kurosawa and Ogata suggested the "bit-slice auction", which is an auction protocol that is more efficient than the one based on the millionaire's problem [8].

Boneh, Goh and Nissim suggested a public evaluation system for 2-DNF formula based on an encryption of Boolean variables [3]. Their protocol is based on Pallier's scheme [13], so it has additive homomorphism in addition to the bilinear map, which allows one multiplication on encrypted values. As a result, this property allows the evaluation of multivariate polynomials with the total of degree two on encrypted values.

In this paper, we introduce an efficient secure auction protocol for $M + 1st$ price auction, in which if the bidding price is an integer up to p and the number of bidders is m, the complexity of our protocol is a polynomial of log p and m.

1.2 Related Works

As related works, there are many secure auction protocols, however, they have problems such as those described hereafter. The secure auction scheme for first price auction proposed by Franklin and Reiter [5] does not provide full privacy, since at the end of an auction players can know the other players' bids. Naor, Pinkas and Sumner achieved a secure second price auction by combining Yao's secure computation with oblivious transfer assuming two types of auctioneers [10]. However, the cost of the bidder communication is high because it proceeds bit by bit using the oblivious transfer protocol. Juels and Szydlo improved the efficiency and security of this scheme with two types of auctioneers through verifiable proxy oblivious transfer [7], which still has a security problem in which if both types of auctioneers collaborate they can retrieve all bids. Mitsunaga, Manabe and Okamoto suggested secure auction protocols for first and second price auction. They applied Boneh-Goh-Nissim Encryption to the bit-slice auction protocol to improve computation costs [11].

For $M + 1st$ price auction, Lipmaa, Asokan and Niemi proposed an efficient secure $M+1st$ auction scheme [9]. In their scheme, the trusted auction authority can know the bid statistics. Abe and Suzuki suggested a secure auction scheme for the $M + 1st$ auction based on homomorphic encryption [1]. However in their scheme, a player's bid is not a binary expression. So, its time complexity is $O(m2^p)$ for a m-player and p-bit bidding price auction.

1.3 Our Result

This paper presents an efficient secure auction protocol for $M+1st$ price auction. In our proposed protocol, bidding prices are represented as binary numbers. Thus, when the bidding price is an integer up to p and the number of bidders is m, the complexity of our protocol is a polynomial of log p and m, while in previous secure $M+1st$ price auction protocols[1], the complexity is a polynomial of p and m.

2 Preliminaries

2.1 The Model of Auction and Outline of Auction Protocol

This model involves n players, denoted by $P_1, P_2, ..., P_m$ and assumes that there exists a public board. The players agree in advance on the presentation of the target function, f as a circuit C_f. For each player P_i's bidding price Z_i, the aim of the protocol is for players to compute $f(Z_1, ..., Z_m)$ without revealing any additional information. Its outline is as follows.

1. **Input stage:** Each $P_i (1 \leq i \leq m)$ computes ciphertexts of the bits of Z_i and broadcasts them and proves that the ciphertext represents 0 or 1 by using the zero-knowledge proof technique in [3].
2. **Mix and Match stage:** The players blindly evaluates each gate, G_j, in order.
3. **Output stage:** After evaluating the last gate G_M, the players obtain O_M, a ciphertext encrypting $f(Z_1, ..., Z_m)$. They jointly decrypt this ciphertext value to reveal the output of function f.

2.2 Mix and Match Protocol

Requirements for the Encryption Function. Let E be a public-key probabilistic encryption function. We denote the set of encryptions for a plaintext m by $E(m)$ and a particular encryption of m by $c \in E(m)$.

Function E must satisfy the following properties.

1.Homomorphic property. There exist polynomial time computable operations, $^{-1}$ and $\otimes$, as follows. For a large prime q,

1. If $c \in E(m)$, then $c^{-1} \in E(-m \bmod q)$.
2. If $c_1 \in E(m_1)$ and $c_2 \in E(m_2)$, then $c_1 \otimes c_2 \in E(m_1 + m_2 \bmod q)$.

For a positive integer a, define
$a \cdot e = \underbrace{c \otimes c \otimes \cdots \otimes c}_{a}$.

2.Random re-encryption. Given $c \in E(m)$, there is a probabilistic re-encryption algorithm that outputs $c' \in E(m)$, where c' is uniformly distributed over $E(m)$.

3.Threshold decryption. For a given ciphertext $c \in E(m)$, any t out of n players can decrypt c along with a zero-knowledge proof of the correctness. However, any t-1 out of n players cannot decrypt c.

MIX Protocol. The MIX protocol [4] takes a list of ciphertexts, $(\xi_1, ..., \xi_L)$, and outputs a permuted and re-encrypted list of the ciphertexts $(\xi'_1, ..., \xi'_L)$ without revealing the relationship between $(\xi_1, ..., \xi_L)$ and $(\xi'_1, ..., \xi'_L)$, where ξ_i or ξ'_i can be a single ciphertext c, or a list of l ciphertexts, $(c_1, ..., c_l)$, for some $l > 1$. For all players to verify the validity of $(\xi'_1, ..., \xi'_L)$, we use the universal verifiable MIX net protocol described in [14].

Plaintext Equality Test. Given two ciphertexts $c_1 \in E(v_1)$ and $c_2 \in E(v_2)$, this protocol checks if $v_1 = v_2$. Let $c_0 = c_1 \otimes c_2^{-1}$.

1. (Step 1) For each player P_i (where $i = 1,...,n$):
   P_i chooses a random element $a_i \in \mathbb{Z}_q^*$ and computes $z_i = a_i \cdot c_0$. He broadcasts z_i and proves the validity of z_i in zero-knowledge.
2. (Step 2) Let $z = z_1 \otimes z_2 \otimes \cdots \otimes z_n$. The players jointly decrypt z using threshold verifiable decryption and obtain plaintext v. Then it holds that

$$v = \begin{cases} 0 & if\ v_1 = v_2 \\ random & otherwise \end{cases}$$

Mix and Match Stage. For each logical gate, $G(x_1, x_2)$, of a given circuit, n players jointly computes $E(G(x_1, x_2))$ from $c_1 \in E(x_1)$ and $c_2 \in E(x_2)$ keeping x_1 and x_2 secret. For simplicity, we show the mix-and-match stage for AND gate.

1. n players first consider the standard encryption of each entry in the table shown below.
2. By applying a MIX protocol to the four rows of the table, n players jointly compute blinded and permuted rows of the table. Let the ith row be (a'_i, b'_i, c'_i) for $i = 1,...,4$.
3. n players next jointly find the row i such that the plaintext of c_1 is equal to that of a'_i and the plaintext of c_2 is equal to that of b'_i by using the plaintext equality test protocol.
4. For the row i, it holds that $c'_i \in E(x_1 \wedge x_2)$.

Table 1. Mix-and-match table for AND

x_1	x_2	$x_1 \wedge x_2$
$a'_1 \in E(0)$	$b'_1 \in E(0)$	$c'_1 \in E(0)$
$a'_2 \in E(0)$	$b'_2 \in E(1)$	$c'_2 \in E(0)$
$a'_3 \in E(1)$	$b'_3 \in E(0)$	$c'_3 \in E(0)$
$a'_4 \in E(1)$	$b'_4 \in E(1)$	$c'_4 \in E(1)$

2.3 Evaluating 2-DNF Formulas on Ciphertexts

Given encrypted Boolean variables $x_1, ..., x_n \in \{0, 1\}$, a mechanism for public evaluation of a 2-DNF formula was suggested in [3]. They presented a homomorphic public key encryption scheme based on finite groups of composite order that supports a bilinear map. In addition, the bilinear map allows for one multiplication on encrypted values. As a result, their system supports arbitrary additions and one multiplication on encrypted data. This property in turn allows the evaluation of multivariate polynomials of a total degree of two on encrypted values.

Bilinear Groups. Their construction makes use of certain finite groups of composite order that supports a bilinear map. We use the following notation.

1. $\mathbb{G}$ and $\mathbb{G}_1$ are two (multiplicative) cyclic groups of finite order n.
2. g is a generator of $\mathbb{G}$.
3. e is a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$.

Subgroup Decision Assumption. We define algorithm $\mathcal{G}$ such that given security parameter $\tau \in \mathbb{Z}^+$ outputs a tuple $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$ where $\mathbb{G}, \mathbb{G}_1$ are groups of order $n = q_1 q_2$ and $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$ is a bilinear map. On input τ, algorithm $\mathcal{G}$ works as indicated below,

1. Generate two random τ-bit primes, q_1 and q_2 and set $n = q_1 q_2 \in \mathbb{Z}$.
2. Generate a bilinear group $\mathbb{G}$ of order n as described above. Let g be a generator of $\mathbb{G}$ and $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$ be the bilinear map.
3. Output $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$.
   We note that the group action in $\mathbb{G}$ and $\mathbb{G}_1$ as well as the bilinear map can be computed in polynomial time.

Let $\tau \in \mathbb{Z}^+$ and let $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$ be a tuple produced by $\mathcal{G}$ where $n = q_1 q_2$. Consider the following problem. Given $(n, \mathbb{G}, \mathbb{G}_1, e)$ and an element $x \in \mathbb{G}$, output '1' if the order of x is q_1 and output '0' otherwise, that is, without knowing the factorization of the group order n, decide if an element x is in a subgroup of $\mathbb{G}$. We refer to this problem as the subgroup decision problem.

Homomorphic Public Key System. We now introduce the public key system which resembles the Pallier [13] and the Okamoto-Uchiyama encryption schemes [12]. We describe the three algorithms comprising the system.

1.KeyGen Given a security parameter $\tau \in \mathbb{Z}$, run $\mathcal{G}$ to obtain a tuple $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$. Let $n = q_1 q_2$. Select two random generators, g and $u \xleftarrow{R} \mathbb{G}$ and set $h = u^{q_2}$. Then h is a random generator of the subgroup of $\mathbb{G}$ of order q_1. The public key is $PK = (n, \mathbb{G}, \mathbb{G}_1, e, g, h)$. The private key is $SK = q_1$.

2.Encrypt(PK, M) We assume that the message space consists of integers in set $\{0, 1, ..., T\}$ with $T < q_2$. We encrypt the binary representation of bids in our main application, in the case $T = 1$. To encrypt a message m using public key PK, select a random number $r \in \{0, 1, ..., n-1\}$ and compute

$$C = g^m h^r \in \mathbb{G}.$$

Output C as the ciphertext.

3.Decrypt(SK, C) To decrypt a ciphertext C using the private key $SK = q_1$, observe that $C^{q_1} = (g^m h^r)^{q_1} = (g^{q_1})^m$. Let $\hat{g} = g^{q_1}$. To recover m, it suffices to compute the discrete log of C^{q_1} base $\hat{g}$.

Homomorphic Properties. The system is clearly additively homomorphic. Let $(n, \mathbb{G}, \mathbb{G}_1, e, g, h)$ be a public key. Given encryptions C_1 and $C_2 \in \mathbb{G}_1$ of messages m_1 and $m_2 \in \{0, 1, ..., T\}$ respectively, anyone can create a uniformly distributed encryption of $m_1 + m_2 \bmod n$ by computing the product $C = C_1 C_2 h^r$ for a random number $r \in \{0, 1, ..., n-1\}$. More importantly, anyone can multiply two encrypted messages once using the bilinear map. Set $g_1 = e(g, g)$ and $h_1 = e(g, h)$. Then g_1 is of order n and h_1 is of order q_1. Also, write $h = g^{\alpha q_2}$ for some (unknown)$\alpha \in \mathbb{Z}$. Suppose we are given two ciphertexts $C_1 = g^{m_1} h^{r_1} \in \mathbb{G}$ and $C_2 = g^{m_2} h^{r_2} \in \mathbb{G}$. To build an encryption of product $m_1 \cdot m_2 \bmod n$ given only C_1 and C_2, 1) select random $r \in \mathbb{Z}_n$, and 2) set $C = e(C_1, C_2) h_1^r \in \mathbb{G}_1$. Then

$$C = e(C_1, C_2) h_1^r = e(g^{m_1} h^{r_1}, g^{m_2} h^{r_2}) h_1^r$$
$$= g_1^{m_1 m_2} h_1^{m_1 r_2 + r_2 m_1 + q_2 r_1 r_2 \alpha + r} = g_1^{m_1 m_2} h_1^{r'} \in \mathbb{G}_1$$

where $r' = m_1 r_2 + r_2 m_1 + q_2 r_1 r_2 \alpha + r$ is distributed uniformly in $\mathbb{Z}_n$ as required. Thus, C is a uniformly distributed encryption of $m_1 m_2 \bmod n$, but in the group $\mathbb{G}_1$ rather than $\mathbb{G}$ (this is why we allow for just one multiplication). We note that the system is still additively homomorphic in $\mathbb{G}_1$. For simplicity, in this paper we denote an encryption of message m in $\mathbb{G}$ as $E_G(m)$ and one in $\mathbb{G}_1$ as $E_{G_1}(m)$.

2.4 Key Sharing

In [2], efficient protocols are presented for a number of players to jointly generate RSA modulus $N = pq$ where p and q are prime, and each player retains a share of N. In this protocol, none of the players can know the factorization of N. They then show how the players can proceed to compute a public exponent e and the shares of the corresponding private exponent. At the end of the computation the players are convinced that N is a product of two large primes by using zero-knowledge proof. Their protocol was based on the threshold decryption that m out of m players can decrypt the secret. The cost of key generation for the shared RSA private key is approximately 11 times greater than that for simple RSA key generation. However the cost for computation is still practical. We use this protocol to share private keys among auction managers. We can assume that auction managers are either a subset of players or a different group such as management group for auctions.

3 New Efficient Auction Protocol

In this section, we show an efficient $M + 1st$ price auction based on bit-slice auction protocols. Compared to previous works on secure $M+1st$ price auctions, proposed protocol is more efficient because bidding prices are represent as binary numbers, however it needs high computation costs if a quite large number of players participate an auction. Because complexity of proposed protocol is a polynomial of m for the m-player auction.

3.1 Proposed $M + 1st$ Price Auction Protocol

We define three types of player's status on j-th bit as $W_j(Winner)$, $C_j(Candidate)$ and $S_j(Survivor)$ shown as below and the numbers of players in W_j and S_j as $\mid W_j \mid$ and $\mid S_j \mid$. We define the status of players for m-player and k-bit bidding price shown as below,

$W_j[1...m]$: $W_j[i]$=1 if player P_i is decided to be a winner by upper $k-j$ bits of the bids.

$C_j[1...m]$: $C_j[i]$=1 if player P_i is not decided to be a winner but has a possibility of $M+1st$ highest bidder by upper $k-j$ bits of the bids.

$S_j[1...m]$: $S_j[i]$=1 if $C_j[i]$=1 and j-th bit of P_i's bid is 1.

Suppose that $B_{M+1st} = (b_{M+1st}^{(k-1)}, ..., b_{M+1st}^{(0)})_2$ is the $M+1st$ highest bidding price and $Z_i = (z_i^{(k-1)}, ..., z_i^{(0)})_2$ is the bid of player i, where $()_2$ is the binary expression. The winners and winning price are found by the following protocol.

As initial setting, we set $W_k[i]$=0 $(1 \le i \le m)$ and $C_k[i]$=1 $(1 \le i \le m)$.

For $j = k$-1 to 0
  $S_j[i]$=$C_{j+1}[i]$ * $z_i^{(j)}$ $(1 \le i \le m)$
  if $\mid W_{j+1} \mid + \mid S_j \mid > M$ then
    $b_{M+1st}^{(j)}$=1
    $C_j[i] = S_j[i]$ $(1 \le i \le m)$
    $W_j[i] = W_{j+1}[i]$ $(1 \le i \le m)$
  else
    $b_{M+1st}^{(j)}$=0
    $W_j[i] = W_{j+1}[i] + S_j[i]$ $(1 \le i \le m)$
    $C_j[i] = C_{j+1}[i]$ - $S_j[i]$ $(1 \le i \le m)$
  end
end

If the number of Winners on $(j+1)$-th bit and Survirors on j-th bit is more than M, we keep Winners remained and update Candidates to eliminate players i in a set of $(C_j[i] - S_j[i])$, because they have no possibility to win the auction.

If the number of Winners on $(j+1)$-th bit and Survirors on j-th bit is less than or equal to M, Survivors on j-th bid are determined as Winners, so we update W_j as $W_{j+1}[i] + S_j[i]$ and eliminate players i of $S_j[i]$ from $C_{j+1}[i]$.

3.2 Example

We show an example 5-player auction for 3 goods (M=3). The information we need to find are the first, second and third highest bidders as the winners of the auction and the forth highest bidding price as the winning price. Assume each player's bid as follows,

$P_1 = 11 = (1011)_2$
$P_2 = \ 7 = (0111)_2$
$P_3 = \ 5 = (0101)_2$

Table 2. Example of 5-player auction for 3 goods

	C_5	W_5	K_4	S_4	C_4	W_4	K_3	S_3	C_3	W_3	K_2	S_2	C_2	W_2	K_1	S_1	C_1	W_1
P_1	1	0	1	1	0	1	0	0	0	1	1	0	0	1	1	0	0	1
P_2	1	0	0	0	1	0	1	1	1	0	1	1	0	1	1	0	0	1
P_3	1	0	0	0	1	0	1	1	1	0	0	0	1	0	1	1	0	1
P_4	1	0	0	0	1	0	1	1	1	0	0	0	1	0	0	0	1	0
P_5	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0
$\mid W \mid$ and $\mid S \mid$		0		1		1		3		1		1		2		1		3
B_{M+1st}				0				1				0				0		

$P_4 = \ 4 = (0100)_2$
$P_5 = \ 1 = (0001)_2$

So, the winners are P_1, P_2 and P_3 and the winning price is 4. We denote K_j be the vector of players' j-th bid as $K_j = (z_1^{(j)}, z_2^{(j)}, z_3^{(j)}, z_4^{(j)}, z_5^{(j)})$.

We also denote W_j, C_j and S_j be the vector of players' status, Winner, Candidate and Survivors on j-th bid respectively.

For initial setting j =5, all players have possibilities to win the auction, so according to the definition of the player status all players are Candidates and they are not decided to win the auction yet, so none of them are Winners.

Next step j =4, only P_1's bid is 1, so P_1 is decided to be Survivor and the number of Winner on upper bit and Survivor on 4th bid is 1. Then, by following the protocol, P_1 is the Winner and is removed from Candidate. The other players are kept to be Candidates because they have still possibilities to win the auction.

Next step j =3, bids of P_2, P_3 and P_4 are 1, so they are decided as Suvivors. The number of Winner on upper bit and Survivor on 3rd bid is 4, which means P_2, P_3 and P_4 can not decided to be Winners but kept to be Candidates and P_5 already loses the auction.

Following the protocol, from the 1st bits of the bids P_1, P_2 and P_3 are decided to be Winners. The winning price is shown in the row of B_{M+1st} in the table 2.

3.3 Secure $M + 1st$ Price Auction Using 2-DNF Scheme and Mix-and-Match Protocol

We assume n players, $P_1, ..., P_n$ and a set of auction managers, AM. The players bid their encrypted prices and broadcast them. The AM runs an auction protocol with the encrypted bids and after the auction AM jointly decrypts the results of the protocol and broadcast it to the players. Players can verify the winning price (the $M + 1st$ price) and the winners from the encrypted bidding prices by using verification protocols. To maintain secrecy of the players' bidding prices through the protocol, we need to use the mix-and-match protocol. Here, we define two types of new tables, MAP_1 and MAP_2. In the proposed protocol, the MAP_1 and MAP_2 tables are created among AM before an auction. The AM jointly computes values in the mix-and-match table for distributed decryption of plaintext equality test. The function of table MAP_1, shown in Table 2, is a

mapping $x_1 \in \{E_{G_1}(0), E_{G_1}(1)\} \rightarrow x_2 \in \{E_G(0), E_G(1)\}$. The table MAP_2, shown in Table 3, is the one for mapping $x_1 \in \{E_{G_1}(0), E_{G_1}(1), ..., E_{G_1}(m)\} \rightarrow x_2 \in \{E_G(0), E_G(1)\}$. These tables can be constructed using the mix-and-match protocol because the Boneh-Goh-Nissim encryption has homomorphic properties.

Setting. AM jointly generates and shares private keys among themselves using the technique described in [2].

Bidding Phase. Suppose that $B_{M+1st} = (b_{M+1st}^{(k-1)}, ..., b_{M+1st}^{(0)})_2$ is the $M+1st$ highest bidding price and a bid of a player i is $Z_i = (z_i^{(k-1)}, ..., z_i^{(0)})_2$, where $()_2$ is the binary expression. Each player P_i computes a ciphertext of his bidding price, Z_i, as

$$ENC_i = (b_i^{k-1}, ..., b_i^0)$$

where $b_i^j \in E_G(z_i^{(j)})$, and publishes ENC_i on the bulletin board. He also proves in zero-knowledge that $z_i^{(j)} = 0$ or 1 by using the technique described in [3].

Opening Phase. Let $C_k = (c_1^k, .., c_m^k)$, where each $c_i^k \in E_G(1)$ and $W_k = (w_1^k, .., w_m^k)$, where each $w_i^k \in E_{G_1}(0)$.

(Step 1) For $j = k$ -1 to 0, perform the following.
(Step 1-a) For $C_j = (c_1^j, ..., c_m^j)$, AM computes $s_i^j = Mul(b_i^j, c_i^j)$ for each player i, and

$$S_j = (Mul(c_1^j, b_1^j), ..., Mul(c_m^j, b_m^j))$$
$$h_j = Mul(b_1^j, c_1^j) \otimes \cdots \otimes Mul(b_m^j, c_m^j)$$
$$d_j = w_1^j \otimes \cdots \otimes w_m^j$$

(Step 1-b) The AM uses table MAP_1 for s_i^j for each i and finds the values of $\tilde{s}_i^j$. Let $\widetilde{S}_j = (\tilde{s}_1^j, ..., \tilde{s}_m^j)$.
(Step 1-c) AM uses table MAP_2 for $d_j \otimes h_j$ and decrypts the output value. The reason MAP_2 is used here is to prevent AM finding any other infomation except $d_j \otimes h_j$ is more than $M+1$ or not. If the output value is 0, the number of winners and survivors are less than $M+1$. Then, AM updates

$W_j = W_{j+1} + S_j = (w_1^{j+1} \otimes s_1^j, .., w_m^{j+1} \otimes s_m^j)$
$C_{j-1} = C_j - \widetilde{S}_j = (c_1^j \otimes (\tilde{s}_1^j)^{-1}, .., c_m^j \otimes (\tilde{s}_m^j)^{-1})$
$b_{M+1st}^{(i)} = 0$

If the output value is 1, then

$W_j = W_{j+1} = (w_1^{j+1}, .., w_m^{j+1})$
$C_{j-1} = \widetilde{S}_j = (\tilde{s}_1^j, ..., \tilde{s}_m^j)$
$b_{M+1st}^{(i)} = 1$

(Step 2) For the final $W_0 = (w_1^0, ..., w_m^0)$, AM decrypts each w_i^0 with verification protocols and obtains the winners of the auction. P_i is the winners if and only

Table 3. Table for MAP_1

x_1	x_2
$a_1 \in E_{G_1}(0)$	$b_1 \in E_G(0)$
$a_2 \in E_{G_1}(1)$	$b_2 \in E_G(1)$

Table 4. Table for MAP_2

x_1	x_2
$a_1 \in E_{G_1}(0)$	$b_1 \in E_G(0)$
$a_2 \in E_{G_1}(1)$	$b_2 \in E_G(0)$
$\cdots$	$b_i \in E_G(0)$
$a_{M+1} \in E_{G_1}(M)$	$b_{M+1} \in E_G(0)$
$a_{M+2} \in E_{G_1}(M+1)$	$b_{M+2} \in E_G(1)$
$\cdots$	$b_i \in E_G(1)$
$a_{m+1} \in E_{G_1}(m)$	$b_{m+1} \in E_G(1)$

if plaintexts of $w_i^0 = 1$ and $\sum w_i^0 = M$. The $M+1st$ highest price is obtained as $B_{M+1st} = (b_{M+1st}^{(k-1)}, ..., b_{M+1st}^{(0)})_2$.

If more than M players bid the same price which is $M+1st$ highest, such as a case four players bid the same price for 5-player auction for 3 goods, this protocol does not work well. At the end of auction, winners and winning price can not be decided.

Verification Protocols

Verification protocols are the protocols for players to confirm that AM decrypts the ciphertext correctly. By using the protocols, each player can verify the results of the auction are correct. We denote b as a palintext and C as a BGN encryption of b ($C = g^b h^r$), where g, h and r are elements used in BGN scheme and $f = C(g^b)^{-1}$. Before a player verifies whether b is the plaintext of C, the player must prove that a challenge ciphertext $C' = g^x f^r$ is created by himself with zero-knowledge proof that he has the value of x.

1. A player proves that he has random element $x \in Z_n^*$ with zero-knowledge proof.
2. The player computes $f = C(g^b)^{-1}$ from the published values, h, g and b, and select a random integer $r \in \mathbb{Z}_n^*$. He sends $C' = g^x f^r$ to AM.
3. The AM decrypts C' and sends value x' to the player.
4. The player verifies whether $x = x'$. AM can decrypt C' correctly only if order$(f) = q_1$, which means that the AM correctly decrypts C and publishes b as the plaintext of C.

3.4 Security

1. **Privacy for bidding prices**
   Each player can not retrieve any information except for the winners and the $M+1st$ highest price. An auction scheme is secure if there is no polynomial

$$
\begin{array}{c}
(PK, SK) \leftarrow KeyGen \\
(m_0, m_1, s) \leftarrow A_1^{o_1}(PK) \\
b \leftarrow \{0, 1\} \\
c \leftarrow Encrypt(PK, m_b) \\
b' \leftarrow A_2^{o_1}(c, s) \\
return\ 1\ \text{iff}\ b\ =\ b'
\end{array}
$$

Fig. 1. $EXPT_{A,\Pi}$

time adversary that breaks privacy with non-negligible advantage $\epsilon(\tau)$. We prove that the privacy for bidding prices in the proposed auction protocols under the assumption that BGN encryption with the mix-and-match oracle is semantically secure. Given a message m, the mix-and-match oracle receives an encrypted value $x_1 \in E_{G_1}(m)$ and returns the encrypted value $x_2 \in E_G(m)$ according to the mix-and-match table shown in Table 3. (which has the same function as MAP_2). Given a message m and the ciphertext $x_1 \in E_{G_1}(m)$, the function of mix-and-match table is to map $x_1 \in E_{G_1}(m) \rightarrow x_2 \in E_G(m)$. The range of the input value is supposed to be {0,1,...,m} and the range of the output is {0,1}. We do not consider cases where the input values are out of the range. Using this mix-and-match oracle, an adversary can compute any logical function without the limit where BGN encryption scheme can use only one multiplication on encrypted values. MAP_1 can also be computed if the range of the input value is restricted in {0,1}. Here, we define two semantic secure games and advantages for BGN encryption scheme and the proposed auction protocols. We also show that if there is adversary $\mathcal{B}$ that breaks the proposed auction protocol, we can compose adversary $\mathcal{A}$ that breaks the semantic security of the BGN encryption with the mix-and-match oracle by using $\mathcal{B}$.

Definition 1
Let Π = (KeyGen, Encrypt, Decrypt) be a BGN encryption scheme, and let $A^{O_1} = (A_1^{O_1}, A_2^{O_1})$, be a probabilistic polynomial-time algorithm, that can use the mix-and-match oracle O_1.

$$\text{BGN-}Adv(\tau) = \Pr[EXPT_{A,\Pi}(\tau) \Rightarrow 1] - 1/2$$

where, $EXPT_{A,\Pi}$ is a semantic security game of the BGN encryption scheme with the mix-and-match oracle shown in Fig. 1.

We then define an adversary $\mathcal{B}$ for an auction protocol and an advantage for $\mathcal{B}$.

Definition 2
Let Π = (KeyGen, Encrypt, Decrypt) be a BGN encryption scheme, and let B be two probabilistic polynomial-time algorithm B_1 and B_2.

$$
\begin{array}{c}
(PK, SK) \leftarrow KeyGen \\
(b_1, b_2, ..., b_{m-1}, b_{m0}, b_{m1}, s) \leftarrow B_1(PK) \\
b \leftarrow \{0,1\} \\
c \leftarrow (Encrypt(PK, b_1), Encrypt(PK, b_2), ..., Encrypt(PK, b_{m-1}), Encrypt(PK, b_{mb})) \\
execute\ auction\ protocols\ using\ c\ as\ players'\ bids \\
and\ x\ is\ transcript\ of\ the\ auction\ protocol. \\
b' \leftarrow B_2(c, s, x) \\
return\ 1\ \text{iff}\ b = b'
\end{array}
$$

Fig. 2. $EXPT_{B,\Pi}$

$$\text{Auction-}Adv(\tau) = \Pr[EXPT_{B,\Pi} = 1] - 1/2$$

where $EXPT_{B,\Pi}$ is a semantic security game of the privacy of the auction protocol shown in Fig. 2.

First of all, B_1 generates k-bit integers, $b_1, b_2, ..., b_{m-1}$ as plaintexts of bidding prices for player 1 to $m-1$, and two challenge k-bit integers as b_{m_0}, b_{m_1} where b_{m_0} and b_{m_1} are the same bits except for i-th bit m_0^i and m_1^i. We assume b_{m_0} and b_{m_1} are not the $M+1st$ highest price. Then the auction is executed with $(Encrypt(PK, b_1), Encrypt(PK, b_2), ..., Encrypt(PK, b_{m-1}), Encrypt(PK, b_{m_b}))$ as the players' encrypted bidding prices where $b \xleftarrow{r} \{0,1\}$. After the auction, B_2 outputs b' $\in \{0,1\}$ as a guess for b. $\mathcal{B}$ wins if $b = b'$.

Theorem 1. *The privacy of the auction protocols is secure under the assumption that the BGN encryption is semantically secure with a mix-and-match oracle.*

We show if there is adversary $\mathcal{B}$ that breaks the security of the proposed auction protocol, we can compose adversary $\mathcal{A}$ that breaks the semantic security of the BGN encryption with the mix-and-match oracle. $\mathcal{A}$ receives two challenge k-bit integers as b_{m_0} and b_{m_1} from $\mathcal{B}$ and then $\mathcal{A}$ uses m_0^i and m_1^i as challenge bits for the challenger of the BGN encryption. Then $\mathcal{A}$ receives $Encrypt(PK, m_b^i)$ and executes a secure auction protocol with the mix-and-match oracle. When calculation of plain equality test or mix-and-match is needed such as checking whether h_j is 0 and updating $\widetilde{W}$, $\mathcal{A}$ uses mix-and-match oracle to transfer encrypted value over E_{G1} to E_G. b_{m_0} and b_{m_1} are not the winning bidding prices and $\mathcal{A}$ knows all the input values, $b_1, b_2, ..., b_{m-1}$ except the i-th bit of b_{m_b}. So, $\mathcal{A}$ with mix-and-match oracle can simulate an auction for the adversary of auction $\mathcal{B}$. Through the auction, $\mathcal{B}$ observes the calculation of the encrypted values and the results of the auction. After the auction, $\mathcal{B}$ outputs b', which is the guess for b. $\mathcal{A}$ outputs b', which is the same guess with $\mathcal{B}$'s output for b_{m_b}. If $\mathcal{B}$ can break the privacy of the bidding prices in the proposed auction protocol with advantage $\epsilon(\tau)$, $\mathcal{A}$ can break the semantic security of the BGN encryption with the same advantage.

2. **Correctness**
   For correct players' inputs, the protocol outputs the correct winner and price. From Theorem 1 introduced in Section 1.4, the bit-slice auction protocol obviously satisfies the correctness.
3. **Verification of the evaluation**
   To verify whether the protocol works, players need to validate whether the AM decrypts the evaluations of the circuit on ciphertexts through the protocol. We use the verification protocols introduced above so that each player can verify whether the protocol is computed correctly.

4 Comparison of Auction Protocols

The protocol proposed in [1] based on homomorphic encryption. Each player encrypts his bidding price k as an integer. When m players and the bidding prices are in the range of $[1, p]$, AM calculates multiplications of ciphertexts $2mp$ times. Mixing and decrypting is used for PET (plain equality test) in the opening phase to check whether the number of i-th bid is more than $M + 1$ or not for each price i in $[1, p]$ using binary search. Binary search for p needs log p comparisons and one comparison needs M+1 PETs for each bid to check whether it is more than $M + 1$. And m decryptions are used to decide the winner of the auction. In our protocol each player's bidding price is represented as a binary expression. We use PET Mp times when AM calculates $\tilde{s}_i^j$ from player j's i-th bid for all i and j. We also use PET when AM detects whether $b_{M+1st}^{(i)}$ is more than M or not. And log$p$ decryptions are used to open the winning price and m decryptions are used to to open the winners of auction.

5 Conclusion

We introduced new efficient secure $M + 1st$ price auction protocols based on the mix-and-match protocol and the BGN encryption. As a topic of future work, we will try to compose a secure auction protocol without using the mix-and-match protocol.

Table 5. The Comparision of computational complexity

	[AS02]	Proposed
Bidding(per one bidder)	p encryptions	log$p$ encyrptions
Running auction (Calculation over group)	$2mp$ multiplications	$m$log$p$ multiplications $m$log$p$ pairing
Running auction(Mix and Match)	log p times on $M + 1$ inputs	log p times on $M + 1$ inputs
Decrypting to decide the winners	m	m
Decrypting to decide the winning price	log$p$	log$p$

References

1. Abe, M., Suzuki, K.: M + 1st price auction using homomorphic encryption. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 115–124. Springer, Heidelberg (2002)
2. Boneh, D., Franklin, M.K.: Efficient Generation of Shared RSA keys. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 425–439. Springer, Heidelberg (1997)
3. Boneh, D., Goh, E., Nissim, K.: Evaluating 2-DNF Formulas on Ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer, Heidelberg (2005)
4. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM, 84–88 (1981)
5. Franklin, M.K., Reiter, M.K.: The design and implementation of a secure auction service. IEEE Transactions on Software Engineering 22(5), 302–312 (1995)
6. Jakobsson, M., Juels, A.: Mix and Match: Secure Function Evaluation via Ciphertexts. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 162–177. Springer, Heidelberg (2000)
7. Juels, A., Szydlo, M.: A Two-Server Sealed-Bid Auction Protocol. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 72–86. Springer, Heidelberg (2003)
8. Kurosawa, K., Ogata, W.: Bit-Slice Auction Circuit. In: Gollmann, D., Karjoth, G., Waidner, M. (eds.) ESORICS 2002. LNCS, vol. 2502, pp. 24–38. Springer, Heidelberg (2002)
9. Lipmaa, H., Asokan, N., Niemi, V.: Secure Vickrey auctions without threshold trust. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 87–101. Springer, Heidelberg (2003)
10. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Proceedings of the 1st ACM Conference on Electronic Commerce (ACM-EC), pp. 129–139. ACM press, New York (1999)
11. Mitsunaga, T., Manabe, Y., Okamoto, T.: Efficient Secure Auction Protocols Based on the Boneh-Goh-Nissim Encryption. In: Echizen, I., Kunihiro, N., Sasaki, R. (eds.) IWSEC 2010. LNCS, vol. 6434, pp. 149–163. Springer, Heidelberg (2010)
12. Okamoto, T., Uchiyama, S.: A new public-key cryptosystem as secure as factoring. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 308–318. Springer, Heidelberg (1998)
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. Park, C.-s., Itoh, K., Kurosawa, K.: Efficient anonymous channel and all/Nothing election scheme. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 248–259. Springer, Heidelberg (1994)

Cryptographic Pairings Based on Elliptic Nets

Naoki Ogura[1], Naoki Kanayama[2],
Shigenori Uchiyama[1], and Eiji Okamoto[2]

[1] Graduate School of Science and Engineering,
Tokyo Metropolitan University,
1-1, Minami-Ohsawa, Hachioji, Tokyo, 192-0397, Japan
ogura-naoki@ed.tmu.ac.jp, uchiyama-shigenori@tmu.ac.jp
[2] Faculty of Systems and Information Engineering,
University of Tsukuba,
1-1-1, Ten-nohdai, Tsukuba-shi, Ibaraki-ken, 305-8573 Japan
{kanayama,okamoto}@risk.tsukuba.ac.jp

Abstract. In 2007, Stange proposed a novel method for computing the Tate pairing on an elliptic curve over a finite field. This method is based on elliptic nets, which are maps from $\mathbb{Z}^n$ to a ring and satisfy a certain recurrence relation. In the present paper, we explicitly give formulae based on elliptic nets for computing the following variants of the Tate pairing: the Ate, Ate$_i$, R-Ate, and optimal pairings. We also discuss their efficiency by using some experimental results.

Keywords: Tate pairing, Ate pairing, R-Ate pairing, Optimal pairing, elliptic net, normalization.

1 Introduction

Recently, pairing-based cryptography have been one of the most attractive research topics in public-key cryptography since the proposals of some useful cryptographic schemes, such as the identity-based key agreement, the tripartite Diffie–Hellman key exchange, and the identity-based encryption schemes [3], [9], [15]. With respect to the efficient implementation of pairing-based cryptographic schemes, the computation of pairings, such as the Weil and Tate pairings, is the bottleneck. Currently, the most suitable pairing for the efficient implementation of pairing-based cryptographic schemes is the Tate pairing. Therefore, many algorithms for the efficient computation of the Tate pairing and some of its variants have been proposed, including the η_T [1], Duursma–Lee [6], Ate [8], Ate$_i$ [20], R-Ate [10], and optimal [21] pairings.

A standard algorithm for computing pairings is Miller's algorithm [11], [12]. A generic implementation of Miller's algorithm uses a classical double-and-add line-and-tangent method. Therefore, the time required using Miller's algorithm is linear with respect to the size of some input parameter r, as well as depending on the Hamming weight of r. Most improvements of pairing computation attempt to shorten the number of iterations of a loop in the algorithm, the so-called

T. Iwata and M. Nishigaki (Eds.): IWSEC 2011, LNCS 7038, pp. 65–78, 2011.

Miller loop. In fact, the Ate, Ate$_i$, R-Ate, and optimal pairings are truncated loop variants of the Tate pairing.

In 2007, Stange [18] defined elliptic nets and proposed an alternative method for the Tate pairing computation based on elliptic nets. Elliptic nets are a generalization of elliptic divisibility sequences, which are certain non-linear recurrence sequences related to elliptic functions. In 1948, Ward [22] first studied the arithmetic properties of elliptic divisibility sequences. As in the case of Miller's algorithm, a generic implementation of elliptic net algorithms proposed by Stange uses the double-and-add method, and so, as in the case of Miller's algorithm, the time required using the algorithm is linear with respect to the size of r. Both Miller's and elliptic net algorithms include two internal steps, referred to as Double and DoubleAdd [18]. In Miller's algorithm, the cost of DoubleAdd is about twice that of Double. In contrast, in the elliptic net algorithm, these two steps require almost the same amount of time. In particular, the running time is independent of the Hamming weight of r.

Because the efficiency of the algorithm is comparable to that of Miller's algorithm, by using further improvements and optimizations, we expect the elliptic net algorithm to be an efficient alternative to Miller's algorithm. Therefore, from both theoretical and practical points of view, it is important to investigate explicit formulae for computing some variants of the Tate pairing, based on elliptic nets.

In the present paper, we explicitly give formulae based on elliptic nets for computing the following variants of the Tate pairing: the Ate, Ate$_i$, R-Ate, and optimal pairings.

These pairings are defined as "point-evaluation" pairings, although the Tate pairing is originally "divisor-evaluation" pairing. These point-evaluation pairings are defined using normalized functions (see Section 2). Hence, we need to formulate a normalization of elliptic nets. In the present paper, we give a normalization of elliptic nets and then the formulae of the above-listed point-evaluation pairings. We also discuss their efficiency by using some experimental results.

The remainder of this paper is organized as follows. Section 2 gives a brief mathematical description of pairings and elliptic nets. Section 3 contains our main results concerning pairings described by elliptic nets. In Section 4, we will show our experimental results. We draw conclusions in Section 5.

2 Mathematical Preliminaries

2.1 Pairings

Let E be an elliptic curve over a finite field $\mathbb{F}_q$ with q elements. The set of $\mathbb{F}_q$-rational points of E is denoted as $E(\mathbb{F}_q)$. Let $E(\mathbb{F}_q)[r]$ denote the subgroup of r-torsion points in $E(\mathbb{F}_q)$. We write O for the point at infinity on E. Consider a large prime r such that $r \mid \#E(\mathbb{F}_q)$ and denote the embedding degree by k, which is the smallest positive integer such that r divides $q^k - 1$. Let π_q be the Frobenius endomorphism $\pi_q : E \to E : (x, y) \mapsto (x^q, y^q)$. We denote the trace of

Frobenius by t, i.e., $\#E(\mathbb{F}_q) = q + 1 - t$. Finally, let $\mu_r (\subset \mathbb{F}_{q^k}^{\times})$ be the group of r-th roots of unity.

Weil Pairing. The Weil pairing $e_r(\cdot, \cdot)$ is defined by

$$\begin{aligned} e_r(\cdot,\cdot) : E(\mathbb{F}_{q^k})[r] \times E(\mathbb{F}_{q^k})[r] &\to \mu_r, \\ (P, Q) &\mapsto e_r(P, Q) := f_{r,P}(D_Q)/f_{r,Q}(D_P), \end{aligned}$$

where D_P is a divisor equivalent to $(P)-(O)$ and $f_{s,P}$ is a rational function on E such that $\mathrm{div}(f_{s,P}) = rD_P$. Similarly, $\mathrm{div}(f_{s,Q}) = rD_Q$, where D_Q is equivalent to $(Q) - (O)$. We assume that D_P and D_Q are chosen with disjoint supports.

Note that the Weil pairing does not depend on the choice of D_P and D_Q. Furthermore, the Weil pairing is bilinear and non-degenerate.

Tate Pairing. Let $P \in E(\mathbb{F}_{q^k})[r]$ and $Q \in E(\mathbb{F}_{q^k})$. Choose a point $R \in E(\mathbb{F}_{q^k})$ such that the support of $\mathrm{div}(f_{r,P}) = r(P) - r(O)$ and $D_Q := (Q + R) - (R)$ are disjoint. Then, the Tate pairing is defined by

$$\begin{aligned} \langle \cdot, \cdot \rangle_r : E(\mathbb{F}_{q^k})[r] \times E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k}) &\to \mathbb{F}_{q^k}^{\times}/(\mathbb{F}_{q^k}^{\times})^r, \\ (P, Q) &\mapsto \langle P, Q \rangle_r := f_{r,P}(D_Q) \mod (\mathbb{F}_{q^k}^{\times})^r . \end{aligned}$$

It has been shown that $\langle P, Q \rangle_r$ is bilinear and non-degenerate.

For cryptography applications, it is convenient to define pairings whose outputs are unique values rather than equivalence classes. Thus, herein, we consider the reduced Tate pairing defined by

$$\begin{aligned} \tau_r : E(\mathbb{F}_{q^k})[r] \times E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k}) &\to \mu_r, \\ \tau_r(P, Q) = \langle P, Q \rangle_r^{(q^k-1)/r}&. \end{aligned}$$

We call the operation $z \mapsto z^{(q^k-1)/r}$ final exponentiation.

The Weil Tate pairings satisfy that

$$e_r(P, Q) = \frac{\langle P, Q \rangle_r}{\langle Q, P \rangle_r} \quad \text{up to } r\text{-th powers.} \tag{1}$$

Thus, if the cost of final exponentiation is sufficiently small, the cost of computing the Tate pairing is almost half of that of computing the Weil pairing. Because of this, the Tate pairing is widely used in cryptography and there are numerous improved versions, such as the Ate pairing.

As mentioned in Section 1, a classical and currently standard algorithm for computing pairings is Miller's algorithm [11], [12]. One of the efficiency benchmarks of pairing computation is based on the Miller loop. The length of the Miller loop is $\log_2(r)$ in the case of the Tate pairing $\langle \cdot, \cdot \rangle_r$. Most improvements of pairing computation attempt to shorten the Miller loop.

Barreto et al. [2] pointed out that $\tau_r(P,Q)$ can be computed by $\tau_r(P,Q) = f_{r,P}(Q)^{(q^k-1)/r}$ if $P \in E(\mathbb{F}_q)[r]$ and $k > 1$.

For cryptographic applications, it is usually assumed that points P and Q are respectively elements in the following groups:

$$\mathbb{G}_1 = E(\mathbb{F}_q)[r] = E(\mathbb{F}_{q^k})[r] \cap \mathrm{Ker}(\pi_q - 1),$$
$$\mathbb{G}_2 = E(\mathbb{F}_{q^k})[r] \cap \mathrm{Ker}(\pi_q - q)$$

Hereafter, we assume that $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$.

We give a brief review of the following variants of the Tate pairing: the Ate [8], Ate$_i$ [20], R-Ate [10], and optimal [21] pairings. These pairings are defined on $\mathbb{G}_2 \times \mathbb{G}_1$ and $\mathbb{G}_1 \times \mathbb{G}_2$. In the present paper, we consider the case of $\mathbb{G}_2 \times \mathbb{G}_1$. See the appropriate papers cited above for the case of $\mathbb{G}_1 \times \mathbb{G}_2$. We use normalized functions to define the above pairings on $\mathbb{G}_2 \times \mathbb{G}_1$; therefore, we will first define this normalization as follows.

Normalization of Rational Functions. For $s \in \mathbb{Z}$, we define $f_{s,Q}$ as the rational function satisfying the equation $\mathrm{div}(f_{s,Q}) = s(Q) - (sQ) - (s-1)(O)$. This function $f_{s,Q}$ is determined uniquely up to multiplication by a constant. Uniqueness is obtained by normalization. We will denote the normalized form of $f_{s,R}$ by $f_{s,R}^{\mathrm{norm}}$ and refer to the latter as the normalized function.

Let u_O be a uniformizer of E on O. We may choose as this uniformizer $u_O = -\frac{x}{y}$. Then the normalized function $f_{s,R}^{\mathrm{norm}}$ is defined by

$$f_{s,R}^{\mathrm{norm}} = f_{s,R}/c, \quad \text{where} \quad c = (u_O^{s-1} f_{s,R})(O). \tag{2}$$

From now on, we may assume that all rational functions on elliptic curves are normalized.

Ate Pairing. The Ate pairing, proposed by Hess et al. [8], is a generalization of the η_T pairing [1]. The Ate pairing can be applied to not only supersingular but also ordinary elliptic curves.

Let $T = t - 1$. We choose integers N and L such that $N = \gcd(T^k - 1, q^k - 1)$ and $T^k - 1 = LN$. We assume that r^2 does not divide $q^k - 1$. Then the Ate pairing is defined by $f_{T,Q}(P)(Q \in \mathbb{G}_2$ and $P \in \mathbb{G}_1)$. We denote by $\alpha(Q,P)$ the reduced Ate pairing: $\alpha(Q,P) := f_{T,Q}(P)^{(q^k-1)/r}$. The length of the Miller loop for computing the Ate pairing $f_{T,Q}(P)$ is $\log_2 |T|$.

Ate$_i$ Pairing. The Ate$_i$ pairing was proposed by Zhao et al. [20]. Let $T_i := q^i \pmod r$ for $i = 1, 2, \cdots, k-1$. For each i, we define the following quantities similarly to those for the Ate pairing: a_i is the smallest positive integer such that $T_i^{a_i} \equiv 1 \pmod r$, $N_i := \gcd(T_i^{a_i} - 1, q^k - 1)$, and L_i is the positive integer such that $T_i^{a_i} - 1 = L_i N_i$.

The Ate$_i$ pairing on $\mathbb{G}_2 \times \mathbb{G}_1$ is defined by $f_{T_i,Q}(P)(Q \in \mathbb{G}_2$ and $P \in \mathbb{G}_1)$. Analogous to the case for Ate pairing, we denote by $\alpha_i(Q,P)$ the reduced Ate$_i$

pairing: $\alpha_i(Q,P) := f_{T_i,Q}(P)^{(q^k-1)/r}$. The length of the Miller loop for computing $f_{T_i,Q}(P)$ is $\log_2(T_i)$.

If $T_n := \min\{T_i : i = 1, 2, \cdots, k-1, 0 \le T_i \le r-1\}$, then $f_{T_n,Q}(P)$ can be computed faster than the Ate pairing $f_{T,Q}(P)$.

R-Ate Pairing. The R-Ate pairing was proposed by Lee et al. [10] Let A, B, a, b be integers such that $A = aB + b$. We define the R-Ate pairing to be

$$R_{A,B}(Q,P) := f_{a,[B]Q}(P) \cdot f_{b,Q}(P) \cdot G_{[aB]Q,[b]Q}(P),$$

where G_{P_1,P_2} is a rational function on E such that $\mathrm{div}(G_{P_1,P_2}) = (P_1) + (P_2) - (P_3) - (O)$ $(P_3 = P_1 + P_2)$.

Lee et al. showed that $R_{A,B}(Q,P)$ is bilinear and non-degenerate under some conditions (see Theorem III.2 of [10]). Furthermore, they also gave the following examples in which $R_{A,B}(Q,P)$ is bilinear and non-degenerate: $(A,B) = (q^i, r)$, $(A,B) = (q, T_1)$ where $q > T_1$, $(A,B) = (T_i, T_j)$, and $(A,B) = (r, T_j)$. See Corollary III.3. in [10].

Optimal Pairing. Optimal pairing was proposed by Vercauteren [21]. Optimal pairing can be computed in $\log_2 r/\phi(k) + \epsilon(k)$ Miller loop iterations ($\phi(k)$ is the Euler function of k and $\epsilon(k) \le \log_2 k$).

Theorem 1 ([21] Theorem 1). *Let λ be an integer such that $r|\lambda$ and $r^2 \nmid \lambda$. We express λ as $\lambda = \sum_{i=0}^{l} c_i q^i$. Then*

$$a_{[c_0,c_1,\cdots,c_l]} : \mathbb{G}_2 \times \mathbb{G}_1 \to \mu_r$$
$$(Q,P) \mapsto \Big(\prod_{i=0}^{l} f_{c_i,Q}^{q^i}(P) \cdot \prod_{i=0}^{l-1} \frac{l_{[s_{i+1}]Q,[c_i q^i]Q}(P)}{v_{[s_i]Q}(P)} \Big)^{\frac{q^k-1}{r}}$$

(where $s_i = \sum_{j=i}^{l} c_j q^j$) defines a bilinear map. Furthermore, if

$$\frac{\lambda}{r} k q^{k-1} \not\equiv \frac{q^k - 1}{r} \sum_{i=0}^{l} i c_i q^{i-1} \pmod{r},$$

$a_{[c_0,c_1,\cdots,c_l]}(Q,P)$ is non-degenerate.

Note that we may consider $l = \phi(k) - 1$ because $r \mid \Phi_k(q)$, where $\Phi_k(X)$ is the k-th cyclotomic polynomial. The pairing $a_{[c_0,c_1,\cdots,c_l]}(Q,P)$ is called the optimal pairing because it can be computed very efficiently if $c_0, c_1, \cdots, c_l$ can be chosen very small.

2.2 Elliptic Nets

In 2007, Stange [18] defined elliptic nets as maps from $\mathbb{Z}^n$ to a ring and they satisfy a certain recurrence relation associated with elliptic curves. In general, an elliptic net W is a map from a finitely generated abelian group $\mathcal{A}$ to an integral domain $\mathcal{R}$ such that

$$\begin{aligned} &W(p+q+s)W(p-q)W(r+s)W(r) \\ &+ W(q+r+s)W(q-r)W(p+s)W(p) \\ &+ W(r+p+s)W(r-p)W(q+s)W(q) = 0 \end{aligned}$$

for $p, q, r, s \in \mathcal{A}$. Elliptic divisibility sequences arise from an elliptic curve defined over the rational numbers and a rational point of that curve. These sequences are strongly related to elliptic functions and the division polynomials of an elliptic curve. For cryptographic applications, the division polynomials of an elliptic curve are the main tools of Schoof's algorithm [16]. As we will see later, the division polynomials of an elliptic curve also play an important role in the computation of elliptic net-based pairings.

Stange introduced the concept of elliptic nets associated with elliptic curves and described Tate pairing by using elliptic nets. In this section, we briefly review elliptic nets. See [18] for detail.

First, we consider a function, denoted by Ψ, associated with elliptic curves over $\mathbb{C}$ by using an elliptic σ-function. We define an elliptic net W (in $\mathbb{C}$) using Ψ. Next, we construct a function associated with Ψ, denoted by Ω, that is defined in finite fields by applying a reduction theorem (see Theorem 3 in [18]). Thus, we are able to consider W in finite fields and construct the Tate pairing in finite fields.

To describe the Tate pairing $f_{r,P}(D_Q)$ by using elliptic nets, Stange showed a formula for a function $f_{r,P}$ with $\mathrm{div}(f_P) = r(P) - r(O)$ as $f_{r,P} = \dfrac{\Omega_{1,0,0,}(-S,P,Q)}{\Omega_{1,r,0}(-S,P,Q)}$, where $\Omega_{1,v_2,v_3}(-S,P,Q)(v_i \in \mathbb{Z})$ is a function in S and the divisor of $\Omega_{1,v_2,v_3}(-S,P,Q)$ on a variable S is $([v_2]P + [v_3]Q) - v_2(P) - v_3(Q) - (1 - v_2 - v_3)(O)$. Then a formula for $f_{r,P}(D_Q)$, where D_Q is a divisor equivalent to $(-S) - (-S - Q)$, as a function in variable S is computed. The following result is obtained by setting $S = P$ the formula of $f_{r,P}(D_Q)$.

Theorem 2 ([18]). *Let E be an elliptic curve over a finite field K. For $P \in E(K)[r]$, $Q \in E(K)$,*

$$f_{r,P}(D_Q) = \frac{W_{P,Q}(r+1,1)W_{P,Q}(1,0)}{W_{P,Q}(r+1,0)W_{P,Q}(1,1)}, \tag{3}$$

where $W_{P,Q}(r+1,i) = \Omega_{1,r,i}(-S,P,Q)|_{S=P}$.

Remark 1. *By using the above theorem and the equation (1), we can easily obtain the Weil pairing formula using elliptic nets as the following. For $P, Q \in E(\mathbb{F}_{q^k})[r]$,*

$$e_r(P,Q) = \frac{W_{P,Q}(r+1,1)W_{Q,P}(r+1,0)}{W_{P,Q}(r+1,0)W_{Q,P}(r+1,1)} \quad \textit{up to } r\textit{-th powers.}$$

Here, we assume that an elliptic curve E has a Weierstrass equation of the form $Y^2 = X^3 + AX + B$. Let $\psi_n(x,y)$ denote the n-th division polynomial of an elliptic curve. For simplicity, we write $W_{P,Q}(i,j) = W(i,j)$. Initial values of

elliptic nets $W(i,0)$ and $W(i,1)$ are obtained by the following definition (see [18]): if $P=(x_1,y_1)$ and $Q=(x_2,y_2)$, then

$$\begin{aligned}
&W(1,0)=1,\\
&W(2,0)=2y_1,\\
&W(3,0)=3x_1^3+6Ax_1^2+12Bx_1-A^2,\\
&W(4,0)=4y_1(x_1^6+5Ax_1^4+20Bx_1^3-5A^2x_1^2-4ABx_1-8B^2-A^3),\\
&W(0,1)=W(1,1)=1,\\
&W(2,1)=2x_1+x_2-(\frac{y_2-y_1}{x_2-x_1})^2,\\
&W(-1,1)=x_1-x_2,\\
&W(2,-1)=(y_1+y_2)^2-(2x_1+x_2)(x_1-x_2)^2.
\end{aligned}$$

Elliptic nets $W(i,0)$ and $W(j,1)$ can be computed by the following recursive formulae.

Proposition 1 ([18])

$$\begin{aligned}
W(2i-1,0)&=W(i+1,0)W(i-1,0)^3-W(i-2,0)W(i,0)^3,\\
W(2i,0)&=\frac{W(i,0)W(i+2,0)W(i-1,0)^2-W(i,0)W(i-2,0)W(i+1,0)^2}{W(2,0)},\\
W(2i-1,1)&=\frac{W(i+1,1)W(i-1,1)W(i-1,0)^2-W(i,0)W(i-2,0)W(i,1)^2}{W(1,1)},\\
W(2i,1)&=W(i-1,1)W(i+1,1)W(k,0)^2-W(i-1,0)W(i+1,0)W(i,1)^2,\\
W(2i+1,1)&=\frac{W(i-1,1)W(i+1,1)W(i+1,0)^2-W(i,0)W(i+2,0)W(i,1)^2}{W(-1,1)},\\
W(2i+2,1)&=\frac{W(i+1,0)W(i+3,0)W(i,1)^2-W(i-1,1)W(i+1,1)W(i+2,0)^2}{W(2,-1)}.
\end{aligned}$$

Note that $W(i,0)=W_{P,Q}(i,0)$ is equal to $\psi_i(x_1,y_1)$ because $W_{P,Q}(i,0)=\psi_i(x_1,y_1)$ for $i=1,2,3,4$ and the recursive formulae for computing $W(2i-1,0)$ and $W(2i-1,0)$ are the same as the recursive formulae for division polynomials. Therefore, if E is defined over K and $P\in E(K)$, $W_{P,Q}(i,0)\in K$ for all i and they are killed by final exponentiation.

See [18] for algorithms for computing elliptic nets.

3 The Main Results

In this section, we describe variants of the Tate pairing, the Ate, Ate$_i$, R-Ate, and optimal pairings by using elliptic nets.

As seen in Section 2, these pairings are point-evaluation pairings and are defined by using normalized functions. Therefore, we need to formulate a normalization of elliptic nets in order to describe the formulae of the above-listed point-evaluation pairings.

3.1 Normalization of Elliptic Nets

First, we present the following lemma, which can be proved by using a straight forward calculation.

Lemma 1. *Let* $\wp(z;\Lambda) := \frac{1}{z^2} + \sum_{\omega\in\Lambda\setminus\{0\}}(\frac{1}{(z-\omega)^2} - \frac{1}{\omega^2})$ *be the Weierstrass* $\wp$ *function and* $\sigma(z;\Lambda) := z\prod_{\omega\in\Lambda\setminus\{0\}}(1-\frac{z}{\omega})e^{z/\omega+(1/2)(z/\omega)^2}$ *be the Weierstrass* σ *function on* $\mathbb{C}$*; then*

$$\left(\frac{\wp(z;\Lambda)}{\wp'(z;\Lambda)\sigma(z;\Lambda)}\right)(0) = -\frac{1}{2}\ .$$

Next, we show the following equation corresponding to equation (2) in Section 2.1.

Proposition 2. *Let* $\Lambda \in \mathbb{C}$ *be a lattice corresponding to the elliptic curve* E*. Fix* $w \in \mathbb{C}\ \setminus\{0\}$*. For* $s \in \mathbb{Z}$*,*

$$(-\wp(z;\Lambda)/\wp'(z;\Lambda))^{1-s}\Psi_{s,1}(w,\ z)|_{z=0} = 2^{s-1}\Psi_{s,0}(w,\ z)\ .$$

Proof. The proposition follows from Lemma 1 and the following fact:

$$\Psi_{s,1}(w,\ z) = \frac{\sigma(sw+z)}{\sigma(w)^{s^2-s}\sigma(w+z)^s\sigma(z)^{1-s}}\ .$$

The uniformizer $u_O = -\frac{x}{y}$ corresponds to $-\frac{\wp(z)}{\wp'(z)}$. Thus, we have the following proposition, which gives the normalization of elliptic nets.

Proposition 3. $\tilde{W}_{P,Q}(s,\ 1)$ *denotes the normalization (by* $-\frac{x}{y}$*) for the elliptic net* $W_{P,Q}(s,\ 1)$*. For* $s \in \mathbb{Z}$*, assume* $[s]P \neq O$*. Then*

$$\tilde{W}_{P,Q}(s,1) = \frac{W_{P,Q}(s,1)}{2^{s-1}W_{P,Q}(s,0)}\ .$$

For practical uses of pairings, we can assume $k > 1$. In this case, $2^{(q^k-1)/r} = 1$, and so we have

$$\tilde{W}_{P,Q}(s,1)^{\frac{q^k-1}{r}} = \left(\frac{W_{P,Q}(s,1)}{W_{P,Q}(s,0)}\right)^{\frac{q^k-1}{r}}.$$

3.2 Elliptic Net-Based Pairings

We explain the key lemma which connects various pairings with elliptic nets. We use $\tilde{W}_{P,S}(s,\ 1)$ to denote the normalization for $W_{P,S}(s,\ 1)$, where $W_{P,S}(s,\ 1)$ is a function in S and P is a fixed point on E.

Lemma 2. *For* $s \in \mathbb{Z}$*, we assume that the point* Q *is neither a zero nor a pole of* $f_{s,P}$*. Then*

$$f_{s,P}(Q) = \tilde{W}_{-P,Q}(s,1)^{-1}.$$

Proof. Let $W_{-P,S}(s,\ 1) = \Omega_{s,1}(-P,\ S)$ be a rational function in variable S. Similar to in [18], the divisor of $W_{-P,S}(s,\ 1)$ in S is

$$\begin{aligned}\mathrm{div}_S(\Omega_{s,1}(-P,\ S)) &= ([-s](-P)) - s(P) - (1-s)(O) \\ &= -\{s(P) - ([s]P) - (s-1)(O)\} \\ &= -\mathrm{div}_S(f_{s,P})\ .\end{aligned}$$

Hence, $f_{s,P} = \tilde{W}_{-P,S}(s,1)^{-1}$ from the uniqueness of the normalized function. Finally, we obtain the desired result by taking $S = Q$.

The following theorem derives formulae for elliptic net-based pairings.

Theorem 3. *If the following function on P and Q,*

$$A(P,\ Q) = \prod_{i=0}^{l_1} f_{t_i,P}^{\alpha_i}(Q) \prod_{j=0}^{l_2} G_{[u_j]P,[v_j]P}^{\beta_j}(Q),$$

is bilinear, then

$$A(P,\ Q) = \prod_{i=0}^{l_1} \tilde{W}_{P,Q}^{\alpha_i}(t_i,\ 1) \prod_{j=0}^{l_2} G_{[-u_j]P,[-v_j]P}^{-\beta_j}(Q)\ .$$

Proof. Using Lemma 2 and the bilinearity of $A(P,\ Q)$,

$$\begin{aligned}A(P,\ Q) &= A(-P,\ Q)^{-1} \\ &= \prod_{i=0}^{l_1} f_{t_i,-P}^{-\alpha_i}(Q) \prod_{j=0}^{l_2} G_{[-u_j]P,[-v_j]P}^{-\beta_j}(Q) \\ &= \prod_{i=0}^{l_1} \tilde{W}_{P,Q}^{\alpha_i}(t_i,\ 1) \prod_{j=0}^{l_2} G_{[-u_j]P,[-v_j]P}^{-\beta_j}(Q)\ .\end{aligned}$$

□

We consider the case of the optimal pairing. In this case, we need to compute scalar multiplications $[c_i q^i]Q (i = 0, 1, \cdots, l)$ using elliptic nets.

Note that $Q := (x_Q, y_Q)$ satisfies $[c_i q^i]Q = [q^i]([c_i]Q) = \pi_q^i([c_i]Q)$ because $Q \in E(\mathbb{F}_{q^k})[r] \cap \mathrm{Ker}(\pi_q - q)$.

Furthermore, as seen in Section 2 of [18], $W_{Q,P}(n,0) = \psi_n(x_Q, y_Q)$. Thus, we are able to express $[n]Q$ in terms of elliptic nets by using the following famous multiplication formula:

$$[n](x,y) = \left(x - \frac{\psi_{n-1}\psi_{n+1}}{\psi_n^2}(x,y), \frac{\psi_{n-1}^2\psi_{n+2} - \psi_{n+1}^2\psi_{n-2}}{4y\psi_n^3}(x,y)\right).$$

Hence, we obtain $[c_i q^i]Q = \pi_q^i([c_i]Q) = (x_{[c_i]Q}^{q^i}, y_{[c_i]Q}^{q^i})$, where

$$x_{[c_i]Q}^{q^i} = \left(x_Q - \frac{W_{Q,P}(c_i-1,0)W_{Q,P}(c_i+1,0)}{W_{Q,P}(c_i,0)^2}\right)^{q^i},$$

$$y_{[c_i]Q}^{q^i} = \left(\frac{W_{Q,P}(c_i-1,0)^2 W_{Q,P}(c_i+2,0) - W_{Q,P}(c_i+1,0)^2 W_{Q,P}(c_i-2,0)}{2W_{Q,P}(2,0)W_{Q,P}(c_i,0)^3}\right)^{q^i}$$

To summarize, we show formulae of cryptographic pairings:

Theorem 4. *Let E be an elliptic curve over a finite field $\mathbb{F}_q$ and $\pi_q : (x, y) \mapsto (x^q, y^q)$ the q-Frobenius endomorphism on E. We assume that the embedding degree $k > 1$. Let r be a large prime number with $r|\#E(\mathbb{F}_q)$ and $(r, q) = 1$, and also $T \equiv q \pmod r$ and $T_i \equiv q^i \pmod r$. Let $\lambda = \sum_{i=0}^{l} c_i q^i$ be such that $r|\lambda$ and $r^2 \nmid \lambda$. We define $s_i = \sum_{j=i}^{l} c_j q^j$.*

Then, we have the following.

Tate Pairing: *For $P \in E(\mathbb{F}_{q^k})[r]$ and $Q \in E(\mathbb{F}_{q^k})$,*

$$\tau_r(P, Q) = f_{r,P}(Q)^{\frac{q^k-1}{r}} = \tilde{W}_{P,Q}(r + 1, 1)^{\frac{q^k-1}{r}} .$$

Variants of the Tate Pairing: *For $P \in \mathbb{G}_1 = E(\mathbb{F}_{q^k})[r] \cap \mathrm{Ker}(\pi_q - 1)$ and $Q \in \mathbb{G}_2 = E(\mathbb{F}_{q^k})[r] \cap \mathrm{Ker}(\pi_q - q)$,*

- *Ate*

$$\alpha(Q, P) = f_{T,Q}(P)^{\frac{q^k-1}{r}} = \tilde{W}_{Q,P}(T, 1)^{\frac{q^k-1}{r}} ;$$

- *Ate$_i$*

$$\alpha_i(Q, P) = f_{T_i,Q}(P)^{\frac{q^k-1}{r}} = \tilde{W}_{Q,P}(T_i, 1)^{\frac{q^k-1}{r}} ;$$

- *R-Ate*

$$\begin{aligned} R_{A,B}(Q, P)^{\frac{q^k-1}{r}} &= \left\{ f_{a,[B]Q}(P) \cdot f_{b,Q}(P) \cdot G_{[aB]Q,[b]Q}(P) \right\}^{\frac{q^k-1}{r}} \\ &= \left\{ \tilde{W}_{[B]Q,P}(a, 1) \cdot \tilde{W}_{Q,P}(b, 1) \cdot G_{[-aB]Q,[-b]Q}{}^{-1}(P) \right\}^{\frac{q^k-1}{r}} , \end{aligned}$$

  where $A = aB + b$;

- *optimal*

$$\begin{aligned} a_{[c_0,c_1,\ldots,c_l]}(Q, P) &= \Big\{ \prod_{i=0}^{l} f_{c_i,Q}(P)^{q^i} \cdot \prod_{i=0}^{l-1} G_{[s_{i+1}]Q,[c_i q^i]Q}(P) \Big\}^{\frac{q^k-1}{r}} \\ &= \Big\{ \prod_{i=0}^{l} \tilde{W}_{Q,P}(c_i, 1)^{q^i} \cdot \prod_{i=0}^{l-1} G_{[-s_{i+1}]Q,[-c_i q^i]Q}{}^{-1}(P) \Big\}^{\frac{q^k-1}{r}} . \end{aligned}$$

For Tate pairings, we have the following stronger result.

Theorem 5. *Let E be an elliptic curve over a finite field $\mathbb{F}_q$. We assume that the embedding degree $k > 1$. Let r be a large prime number with $r|\#E(\mathbb{F}_q)$ and $(r, q) = 1$. Then, for $P \in E(\mathbb{F}_q)[r]$ and $Q \in E(\mathbb{F}_{q^k})$,*

$$\tau_r(P, Q) = f_{r,P}(Q)^{\frac{q^k-1}{r}} = W_{P,Q}(r, 1)^{\frac{q^k-1}{r}} . \tag{4}$$

Proof. Note that Tate pairing $f_{r,P}(Q)$ is uniquely defined over $\pmod{(\mathbb{F}_{q^k}^{\times})^r}$ even though $f_{r,P}$ is not normalized since $P \in E(\mathbb{F}_q)[r]$. Then, just as in the proof of the Lemma 2,

$$f_{r,P}(Q) \equiv W_{-P,Q}(r,1)^{-1} \mod (\mathbb{F}_{q^k}^{\times})^r .$$

Therefore, from the bilinearity of $\tau_r(P,Q) = f_{r,P}(Q)^{\frac{q^k-1}{r}}$,

$$\tau_r(P,Q) = \tau_r(-P,Q)^{-1} = W_{P,Q}(r,1)^{\frac{q^k-1}{r}} .$$

Remark 2. *The differences between (3) in Theorem 2 (see p.7) and (4) are explained as follows. In [18], Stange gave a general formula of the Tate pairing with a parameter S by using the divisor D_Q. We obtain (3) by putting $S = P$. On the other hand, we need to compute only $W_{P,Q}(r,1)$ because we evaluate the function f_P at the point Q. We can verify $W_{P,Q}(r,1) \equiv W_{P,Q}(r+1,1) \pmod{(F_{q^k}^{\times})^r}$ because $f_{r,P}(Q) = f_{r+1,P}(Q)$ if $[r]P = O$. (Here we note that $f_{r,P}$, $f_{r+1,P}$ are normalized.) Since we assume that P is an $\mathbb{F}_q$ rational point on E, we can compute the Tate pairing $\langle P,Q\rangle_r$ by evaluating $f_{r,P}$ at Q. Thus, the equation (4) is a special case of (3). However, (4) is sufficient and efficient for cryptographic use.*

4 Implementation

In this section, we will show some experimental results for implementations of various pairings using elliptic nets.

The computer specifications are the following: CPU, a 2 GHz AMD Opteron 246; memory, 4 GB; and hard disk, 160 GB. Magma [23] was used as the software for writing the program.

We used the following elliptic curves for our experiments.

1 $y^2 = x^3 + 4$ [4]
$k = 12$,
$q = 2349801752596847369029608311386467706368831787348451364102015842544 7$ (224 bit),
$r = 17064817657290063780567158346925100943102388 33$ (151 bit),
$T = T_n = 203247593908$.

2 $y^2 = x^3 + 3$ [5]
$k = 12$,
$q = 1461501624496790265145448589920785493717258890819$ (160 bit),
$r = 1461501624496790265145447380994971188499300027613$ (160 bit),
$T = T_n = 1208925814305217958863206$.

3 $y^2 = x^3+2x+2557544131752059464799627850932759581478117758360748682544 75\backslash$
$5542022504589304559812663114754842137$ [13]
$k = 10$,
$q = 2691656114049822988376675914574795422806785455749627181432 97\backslash$

$$962763087823609651608159505713306695 69 \text{ (324 bit)},$$
$$r = 1184972659906501436389408869130632556884221748131065689 61 \text{ (187 bit)},$$
$$T = -12131133023075412575000611486055266851595610191692815,$$
$$T_n = 104334294221056.$$

The Tables 1 and 2 show the experimental results of our implementations. The column "EN" indicates a computation using elliptic nets. The column "Miller" indicates a computation using Miller's algorithm. Note that we did not use built-in functions in Magma (such as "ReducedTatePairing") but rewrote Miller's algorithm by using the Magma language.

The column "R-Ate (i)" corresponds to the index i in Corollary 3.3 of [10]. Note that showing values in some cells parenthetically indicates that those values correspond to values in other cells. For example, the calculation of the Ate_i pairing is sometimes equivalent to that of the Ate pairing.

Our experimental results show that pairing computations using elliptic nets is comparable to those using Miller algorithm in terms of efficiency. However, our implementations were not optimized, and so we need to study these algorithms in detail and optimize their implementations of various pairings.

Table 1. Experimental Results for Tate, Ate, and Ate_i Pairings

	Tate		Ate		Ate_i	
curve	EN[s]	Miller[s]	EN[s]	Miller[s]	EN[s]	Miller[s]
1	0.19	0.26	0.22	0.19	(0.22)	(0.19)
2	0.13	0.21	0.24	0.21	(0.24)	(0.21)
3	0.21	0.31	0.39	0.37	0.23	0.22

Table 2. Experimental Results for R-Ate and Optimal Pairings

	R-Ate (2)		R-Ate (3)		R-Ate (4)		Optimal	
curve	EN[s]	Miller[s]	EN[s]	Miller[s]	EN[s]	Miller[s]	EN[s]	Miller[s]
1	0.65	0.51	0.38	0.31	0.39	0.32	0.98	0.76
2	0.34	0.27	0.33	0.27	0.34	0.26	0.74	0.56
3	0.73	0.67	0.36	0.34	0.40	0.38	1.07	0.94

5 Conclusion

In this paper, we explicitly gave a normalization of elliptic nets and gave formulae based on elliptic nets for computing some variants of the Tate pairing: the Ate, Ate_i, R-Ate, and optimal pairings. We also discussed their efficiency by using some experimental results. Further improvement and optimization of these elliptic net-based algorithms are expected in future work.

Acknowledgment. The authors would like to thank the anonymous referees for various improvements. This work was supported Grants-in-Aid for Scientific Research 20540125, 22500005 and 22300002.

References

1. Barreto, P.S.L.M., Galbraith, S.D., ÓhÉigeartaigh, C., Scott, M.: Efficient pairing computation on supersingular abelian varieties. Designs, Codes and Cryptography 42(3), 239–271 (2007)
2. Barreto, P.S.L.M., Kim, H.Y., Lynn, B., Scott, M.: Efficient algorithms for pairing-based cryptosystems. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 354–369. Springer, Heidelberg (2002)
3. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–369. Springer, Heidelberg (2001)
4. Barreto, P.S.L.M., Lynn, B., Scott, M.: Constructing elliptic curves with prescribed embedding degrees. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 2002. LNCS, vol. 2576, pp. 257–267. Springer, Heidelberg (2003)
5. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006)
6. Duursma, I., Lee, H.-S.: Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 111–123. Springer, Heidelberg (2003)
7. Galbraith, S.D.: Pairings. In: Blake, I., Seroussi, G., Smart, N. (eds.) Advances in Elliptic Curve Cryptography, Ch. IX, Cambridge University Press, Cambridge (2005)
8. Hess, F., Smart, N.P., Vercauteren, F.: The Eta pairing revisited. IEEE Transaction on Information Theory 52(10), 4595–4602 (2006)
9. Joux, A.: A one round protocol for tripartite Diffie–Hellman. In: Bosma, W. (ed.) ANTS 2000 Part IV. LNCS, vol. 1838, pp. 385–393. Springer, Heidelberg (2000)
10. Lee, E., Lee, H.S., Park, C.M.: Efficient and generalized pairing computation on abelian varieties. IEEE Transactions on Information Theory 55(4), 1793–1803 (2009)
11. Miller, V.S.: Short programs for functions on curves (1986), http://crypto.stanford.edu/miller/miller.pdf
12. Miller, V.S.: The Weil pairing and its efficient calculation. Journal of Cryptology 17(4), 235–261 (2004)
13. Murphy, A., Fitzpatrick, N.: Elliptic Curves for Pairing Applications. Cryptology ePrint Archive, Report 2005/302 (2005), http://eprint.iacr.org/2005/302.pdf
14. Ogura, N., Uchiyama, S., Kanayama, N., Okamoto, E.: A note on the pairing computation using normalized Miller functions, to appear in IEICE Transaction on Fundamentals of Electronics, Communications and Computer Sciences
15. Sakai, R., Ohgshi, K., Kasahara, M.: Cryptosystems based on pairings. In: Symposium on Cryptography and Information Security 2000, SCIS (2000)
16. Schoof, R.: Elliptic curves over finite fields and computation of square roots mod p. Math. Comp. 44, 483–494 (1985)
17. Silverman, J.H.: The arithmetic of elliptic curves. Graduate Texts in Mathematics, vol. 106. Springer, Heidelberg (1986)

18. Stange, K.E.: The tate pairing via elliptic nets. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 329–348. Springer, Heidelberg (2007)
19. Taylor, G.: Stange's algorithm for elliptic nets, http://maths.straylight.co.uk/archives/102
20. Zhao, C.-A., Zhang, F., Huang, J.: A note on the Ate pairing. International Journal of Information Security 6(7), 379–382 (2008)
21. Vercauteren, F.: Optimal pairings. IEEE Transactions on Information Theory 56(1), 455–461 (2010)
22. Ward, M.: Memoir on elliptic divisibility sequence. American Journal of Mathematics 70, 31–74 (1948)
23. MAGMA group, Magma, http://magma.maths.usyd.edu.au/magma/

Identity-Based Deterministic Signature Scheme without Forking-Lemma

S. Sharmila Deva Selvi, S. Sree Vivek, and C. Pandu Rangan*

Theoretical Computer Science Laboratory,
Department of Computer Science and Engineering,
Indian Institute of Technology Madras,
Chennai, India
{sharmila,svivek}@cse.iitm.ac.in, prangan@cse.iitm.ac.in

Abstract. Since the discovery of identity based cryptography, a number of identity based signature schemes were reported in the literature. Although, a lot of identity based signature schemes were proposed, the only identity based deterministic signature scheme was given by Javier Herranz. This signature scheme uses Schnorr signature scheme for generating the private key of the users and uses BLS short signature scheme for generating users signature. The security of this scheme was proved in the random oracle model using forking lemma. In this paper, we introduce a new identity based deterministic signature scheme and prove the security of the scheme in the random oracle model, without the aid of forking lemma. Hence, our scheme offers tighter security reduction to the underlying hard problem than the existing identity based deterministic signature scheme.

Keywords: Identity Based Cryptography, Deterministic, Signature, Tight Security, Random Oracle Model, Provable Security, Without Forking-Lemma.

1 Introduction

The concept of using the identity of an entity for deriving the public key is known as Identity Based Cryptography (IBC). This technique was introduced by Adi Shamir in his seminal paper [15] in 1984. This paved way for eliminating the use of certificates for authenticating the public keys of a user (in PKI based system). In identity based system, a trusted authority called Private Key Generator (PKG) generates the private key for the users. The PKG possesses a master public key and master private key and uses the master private key to generate the private key of the users registered with the system. The private key of the user is the signature on the identity of the user (as message) generated by the PKG with the master private key. The user makes use of his/her private key to generate a signature on a message.

* Currently Head, Indian Statistical Institute, Chennai, India.

T. Iwata and M. Nishigaki (Eds.): IWSEC 2011, LNCS 7038, pp. 79–95, 2011.

Thus the complete description of an identity based signature scheme can be conveniently split as the description of the signature scheme employed by the PKG to create private keys for the users and description of signature generation algorithm executed by a user on a message. The signature schemes used in these two parts may resemble some well known signature schemes or customized signature scheme or they may be deterministic or probabilistic. For example, the identity based signature scheme by Cha and Cheon [5] may be viewed as BLS [4] + a customized scheme, where the BLS signature scheme is used by the PKG to generate private keys of users and users themselves use the customized scheme to produce signed documents. For this scheme, the private key generation is deterministic while the signature generation process is probabilistic. As another example, the scheme by Galindo et al. [6] uses Schnorr signature for private key generation (by PKG) and again a Schnorr signature scheme [14] for the signature generation (by a user). For this scheme, the private key generation as well as signature generation is probabilistic. The scheme by Javier Herranz [9] uses Schnorr signature for private key generation (by PKG) and BLS signature scheme for the signature generation (by a user). Thus in this scheme, the private key generation is probabilistic and the signature generation is deterministic. Table-1 gives a summary of properties of existing identity based signature schemes.

Table 1. Properties of ID-Based Signatures
P - Probabilistic Signature, D - Deterministic Signature, Custom - Custom designed signing algorithm

Scheme	**Private Key**	**Signing Algorithm**	**Type of Scheme**		**Pairing Computation**	
			Key	Sign	Sign	Verify
Cha-Cheon [5]	BLS	Custom	D	P	No	Yes
Sakai [13]	BLS	Custom	D	P	No	Yes
Barreto [1]	[16]	Custom	D	P	No	Yes
Galindo [6]	Schnorr	Schnorr	P	P	No	No
Javier [9]	Schnorr	BLS	P	D	No	Yes
Ours	$Basic_{Sign}$	custom	P	D	No	Yes

Tightness of Security Reduction: In the computational model, proof of security for a signature follows if there does not exist a polynomial time algorithm with the following ability:

The reduction algorithm makes use of a polynomial time algorithm that forges a signature, to construct a polynomial time algorithm that solves the computational hard problem. If there is no polynomial time algorithm for solving the computational hard problem then the existence of such reduction implies that the signature scheme is not breakable in polynomial time.

This security argument is asymptotic. In CDH based signature schemes, forging signatures is infeasible in prime order groups where the size of the security parameter is above some threshold value. For practice, we should exactly know

Table 2. Tightness Comparison with the Existing Scheme

T - Tight, NT - Not Tight (uses forking-lemma), P - Probabilistic Signature, D - Deterministic Signature, † - The eight fold increase is due to loose reduction through forking lemma, We consider Elliptic Curve CDH is hard in 320 bits.

Scheme	Tightness		Implication on size of $\|p\|$		Size of one group element	Type
	Key	Sign	Key Size	Sign Size		
Javier [9]	NT	NT	8*320=2560†	8*320=2560†	2560	D
Ours	T	T	320	320	320	D

what should be the constraint on security parameter to impose a sufficient infeasible computational bound on the adversary.

Bellare and Rogaway [3] gave the method for exact security analysis that focuses on the computational efficiency of the reduction algorithm. This allows one to quantify the relation between the difficulty of forging a signature and hardness of the underlying hard problem. The relative hardness of forging the signature to that of breaking the computational assumption can be loose, close or tight as pointed out by Micali and Reyzin [10]. In [7], Goh et al. showed that the application of forking lemma [12], for proving security of Fiat-Shamir based signatures makes it inefficient by imposing an increase in the length of the modulus p. In any discrete-log based system of a prime field $\mathbb{Z}_p$, breaking the discrete-log in the index-calculus method works in $\mathcal{O}(exp(\sqrt[3]{|p|}))$. Thus, a factor α increase in the security parameter implies a α^3 increase in the size of the modulus p. This is why, the reduction with forking lemma for Schnorr signature scheme implies that the scheme is secure only with a field modulus 8000 bits, if we consider that discrete-log problem is hard for 1000 bit modulus. In Table-2, we do not consider the schemes reported in [5,13,1,6] because they are all probabilistic signature schemes. We consider the scheme in [9] for comparing with our scheme.

Application: Aggregation of several signatures is an important computation done on several signatures in order to optimize communication, computation and storage costs. Depending on the size of the aggregated output, we refer a particular aggregation scheme as Naive, Partial or Full aggregation. By using the identity based signature scheme by Herranz [9] partial aggregation is possible. His scheme allows a more compact aggregation where the length of the resulting aggregate signature will not depend on the number of signed messages, but on the number of signers. This improvement, is considered to be a major improvement in [9] because in situations where devices have to store many signatures coming from a small set of users, the size of the aggregate signature gets compact. This is because, the key generation is probabilistic and the randomness used to compute the key can be stored by the verifier and since the signature is deterministic, there is no randomness to be propagated with the signature and hence the aggregate signature is more compact than the aggregate signatures generated by probabilistic identity based signature schemes.

Our Contribution: Our first contribution is a novel probabilistic PKI based signature scheme (and this is of independent interest) described in section 4.

PKG uses this to generate the private keys for users. The next section (section 5) contains the details of an identity based deterministic signature scheme, and again this scheme is different from all the existing ones. Our scheme does not use pairing in the generation process. Of course, in the verification process, we employ pairing computations. The significant advantage of our scheme is that it allows a tight reduction to the GDH problem. For all the schemes that are available so far, the reduction is not tight. However, we show a tight reduction of the security of our scheme to the GDH problem. Ours is the first and only system with this property. Due to this property both the key size and signature size are substantially smaller than the best previously known schemes. Since our identity based signature scheme offers tight reduction to the GDH problem, it can be used to generate more compact aggregate signatures using smaller security parameter values.

2 Preliminaries

Bilinear Pairing: Let $\mathbb{G}_1$ be an additive cyclic group generated by P, with prime order q, and $\mathbb{G}_2$ be a multiplicative cyclic group of the same order q. A bilinear pairing is a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ with the following properties.

- ***Bilinearity.*** For all $P, Q, R \in \mathbb{G}_1$,
  - $\hat{e}(P + Q, R) = \hat{e}(P, R)\hat{e}(Q, R)$
  - $\hat{e}(P, Q + R) = \hat{e}(P, Q)\hat{e}(P, R)$
  - $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$ [Where $a, b \in_R \mathbb{Z}_p$]
- ***Non-Degeneracy.*** There exist $P, Q \in \mathbb{G}_1$ such that $\hat{e}(P, Q) \neq I_{\mathbb{G}_2}$, where $I_{\mathbb{G}_2}$ is the identity element of $\mathbb{G}_2$.
- ***Computability.*** There exists an efficient algorithm to compute $\hat{e}(P, Q)$ for all $P, Q \in \mathbb{G}_1$.

Computational Assumptions: In this section, we review the computational assumptions related to bilinear maps that are relevant to the protocol we discuss.

Definition 1. *Computation Diffie-Hellman Problem (CDHP): Given* $(P, aP, bP) \in \mathbb{G}_1^3$ *for unknown* $a, b \in \mathbb{Z}_p$*, the CDH problem in* $\mathbb{G}_1$ *is to compute* abP*. The advantage of any probabilistic polynomial time algorithm* $\mathcal{A}$ *in solving the CDH problem in* $\mathbb{G}_1$ *is defined as:*

$$Adv_{\mathcal{A}}^{CDH} = Pr\left[\mathcal{A}(P, aP, bP) = abP \mid a, b \in \mathbb{Z}_p\right]$$

The CDH Assumption is that, for any probabilistic polynomial time algorithm $\mathcal{A}$*, the advantage* $Adv_{\mathcal{A}}^{CDH}$ *is negligibly small.*

Definition 2. *Decisional Diffie-Hellman Problem (DDHP): Given* $(P, aP, bP, Q) \in \mathbb{G}^4$ *for unknown* $a, b \in \mathbb{Z}_p$*, the DDH problem in* $\mathbb{G}$ *is to check whether* $Q \stackrel{?}{=} abP$*. The advantage of any probabilistic polynomial time algorithm* $\mathcal{A}$ *in solving the DDH problem in* $\mathbb{G}_1$ *is defined as:*

$$Adv_{\mathcal{A}}^{CDH} = \left|Pr\left[\mathcal{A}(P, aP, bP, Q) = 1\right] - Pr\left[\mathcal{A}(P, aP, bP, abP) = 1\right] \mid a, b \in \mathbb{Z}_p\right|$$

The DDH Assumption is that, for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage $Adv_{\mathcal{A}}^{DDH}$ is negligibly small. Here $\mathbb{G}$ is a multiplicative group

Definition 3. *Gap Diffie-Hellman Problem (GDHP) [11][5]: We call $\mathbb{G}$ a GDH group if DDHP can be solved in polynomial time but no probabilistic algorithm can solve CDHP with non-negligible advantage within polynomial time.*

3 Identity Based Signature Scheme

In this section, we describe the generic frame work for an identity based signature scheme. The frame work of an identity based deterministic signature scheme consists of the algorithms described below, namely **Setup**, **Extract**, **Sign** and **Verify**. An identity based signature scheme is deterministic if the signature on a message by the same user is always the same.

3.1 Definition

- ***Setup:*** *The private key generator (PKG) provides the security parameter κ as the input to this algorithm, generates the system parameters params and the master private key msk. PKG publishes params and keeps msk secret.*
- ***Extract:*** *The user provides his identity ID to the PKG. The PKG runs this algorithm with identity ID, params and msk as the input and obtains the private key D. The private key D is sent to user through a secure channel.*
- ***Sign:*** *For generating a signature on a message m, the user provides his identity ID, his private key D, params and the message m as input. This algorithm generates a valid signature σ on message m by the user.*
- ***Verify:*** *This algorithm on input a signature σ on message m by the user with identity ID, params, checks whether σ is a valid signature on message m by ID. If true it outputs "$Valid$", else it outputs "$Invalid$".*

3.2 Security Model for Existential Unforgeability

An IBDS scheme is secure against existential forgery under adaptive chosen identity and message attack, if no probabilistic polynomial time algorithm $\mathcal{F}$ has non-negligible advantage in the following game.

- **Setup phase:** The challenger $\mathcal{C}$ runs the setup algorithm and generates the system public parameters *params* and the master secret key *msk*. Now, $\mathcal{C}$ gives *params* to the forger $\mathcal{F}$ and keeps *msk* secret.
- **Training phase:** After the setup is done, $\mathcal{F}$ starts interacting with $\mathcal{C}$ by querying the various oracles provided by $\mathcal{C}$ in the following way:
  - **KeyGen oracle:** When $\mathcal{F}$ makes a query with an identity ID as input, $\mathcal{C}$ outputs D, the private key of ID to $\mathcal{F}$, provided $\mathcal{C}$ knows the private key for the queried identity.
  - **Signing oracle:** When $\mathcal{F}$ makes a signing query with identity ID and message m, $\mathcal{C}$ outputs a valid signature σ on m by ID.

– **Forgery phase:** $\mathcal{F}$ identifies an identity, message pair (ID_T, m^*), where
  - $\mathcal{F}$ has not queried the $KeyGen$ query on ID_T and
  - $\mathcal{F}$ has not asked the signature for the pair (ID_T, m^*).

  $\mathcal{F}$ outputs a signature σ, with ID_T as signer, and on message m^*. $\mathcal{F}$ wins the game if σ is a valid signature.

$$Adv_{\mathcal{F}}^{IBDS} = \{Pr[\mathcal{F}(Verify(\sigma) = valid)\}$$

3.3 Existing Identity Based Signatures

Here, we review the most important identity based signature schemes.

Table 3. Brief Survey of existing schemes

MSK - Master Private Key, MPK - Master Public Key, $\hat{H}, \bar{H}$ - Cryptographic hash functions

Scheme	Master key	Private Key	Signature
Cha-Cheon [5]	$MSK = s$ $MPK = sP$	$D_A = sQ_A \in \mathbb{G}_1$ $Q_A = \hat{H}(ID_A) \in \mathbb{G}_1$	$r \in_R \mathbb{Z}_q^*$ $U = rQ_A \in \mathbb{G}_1, h = \bar{H}(m, U) \in \mathbb{Z}_q^*$ $V = (r + h)D_A \in \mathbb{G}_1$, $\sigma = \langle U, V \rangle$
Sakai [13]	$MSK = s$ $MPK = sP$	$D_A = sQ_A \in \mathbb{G}_1$ $Q_A = \hat{H}(ID_A) \in \mathbb{G}_1$	$r \in_R \mathbb{Z}_q^*$ $U = rP \in \mathbb{G}_1, H = \bar{H}(m, U) \in \mathbb{G}_1$ $V = rH + D_A \in \mathbb{G}_1$, $\sigma = \langle U, V \rangle$
Barreto [1]	$MSK = s$ $MPK = sP$	$D_A = \frac{1}{s+q_A} P \in \mathbb{G}_1$ $q_A = \hat{H}(ID_A)$	$r \in_R \mathbb{Z}_q^*$ $U = rP \in \mathbb{G}_1, h = \bar{H}(m, U) \in \mathbb{Z}_q^*$ $V = (r + h)D_A \in \mathbb{G}_1$, $\sigma = \langle U, V \rangle$
Galindo [6]	$MSK = s$ $MPK = sP$	$x_A \in_R \mathbb{Z}_q^*, X_A = x_A P$ $d_A = x_A + sq_A \in \mathbb{Z}_q^*$ $q_A = \hat{H}(ID_A, X_A)$	$r \in_R \mathbb{Z}_q^*$ X_A, $U = rP, h = \bar{H}(m, U) \in \mathbb{Z}_q^*$ $V = r\bar{h} + d_A \in \mathbb{G}_1$, $\sigma = \langle X_A, U, V \rangle$
Javier [9]	$MSK = s$ $MPK = sP$	$x_A \in_R \mathbb{Z}_q^*, X_A = x_A P$ $d_A = x_A + sq_A \in \mathbb{Z}_q^*$ $q_A = \hat{H}(ID_A, X_A)$	X_A $U = d_A \bar{H}(m) \in \mathbb{G}_1$ $\sigma = \langle X_A, U \rangle$

4 Basic Signature Scheme ($Basic_{Sign}$)

We now construct a fully secure public key signature scheme in the random oracle model under the GDH assumption and without using forking lemma. This is a PKI based signature scheme and this will be used by the PKG to generate the private key for the users of our identity based system.

Scheme: Let $\mathbb{G}_1$, $\mathbb{G}_2$ be cyclic prime order groups of order p, where $\mathbb{G}_1$ is an additive group and $\mathbb{G}_2$ be a multiplicative group. Let $P \in_R \mathbb{G}_1$ be the generator of $\mathbb{G}_1$, $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be a bilinear map and $H_1(.), H_2(.)$ be two cryptographic hash functions defined by,

$$H_1\text{: } \{0,1\}^{l_m} \rightarrow \mathbb{G}_1 \text{ and } H_2\text{: } \{0,1\}^{l_m} \times \mathbb{G}_1 \rightarrow \mathbb{Z}_p$$

– **User KeyGen**: Let U_A be a user with public key $PK_A = \langle P_1, P_2 \rangle = \langle s_1 P, s_2 P \rangle$, where s_1, s_2 are random elements from $\mathbb{Z}_p$. Here, the private key of user U_A is $SK_A = \langle s_1, s_2 \rangle$.

– **Sign:** To generate the signature on message m, the user U_A executes this algorithm:

- Pick r randomly from $\mathbb{Z}_p$.
- Compute $Y_m = rP_2$
- Compute $X_m = rH_1(m, Y_m)$.
- Find $q_m = H_2(m, X_m)$.
- Compute $d_m = q_m s_1 + rs_2 \ mod \ p$.
- Output the signature $\sigma = \langle X_m, d_m \rangle$

Important Note: The value Y_m is not sent along with the signature because it can be computed from the second component of σ as follows and the hash value q_m is computable by any one on knowing m and X_m:

$$Y_m = d_m P - q_m P_1 = q_m P_1 + rP_2 - q_m P_1 = rP_2$$

The tuple $\langle P_2, H_1(m, Y_m), Y_m, X_m \rangle = \langle P_2, H_1(m, Y_m), rP_2, rH_1(m, Y_m) \rangle$ is a DH tuple. We verify if $\langle P_2, H_1(m, Y_m), Y_m, X_m \rangle$ is a DH tuple by testing $\hat{e}(X_m, P_2) \stackrel{?}{=} \hat{e}(H_1(m, Y_m), Y_m)$. This suggests the following verification algorithm.

– **Verify**

- On receiving $\sigma = \langle X_m, d_m \rangle$, compute $q_m = H_2(m, X_m)$ and $Y_m = d_m P - q_m P_1$.
- Check if $\hat{e}(X_m, P_2) \stackrel{?}{=} \hat{e}(H_1(m, Y_m), Y_m)$.

If the above check holds accept the signature as "$Valid$" else return "$Invalid$".

4.1 Security

We prove the security of the signature scheme against existential forgery under adaptive chosen-message attacks in the random oracle model. The following theorem shows that the $Basic_{Sign}$ scheme is secure and the security of the scheme follows from the GDH assumption in $(\mathbb{G}_1, \mathbb{G}_2)$.

Theorem 1. *Suppose* $(\mathbb{G}_1, \mathbb{G}_2)$ *be a* (τ, t', ε')*-GDH group pair of order* p*. Then the* $Basic_{Sign}$ *signature scheme on* $(\mathbb{G}_1, \mathbb{G}_2)$ *is* $(t, q_{Sign}, q_{H_1}, q_{H_2}, \varepsilon)$*-secure against existential forgery under adaptive chosen-message attack in the random oracle model, for all* t *and* ε*, that satisfies*

$$\varepsilon \leq \varepsilon' \ \text{and} \ t \geq t' - (q_{H_1} + q_{H_2} + q_{Sign} + \mathcal{O}(1))$$

Proof: Let us assume, $\mathcal{F}$ is a forger algorithm that $(t, q_{Sign}, q_{H_1}, q_{H_2}, \varepsilon)$-breaks the $Basic_{Sign}$ signature scheme on $(\mathbb{G}_1, \mathbb{G}_2)$. We show how to construct a t'-time

algorithm $\mathcal{C}$ that solves GDH on $(\mathbb{G}_1, \mathbb{G}_2)$ with probability at least ε'. Showing this will contradict the fact that $(\mathbb{G}_1, \mathbb{G}_2)$ is a (t', ε')-GDH group pair.

Let P be the generator of $\mathbb{G}_1$. Algorithm $\mathcal{C}$ is provided with the challenge instance $(P, aP, bP) \in \mathbb{G}_1$. The goal of $\mathcal{C}$ is to output $abP \in \mathbb{G}_1$. Algorithm $\mathcal{C}$ simulates the challenger and interacts with $\mathcal{F}$ in the following way:

- **Setup:** Challenger $\mathcal{C}$ starts by giving $\mathcal{F}$ the common reference string $(P, \mathbb{G}_1, \mathbb{G}_2)$ and the public key $(P_1 = aP, P_2 = s_2P)$, where s_2 is chosen at random from $\mathbb{Z}_p$. The private key corresponding to the public keys $(P_1 = aP, P_2 = s_2P)$ are (a, s_2). Note that, $\mathcal{C}$ does not know one of the private keys namely a.
- **Training Phase:** During this phase $\mathcal{F}$ has access to the following oracles:
  - *H_1 Queries:* Forger $\mathcal{F}$ is allowed to query the H_1 oracle at any time. To handle these queries $\mathcal{C}$ maintains a list which is defines as $\langle m, Y_m, h, H_m \rangle$ and we refer this list as $\mathcal{L}_1 - list$. Initially, this list is empty and will be updated as explained below. When $\mathcal{F}$ queries the oracle H_1 with $(m \in \{0,1\}^{l_m}, Y_m \in \mathbb{G}_1)$ as input, $\mathcal{C}$ responds as follows:
    - If (m, Y) already exists as a tuple of the form $\langle m, Y_m, h, H_m \rangle$ in $\mathcal{L}_1 - List$, then $\mathcal{C}$ responds with $H_1(m, Y_m) = H_m \in \mathbb{G}_1$.
    - Otherwise, $\mathcal{C}$ picks a random $h \in \mathbb{Z}_p$ and sets $H_m = hbP$.
    - $\mathcal{C}$ stores the tuple $\langle m, Y_m, h, H_m \rangle$ in $\mathcal{L}_1 - List$ and responds with $H_1(m, Y_m) = H_m \in \mathbb{G}_1$.
  - *H_2 Queries:* $\mathcal{F}$ can query this oracle at any time and $\mathcal{C}$ maintains a list of tuples $\langle m, X_m, q_m \rangle$. This list is called $\mathcal{L}_2 - list$. When $\mathcal{F}$ issues a query for (m, X_m) to the H_2 oracle, $\mathcal{F}$ responds in the following way:
    - If (m, X_m) already appears in $\mathcal{L}_2 - list$ as a tuple $\langle m, X_m, q_m \rangle$, then $\mathcal{C}$ responds with $H_2(m, X_m) = q_m \in \mathbb{Z}_p$.
    - Otherwise, $\mathcal{C}$ randomly picks a $q_m \in \mathbb{Z}_p$, stores the tuple $\langle m, X_m, q_m \rangle$ in $\mathcal{L}_1 - list$ and responds with $H_2(m, X_m) = q_m \in \mathbb{Z}_p$.
  - *Signature Queries:* When a signature query is issued by $\mathcal{F}$ for message m, $\mathcal{C}$ responds as follows:
    - $\mathcal{C}$ randomly picks $d_m, q_m, h \in \mathbb{Z}_p$.
    - Then, $\mathcal{C}$ sets $H_m = hP$, $Y_m = d_mP - q_mP_1 \in \mathbb{G}_1$ and $X_m = \left(\frac{h}{s_2}\right) Y_m \in \mathbb{G}_1$.
    - If a tuple of the form $\langle m, X_m, q_m \rangle$ appears in the list $\mathcal{L}_1 - list$ or a tuple $\langle m, Y_m, h, H_m \rangle$ appears in the list $\mathcal{L}_2 - list$, then repeat the process by picking new set of random values $d_m, q_m, h \in \mathbb{Z}_p$.
    - $\mathcal{C}$ stores the tuple $\langle m, X_m, q_m \rangle$ in $\mathcal{L}_1 - list$ and $\langle m, Y_m, h, H_m \rangle$ in $\mathcal{L}_2 - list$.
    - $\mathcal{C}$ gives the signature $\sigma = \langle X_m, d_m \rangle$ to $\mathcal{F}$.

    Correctness: The simulated signature is valid and passes the verification test $\hat{e}(X_m, Y) \stackrel{?}{=} \hat{e}(H_1(m, Y_m), Y_m)$. The correctness is shown below:

$$\text{LHS} = \hat{e}(X_m, Y) = \hat{e}((\tfrac{h}{s_2})Y_m, Y) = \hat{e}((\tfrac{h}{s_2})Y_m, s_2P)$$
$$= \hat{e}(hY_m, P) = \hat{e}(Y_m, hP) = \hat{e}(Y_m, H_1(m, Y_m)) = \text{RHS}$$

– **Forgery:** On getting sufficient training, algorithm $\mathcal{F}$ produces a message-signature pair $(m^*, \sigma^* = \langle X_m^*, d_m^* \rangle)$ such that σ^* is not the output generated by sign oracle for message m^* and σ^* is valid. Now, $\mathcal{C}$ may compute the solution to the hard problem as given below.

- $\mathcal{C}$ computes $q_m^* = H_2(m^*, X_m^*)$ and $\delta = \dfrac{1}{q_m^*}\left(d_m^*(bP) - \dfrac{s_2}{h^*}X_m^*\right)$.
- According to the signature definition $X^* = r^*H_m^* = r^*h^*bP$, $Y_m^* = r^*P_2$ and $d_m^* = q_m^*a + r^*s_2$, by the definition of H_2.
- Therefore, $\delta = \dfrac{1}{q_m^*}\left((q_m^*a + r^*s_2)bP - \dfrac{s_2}{h^*}r^*h^*bP\right) = abP$.

This completes the description of algorithm $\mathcal{C}$. Now, we have to show that $\mathcal{C}$ solves the GDH problem on $(\mathbb{G}_1, \mathbb{G}_2)$ with probability at least ε'. Note that, in this simulation there is almost no aborting scenario for training phase and forgery phase. Hence $\mathcal{C}$ solving the GDH problem happens almost with the same advantage of $\mathcal{F}$.

The hard problem is solved after q_{H_1} queries to the H_1 oracle, q_{H_2} queries to the H_2 oracle and q_{Sign} sign oracle queries and getting the forged signature. The challenger has to spend $\mathcal{O}(1)$ computation to extract the solution to GDH problem from the forgery generated by the adversary. Therefore the total time t taken for solving the hard problem is given by $t \leq t' + (q_{H_1} + q_{H_2} + q_{Sign} + \mathcal{O}(1))$. □

5 Identity Based Deterministic Signature Scheme (Det-IBS)

Inspired by the impact of tightness of security reduction for a signature scheme, we present the first identity based deterministic signature scheme with tight security reduction to GDH problem. The only identity based deterministic signature by Herranz [9], employs Schnorr signature scheme for generating the private key of the user and uses BLS short signature scheme for producing signature on the message by the user. This system was shown to be secure under GDH problem on $(\mathbb{G}_1, \mathbb{G}_2)$. The reduction given for the scheme in [9] use forking lemma and hence considered to be loose. We present a signature that works on GDH group pair $(\mathbb{G}_1, \mathbb{G}_2)$. We prove the security of the scheme in the random oracle model and show how it leads to a tight reduction. The scheme uses $Basic_{Sign}$ signature scheme for generating the private key of users and BLS short signature for generating the signature on message.

Scheme: Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a (t, ε)-GDH group pair with same prime order p and $\hat{e}$ be a bilinear map defined by $\mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. The signature scheme comprises of setup, extract, sign and verify algorithms. The scheme makes use of three cryptographic hash functions $H_1 : \{0,1\}^{l_1} \times \mathbb{G}_1 \to \mathbb{G}_1$, $H_2 : \{0,1\}^{l_1} \times \mathbb{G}_1 \to \mathbb{Z}_p$ and $H_3 : \{0,1\}^{l_m+1} \times \{0,1\}^{l_1} \to \mathbb{G}_1$, where l_1 is the size of the identity string and l_m is the size of the message.

- **Det-IBS.Setup:** PKG picks at random $s_1, s_2 \in \mathbb{Z}_p$, and $P \in \mathbb{G}_1$, sets $P_1 = s_1 P \in \mathbb{G}_1$, and $P_2 = s_2 P \in \mathbb{G}_1$. The master public key is (P_1, P_2). The master private key is (s_1, s_2).
- **Det-IBS.Extract:** Given the master private (s_1, s_2), and the user identity $ID_A \in \{0,1\}^{l_1}$, perform the following:
  - Pick $r_A \in_R \mathbb{Z}_p$.
  - Compute $Y_A = r_A P_2 \in \mathbb{G}_1$.
  - Find $H_A = H_1(ID_A, Y_A)$ and set $X_A = r_A H_A \in \mathbb{G}_1$.
  - Compute $d_A = s_1 q_A + s_2 r_A \, mod p$, where $q_A = H_2(ID_A, X_A)$.
  - The private key is $D_A = \langle d_A, X_A, Y_A \rangle$.

  Note: However, in our identity based deterministic signature scheme, we provide Y_A explicitly along with the private key How ever Y_A is computable by the user with identity ID_A on knowing d_A and X_A.
- **Det-IBS.Sign:** Given a message m, user identity $ID_A \in \{0,1\}^{l_1}$ and the user private key $D_A = \langle d_A \in \mathbb{Z}_p, X_A \in \mathbb{G}_1, Y_A \in \mathbb{G}_1 \rangle$, choose $\lambda \in_R \{0,1\}$, compute $H_m = H_3(m \| \lambda, ID_A)$ and $V = d_A H_m$. The signature is $\sigma = \langle V, \lambda, X_A, Y_A \rangle \in \mathbb{G}_1^3 \times \{0,1\}$.

  Note: λ can be generated using a pseudo-random function with the identity ID_A, message m and the private key of the user as input. This helps to preserve the determinism because each time a message is signed by a user, the bit λ is going to be the same. (Goh et al. [8]).
- **Det-IBS.Verify:** Given an identity $ID_A \in \{0,1\}^{l_1}$, a message $m \in \{0,1\}^{l_m}$, and a signature $\sigma = \langle V \in \mathbb{G}_1, \lambda \in \{0,1\}, X_A \in \mathbb{G}_1, Y_A \in \mathbb{G}_1 \rangle$, compute $q_A = H_2(ID_A, X_A)$, $H_A = H_1(ID_A, Y_A)$, and $H_m = H_3(m \| \lambda, ID_A)$ and check,

$$\hat{e}(V, P) \stackrel{?}{=} \hat{e}(H_m, q_A P_1 + Y_A) \text{ ---(1)}$$
$$\hat{e}(X_A, P_2) = \hat{e}(H_A, Y_A) \text{ ---(2)}$$

  If both the check passes, output "*Valid*"; if not, output "*Invalid*"

Theorem 2. *The signature scheme Det-IBS is consistent*

Proof: We need to show that, for all private key tuples , and for all messages, any signature generated by the signing algorithm verifies as a valid signature under the respective user identity. Indeed, we have for equation (1)

$$\begin{aligned} \text{LHS} = \hat{e}(V, P) &= \hat{e}(d_A H_m, P) = \hat{e}((q_A s_1 + r_A s_2) H_m, P) \\ &= \hat{e}(H_m, (q_A s_1 + r_A s_2) P) = \hat{e}(H_m, q_A P_1 + r_A P_2) \\ &= \hat{e}(H_m, q_A P_1 + Y_A) = \text{RHS} \end{aligned}$$

and also, for equation (2)

$$\text{LHS} = \hat{e}(X_A, P_2) \ = \hat{e}(r_A H_A, P_2) = \hat{e}(H_A, r_A P_2) = \hat{e}(H_A, Y_A) = \text{RHS}.$$

5.1 Security

We prove the security of the identity based deterministic signature scheme against existential forgery under adaptive chosen-message attacks in the random oracle model. The following theorem shows that the *Det-IBS* scheme is secure and the security of the scheme follows from GDH assumption on the groups $(\mathbb{G}_1, \mathbb{G}_2)$.

Theorem 3. *Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a $(\tau_1, t_1, \varepsilon_1)$ GDH group pair of order p then the identity based deterministic signature scheme on $(\mathbb{G}_1, \mathbb{G}_2)$ is $(t_2, q_s, q_{H_1}, q_{H_2}, q_{H_3}, \varepsilon_2)$ - secure against existential forgery under an adaptive chosen message attack in random oracle model, for all t_2 and ε_2 satisfying:*

$$\varepsilon_2 \geq 2q_{H_1}\varepsilon_1 \, and \, t_2 \leq t_1 - (q_{H_1} + q_{H_2} + q_{H_3} + 2q_s + \mathcal{O}(1))$$

Here, q_H is the total number of identities generated.

Proof: Consider $\mathcal{F}$ to be a forger that is assumed to $(t_2, q_s, q_{H_1}, q_{H_2}, q_{H_3}, \varepsilon_2)$ - break the signature scheme. We show how to construct an algorithm $\mathcal{C}$ that solves GDHP on $(\mathbb{G}_1, \mathbb{G}_2)$ with probability at least ε_1. This will contradict the fact that $(\mathbb{G}_1, \mathbb{G}_2)$ is a GDH group pair.

For doing this, let us assume P be the generator of $\mathbb{G}_1$ and $(P, aP, bP) \in \mathbb{G}_1^3$ be the GDH problem instance given to $\mathcal{C}$. The goal of $\mathcal{C}$ is to find $abP \in \mathbb{G}_1$. $\mathcal{C}$ simulates the challenger and interacts with $\mathcal{F}$ as defined in the EUF-CMA game. The game is viewed as given below:

- **Setup:** $\mathcal{C}$ starts interaction with $\mathcal{F}$ by providing $P \in \mathbb{G}_1$, $P_1 = aP \in \mathbb{G}_1$ and $P_2 = s_2P$, where $s_2 \in_R \mathbb{Z}_p$. Here, the master private key a is not known to $\mathcal{C}$. $\mathcal{C}$ also chooses $1 \leq T \leq q_H$ randomly and sets the T^{th} unique identity queried to the H_1 hash oracle as the target identity. Without loss of generality, we assume ID_T to be the target identity (not known to $\mathcal{F}$ and $\mathcal{C}$ at the start of the game.)
- **Training Phase:** $\mathcal{C}$ interacts with $\mathcal{F}$ in the following manner:

  $\underline{H_1 \ Oracle:}$ $\mathcal{F}$ queries to this oracle with inputs $\langle ID_i, Y_j \rangle$. $\mathcal{C}$ maintains the list L_{H_1}, consisting of tuples of the form $\langle ID_i, Y_j, H_j, \hat{x}_j \rangle$ and responds to $\mathcal{F}$'s queries in the following way:

  - If the tuple $\langle ID_i, Y_j, H_j, \hat{x}_j \rangle$ is already available in the L_{H_1} list, retrieve and return H_j.
  - If $i \neq T$, choose $\hat{x}_j \in_R \mathbb{Z}_p$ and set $H_j = \hat{x_j}P \in \mathbb{G}_1$. Store the tuple $\langle ID_i, Y_j, H_j, \hat{x}_j \rangle$ to L_{H_1} and return H_j.
  - If $i = T$, choose $\hat{x}_j \in_R \mathbb{Z}_p$ and set $H_j = \hat{x}_j(bP)$. Store the tuple $\langle ID_i, Y_j, H_j, \hat{x}_j \rangle$ to L_{H_1} and return H_j to $\mathcal{F}$.

  $\underline{H_2 \ Oracle:}$ To respond to the queries by $\mathcal{F}$, $\mathcal{C}$ maintains the list L_{H_2}, consisting of tuples of the form $\langle ID_i \in \{0,1\}^{l_1}, X_j \in \mathbb{G}_1, q_j \in \mathbb{Z}_p \rangle$. The list is initially empty. When $\mathcal{F}$ queries with (ID_i, X_j), $\mathcal{C}$ responds as follows:

- If the tuple $\langle ID_i, X_j, q_j \rangle$ already exists in L_{H_2} list, retrieve and return q_j corresponding to (ID_i, X_j) to $\mathcal{F}$.
- Else, pick $q_j \in_R \mathbb{Z}_p$ store (ID_i, X_j, q_j) in L_{H_2} list and return q_j to $\mathcal{F}$.

H_3 Oracle: The input to this oracle are $m_j \| \lambda, ID_i$, where $\lambda \in \{0, 1\}$. To respond to the queries by $\mathcal{F}$, $\mathcal{C}$ maintains the list L_{H_3}, consisting of tuples of the form $\langle m_j, \lambda, ID_i, H_{j,\lambda}, y_{j,\lambda}, \star \rangle$. Here $\star$ is either '$\perp$' or '$\top$', where '$\perp$' represents $H_{j,\lambda} = y_{j,\lambda} P$ and '$\top$' represents $H_{j,\lambda} = y_{j,\lambda} bP$. $\mathcal{C}$ respond to $\mathcal{F}$ in the following way:

- If the tuple $\langle m_j, \lambda, ID_i, H_{j,\lambda}, y_{j,\lambda}, \star \rangle$ is already there in the list L_{H_3}, retrieve and respond with the corresponding $H_{j,\lambda}$.
- Else,
  - Pick $y_{j,0}, y_{j,1} \in_R \mathbb{Z}_p$.
  - Flip a coin $c \in \{0, 1\}$.
  - Set $H_{j,c} = y_{j,c} bP$ and store the tuple $\langle m_j, c, ID_i, H_{j,c}, y_{j,c}, \top \rangle$ in list L_{H_3}.
  - Set $H_{j,\overline{c}} = y_{j,\overline{c}} P$ and store the tuple $\langle m_j, \overline{c}, ID_i, H_{j,\overline{c}}, y_{j,\overline{c}}, \perp \rangle$ in list L_{H_3}.
  - If $c = \lambda$, return $H_{j,c}$ else, return $H_{j,\overline{c}}$.

Note that $H_{j,\lambda}$ is uniform in $\mathbb{G}_1$ and is independent of $\mathcal{F}$'s current view as required.

Extract Oracle: To respond to this query, $\mathcal{C}$ maintains the L_E list, consisting of tuples of the form $\langle ID_i, d_i, X_i, Y_i \rangle$. When $\mathcal{F}$ makes a query with ID_i as input, $\mathcal{C}$ checks whether $i = T$, if so *aborts*. Otherwise, $\mathcal{C}$ performs the following:

- If the tuple corresponding to ID_i is available in list L_E, then retrieve and return (d_i, X_i, Y_i) as the private key corresponding to ID_i to $\mathcal{F}$.
- Otherwise, $\mathcal{C}$ performs the following:
  - Pick $d_i, y_i \in_R \mathbb{Z}_p$.
  - Set $X_i = \frac{d_i \hat{x}_i}{s_2} P - \frac{q_i \hat{x}_i}{s_2} P_1 = (d_i - aq_i) \frac{1}{s_2} H_i$; where $H_i = \hat{x}_i P$.
  - Compute $Y_i = d_i P - q_i P_1 = (d_i - aq_i) P$.
  - Store the tuple $\langle ID_i, X_i, q_i \rangle$ in L_{H_2} list, the tuple $\langle ID_i, Y_i, H_i, \hat{x}_i \rangle$ in L_{H_1} list and the tuple $\langle ID_i, d_i, X_i, Y_i \rangle$ to L_E list.
  - Output (d_i, X_i, Y_i) as the private key.

Without loss of generality, we assume that any identity is queried only once to this oracle.

Signature Oracle: Let (m_j, ID_i) be the message identity pair for which $\mathcal{F}$ request the signature. $\mathcal{C}$ performs the following:

- If there are no entries corresponding to $m_j \| 0, ID_i$ and $m_j \| 1, ID_i$ in the list L_{H_3}, then query the H_3 oracle with input $m_j \| 0, ID_i$.
- Retrieve the entries corresponding to $m_j \| 0, ID_i$ and $m_j \| 1, ID_i$ in list L_{H_3}. Let the two tuples retrieved be $\langle m_j, \lambda, ID_i, H_{j,\lambda}, y_{j,\lambda}, \perp \rangle$ and $\langle m_j, \overline{\lambda},$

$ID_i, H_{j,\overline{\lambda}}, y_{j,\overline{\lambda}}, \top\rangle$ (Note that according to the definition of the H_3 oracle, one of the entries will have $\perp$ and the other one will have $\top$ as the last entry in the tuple.). Pick the entry corresponding to $\perp$, here $\langle m_j, \lambda, ID_i, H_{j,\lambda}, y_{j,\lambda}, \perp\rangle$ is the required tuple.

- If $i \neq T$, then perform the following:
  - Set $H_j = H_{j,\lambda}$
  - Set $V = d_i H_j$. (Note that $\mathcal{C}$ knows the private key d_i corresponding to ID_i)
- If $i = T$, then perform the following:
  - Set $y_j = y_{j,\lambda}$
  - Set $V = y_j(q_T P_1 + x_T P_2)$. (Notice that $V = y_j(q_T P_1 + x_T P_2) = (q_T s_1 + x_T s_2) y_j P = d_T y_j P = d_T H_j$.)
- Return $\sigma = \langle V, X_i, Y_i, \lambda\rangle$ as the signature on the message m_j.

– **Forgery:** Eventually, after getting enough training, $\mathcal{F}$ produces a forgery $m^*, ID_S, \sigma^* = (V, X_S, Y_S, \lambda)$. $\mathcal{C}$ *aborts* if any of the following is true:

- $S \neq T$ (i.e., ID_S is not the target identity set by the challenger).
- The last field of the tuple corresponding to $m^* \| \lambda, ID_S$ in list L_{H_3} is $\perp$ (i.e., $\langle m^*, \lambda, ID_S, H_{j,\lambda}, y_{j,\lambda}, \perp\rangle \in L_{H_3}$).
- σ^* corresponding to m^* is invalid. (Since it is a deterministic signature, m^* should not be queried to the sign oracle with ID_S as the signer.)

Otherwise, $\mathcal{C}$ does the following:

- Find $q_S = H_2(ID_T, X_S)$, $H_S = H_1(ID_T, Y_S)$.
- Retrieve $\hat{x}_S$ corresponding to $\langle ID_T, Y_S, H_S, \hat{x}_S\rangle$ in the list L_{H_1}.
- Compute $\Delta = [q_S^{-1} y_S^{-1}(V - s_2 y_S \hat{x}_S^{-1} X_S)] = abP$.

Note that $\mathcal{C}$ can solve the GDH problem instance irrespective of X_S and Y_S, that is $X_S = X_T$ or $X_S \neq X_T$ and $Y_S = Y_T$ or $Y_S \neq Y_T$

Lemma 1. *Let $\langle m^*, \lambda, ID_S, H_{j,\lambda}, y_{j,\lambda}, \top\rangle$ be the tuple in the list L_{H_3} corresponding to $m^* \| \lambda, ID_S$ and $y_S = y_{j,\lambda}$. If (ID_T, σ^*) is a valid forgery on m^* then $q_S^{-1} y_S^{-1}(V - s_2 y_S \hat{x}_S^{-1} X_S) = abP$ with $P_1 = aP$, $P_2 = s_2 P$, $H_1(ID_T, Y_S) = \hat{x}_S bP$ and $H_{m^*} = H_3(m^*, ID_T) = y_S bP$.*

Proof: The proof is straight forward and is given below:

$$\begin{aligned} LHS &= q_S^{-1} y_S^{-1}(V - s_2 y_S \hat{x}_S^{-1} X_S) \\ &= q_S^{-1} y_S^{-1}(d_S H_{m^*} - s_2 y_S \hat{x}_S^{-1} X_S) \\ &= q_S^{-1} y_S^{-1}((x_S s_2 + a q_S) H_{m^*} - s_2 y_S \hat{x}_S^{-1} x_S \hat{x}_S bP) \\ &= q_S^{-1} y_S^{-1}(x_S s_2 y_S bP + a q_S y_S bP - s_2 y_S x_S bP) \\ &= q_S^{-1} y_S^{-1}(q_S y_S abP) = abP = RHS \end{aligned}$$

□

This completes the description of the game between $\mathcal{C}$ and $\mathcal{F}$. Now, we show how $\mathcal{C}$ solves the GDH instance (P, aP, bP) with probability at least ε_1. For showing this we have to analyze the probability related to the following events:

- $E_1 : \mathcal{C}$ does not abort as a result of Extract query
- $E_2 : \mathcal{F}$ generates a valid message - signature forgery (m^*, σ^*) for $ID_S = ID_T$.
- E_3 : This event occurs for $m^* \| \lambda, ID_S$ such that the last field of the tuple corresponding to $m^* \| \lambda, ID_S$ in list L_{H_3} is $\top$ (i.e., $\langle m^*, \lambda, ID_S, H_{j,\lambda}, y_{j,\lambda}, \top \rangle \in L_{H_3}$).

Let q_{H_1} and q_E denote the number or queries made to the H_1 and Extract oracles. The probability of the above events to occur is discussed below:

- Probability of $\mathcal{C}$ aborting during an extract query is $\frac{1}{q_{H_1}}$. There are totally q_E extract queries. Thus the probability that $\mathcal{C}$ does not abort in any of the extract queries is $1 - \frac{q_E}{q_{H_1}}$ (i.e., $Pr[E_1] = \left(1 - \frac{q_E}{q_{H_1}}\right)$)
- There are totally $q_{H_1} - q_E$ identities are the eligible entities for being a valid ID_S and thus $ID_S = ID_T$ happens with probability $\frac{1}{q_{H_1} - q_E}$ (i.e., $Pr[E_2] = \frac{1}{q_{H_1} - q_E}$)
- Assuming E_2 has happened, the probability that the message $m^* \| \lambda$ being a fruitful instance (i.e., $\langle m^*, \lambda, ID_S, H_{j,\lambda}, y_{j,\lambda}, \top \rangle \in L_{H_3}$) is $\frac{1}{2(q_{H_1} - q_E)}$ (i.e.,$Pr[E_3 | E_2] = \frac{1}{2(q_{H_1} - q_E)}$). Note that every message corresponding to ID_S carries the hard problem instance bP with probability $\frac{1}{2}$.

Now, the probability of $\mathcal{C}$ solving the GDH is $\varepsilon_1 \leq \varepsilon_2 \left(\frac{1}{2(q_{H_1} - q_E)}\right) \left(1 - \frac{q_E}{q_{H_1}}\right) = \varepsilon_2 \left(\frac{1}{2q_{H_1}}\right)$.

Important Remark: The generic method given by Bellare et al. [2] to construct an identity based signature scheme is to use certificate for the public key PK and the identity ID of the user, and then use the certificate and the private key SK corresponding to the public key PK for signing the message. Using this approach, to construct a deterministic identity based signature we require two invocations of BLS signature [4]. The first application of BLS signature is by the PKG to sign ID and PK, and the second BLS signature is by the user to sign the message m using SK. Let the scheme described above be denoted as Γ. While proving the security of the scheme Γ, the advantage of $\mathcal{C}$ in solving the GDH problem is given as $\varepsilon_2 \leq \varepsilon_1 \frac{1}{q_{H_1}^2}$, where ε_1 is the advantage of the forger in breaking the signature scheme Γ. This bound can be further improved with the help of Goh et al.'s [8] technique which leads to a boost in the advantage, which is $\varepsilon_2 \leq \varepsilon_1 \frac{1}{2q_{H_1}}$. Thus the security bounds for the scheme Γ and for our new scheme are equivalent, our scheme achieves this bound without certificates. Also, the size of the signature generated are the same. The signature size of both the schemes will be $\mathbb{G}^3 + |ID|$.

6 Application to Efficient Signature Aggregation

The aggregate signatures generated using our identity based deterministic signature scheme produces an aggregate signature similar to the aggregate signatures described in [9]. The size of the aggregate signature depends on the number of distinct signers and not the number of messages signed. The scheme in [9] uses multiple forking-lemma and hence the size of the security parameter should be increased to achieve sufficient security. Since our scheme does not require forking-lemma in the reduction, the security parameter need not be blown-up as in [9] and as a result we have signature schemes with small size. The details of our aggregate signature scheme follows:

The Setup, Extract, Sign algorithm are the same as Det-IBS.Setup, Det-IBS. Extract and Det-IBS.Sign algorithms. The algorithms aggregate sign and aggregate verify are explained below:

- **Det-IBS.AggSign:** Given n signatures $\sigma_1 = \langle V_1, X_1, Y_1\rangle, \ldots, \sigma_n = \langle V_n, X_n, Y_n\rangle$ on n messages $m_1, \ldots, m_n$ by users with identities $ID_1, \ldots, ID_t$, where $t < n$ and a list $\mathcal{L}$ which provides the details about which message is signed by whom, the aggregate signature σ_{Agg} is computed as follows:
  - Computes $V_{Agg} = \sum_{i=1}^{n} V_i$ and sets $\sigma_{Agg} = \langle V_{Agg}, X_1, \ldots, X_t, Y_1, \ldots, Y_t\rangle$
- **Det-IBS.AggVerify:** Given an aggregate signature σ_{Agg} on n messages $m_1, \ldots, m_n$ by users with identities $ID_1, \ldots, ID_t$, where $t < n$ and a list $\mathcal{L}$, σ_{Agg} is verified as follows:

  Perform the following checks:

$$\hat{e}(V_{Agg}, P) \stackrel{?}{=} \prod_{i=1}^{n} (\hat{e}(H_3(m_i, ID_i), q_i P_1 + Y_i)) \text{ ——(3)}$$

$$\hat{e}(\sum_{j=1}^{t} X_j, P_2) = \prod_{j=1}^{t} \hat{e}(H_1(ID_j, Y_j), Y_j) \text{ ——(4)}$$

  If the checks in (3) and (4) pass, output "$Valid$"; if not, output "$Invalid$"

Note that in the first verification, if a signer has signed more than one message (This will be documented in the list $\mathcal{L}$), the corresponding Y_i and identity ID_i will be reused along with the corresponding message m_i. The second verification needs to be done only for t values as the number of distinct signers is only t. If a single signer, say U_A with identity ID_A signs more than one message, our aggregate signature scheme can be used to generate very efficient aggregate signatures with only three group elements namely, V_{Agg}, X_A and Y_A. This optimizes the storage and communication complexity of signatures generated and communicated by a single user. This cannot be achieved by any of the probabilistic identity based aggregate signature schemes.

We argue the proof of security of the aggregate signature scheme informally. Consider the forger sends an aggregate signature (with one of the signatures that is aggregated as a signature by the target identity and the sign oracle was not queried by the adversary with the corresponding message with the target identity

as the signer), as the forgery in the EUF-CMA (Existential Unforgeability) game. The challenger can remove all other signatures except the one corresponding to the target identity from the aggregate signature by generating them using the values the challenger has obtained from the random oracle lists. Now, the resulting signature is a valid individual signature by the target identity on the target message and that will be a forgery to the basic identity based signature scheme. This is a contradiction since the basic identity based signature scheme is EUF-CMA secure the aggregate signature is also secure.

7 Conclusion

In this paper, we have designed an identity based deterministic signature scheme that has tight reduction to GDH problem. Our scheme is completely different from all the existing schemes. The PKG uses a novel PKI based signature scheme to generate the private keys for users and the PKI based signature scheme itself is of independent interest. We have also proposed a novel scheme for the users to generate signed documents. Both the schemes allow tight reductions and this results in substantially smaller keys and signatures than the ones of the only other identity based deterministic signature scheme by Javier Herranz [9]. Improving the tightness by avoiding the abort scenario during the extract phase in the deterministic identity based signature scheme is considered to be an open challenge and an interesting direction to proceed.

Acknowledgement. We would like to thank the anonymous referees of IWSEC-2011 for their insightful remarks, which helped in improving our paper greatly. Special thanks goes to Prof. Willy Susilo for shephard heading our paper and helping in improving the paper.

References

1. Barreto, P.S.L.M., Libert, B., McCullagh, N., Quisquater, J.-J.: Efficient and provably-secure identity-based signatures and signcryption from bilinear maps. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 515–532. Springer, Heidelberg (2005)
2. Bellare, M., Namprempre, C., Neven, G.: Security proofs for identity-based identification and signature schemes. Journal of Cryptology 22(1), 1–61 (2009)
3. Bellare, M., Rogaway, P.: The exact security of digital signatures - how to sign with RSA and rabin. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996)
4. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. Journal of Cryptology 17(4), 297–319 (2004)
5. Cha, J.C., Cheon, J.H.: An identity-based signature from gap diffie-hellman groups. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 18–30. Springer, Heidelberg (2002)
6. Galindo, D., Garcia, F.D.: A schnorr-like lightweight identity-based signature scheme. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 135–148. Springer, Heidelberg (2009)

7. Goh, E.-J., Jarecki, S.: A signature scheme as secure as the diffie-hellman problem. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 401–415. Springer, Heidelberg (2003)
8. Goh, E.-J., Jarecki, S., Katz, J., Wang, N.: Efficient signature schemes with tight reductions to the diffie-hellman problems. Journal of Cryptology 20(4), 493–514 (2007)
9. Herranz, J.: Deterministic identity-based signatures for partial aggregation. The Computer Journal 49(3), 322–330 (2006)
10. Micali, S., Reyzin, L.: Improving the exact security of digital signature schemes. Journal of Cryptology 15(1), 1–18 (2002)
11. Okamoto, T., Pointcheval, D.: The gap-problems: A new class of problems for the security of cryptographic schemes. In: Kim, K.-c. (ed.) PKC 2001. LNCS, vol. 1992, pp. 104–118. Springer, Heidelberg (2001)
12. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (1996)
13. Sakai, R., Ohgishi, K., Kasahara, M.: Cryptosystems based on pairing. In: The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan, pp. 135–148 (January 2000)
14. Schnorr, C.-P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (1990)
15. 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)
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)

Nitro: Hardware-Based System Call Tracing for Virtual Machines

Jonas Pfoh, Christian Schneider, and Claudia Eckert

Technische Universität München, Munich, Germany
{pfoh,schneidc,eckertc}@in.tum.de

Abstract. Virtual machine introspection (VMI) describes the method of monitoring and analyzing the state of a virtual machine from the hypervisor level. This lends itself well to security applications, though the hardware virtualization support from Intel and AMD was not designed with VMI in mind. This results in many challenges for developers of hardware-supported VMI systems. This paper describes the design and implementation of our prototype framework, *Nitro*, for system call tracing and monitoring. Since Nitro is a purely VMI-based system, it remains isolated from attacks originating within the guest operating system and is not directly visible from within the guest. Nitro is extremely flexible as it supports all three system call mechanisms provided by the Intel x86 architecture and has been proven to work in Windows, Linux, 32-bit, and 64-bit environments. The high performance of our system allows for real-time capturing and dissemination of data without hindering usability. This is supported by extensive testing with various guest operating systems. In addition, Nitro is resistant to circumvention attempts due to a construction called hardware rooting. Finally, Nitro surpasses similar systems in both performance and functionality.

1 Introduction

Virtual machine introspection (VMI) lends itself very well to security applications [5]. This is, in part, due to the fact that security mechanisms running within the hypervisor are isolated from attacks that occur within a virtual machine (VM) and that the hypervisor maintains a complete and untainted view of a VM's system state.

In order to leverage the full potential that VMI provides, identifying and isolating the relevant guest operating system (OS) state information becomes crucial. This process requires some semantic knowledge about the guest and is referred to as the *semantic gap* issue [3]. Bridging this semantic gap has been classified into three fundamental *view generation patterns* [12]. One of these patterns relies on knowledge of the hardware architecture to derive semantic information about the guest OS. Making use of the hardware architecture allows one to construct mechanisms that are resistant to evasion attempts through a method called hardware rooting [13]. This makes hardware-based information extraction particularly interesting for security approaches that are intended to detect malicious activity within a monitored VM.

T. Iwata and M. Nishigaki (Eds.): IWSEC 2011, LNCS 7038, pp. 96–112, 2011.

One promising way of detecting such malicious activity is to monitor system calls. System calls facilitate communication between the kernel and user space within an OS and are interesting from a security perspective. System call traces may be used to classify the actions of a process as benign or malicious with machine learning approaches [9,6,14]. In addition, prevalent sandboxing environments, such as CWSandbox [7], Anubis [2], or the Norman Sandbox product line, incorporate system call or API monitoring to create their reports. Finally, particular system calls of interest are monitored in live forensic applications, such as those found in honeypot environments.

In this paper, we describe the implementation of our prototype VMI framework, *Nitro*, for hardware-based system call tracing and monitoring. Due to the properties of VMI, it is isolated from malicious activities within the VM and remains hidden from the guest OS. To our knowledge, Nitro is the first VMI-based system that supports all three system call mechanisms provided by the Intel x86 architecture and has been proven to work for Windows, Linux, 32-bit, and 64-bit guests. Moreover, this framework is flexible enough to feasibly support almost any OS built upon the x86 architecture. Capturing and disseminating data is done in real-time without hindering usability of the guest, as our performance tests show. Finally, Nitro is resistant to attempts at evasion due to hardware anchors. We will discuss the foundations of this approach, its implementation, and its properties throughout this work.

The remainder of this paper is organized as follows: We start by presenting some related work in Section 2. We go on to introduce the requirements for desirable properties of a VMI system in Section 3. The implementation of Nitro is detailed in Section 4, followed by a discussion of how it meets the aforementioned requirements in Section 5. In Section 6, we explain our performance evaluation and present the results, which we compare to a similar system called Ether [4] in Section 7. Finally, we draw our conclusions in Section 8.

2 Related Work

At the time of this writing, to our knowledge, there are three other systems that make use of virtualization extensions to implement systems that are capable of producing system call traces for security applications. The first system, Lares [11], was the pioneer in this area introducing a mechanism for creating arbitrary hooks within a Windows guest OS. Lares was developed on the Xen hypervisor and required drivers to be installed within the Windows guest to facilitate hooking. Since our system does not require any guest OS support at all, it achieves a much higher level of portability and robustness, as we will show. Thus, a detailed comparison between Lares and Nitro would not be very meaningful and is not provided in this paper.

In addition to Lares, the Ether system [4] provides the capability to produce system call traces. This system is also built upon the Xen hypervisor and takes a similar approach to that of Nitro. For a detailed comparison, please refer to Section 7.

Finally, HyperSleuth [10] also provides the ability to trace system calls, though the authors indicate that the "approach we use to trace system calls is thus inspired by Ether". For this reason we forgo a detailed comparison and direct the readers attention to our comparison between Nitro and Ether.

3 Properties

In Section 1 we outline the key properties of Nitro. These properties, a description, and the requirements to achieve each property are discussed within this section in a general manner. How exactly Nitro meets these requirements follows later in Section 5.

Guest OS Portability. Guest OS portability refers to a property that allows the same VMI mechanism to work for various guest OSs without major changes. Ideally, a guest OS portable VMI mechanism would work on any guest OS with no change, however we tolerate some minor configuration changes to the VMI mechanism, as long as the basic mechanics of that approach is shared among all guests.

In order to achieve guest OS portability, the underlying VMI mechanism may not rely on knowledge of the guest OS itself, but rather on knowledge of the virtual hardware specifications. For example, Jones et al. make use of the CR3 register in order to track processes [8]. How this register is to be used within the memory management unit (MMU) is specified by the x86 architecture, and all OSs running on this hardware and using the MMU must conform to these specifications. Thus, this basic method can be used to track processes in various guest OSs without change as long as the OSs support virtual memory.

Evasion-Resistance. An evasion-resistant mechanism is a mechanism which is impossible for an attacker to circumvent when *correctly implemented* and deployed in an *ideal system*. We define a correctly implemented mechanism as a mechanism that perfectly enforces the policy that it was designed to enforce with no flaws or errors. In the same manner, we define an ideal system as a system that perfectly implements its design and contains no flaws or errors. Since we know that these ideal properties are impractical, it may be possible to circumvent the mechanism if and only if such a flaw is found and exploited by a malicious entity. This is why we refer to this property as "evasion-resistant" rather than "evasion-proof". We begin this discussion by describing how a mechanism may be rooted in hardware.

In order to interpret the low-level binary state information of a virtual machine, the hypervisor must incorporate knowledge of the hardware architecture or the guest OS to bridge the semantic gap. As Pfoh et al. argue [12], an approach that relies on guest OS knowledge alone might be circumvented by attacks that change the guest OS architecture itself. For example, the manner in which the guest OS uses a particular data-structure may be manipulated by a malicious entity. This stems from the fact that this knowledge of the guest OS is in no way

bound to the running OS kernel. The fact that such attacks against VMI mechanisms have been successfully implemented recently [1] shows, that this threat is not of pure theoretical nature.

If, in contrast, the VMI mechanism bases its knowledge on information about the virtual hardware architecture, these attacks cannot be applied. The guest OS and all software running on it, including any malware, must play by the rules of the virtual hardware. An attacker has no means of changing these rules. Thus, this knowledge of the hardware specifications is bound to the hardware architecture. For example, if the hardware architecture specifies that a control register holds the address of a data-structure, there is nothing a malicious entity can do to circumvent this as the hardware will expect this to be the case in order to run correctly.

This argument can be expanded further to also cover other parts of state information as follows: If we can start at a feature of the virtual hardware specification (e. g., a register) and, from there, follow references in memory, thus building a chain to a critical data-structure, a malicious entity cannot modify that data-structure unnoticed. Figure 1 depicts such a chain. We will therefore refer to a portion of state information as being *rooted in hardware* if such a chain can be built [13].

Having introduced hardware rooting, we now describe the two requirements for an evasion-resistant VMI mechanism. First, the monitored or protected portions of the VM's state must be rooted in the virtual hardware, as described above. Second, each involved piece of VM state along the described reference chain must be protected such that it cannot be manipulated in violation of policy or that any change to such a piece of VM state is ignored by the guest OS. If both of these requirements are met the mechanism is evasion-resistant.

4 Implementation

This section describes the steps we took in implementing our prototype. Nitro is based upon the Linux Kernel Virtual Machine (KVM). KVM is split into two portions, namely a user application that is built upon QEMU and a set of Linux kernel modules.

The user application portion of KVM provides the *QEMU monitor* which is a shell-like interface to the hypervisor. It provides general control over the VM. For example, it is possible to pause and resume the VM as well as to read out CPU registers using the monitor. We modified KVM by adding new commands to the monitor to control Nitro's features. That is, all Nitro commands are input via this monitor.

These commands are then sent to the kernel module portion of KVM through an I/O control interface. The majority of Nitro is implemented in these kernel modules. Finally, the output is realized by making use of the *proc* filesystem. That is, Nitro creates a node in the *proc* filesystem and obtaining its output is as simple as reading from a file.

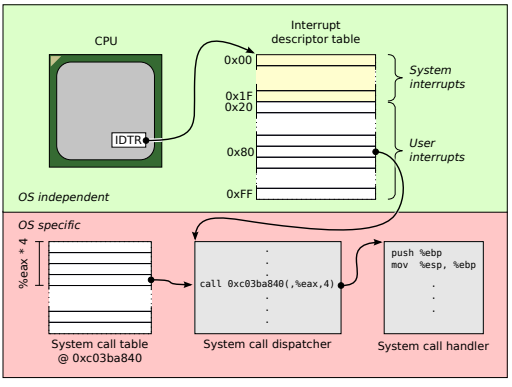


Fig. 1. The relationship between the Interrupt Descriptor Table Register (IDTR), the Interrupt Descriptor Table (IDT), and the system call dispatcher shows that the system call dispatcher is rooted in the IDTR through a chain that includes the IDT

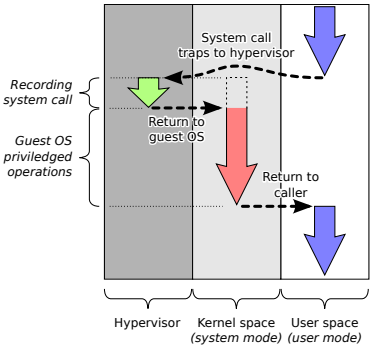


Fig. 2. Control flow of a system call that traps to the hypervisor

4.1 VMI Mechanisms for Trapping System Calls

In some cases the virtualization extensions provided by hardware manufacturers support trapping the specific event one is interested in, which makes the effort straightforward. However, it is often the case, especially for security mechanisms, that the hardware extensions do not support trapping the desired event. In these instances, we must indirectly induce a trap to the hypervisor. Finding these indirect methods for trapping desired events is often a challenge.

As it turns out, trapping to the hypervisor on the event of a system call is not supported on the popular Intel IA-32 (i. e., x86) and Intel 64 (formerly EM64T) architectures. In this case, we must find a way to indirectly cause the trap as discussed above. We do this by forcing system interrupts (e. g., page faults, general protection faults, etc) for which trapping is supported by the Intel Virtualization Extensions (VT-x). Hence, we have effectively created a mechanism for trapping system calls even though the hardware extensions do not natively support this. The resulting control flow is depicted in Figure 2. Since the three system call mechanisms are quite different in their nature, a unique trapping mechanism must be designed for each. These trapping mechanisms and their implementations are described below.

Interrupt-Based System Calls. System calls may be implemented as a user-defined interrupt. The x86 architecture handles interrupts through an Interrupt Descriptor Table (IDT). This IDT may have as many as 256 entries, each of which is 8 bytes long. The exact size of the IDT is stored in the IDTR along with the address at which the IDT resides in system memory. When an interrupt occurs, the hardware consults the IDT via the IDTR to determine the location for the appropriate handler and continues execution there as shown in Figure 1.

Intel's VT-x extensions allow one to trap system interrupts (interrupts 0 to 31) to the hypervisor, but they do not provide a mechanism for trapping user interrupts (interrupt 32 and above) which may be used for system calls.[1] This means that we must design a way to cause this user interrupt to generate a system interrupt.

We can achieve this by virtualizing the IDT, that is, we copy out the guest's IDT into the hypervisor. We must then manipulate the IDTR and trap all write accesses to it, thus disallowing any further manipulation. As the IDT size value stored in the IDTR is added to the base address to get the offset of the last valid byte of the IDT, we can set this size to $32 \cdot 8 - 1 = 255$. This leaves all system interrupts unaffected, however all attempts at invoking a user interrupt (i. e., interrupts greater than 31) will result in a general protection fault as the bounds of the IDT will have been exceeded. The advantage of this approach is that the IDT remains unaffected in memory, but is effectively ignored for user interrupts.

The next step is to trap all general protection faults to the hypervisor, which the virtualization extensions support natively. However, we must still determine the difference between general protection faults that we generated and those that occur naturally[2]. This can be done by inspecting the current instruction and determining whether or not it is the `int` instruction and whether the interrupt number is greater than 31.

If we identify the exception as being natural, we inject this exception into the guest and allow it to continue. However, if we recognize the exception to be caused by a user interrupt, we look at the interrupt number to determine whether we have trapped a system call. If this is the case, we collect data according to the rules specified for Nitro's data collection engine (see Section 4.3). In either case, the `int` instruction must be emulated using the IDT that we copied out of the guest and hand control back to the guest OS.

SYSCALL-Based System Calls. System calls may also be implemented using the `SYSCALL` instruction and its analogue counterpart `SYSRET`. Both of these rely on a set of MSRs, namely STAR_MSR, CSTAR_MSR, and LSTAR_MSR. Exactly which of these registers is used depends on whether the guest OS is running in legacy, long, or compatibility mode. Additionally, this mechanism can effectively be turned on and off by setting and unsetting the SCE flag in the Extended Feature Enable Register (EFER). Making use of either `SYSCALL` or `SYSRET` with the SCE flag not set results in an invalid opcode exception.

Forcing this mechanism to cause a system interrupt is then a matter of unsetting the SCE flag and setting the hypervisor to trap all invalid opcode exceptions, which is natively supported by the virtualization extensions. Once control has passed to the hypervisor, we must once again differentiate between natural exceptions and those caused by our introspection. This is achieved by looking at the violating instruction and if this instruction is *not* either `SYSCALL` or `SYSRET`,

[1] In contrast, AMD's SVM virtualization extensions do provide a mechanism for trapping user interrupts.

[2] We refer to exceptions that are not caused by our changes as *natural* exceptions.

we inject an invalid opcode exception into the guest OS and return control to it. However, if the violating instruction is, in fact, `SYSCALL`, Nitro collects the desired information, emulates this instruction, and returns control back to the guest OS.

In addition to emulating the `SYSCALL` instruction, Nitro must be capable of handling exceptions caused by the `SYSRET` instruction and emulating this instruction as well. This is due to the fact that the changes made to the EFER affect the `SYSRET` instruction in the same manner that they affect the `SYSCALL` instruction. Thus, use of the `SYSRET` instruction will also cause an invalid opcode exception and must be handled accordingly. In doing so, Nitro is also able to collect the return value of the invoked system call if the application requires this information.

SYSENTER-Based System Calls. Similar to `SYSCALL` and `SYSRET`, the `SYSENTER` and `SYSEXIT` pair of instructions also rely on a set of MSRs, namely SYSENTER_CS_MSR, SYSENTER_ESP_MSR, and SYSENTER_EIP_MSR. The values in each of these MSRs are copied into specific system registers upon a call to `SYSENTER`. Specifically and most interesting for the development of Nitro, the value of the SYSENTER_CS_MSR is copied into the CS register when `SYSENTER` is executed and an attempt to load the CS register with a null value results in a general protection exception. Hence, causing a system interrupt is a matter of saving the current value of the SYSENTER_CS_MSR register in the hypervisor and loading it with a null value. This will cause each `SYSENTER` operation to attempt to load a null value into the CS register, thus causing a system interrupt that the hypervisor can trap.

Once the hypervisor has trapped a general protection exception, differentiating between natural and forced exceptions is once more a matter of checking the current instruction at the time of the exception. If we come across a natural exception, as with the previous system call mechanisms, we inject the exception into the guest OS and allow it to continue. In the case that we come across general protection exception and the current instruction is `SYSENTER`, we collect the relevant data, emulate the instruction using the saved value of the SYSENTER_CS_MSR, and return control to the guest OS.

As with the `SYSCALL`/`SYSRET`-based system call mechanism, the change that we make to the guest in order to induce a system interrupt also affects the `SYSEXIT` instruction. Consequently, we must also emulate this instruction and with that, get a chance to easily extract the return value of the system call invoked.

4.2 Process Identification

It is always important to be able to determine which process produced a system call. This requires that we collect information which is unique to a process each time a system call is interrupted. Nitro collects the value of the CR3 register along with the value of the first valid entry in the corresponding top-level page directory. This allows us to identify a process due to the fact that the value in

the CR3 register (i. e., the address of the top-level page directory) is unique for a single process. In order to handle the case in which a newly created process receives a top-level page directory which is located at the same location of a previously destroyed process's top-level page directory, we also consider the first valid entry in the corresponding top-level page directory in order to create a truly unique identifier.

4.3 Collection of System Call Data

In our experience, different applications for system call traces depend on varying amounts of information. In some cases a simple sequence of system call numbers without arguments may suffice, while other scenarios may require detailed information including register values, stack-based arguments, and return values from a small subset of system calls. As we cannot foresee every guest OS type and possible application of system call tracing, Nitro does not deliver a fixed set of data per system call. Instead, it allows the user to define flexible rules to control the data collection during system call tracing in a fine-grained manner.

For example, the user can specify where exactly the guest OS stores the system call number (generally in the EAX register). Nitro can then extract this system call number along with a process identifier as described in Section 4.2. This information is often enough for certain machine learning techniques used for detection of malware or malicious behavior in processes [9,6].

In other instances, system call arguments or even dereferenced memory variables pointed to by arguments are crucial. To meet these requirements, Nitro's rules are expressive enough to account for both stack-based as well as register-based argument passing in addition to printing register values directly or dereferencing them. The syntax of such a rule takes the following format:

```
add_scmon_rule CONDITION_REG CONDITION_VAL ACTION_REG OFFSET ACTION,
```

where `CONDITION_REG` contains the name of the register that should be tested to determine whether information should be collected, `CONDITION_VAL` contains the value the `CONDITION_REG` should contain in order for further information to be collected, `ACTION_REG` contains the name of the register that contains the base value we are interested in, `OFFSET` contains the offset (positive or negative) from the `ACTION_REG` for the data we are interested in when collecting dereferenced values, and `ACTION` defines whether the `ACTION_REG` should be dereferenced as well as the format the output should take. This may result in printing or dereferencing the data as hexadecimal, integer, unsigned integer, or string. We provide a description of the rules in Backus-Naur Form in Appendix A. As an example, it is easy to specify a rule that dereferences and outputs the string being written every time a user process makes use of the write system call within a Linux guest. This rule would look as follows:

```
add_scmon_rule rax 4 rcx 0 derefstr
```

This rules-based method of requesting information makes Nitro very flexible and contributes to its OS agnostic nature.

Keeping the design goals for Nitro in mind, we collect only information whose location and format is defined by the hardware specifications or for which a rule is specified. However, the flexible design of Nitro allows an easy incorporation of guest OS specific knowledge in order to collect additional information about the calling process. We have successfully combined Nitro with other projects within our research group to include information such as process and user IDs into the output. However, we keep these projects separate in order to keep Nitro as simple and flexible as possible. This allows Nitro to be applicable in a greater range of applications. When combined with a memory analysis tool we call InSight, we are able to produce output as shown in Figure 3. This provides additional guest OS specific information, such as the type of descriptor being written to, while allowing Nitro to remain applicable of a wide range of guest OSs.

```
Jun 20 17:58:20: sys_write: unsigned int fd, const char
      __user *buf, size_t count
  fd: unsigned int: 0x3 → (socket) → [...]: (SOCK_STREAM)
      flags: ()
  buf: const char __user*: 0x7FFF702FF320 →
    buffer content hex (of size 107):
      47 45 54 20 2f 20 48 54 54 50 2f 31 2e 30 da 55 73 65
      72 2d 41 67 65 6e 74 3a 20 57 67 65 74 2f 31 2e 31 32
      20 28 6c 69 6e 75 78 2d 67 6e 75 29 da 41 63 63 65 70
      74 3a 20 2a 2f 2a da 48 6f 73 74 3a 20 67 6f 6f 67 6c
      65 2e 64 65 da 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 4b
      65 65 70 2d 41 6c 69 76 65 da da
    buffer content string:
      GET / HTTP/1.0
      User-Agent: Wget/1.12 (linux-gnu)
      Accept: */*
      Host: google.de
      Connection: Keep-Alive
  count: size_t: 0x6B
```

Fig. 3. Output of Nitro when combined with a memory analysis tool

5 Discussion and Evaluation

In most VMI-based mechanisms, performance overhead becomes a concern. For this reason, it is important to keep unnecessary traps to the hypervisor at an absolute minimum. In all the mechanisms described in Section 4, we make use of a system interrupt to facilitate the trap to the hypervisor due to the fact that system calls are not natively trappable.

For our implementation, we looked at the individual system call mechanisms and determined all system interrupts that each system call mechanism could

be made to produce and how to induce them. We then inspected all feasible solutions and considered them in terms of their impact on performance. For example, all three system call mechanisms can be made to produce a page-fault, however we passed on this for two reasons. First, page faults occur often (relative to other system interrupts) in regular system activity. This means that each page fault would result in a costly trap to the hypervisor to distinguish between forced and "natural" page faults, most of which would be natural. Second, this would essentially nullify any performance improvement that comes from using Extended Page Tables or Nested Page Tables.[3] In general, we strove for a system interrupt that occurs *infrequently* during normal operation and one whose use would not counteract performance enhancements in other parts of the system. This is how we came to the implementation and made our system viable for live collection.

Revisited: Guest OS Portability. Nitro is guest OS portable due to the fact that all three mechanisms described in Section 4 make sole use of hardware knowledge. This allows the mechanisms to work for any guest OS that is compatible with the x86 or the Intel 64 architecture. The IDTR and IDT as well as all the involved MSRs and their uses are specified by the hardware architecture and must be used in the way specified. That is, any guest OS must use these hardware mechanisms according to the specifications regardless of the guest OS.[4]

One potential hindrance for guest OS portability is the fact that how information is passed between kernel and user space is left to the OS designer. For example, some OSs are designed such that system call arguments are passed in registers, while others pass arguments on the stack. Nitro addresses this by providing the flexible set of rules described in Section 4.3. That is, the user can control which data is collected by specifying rules at run-time. This allows Nitro to be used across all guest OSs by simply changing the rule-set.

Revisited: Evasion-Resistance. We make the reasonable assumption that the hypervisor itself is secure. In addition, any components that reside within the hypervisor are safe from attacks originating from within a guest OS due to the hypervisor's isolation property.

While we have the aforementioned assumption with regard to the hypervisor itself, this alone is not enough. This is due to the fact that our VMI mechanisms make changes to the state of the guest VM. These state changes are clearly not protected by the isolation property as they take place within the guest OS itself. A malicious entity might simply revert the changes we made to the system state to circumvent our security mechanisms. For this reason we took special care to make sure that Nitro is evasion-resistant.

As stated in Section 3, evasion-resistance requires that the VMI mechanism is rooted in hardware and that each involved piece of VM state is protected

[3] These are hardware extensions implemented by Intel and AMD, respectively, to counter the performance degradation caused by using shadow page tables.

[4] Technically, an OS could also implement system calls entirely in software, but would then lack the privilege level feature of the CPU, leading to an insecure OS kernel.

against manipulation. Since the VMI mechanisms for the fast system call mechanisms and the interrupt-based mechanism differ slightly in this regard, they are discussed separately in the following.

In order to achieve an evasion-resistant VMI mechanism for fast system calls, it is rooted in either the SYSENTER_CS_MSR (`SYSENTER`-based) or the EFER (`SYSCALL`-based) as discussed in Section 3. In addition, the VMI mechanism may protect each of these registers from manipulation, which is directly supported by the virtualization extensions provided. This is enough to achieve evasion-resistance because Nitro's manipulation of the system call mechanisms are constrained to these registers. That is, due to the changes we made to the system, all fast system calls are trapped to the hypervisor and there is no way for a malicious entity to circumvent this without making changes to exactly those parts of the system that Nitro protects. Hence, this approach is both rooted in hardware and protects all involved pieces of VM state, resulting in an evasion-resistant mechanism.

Making the interrupt-based system call traps evasion-resistant is similar to the method described for fast system calls with one additional step. The mechanism is rooted in the IDTR and this register is protected against malicious manipulation by the hypervisor. In addition, a shadow copy of the original IDT is created within the hypervisor at boot time. This already is enough to achieve evasion-resistance as the changes to the guest OS are limited to this register. In addition, only the shadow IDT is referred to for each user interrupt. That is, any changes to the IDT within the guest OS do not affect the ability to trap user interrupts. In order to hinder this, a malicious entity would have to manipulate the IDTR directly or the shadow IDT within the hypervisor, both of which are protected with the help of the virtualization extensions.

6 Performance Testing

In this section, we present our general performance testing results for all guest OSs tested, which include: Windows XP SP2 (32-bit), Ubuntu Linux 9.04 Server (32-bit), and Ubuntu Linux 9.04 Server (64-bit). The tests were performed on an Intel Core 2 Duo processor at 2.4 GHz with 2 GB of RAM. We used a Debian Lenny (5.0.6 64-bit) host system for all tests. Finally, we used KVM 0.12.4 to act as the hypervisor.

These tests were performed by running benchmarks on the guest OSs once with Nitro disabled, then once with Nitro enabled and comparing these results. While the results themselves are of interest, we focus primarily on the amount of degradation observed because the degradation is a strong indicator for the overhead incurred by our system call tracing.

Throughout these tests, we made sure to test each mechanism. That is, we present tests that measure the performance of our mechanism for `SYSENTER`-based, `SYSCALL`-based, and interrupt-based system calls. It is important to note that the mechanism implemented at the time of this testing for the interrupt-based system calls function by redirecting the interrupt to a new gate descriptor

Table 1. Windows XP SP2 (32-bit) performance comparison between KVM/Nitro (`SYSENTER`-based) and Xen/Ether

Benchmark	*Xen/Ether*			*KVM/Nitro (SYSENTER)*		
	Tracing disabled	*Tracing enabled*	*Degradation*	*Tracing disabled*	*Tracing enabled*	*Degradation*
HTML Render [pg/s]	3.277	0.598	81.74%	2.826	**2.034**	**28.04%**
File Decryption [MB/s]	65.561	64.561	1.53%	64.697	**64.654**	**0.07%**
HDD [MB/s]	45.198	7.215	84.04%	46.545	**9.726**	**79.10%**
Text Edit [pg/s]	89.066	17.246	80.64%	84.743	**40.032**	**52.76%**
Image Decompression [MPix/s]	33.856	32.951	2.67%	33.364	**33.103**	**0.78%**
File Compression [MB/s]	2.737	2.677	2.19%	2.744	**2.741**	**0.10%**
File Encryption [MB/s]	15.821	15.515	1.94%	15.853	**15.826**	**0.17%**
Virus Scan [MB/s]	333.988	85.307	74.46%	314.118	**155.718**	**50.43%**
Mem. Latency [MemAcc/s]	6.735	3.580	46.84%	6.231	**3.782**	**39.30%**
PerformanceTest [score]	586.500	383.020	34.69%	628.700	**540.260**	**14.07%**

within the IDT, rather than emulating the `int` instruction. The following subsections present our results. All scores and times presented are a mean over three scores or runs.

Windows XP. In testing a Windows XP guest OS we made use of two commercial benchmarking products, namely PCMark05 from Futuremark and PerformanceTest from PassMark.[5] These tools perform various CPU, memory, disk drive, and graphics tests. Each make heavy use of system calls as is evidenced by the output of Nitro. PCMark05 returns a value for each performed test, while PassMark outputs a single combined score.

The standard deviation of all tests were negligible, except for the 'HDD' and 'Virus Scan' tests where we observed standard deviations of 6.3 and 61.0, respectively. We hypothesize that this is due to the fact that these are both disk I/O intensive tests. In any case, we present these results for the sake of completeness, however, due to their high deviation from the mean, we do not draw any conclusions from these values.

For these tests we were able to modify the virtual hardware such that the guest OS determined that the `SYSENTER` and `SYSEXIT` instructions were not available and thus resorted to the interrupt-based mechanism for system calls. This allowed us to test the performance of both `SYSENTER`-based (Table 1) and interrupt-based (Table 2) system call mechanisms.

It is interesting to note that across both sets of tests the degradation varies greatly from benchmark to benchmark, however the benchmarks with the lowest degradation ($< 10\%$) all perform some sort of compression, decompression, encryption, or decryption. Such functions are highly arithmetic and perform

[5] Available from `http://www.futuremark.com/products/pcmark05/` and `http://www.passmark.com/products/pt.htm`, respectively.

Table 2. Windows XP SP2 (32-bit) performance results with interrupt-based system calls on KVM/Nitro

	KVM/Nitro (interrupt)		
Benchmark	*Tracing disabled*	*Tracing enabled*	*Degradation*
HTML Render [pg/s]	3.05	2.28	25.12%
File Decryption [MB/s]	65.87	65.57	0.45%
HDD [MB/s]	46.06	9.13	80.17%
Text Edit [pg/s]	83.61	56.44	32.49%
Image Decomp. [MPix/s]	33.75	32.24	4.46%
File Compression [MB/s]	2.81	2.64	6.07%
File Encryption [MB/s]	16.10	15.23	5.42%
Virus Scan [MB/s]	325.34	102.63	68.45%
Mem. Latency [MemAcc/s]	6.23	4.67	25.00%
PerformanceTest [score]	623.16	557.66	10.51%

Table 3. Ubuntu Linux 9.04 Server performance results on KVM/Nitro

	Apache Compile Results		
Linux Guest OS	*Tracing disabled*	*Tracing enabled*	*Degradation*
32-bit Interrupt-based	168.989s	195.254s	15.54%
32-bit `SYSENTER`-based	167.916s	212.492s	26.55%
64-bit `SYSCALL`-based	179.166s	232.640s	29.85%

relatively few system calls since these arithmetic operations do not require OS support. We believe that this is the reason for the variation in degradation across the benchmarks. While the PCMark05 tests (the first nine benchmarks in Tables 1 and 2) are great for identifying to which degree an operation is affected by overhead in the system call mechanism, we feel that the results delivered by PerformanceTest give a better overall impression of the performance degradation in the guest OS as a whole.

Ubuntu Linux. For testing all Linux guest OSs we created a script that makes use of the 'time' command in Linux. Using this utility we measured the compile time of the Apache web server 2.2.16. The time utility makes use of the hardware clock and we verified beforehand that the hardware clock within the VM is consistent with the host system's hardware clock. We used this as a benchmark as it is resource intensive enough to show performance degradation and makes extensive use of system calls as is evidenced by the output of Nitro.

Presented in Table 3 are the test results when performed on a Ubuntu Linux 9.04 Server (32-bit) guest OS. As with our testing for the Windows XP SP2 guest, we manipulated the virtual hardware in order to be able to report results for interrupt-based and `SYSENTER`-based system call mechanisms. Considering these

results, we notice that the interrupt-based guest OS incurs less degradation than the `SYSENTER`-based guest. This is due to the fact that the mechanism in place for trapping the `SYSENTER` instruction requires emulating that instruction, while the mechanism in place for trapping the `int` instruction does not.

The testing process we used for a 64-bit Linux guest OS is identical to the processes we used for the 32-bit Linux guest with the obvious exception that we use Ubuntu Linux 9.04 Server (64-bit). One noteworthy difference between the 32-bit and 64-bit version of this operating system is that the 64-bit version makes use of the `SYSCALL`-based system call mechanism, making this the only test case that makes use of this instruction. These results are also presented in Table 3. Comparing the degradation of this guest to its 32-bit counterpart reveals that this OS incurred the most degradation among the Linux guests.

7 Comparison

We chose to compare our system to the Ether system [4] because Ether is the only other system to our knowledge (aside from HyperSleuth [10], which bases its system call tracing on Ether's approach) that supports some forms of system call tracing using VMI without having to install drivers or modules in the guest OS. Both in function and performance, Nitro surpasses Ether with regard to system call tracing and monitoring. In this section we discuss these differences in further detail.

Functional Differences. The largest functional difference between Ether and Nitro is the hypervisor that they are built upon. Ether builds upon the Xen hypervisor, while Nitro builds upon KVM. Nitro and Ether's system call tracing mechanism are similar in respect to the output they provide, though Ether's output is guest OS specific. That is, the output is Windows specific. Despite this, we tried Ether on further guest OSs in order to determine whether the underlying mechanism may be used for other guest OSs.

We tested both systems for functionality on Windows XP SP2 (32-bit), Ubuntu Linux 9.04 Server (32-bit), and Ubuntu Linux 9.04 Server (64-bit). Nitro proved functional on all tested platforms, while Ether proved 100% functional only on Windows XP SP2. While we expected Ether to be functional on Ubuntu Linux 9.04 Server (32-bit) because this OS uses the same system call mechanism as Windows XP SP2, the guest OS became very unstable and was not usable from a user's perspective when system call tracing was enabled. Finally, Ether was unable to provide any system call information for Ubuntu Linux 9.04 Server (64-bit) although the guest OS continued to run without issue. We believe that this is due to the fact that Ether fails to consider the `SYSCALL/SYSRET` mechanism for system calls completely. We see this as a major detractor as this limits the number of guest OSs for which system call tracing or monitoring will work.

Performance Comparison. We performed all presented tests on the same hardware and host OS as described in Section 6. In addition, we used the same Windows XP SP2 (32-bit) image for all tests to ensure the consistency of the

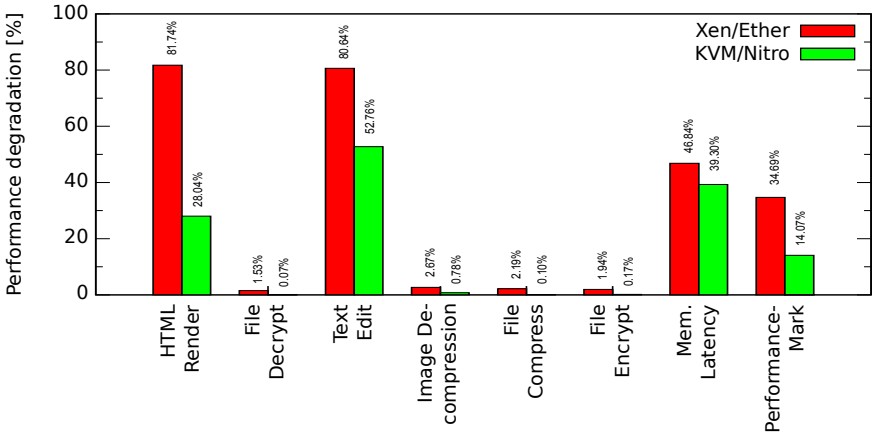


Fig. 4. Performance degradation of Xen/Ether and KVM/Nitro when tracing a Windows XP SP2 guest OS (ref. Table 1)

guest OS. Finally, for Ether we used the recommended Xen 3.1.0 as hypervisor. It is also important to note that we chose the specific benchmarks out of the PCMark05 suite due to the fact that these are the same benchmarks that Ether's authors used when testing their system originally. We felt it was important to include the same set of tests in the interest of equity.

Nitro and Ether are based on two different hypervisors, namely KVM and Xen. As our intentions were not to compare the performance between KVM and Xen but to compare the efficiency of the different implementations for system call tracing, we look at the relative performance degradation between the unmodified version of KVM and Nitro and compare this to the relative performance degradation between the unmodified version of Xen and Ether. This way we can measure the performance overhead incurred by each VMI implementation and do a fair comparison of Nitro and Ether.

As Ether only worked correctly for a Windows XP SP2 guest OS, we were only able to compare the performance of Ether and Nitro on this guest. These results are presented in Table 1 and Figure 4. Again, we do not draw any conclusions from the 'HDD' and 'Virus Scan' results due to their high observed standard deviation (this high standard deviation was observed for Xen/Ether as well as KVM/Nitro), however the results are present in Table 1 for the curious reader. Despite this, we see that Nitro outperforms Ether both on the absolute scores and the amount of degradation in all tests performed. In the benchmarks that focus on arithmetic operations (i. e., use relatively fewer system calls), for example compression and encryption tests, Nitro outperforms Ether only nominally. However in the case of HTML rendering, text editing, and memory latency, Ether's degradation is between 5 and 54 percentage points greater than that of Nitro. As mentioned in Section 6, while the PCMark05 benchmarks nicely reveal which specific operations incur the greatest degradation, the PerformanceTest benchmark is a better indicator of the overall degradation of the system. We see

that with this benchmark Ether's degradation is over 20 percentage points more than that of Nitro.

We hypothesize that Ether's greater degradation is primarily due to the fact that Ether forces a page fault interrupt to perform system call tracing. This adds additional overhead to a part of the hypervisor which is already responsible for incurring a large performance overhead. Additionally, this design decision effectively counteracts any benefits one might have from using Ether with a hypervisor which makes use of Extended Page Tables.

8 Conclusion

We have shown that Nitro is a powerful and flexible tool for system call tracing and monitoring. It supports all three system call mechanisms provided by Intel's x86 architecture for both 32-bit and 64-bit environments. In fact, we have successfully collected system call traces with Nitro for Windows, Linux, 32-bit, and 64-bit guests and we are confident that it will perform equally well for a variety of additional guest OSs. Further, the proven performance of our implementation allows the collection and dissemination of data in real-time. Finally, all of the VMI mechanisms presented have been shown to be evasion-resistant. That is, these mechanisms cannot be manipulated in a way which allows a malicious entity to circumvent system call tracing or monitoring. Its flexible and secure nature allows Nitro to be used effectively in a variety of applications, such as machine learning approaches to malware detection, honeypot monitoring, as well as sandboxing environments.

As Nitro builds upon KVM which is licensed under the GPLv2, we release the source code of our system under the same license. The source code is available at `http://code.google.com/p/nitro-kvm/`. We hope that this tool is useful for the community and will help to further security research.

Acknowledgements. The authors would like to thank Cornelius Diekmann for his contribution to this work by combining Nitro with other tools to help show its potential.

References

1. Bahram, S., Jiang, X., Wang, Z., Grace, M., Li, J., Xu, D.: DKSM: Subverting virtual machine introspection for fun and profit. In: Proc. of 29th IEEE Int. Symp. on Reliable Distributed Systems (SRDS 2010), New Delhi, India (October 2010)
2. Bayer, U., Kruegel, C., Kirda, E.: TTAnalyze: A tool for analyzing malware. In: 15th European Inst. for Computer Antivirus Research (EICAR 2006) Conf., Hamburg, Germany (April 2006)
3. Chen, P.M., Noble, B.D.: When virtual is better than real. In: Proc. of the 8th Workshop on Hot Topics in Op. Sys., p. 133. IEEE, Washington, DC, USA (2001)
4. Dinaburg, A., Royal, P., Sharif, M., Lee, W.: Ether: malware analysis via hardware virtualization extensions. In: Proc. of the 15th ACM Conf. on Computer and Communications Security, pp. 51–62. ACM, New York (2008)

5. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: Proc. of Network and Distributed Systems Security Symp., pp. 191–206 (2003)
6. Hofmeyr, S.A., Forrest, S., Somayaji, A.: Intrusion detection using sequences of system calls. Journal of Computer Security 6(3), 151–180 (1998)
7. Holz, T., Freiling, F., Willems, C.: Toward automated dynamic malware analysis using CWSandbox. IEEE Security & Privacy 5(2), 32–39 (2007)
8. Jones, S.T., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: VMM-based hidden process detection and identification using Lycosid. In: Proc. of the 4th Int. conf. on Virtual Execution Environments, pp. 91–100. ACM, New York (2008)
9. Kosoresow, A.P., Hofmeyr, S.A.: Intrusion detection via system call traces. IEEE Softw. 14(5), 35–42 (1997)
10. Martignoni, L., Fattori, A., Paleari, R., Cavallaro, L.: Live and trustworthy forensic analysis of commodity production systems. In: Jha, S., Sommer, R., Kreibich, C. (eds.) RAID 2010. LNCS, vol. 6307, pp. 297–316. Springer, Heidelberg (2010)
11. Payne, B.D., Carbone, M., Sharif, M., Lee, W.: Lares: An architecture for secure active monitoring using virtualization. In: Proc. of 2008 IEEE Symp. on Security and Privacy, pp. 233–247. IEEE, Washington, DC, USA (2008)
12. Pfoh, J., Schneider, C., Eckert, C.: A formal model for virtual machine introspection. In: Proc. of the 2nd ACM Workshop on Virtual Machine Security. ACM, New York (2009)
13. Pfoh, J., Schneider, C., Eckert, C.: Exploiting the x86 architecture to derive virtual machine state information. In: Proc. of the 4th Int. Conf. on Emerging Security Information, Systems and Technologies. IEEE, Venice (2010)
14. Rieck, K., Trinius, P., Willems, C., Holz, T.: Automatic analysis of malware behavior using machine learning. Tech. Rep. 18-2009, Berlin Inst. of Technology (2009)

A Nitro Output Rules Definition

We present the flexibility of our rules by expressing them in Backus-Naur Form.

rule	::=	**add_scmon_rule** <condition> <location> <action>
condition	::=	<register> <value>
location	::=	<register> <offset>
register	::=	rax | rbx | rcx | rdx | rsp | rbp | rsi | rdi
value	::=	[0,4294967295]
offset	::=	[-2147483648,2147483647]
action	::=	hex | int | uint | derefhex | derefint | derefuint | derefstr

Taint-Exchange: A Generic System for Cross-Process and Cross-Host Taint Tracking

Angeliki Zavou, Georgios Portokalidis, and Angelos D. Keromytis

Network Security Lab, Department of Computer Science,
Columbia University, New York, NY, USA
{azavou,porto,angelos}@cs.columbia.edu

Abstract. Dynamic taint analysis (DTA) has been heavily used by security researchers for various tasks, including detecting unknown exploits, analyzing malware, preventing information leaks, and many more. Recently, it has been also utilized to track data across processes and hosts to shed light on the interaction of distributed components, but also for security purposes. This paper presents Taint-Exchange, a *generic* cross-process and cross-host taint tracking framework. Our goal is to provide researchers with a valuable tool for rapidly developing prototypes that utilize cross-host taint tracking. Taint-Exchange builds on the *libdft* open source data flow tracking framework for processes, so unlike previous work it does not require extensive maintenance and setup. It intercepts I/O related system calls to transparently multiplex fine-grained taint information into existing communication channels, like sockets and pipes. We evaluate Taint-Exchange using the popular *lmbench* suite, and show that it incurs only moderate overhead.

1 Introduction

Dynamic taint analysis (DTA) has been a prominent technique in the computer security domain, used independently or frequently complementing other systems [9,23,21,19,20,27,26,7], while researchers seem to continuously find new applications for it, many times extending to other domains [12,29]. Originally, taint tracking systems enabled the tracking of marked or "tainted" data throughout the execution of a single process [21], or an entire host in the case of virtual machine (VM)- and emulator-based systems [13,20]. The latter enabled researchers to track the interactions between processes running within a virtual machine.

However, as taint tracking is applied on more domains, like the visualization of information flow among the components of a system [17,28] and the automatic troubleshooting of application misconfigurations [1], systems that can also propagate taint between different hosts over the network have been also developed. Existing cross-application and cross-host taint propagation systems frequently make use of VMs and emulators [17,28], incurring unnecessary overhead and requiring extensive maintenance and setup. Other implementations are very problem-specific, requiring extensive modifications for reuse by the research community to solve new problems.

T. Iwata and M. Nishigaki (Eds.): IWSEC 2011, LNCS 7038, pp. 113–128, 2011.

This paper presents a *generic* cross-process and cross-host taint tracking framework, called Taint-Exchange. Our system, builds on the *libdft* open-source data flow tracking (DFT) framework [14], which performs taint tracking on unmodified binary processes using Intel's Pin dynamic binary instrumentation framework [15]. We have extended *libdft* to enable *transfer* of taint information for data exchanged between hosts through network sockets, and between processes using pipes and unix sockets. Taint information is transparently *multiplexed* with user data through the same channel (*i.e.,* socket or pipe), allowing us to mark individual bytes of the communicating data as tainted. Additionally, users have the flexibility to specify which communication channels will propagate or receive taint information. For instance, a socket from *HOST A* can contain fine-grained taint information, while a socket from *HOST B* may not contain detailed taint transfer information, and all data arriving can be considered as tainted. Similarly, users can also configure Taint-Exchange to treat certain files as tainted. Currently, entire files can be identified as a source of tainted data.

Most real-world services consist of multiple applications exchanging data, that in many cases run on different hosts, *e.g.,* Web services. Taint-Exchange can be a valuable asset in such a setting, providing transparent propagation of taint information, along with the actual data, and establishing accurate cross-system information flow monitoring of *interesting* data. Taint-Exchange could find many applications in the system security field. For example, in tracking and protecting privacy-sensitive information as it flows throughout a multi-application environment (*e.g.,* from a database to a web server, and even to a browser). In such a scenario, the data marked with a "sensitive" tag, will maintain their taint-tag throughout their lifetime, and depending on the policies of the system, Taint-Exchange can be configured to raise an alert or even restrict their use on a security-sensitive operation, *e.g.,* their transfer to another host. In a different scenario, a Taint-Exchange-enabled system could also help improve the security of Web applications by tracking unsafe user data, and limiting their use in JavaScript and SQL scripts to protect buggy applications from XSS and SQL-injection attacks.

Taint-Exchange, along with libdft, provides a stable and reusable cross-host taint tracking platform that can promote new research and expedite the development of research prototypes. The main contributions of this paper are summarized in the following:

- We designed and implemented a *reusable* cross-process and cross-host taint tracking framework. Taint-Exchange is based on *libdft* [14], a customizable DFT framework that offers an extensive API for creating tools
- Taint-Exchange operates transparently on unmodified x86 Linux binaries, allowing real-world legacy applications to take advantage of our framework transparently
- We offer flexible configuration of taint sources, as well as allowing mixing our own fine-grained taint transferring sockets with ordinary sockets. For example, many security-oriented DTA implementations [19] do not support configurable taint sources, and mark all incoming network as tainted

– We improved on inter-process taint tracking over previous system-wide tracking systems (e.g. Minos [9], TaintBochs [7], Rakscha [10], RIFLE [24]), which are based on slow full-system emulators (e.g. Xen [2], QEMU [3], Bochs [4]), by enabling cross-host and cross-process tracking on the communication channels that matter to the target applications, rather than overloading every operation in the entire system with unnecessary heavyweight taint tracking operations
– We evaluate the overhead imposed by Taint-Exchange, and show that it incurs minimal overhead over the baseline tool *libdft*

The rest of the paper is organized as follows. Section 2 introduces the concept of dynamic taint tracking and presents the most important implementations in this research area. In Sect. 3 we present our protocol and our system. In Sect. 4, we highlight the implementation choices made when we built our system. In Sect. 5, we evaluate our system. We discuss future work in Sect. 6, and finally, our conclusions follow in Sect. 7.

2 Background and Related Work

2.1 Dynamic Taint Tracking

Dynamic taint tracking is the mechanism of monitoring the flow of tainted data, at runtime, within an instance of a software application (process) or a system, after "recognizing" the *data of interest* according to a predefined taint configuration, and associating it with metadata, usually referred to as *taint tags*. Therefore, most dynamic taint analysis implementations can be described by three elements the *taint sources*, the *propagation policy* and the *taint sinks*. Regarding taint-tags, in most cases one bit of taint is sufficient, but there are situations where multiple bits are useful. For instance, to distinguish between multiple input sources or to distinguish between trust levels.

Dynamic taint tracking is not a new concept. One of its first practical instantiations was employed in detecting and defending against software attacks [19], while it has found many more applications since then. Currently, dynamic tracking approaches range from *per-process* taint tracking [6,8,14,19,21,25,29], to *whole-system* tracking [9,10,20,27] using emulation environments and hardware extensions.

2.2 Single-Process Taint Tracking

Most application-level taint tracking tools, like TaintCheck [19], TaintTrace [6], *libdft* [14], Dytan [8], and LIFT [21] use dynamic binary instrumentation (DBI) frameworks, like PIN [15], StarDBT [5] and Valgrind [18]. While quite effective and useful, as they do not require any modifications to source code or customized hardware, they impose significant impact on the performance, as every instruction needs to be instrumented, and additional storage, usually called *shadow*

memory, is required for storing the tags. As a result, there has been great interest in optimization techniques in order to improve their performance. TaintTrace achieved significantly faster taint-tracking by using more efficient instrumentation based on DynamoRIO, combined with simple static analysis to speed up the taint-tag access. LIFT also achieved significant additional performance benefits by using better static analysis and faster instrumentation techniques.

2.3 Cross-Process and Cross-Host Taint Tracking

A large body of research has also focused on cross-process or system-wide taint tracking, leading to the creation of many tools [9,10,27,28], mostly based on emulators and hardware extensions to efficiently handle data tracking for an entire operating system (OS). For instance, the whole system emulator QEMU [3] is employed by various solutions that implement DTA [13,20,27], while TaintBochs [7] builds on the Bochs IA-32 emulator. The architecture community attempted to integrate or assist dynamic taint tracking with hardware extensions [9,10,23,24], to alleviate the significant performance impact due to extra tag processing from DBI frameworks and emulators.

While there is much research aiming at intra-process and system-wide DTA implementations, it was not until very recently that interest has risen for efficient cross-host taint propagation systems [1,11,28]. Most of these techniques are more problem-specific, and therefore it would be difficult to adapt the techniques and tools developed for use in other contexts. For instance, DBTaint [11] is targeting taint information flow tracking specifically for databases, whereas ConfAid [1], which is the closest to our design, tackles the problem of discovering a set of possible root causes in configuration files that may be responsible for software misconfigurations. System tomography [17], which also looks into the concept of propagating taint information remotely, builds on the QEMU emulator so it cannot be applied on already deployed software and incurs large slowdowns. Finally, Neon [28] also requires modifications in the underlying system to perform dynamic taint tracking. It uses a modified NFS server for handling the initial tainting, and utilizes a network-filter for monitoring the tainted packets arriving/leaving the server.

In contrast to previous approaches, that use slow-emulators or VMs to perform system-wide taint tracking, in Taint-Exchange we use an already available and fast single-process taint-tracking framework [14], and extend it to perform fine-grained, cross-process, and cross-host transfer of taint information. *Although our design was inspired by prior works, it addresses different challenges, is more general, and completely transparent to applications.*

3 Taint-Exchange

We designed and implemented Taint-Exchange based on the *libdft* data flow tracking framework [14], to produce a *generic* system for efficiently performing cross-process and cross-host taint tracking. Nevertheless, Taint-Exchange could

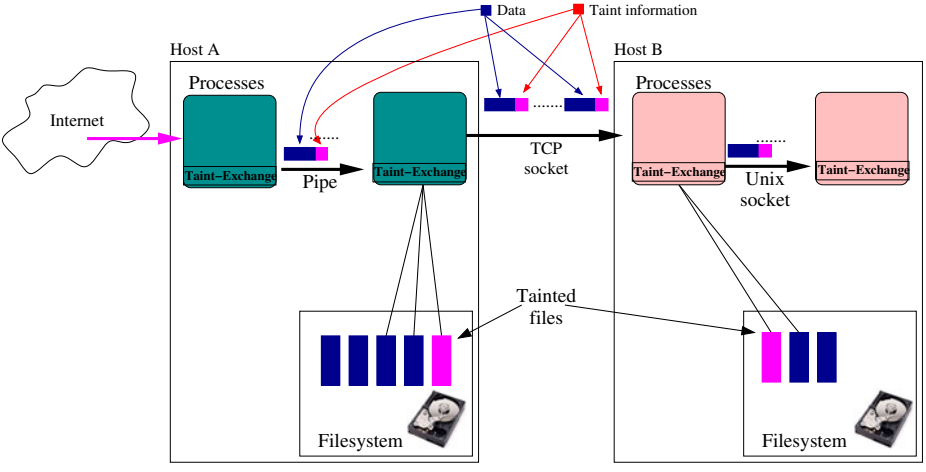


Fig. 1. Taint-Exchange overview. Taint information can be exchanged using network sockets and pipes. Sockets and files can also be configured as taint sources.

be easily retargeted to numerous other taint tracking systems similar to libdft, *e.g.,* TaintCheck [19], TaintTrace [6], LIFT [21], or Dytan [8], as Taint-Exchange is a broadly applicable design. libdft was chosen because it is one of the fastest process-wide taint-tracking frameworks, which performs at least as fast, if not faster than similar systems such as LIFT and Dytan.

3.1 Design Overview

We will present the main aspects of Taint-Exchange, following the three-dimensions described in Sect. 2. Figure 1 shows an overview of Taint-Exchange, the various sources of tainted data that can be configured, and the mechanisms for exchanging taint information between processes and hosts.

Taint Sources. The taint sources are the "starting points" of the system, where taint is assigned to *data of interest*. Our current framework supports configurable taint sources from the two most common input channels, the file-system and the network. In the current implementation, the user of Taint-Exchange directly interacts with the underlying framework (*i.e.,* both Taint-Exchange and libdft) to define the taint sources, as they each time depend on the problem being tackled. Although configuration is straightforward, we stress that a better user interface for the configuration of the taint sources would improve usability, but this is beyond the scope of this paper.

Configuring the filesystem taint sources is straightforward. A shadow file `taint_config` is maintained for listing all tainted files in the system. The designer has to update it with the taint files, using full-path format, and all data originating from files listed in `taint_config` will be marked as tainted.

In addition, network sockets and pipes can also be among the taint sources, so data arriving from them can be also tagged as tainted. Sockets and pipes can be also configured to receive and transmit detailed taint information regarding the data being exchanged (described later in this section).

A global array (*state_sfd*) is used to keep track of the important "channels" that comprise Taint-Exchange's taint sources. The `open()`, `socket()`, `accept()`, `dup()`, and `close()` system calls are intercepted to update the `state_sfd` array accordingly. The marked "channels" will be the ones monitored for tainted data. Briefly, for read-like calls, such as `read()`, `readv()`, `recv()` etc., this includes the extraction of a taint-header from the received data stream, the reception of the taint information, and the appropriate marking of the received data.

Data Tags. Currently our system only supports binary tags (tainted or clean data), but this is only a limitation because of the chosen underlying taint tracking system *libdft* [14]. In fact, according to the authors of libdft, future versions will include support for multiple labels/colors for tracked data. In the future, we plan to port our tool to using the lated libdft version to take advantage of the richer data tags. Of course, we expect larger tags to have a larger impact on the performance of data transfers, but it is something that needs to investigated.

Taint Propagation. There are three cases that we examined for the propagation of taint tags. Firstly, the *intra-propagation* of taint values during the execution of a single process. As we discussed in Sect. 2, this has been thoroughly explored by past work, and there exist many tools [6,8,14,19] for efficiently handling this issue. Generally, all these tools allocate a "shadow storage" for every process to store which data is tainted (*i.e.,* data tags). We will refer to this shadow memory as a `tagmap`. The second case of taint propagation we consider is the *cross-process* propagation of taint tags for the data exchanged between processes. Previous research has mostly addressed this topic by performing system-wide taint tracking using modified VMs and specialized hardware [9,10,20,27], mostly based on emulators and hardware extensions for efficiently handling system-wide tracking of tainted information. The last case we examine is the *cross-host* transfer of tainted tags. Relatively little research has explored this path [1,11,17,28].

For Taint-Exchange, **taint transfer** refers to the propagation of taint information along with the data, when processes on the same host or on different hosts communicate. Our mechanism supports processes that communicate using sockets and pipes. Briefly, the main idea is to monitor the information flow between the taint sources, and intercept the system calls from the `read/receive` and `write/send` families that are used to read from and write to the tainted channels. In the case of data leaving the current process (or host), a **taint-header** is composed and attached to the data, indicating which bytes, if any, are tainted. We will describe the taint-header in detail in Sect. 3.2. On the receiving side, as "extended" data enters the process (or host), and assuming that the source descriptor is among the taint sources described earlier, the taint header will be extracted from the received data, and the taint-tag storage structure of the process will be updated accordingly.

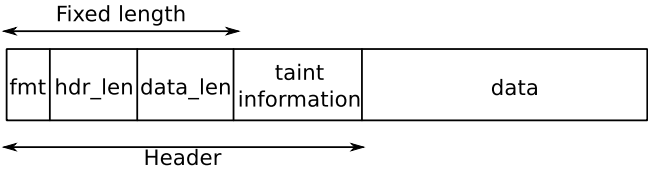


Fig. 2. Taint-Exchange encapsulates data using a header to transparently inject taint information in data transfers

Cross-process taint transfer is handled the same way as cross-host taint transfer. The only difference is the source descriptor, which in the case of pipes is the pid or name of the application we are communicating with. IPC through UNIX sockets is very similar to TCP sockets, so the mechanism remains almost entirely the same.

Taint Sinks. *Taint sinks* refer to the "locations" in the system, where the user needs to perform some assertions on the data. For example, tainted data may not be allowed to be transmitted overa certain socket, or used as program control data (*e.g.,* a function return address). or it should just be logged. Taint sinks are problem-specific, and can be configured by the user. *libdft* offers an extensive API for the users to check for the presence of tainted data on instructions and on system or function calls.

3.2 Taint Headers

Taint Header Structure. To multiplex data and taint information, Taint-Exchange prefixes each data transfer with a taint header, which essentially encapsulates the transferred data into new "packet" protocol, following the format shown in Fig. 2. It consists of the following fields:

`fmt` the format version of that taint information
`hdr_len` the length of the header including that taint information
`data_len` the length of the data payload (*i.e.,* the data the application is actually transferring)
`taint information` fine-grained taint information regarding the payload, and following the format specified by `fmt`
`data` the data payload

Composing the Taint Header. A taint header is created when `write-like` system calls, such as `write()`, `writev()`, `send()` etc., are executed and the destination descriptor of the system call is among the ones configured to transfer taint. The process' `tagmap` is referenced to determine which parts of the outgoing data message is tainted. Depending on the number of tainted bytes and their distribution, Taint-Exchange determines which format to use to encode the taint information for the data. We support two formats for encoding taint

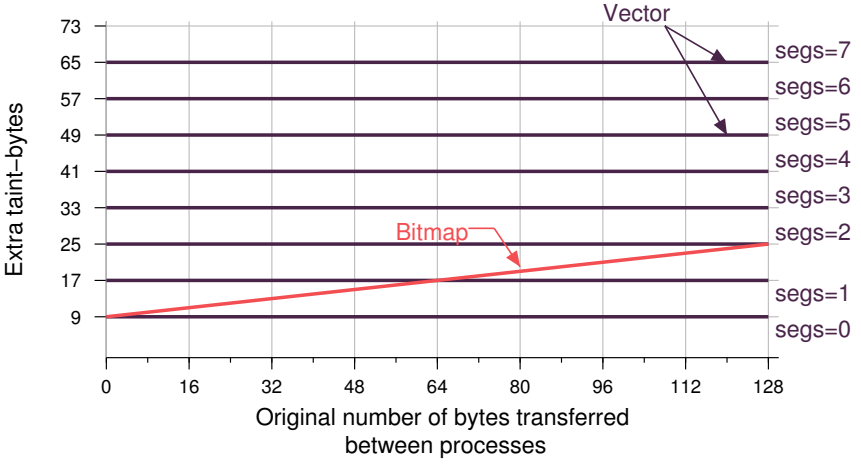


Fig. 3. Space overhead of the different Taint-Exchange taint information transfer formats. For both formats, the fixed header size is 9 bytes. The overhead of the bitmap format is linear with the number of bytes transferred, while with the vector protocol it depends solely on the number of tainted data segments.

information. The first, is a *bitmap* which contains one bit for every byte of data being transferred, and the second using a *vector* for each segment of data that is tainted. For example, if bytes 5 through 15 are tainted, the vector describing this segment is [5, 10]. That is, there is a segment of tainted data starting at offset 5 of the data and lasting for 10 bytes. The space overhead of these two formats is drawn in Fig. 3. We see that depending on the number of tainted segments a different format is preferable. Usually, for large continuous areas of tainted bytes, a vector proves to be a more efficient choice, whereas for sparsely tainted bytes the bitmap is preferable.

4 Implementation

4.1 The libdft Data Flow Tracking Framework

Taint-Exchange operates by intercepting and instrumenting the system calls used for inter-process as well as for cross-host communication, while it relies on an already available tool, *libdft* [14], to perform the taint tracking within each process. For this purpose, we instrumented the `socketcall` family of system calls (*i.e.,* `socket()`, and `accept()`), the `dup()` system call, and the `read/receive`-like and `write/send`-like system calls. We also intercept the `open()`, `close()` and `mmap()` system calls for handling files.

For the intra-process dynamic taint tracking we chose *libdft* [14], a dynamic Data Flow Tracking (DFT) framework, designed to transparently perform DFT on binaries. Although our design is independent of the underlying IFT system,

for our implementation we chose *libdft* because of its availability, well-defined API, and efficient instrumentation, which makes it one of the fastest process-wide taint-tracking frameworks. *libdft* is used as a shared library offering a user-friendly API for customizing *intra-process taint propagation*, and can be used on unmodified multi-threaded and multiprocess applications. It relies on PIN [15], a dynamic binary instrumentation framework (DBI) from Intel, widely used in the implementation of other DTA tools [8,29].

Similarly to PIN's, the *libdft* API allows both instrumentation and analysis of the target process. In particular, *libdft*'s *I/O interface* component, offers an extensive *system call* level API, enabling instrumentation hooks before (`pre_syscall`) and after (`post_syscall`) every system call, while making use of the underlying DFT services. We have registered analysis callbacks for the "interesting" system calls, which get invoked when these system calls are encountered, to observe the process' communication "channels", and to inject taint information along with the native data (*i.e.,* the data the application is communicating). For the system calls that are not explicitly handled by Taint-Exchange, the default behavior is to clear the tags of the data being read (*i.e.,* the data written in the process' memory). This way, over-tainting is avoided since "uninteresting" system calls, that overwrite program memory with new inputs read from the kernel, are not ignored. It is worth to mention that *libdft* does not suffer from taint-explosion (*i.e.,* over-tainting data that should not be tagged), because it does not consider control-flow dependencies and it operates in user-space. Control-flow dependencies, kernel data structures, and pointer tainting have been identified as the prominent causes of taint-explosion [22].

4.2 Taint-Exchange Data Structures

The *tagmap* is the taint-tag storage maintained by *libdft*, reflecting the taint-status of the running process' memory and CPU registers for each running thread. Its implementation plays a crucial role in performance and memory overhead. *libdft* supports byte-level memory tagging, which is mapped to a single-bit tag in the process' tagmap, and four 1-bit tags for every 32-bit GPR. *libdft* offers an extensive API for the update of the taint-tags in tagmap (*e.g.,* `tagmap_setb`, `tagmap_getb`, `tagmap_clrb` are handling the taint-tag per byte of addressable memory).

The *state_sfd* is the global array of tainted descriptors, that designates the important "channels", being monitored for tainted data. It is updated by the system call subset used for handling files and creating/closing sockets, and is indexed by the file (or socket) descriptor. Initially, all elements of the array are empty. The array is updated by the instrumented versions of `open()`, `socket()`, `accept()`, `dup()` and `close()` system calls, and variations of them.

Finally, *taint_config* is the configuration file for filesystem taint sources. The files listed in it should be written in full-path format.

4.3 Filesystem Taint Sources

Currently, our framework supports configurable taint sources from the file-system and the network. The `taint_config` file, lists the files that contain *data of interest*, which should be tainted and tracked throughout the monitored system. This is implemented, by the instrumented `open()` system call, which marks as "tainted" the *state_sfd* elements, that correspond to the files listed in *taint_config*. The descriptors of these files are considered "channels" of incoming tainted data. Therefore, whenever a system call, like `read()`, tries to read data from them the corresponding taint-tags in the *tagmap* structure are updated accordingly.

4.4 Taint Propagation over the Network

The main purpose of Taint-Exchange is the delivery of taint-tags along with the transferred data in all the tainted "channels". To establish information about the TCP channels the `socket()` and `accept()` system calls are instrumented. For simplicity, in the current implementation, every network connection is considered *capable* of propagating tainted data, but this could be easily limited to work only on specific IPs. When a TCP connection is established, *state_sfd* structure is updated accordingly to add the new socket descriptor to the monitored "channels".

The cross-host taint propagation mechanism is handled by the instrumented versions of `write/send`-like system calls on the sending side, and `read/receive`-like system call on the receiving side. When the sender transmits data by invoking a `write()` (or an equivalent) system call, Taint-Exchange constructs the corresponding taint-header according to the relevant taint-tags as reflected in the *tagmap* of the sending process and attaches it to the original data. The receiving side, will invoke an instrumented `read()` call (or an equivalent) to process the "extended data", and update the process' *tagmap* with the taint-tags corresponding to the received data.

4.5 Cross-Process Taint Propagation

Interprocess communication can happen through unix sockets, TCP/UDP sockets, pipes, and shared memory. If the processes are communicating via sockets or pipes, taint tags can propagate between communicating processes in the same manner we described in the previous section. The main difference is the source descriptor, which in the case of pipes is linked to the pid or name of the application the process is communicating with. IPC through UNIX sockets is very similar to TCP sockets, so the mechanism remains almost entirely the same. We are currently not handling data exchanged through shared memory segments.

5 Evaluation

The aim of this section is to demonstrate the communication overhead of Taint-Exchange, when transparently passing taint information, along with real data,

across the network. To assess the impact imposed by Taint-Exchange, we performed several micro benchmarks using utilities from the *lmbench* [16] Linux performance analysis suite. During the tests we only used the bitmap format to represent taint information as the overhead of the vector format can vary significantly depending on the application scenario.

Our testbed consisted of two identical hosts, equipped with two 2.66GHz quad core Intel Xeon X5500 CPUs and 24GB of RAM each, running Linux (Debian "squeeze" with kernel version 2.6.32). The version of Pin used during the evaluation was 2.9 (build 39599). When conducting our experiments, the hosts were idle with no other user processes running under taint-tracking apart from the evaluation suites.

Since Taint-Exchange intercepts socket connection calls to inject the additional taint information, we used lmbench's bandwidth benchmark *bw_tcp* to measuring the impact of our approach when moving the "extended" data over the network. bw_tcp measures TCP bandwidth by creating two processes, a server and a client, that are moving data over a TCP/IP connection. We repeated our tests with data of different sizes (*i.e.,* 64, 128, 256, 512, 1024 and 1047 bytes), and against three different scenarios: (a) using a simple pintool, *null_tool*, which uses minimum PIN instrumentation to add callbacks to system calls without further employing any form of analysis. We developed this as the base case, to establish the lower bound of our instrumentation and analysis overhead as imposed by PIN's runtime environment alone. (b) *libdft-dta*, a tool based on *libdft* to employ basic dynamic taint analysis. We used this tool to achieve an estimation of the overhead imposed by *libdft*. (c) Taint-Exchange.

We repeated the measurements 10 times and calculated the mean and standard deviation of the output. The results are presented in Figure 4. We see that there is an obvious impact on the throughput of TCP sockets, which becomes more severe as the size of the sent data increases. As expected, Taint-Exchange has the largest impact of the three scenarios as the number of data sent every time is more than the other applications. For example, in the case of the 64 bytes buffer sent, the space overhead will be 17 bytes (as described in Section 3.2).

When running our tests we noticed that there was an instability in the measurements, as size of the buffer increased, and especially with the libdft-dta tool. We assume that maybe the measurements are affected also by the instruction instrumentation that libdft-dta employs. Unfortunately, we have not yet determined the exact reasons for this instability as we are not fully aware of lmbench's internal workings. The 1437B buffer in our experiments is a default of the lmbench benchmark, and has to do with the maximum number of (payload) bytes that can be transmitted within a single Ethernet frame. This also explains the oscillation in performance, as the taint information can no longer fit within the same Ethernet frame as the data. We should note that Pin itself introduces significant overhead with small buffer sizes like the ones used by the *lmbench* suite (Fig. 4), reducing throughput by 10x-20x compared with native execution. On the other hand, when using large, 10MB buffers (*i.e.,* the largest buffer measured by *lmbench*), Pin does incur any observable overhead on throughput.

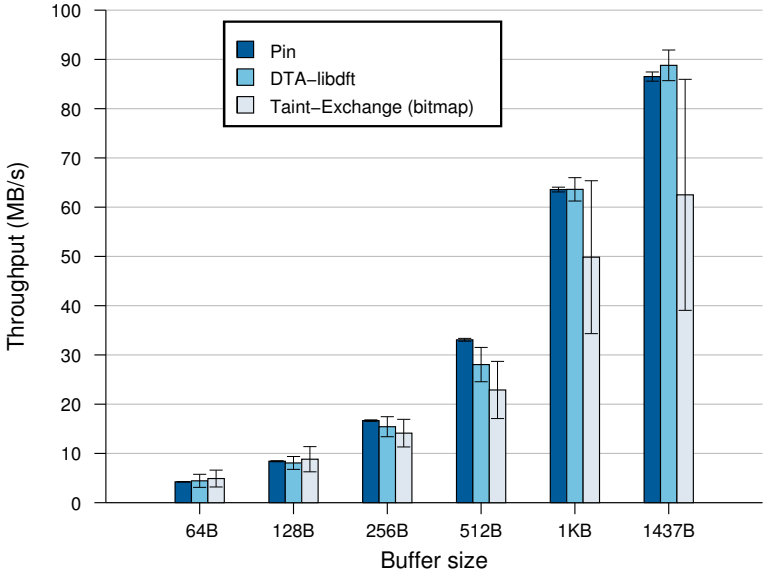


Fig. 4. TCP socket throughput measured with *lmbench* for various buffer sizes. We draw the mean and standard deviation. Note that the effect of Pin with the buffers drawn here is quite pronounced, reducing throughput by 10x-20x compared with native execution. In contrast, it does not incur any observable overhead with large, 10MB buffers.

Since the implementation of Taint-Exchange is mostly based on the instrumentation of system calls, we also employed the *lat_syscall* benchmark to measure the latency impact of the three implementations . We used lat_syscall with the `open`, `read` and `write` system calls, in order to show how these operations are affected. We chose these three system calls as they represent the calls that were affected the most by Taint-Exchange. More specifically, `open()` is handling the initial configuration of tainted "channels" from the file-system, while the `read` and `write` system calls are the ones handling the movement of the tainted information along. In the performed tests, *lat_syscall read* measures how long it takes to read one byte from /dev/zero, whereas *lat_syscall write* measures how long it takes to write one byte to /dev/null. *lat_syscall_open* measures how long it takes to open and then close a file. The results are presented in Figure 5. The conditions during these measurements were the same as with bw_tcp, regarding repetitions and the calculation of the mean and standard deviation from the original measurements.

The observed overhead is attributed to the overhead of PIN for the dynamic instrumentation analysis of the process, as well as the overhead inserted by *libdft* performing the taint-tracking. The additional overhead imposed by Taint-Exchange, apart from the obvious reasons such as the instrumentation of these system calls for handling the taint-headers and the continuous update the taint

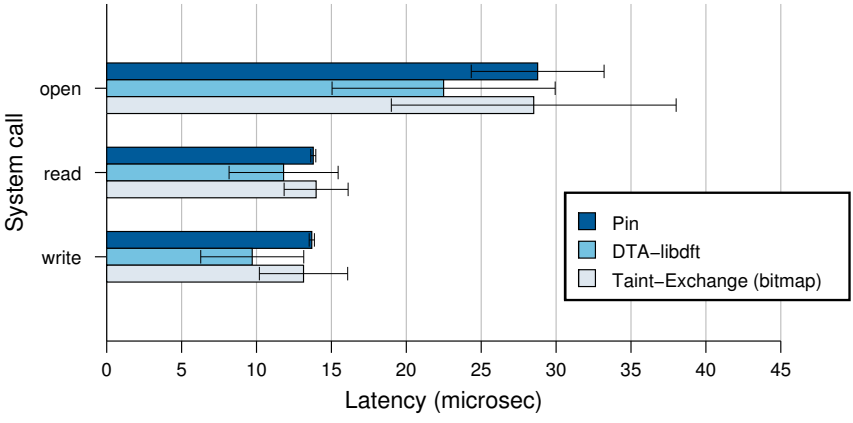


Fig. 5. System call latency measured with *lmbench*. Note that Pin, as probably most DBI frameworks, greatly affects system call latency (approximately 20x slower than native).

data structures, is also implementation-specific. For instance, in our current implementation every `write()` system call performed by the application results in an additional write being performed to inject the taint-header. Similarly, the instrumented version of the `read()` system call is reading the "extended" data in three pieces, inevitably imposing the overhead seen in Figure 5. Note that Pin, as probably most DBI frameworks, greatly affects system call latency (approximately 20x slower than native) because it receives control before and after every call. We attribute Pin's overhead on throughput with small buffer sizes, to the general increase in system call overhead.

6 Future Work

In this paper, we presented a preliminary implementation of Taint-Exchange, our approach for handling cross-application and cross-host transfer of tainted information. There are some obvious extensions to the work presented in this paper, which we plan to address in a next version of Taint-Exchange. In the current implementation, every TCP socket is by default considered among the important "channels", that participate in the taint-propagation process. We are planning to build a more fine-grained configuration procedure, so that certain IPs can be included (or excluded) from participating into the propagation of the taint-tags over the network.

In addition, we plan to support persistent taint information storage for files, to be able to handle both tainted and untainted data stored in a file. An auxiliary file per original file could be used to maintain the information on the tainted bytes of the original file. A similar scheme with the one used for passing taint information over the network will be probably used (*e.g.,* bitmap for files with

interleaved tainted and clean data, and vectors for files that store tainted data in large blocks). Compression may also be utilized to reduce storage requirements. Coarser-grained tracking can already be performed. For instance, when tainted data are written to a file, taint-exchange can consider the entire file as tainted.

Finally, in order to reduce the overhead of the inserted taint header we think that it is a promising direction to explore the use of TCP optional headers to pass the taint information. This option would not only help us trivially implement communication between a native and a taint-exchange application, but it could also potentially improve the overall performance of our proposed mechanism.

7 Conclusion

We presented Taint-Exchange, a *generic* cross-host and cross-process taint tracking framework. Taint-Exchange enables the transfer of fine-grained taint information across processes and the network. It does so by intercepting I/O system calls to transparently inject and extract information regarding the *taintness* of every byte transferred between processes running under Taint-Exchange. It also provides a flexible mechanism for easily customizing the sources of tainted data, be it a network socket, a file, or an IPC mechanism like a pipe or UNIX socket. Our evaluation of Taint-Exchange shows, as expected, that I/O is affected because of the additional data being sent, and the utilization of the same channel to do so. Nonetheless, we believe that the overhead is small, specially when compared with the high overheads imposed by the various dynamic taint tracking systems.

Acknowledgements. This work was supported by the US Air Force and the National Science Foundation through Contract AFRL-FA8650-10-C-7024 and Grant CNS-09-14845, respectively, with additional support by Google and Intel Corp. Any opinions, findings, conclusions or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government, the Air Force, the NSF, Google or Intel. We are also grateful to our shepherd, William Enck, for his assistance in preparing the camera ready version of this paper.

References

1. Attariyan, M., Flinn, J.: Automating configuration troubleshooting with dynamic information flow analysis. In: Proceedings of the 9th Symposium on Operating Systems Design and Implementation (OSDI), pp. 1–11 (2010)
2. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: Proceedings of the 19th Symposium on Operating Systems Principles (SOSP), pp. 164–177 (2003)
3. Bellard, F.: QEMU, a fast and portable dynamic translator. In: Proceedings of the USENIX Annual Technical Conference, pp. 41–46 (April 2005)
4. Bochs: The cross platform IA-32 emulator (2001), http://bochs.sourceforge.net

5. Borin, E., Wang, C., Wu, Y., Araujo, G.: Software-based transparent and comprehensive control-flow error detection. In: Proceedings of the International Symposium on Code Generation and Optimization (CGO), pp. 333–345 (2006)
6. Cheng, W., Zhao, Q., Yu, B., Hiroshige, S.: TaintTrace: Efficient flow tracing with dynamic binary rewriting. In: Proceedings of the IEEE Symposium on Computers and Communications (ISCC), pp. 749–754 (2006)
7. Chow, J., Pfaff, B., Garfinkel, T., Christopher, K., Rosenblum, M.: Understanding Data Lifetime via Whole System Simulation. In: Proceedings of the 13th USENIX Security Symposium, pp. 321–336 (2004)
8. Clause, J., Li, W., Orso, A.: Dytan: a generic dynamic taint analysis framework. In: Proceedings of the 2007 International Symposium on Software Testing and Analysis (ISSTA), pp. 196–206 (2007)
9. Crandall, J.R., Chong, F.T.: Minos: Control data attack prevention orthogonal to memory model. In: Proceedings of the 37th Annual International Symposium on Microarchitecture, pp. 221–232 (2004)
10. Dalton, M., Kannan, H., Kozyrakis, C.: Real-world buffer overflow protection for userspace & kernelspace. In: Proceedings of the 17th USENIX Security Symposium, pp. 395–410 (2008)
11. Davis, B., Chen, H.: DBTaint: cross-application information flow tracking via databases. In: Proceedings of the 2010 USENIX Conference on Web Application Development, WebApps (2010)
12. Enck, W., Gilbert, P., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In: Proceedings of the 9th Symposium on Operating Systems Design and Implementation (OSDI), pp. 393–407 (2010)
13. Ho, A., Fetterman, M., Warfield, C.C.A., Hand, S.: Practical taint-based protection using demand emulation. In: Proceedings of the 1st European Conference on Computer Systems (EuroSys), pp. 29–41 (2006)
14. Kemerlis, V.P.: libdft (2010), http://www.cs.columbia.edu/~vpk/research/libdft/
15. Luk, C.K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: Building customized program analysis tools with dynamic instrumentation. In: Proceedings of the Conference on Programming Language Design and Implementation (PLDI), pp. 190–200 (2005)
16. McVoy, L., Staelin, C.: lmbench (2005), http://lmbench.sourceforge.net/
17. Mysore, S., Mazloom, B., Agrawal, B., Sherwood, T.: Understanding and visualizing full systems with data flow tomography. In: Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 211–221 (2008)
18. Nethercote, N., Seward, J.: Valgrind: A framework for heavyweight dynamic binary instrumentation. In: Proceedings of the Conference on Programming Language Design and Implementation (PLDI), pp. 89–100 (2007)
19. Newsome, J., Song, D.: Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: Proceedings of the 12th Annual Network and Distributed System Security Symposium, NDSS (2005)
20. Portokalidis, G., Slowinska, A., Bos, H.: Argos: an emulator for fingerprinting zero-day attacks. In: Proceedings of the 1st European Conference on Computer Systems (EuroSys), pp. 15–27 (2006)

21. Qin, F., Wang, C., Li, Z., Kim, H.s., Zhou, Y., Wu, Y.: LIFT: A low-overhead practical information flow tracking system for detecting security attacks. In: Proceedings of the 39th Annual International Symposium on Microarchitecture, pp. 135–148 (2006)
22. Slowinska, A., Bos, H.: Pointless tainting? Evaluating the practicality of pointer tainting. In: Proceedings of EuroSys 2009, Nuremberg, Germany (March-April 2009)
23. Suh, G.E., Lee, J., Devadas, S.: Secure program execution via dynamic information flow tracking. In: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 85–96 (2004)
24. Vachharajani, N., Bridges, M.J., Chang, J., Rangan, R., Ottoni, G., Blome, J.A., Reis, G.A., Vachharajani, M., August, D.I.: RIFLE: An architectural framework for user-centric information-flow security. In: Proceedings of the 37th International Symposium on Microarchitecture (MICRO), pp. 243–254 (2004)
25. Wang, T., Wei, T., Gu, G., Zou, W.: TaintScope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 497–512 (2010)
26. Xu, W., Bhatkar, S., Sekar, R.: Taint-enhanced policy enforcement: a practical approach to defeat a wide range of attacks. In: Proceedings of the 15th USENIX Security Symposium (2006)
27. Yin, H., Song, D., Egele, M., Kruegel, C., Kirda, E.: Panorama: capturing system-wide information flow for malware detection and analysis. In: Proceedings of the 14th Conference on Computer and Communications Security (CCS), pp. 116–127 (2007)
28. Zhang, Q., McCullough, J., Ma, J., Schear, N., Vrable, M., Vahdat, A., Snoeren, A.C., Voelker, G.M., Savage, S.: Neon: system support for derived data management. In: Proceedings of the 6th International Conference on Virtual Execution Environments (VEE), pp. 63–74 (2010)
29. Zhu, D., Jung, J., Song, D., Kohno, T., Wetherall, D.: TaintEraser: protecting sensitive data leaks using application-level taint tracking. SIGOPS Operating Systems Review 45, 142–154 (2011)

An Entropy Based Approach for DDoS Attack Detection in IEEE 802.16 Based Networks

Maryam Shojaei, Naser Movahhedinia, and Behrouz Tork Ladani

Department of Computer Engineering, University of Isfahan Hezarjarib, Isfahan 81746, Iran
{m_shojaei,naserm,ladani}@eng.ui.ac.ir

Abstract. Distributed denial of service attacks are great security threats to computer networks, especially to large scale networks such as WiMAX. Detecting this kind of attack is not as easy as some other attacks, because the traffic created by attack is too similar to the traffic of the network in the normal case. So in this paper a novel framework is proposed to detect DDoS attack in IEEE802.16-based networks efficiently. The key idea of the proposed method is to exploit some statistical features of the incoming traffic. In fact we design a system in which some entropy-based features of the traffic are analyzed. Based on these features we decide whether the attack has occurred or not. Previous works have all focused on the entropy of IP address of the incoming packets, while in this system we have comprehensively considered some other entropy-based features which help a lot in detecting the attack rather than just considering the entropy of the incoming IP addresses. Also in the proposed method we have tried to exploit the long range dependency of the traffic to detect the attack. The simulation results show that the proposed method can detect DDoS attacks efficiently.

Keywords: WiMAX, DDoS attack, Entropy, Initial network entry, RNG-REQ Message.

1 Introduction

The WiMAX technology which is based on IEEE 802.16 may use radio transmission for high speed direct access to internet. Due to high bit rate and QoS support, WiMAX networks are able to offer multimedia services such as voice and video streaming and instant data transfers. Today Mobile WiMAX is one of the wireless broadband standards capable of providing the quadruple play technologies data, voice, video and mobility using a single network. Although many solutions have been presented for analyzing and detecting DDoS attacks, but still this threat is an important security problem for these networks [1,2,3]. Traditional wireless technologies such as 2.5G cellular networks are not exposed to DDoS attack since they are mainly based on circuit switching. However, with emergence of broadband wireless services like Mobile WiMAX system that are based on packet switching, cellular networks are no more safe against DDoS attacks [4,5,6]. In [1,7], Nasreldin, et al. studied the security vulnerabilities of IEEE 802.16 and categorized the threats as solved and unsolved ones. However interrupt attacks, such as DoS attacks, in which

T. Iwata and M. Nishigaki (Eds.): IWSEC 2011, LNCS 7038, pp. 129–143, 2011.

an intruding entity blocks information sent from the source entity to the destination entity have not been discussed much, and since WiMAX is a rather recent technology, this vulnerability has not been deservedly studied yet. In the next section of this paper the previous works in similar networks such as internet and heavy traffic campus networks will be reviewed then in section 3 a reference model which is used to simulate an attacked network and its traffic, bandwidth allocation for ranging and vulnerability to DDoS attacks are described. Section 4 presents the network traffic model and the method employed for calculating entropy-based parameters. In section 5 WiMAX network simulations with the mentioned vulnerability and the evaluated statistical parameters in different condition are described. Finally we conclude our paper in section 6.

2 Related Works and Background

2.1 DDoS Attack in Similar Networks

Since DDoS attacks can make a server or base station to go out of service, lots of researches have been done on this attack. In this section some works which have been done in similar networks will be reviewed. The researchers have used entropy and distributions of traffic features to detect DDoS attack and have paid a lot of attention to it. As one of the recent investigations, in [8] Lee, et al. used cluster analysis to detect DDoS attack by selecting some parameters and features of the traffic which show anomalies in the traffic. They used 2000 DARPA Intrusion Detection Scenario data set and as a result they divided the data to 5 groups. By extracting some features of the traffic which includes entropy of source and destination IP address, entropy of source and destination port number, entropy of packet type, occurrence rate of packet type (ICMP, UDP, TCP SYN) and number of packets and clustering them in to 5 groups, they differentiated between the normal and attack traffic. Among the five phases of the DDoS attack, they could detect three phases. In [9] George, et al. extracted two types of features from the network traffic. One type is extracted from the header of the received packets and the other type includes some behavioral features which are extracted from the network traffic. The first type includes source and destination IP address, source and destination port number and flow size distribution. The second type includes the in- and out-degree of each active internal IP address inside the network under consideration. By using CMU, GA Tech, Internet2 and G`EANT data sets and calculating the correlation between the extracted features they showed that there is a strong relationship between port and address distributions while the degree distributions and flow size distribution are weakly correlated with each other and with the port/address distributions. The correlation between the port number and addresses distribution arises due to the underlying traffic templates. In [10], Shui and Wanlei calculated the entropy of flows at a router, if the router entropy is less than a given threshold, then an attack alarm is raised and then the routers on the path of the suspected flow will calculate the entropy rate of the suspected flow. If the entropy rates are the same or the difference is less than a given value, then the traffic is marked as attack traffic and the packets are discarded. In [11] Sumit and Sahoo also used entropy and entropy rate to model an anomaly detection system for DDoS

attacks in grid computing. In their proposed algorithm, first the entropy of the received packets is estimated. If the estimated entropy exceeds the threshold, then the entropy rate will also is calculated and compared with a threshold. If it is more than the specified limit, the traffic is recognized as the attack traffic.

2.2 DDoS Attack in WiMAX Networks

The Mobile WiMAX is a broadband wireless technology that some of its security vulnerabilities are not much explored [1]. In [4] Youngwook et al. exploited the unused most significant 64 bits of the 128-bit Cipher-based Message Authentication Code (CMAC) which is designed to provide the integrity of management message. They used DREG REQ message to extract SAI and use it to defend the Mobile WiMAX network against DDoS attack. In fact they proposed a method to defend the network against the DDoS attack.

Since it is assumed that high-volume attack traffic causes significant changes in the power spectral density of traffic and few works have been done in understanding the analysis capability provided by a set of entropy metrics in conjunction with one another, our proposed method analyzes the network traffic by extracting more effective features. Since WiMAX networks are new technology few works have been done in these networks, so there is no real data set to compare the achieved results with the results of a real attack.

3 DDoS Attack on WiMAX Network

In this section interrupt attacks on WiMAX Networks such as DoS attacks are studied in brief.

3.1 Reference Network Model

The Mobile WiMAX Reference Network Model (RNM) consists of Access Service Network (ASN) and Connectivity Service Network (CSN) as shown in Fig. 1[4]. Constituted of several BSs and an ASN Gateway (ASN-GW) ASN provides radio connectivity service for its mobile subscribers (MS) and CSN offers IP access service to a number of ASN-GWs. A BS provides direct radio access to the MSs in its cell and ASN GW connects the BSs in its paging group to the CSN. (The ASN coordinates traffic across multiple Base Transceiver Stations (BTS) and supports security, handoffs and Quality of Service (QoS). Typically the ASN includes numerous BTSs with one or more ASN gateways. The ASN manages radio resources, MS access, mobility, security and QoS. It acts as a relay for the CSN for IP address allocation and AAA functions. The ASN gateway hosts the Mobile IP Foreign Agent (FA). The CSN performs core network functions, including policy and admission control, IP address allocation, billing and settlement. It hosts the Mobile IP Home Agent (HA), the IP and AAA servers, and PSTN and VoIP gateways. The CSN is also responsible for internetworking with non-WiMAX networks (e.g. 3G, DSL) and for roaming through links to other CSNs.

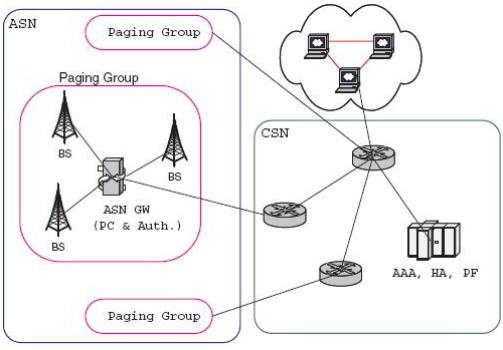


Fig. 1. WiMAX Reference Network Model

3.2 Bandwidth Allocation for Ranging

When a subscriber station (SS) attempts initial entry to the network, first it requests bandwidth by using RNG-REQ and Ranging Response (RNG-RSP) messages [12]. Figure 2 shows the initial entry process.

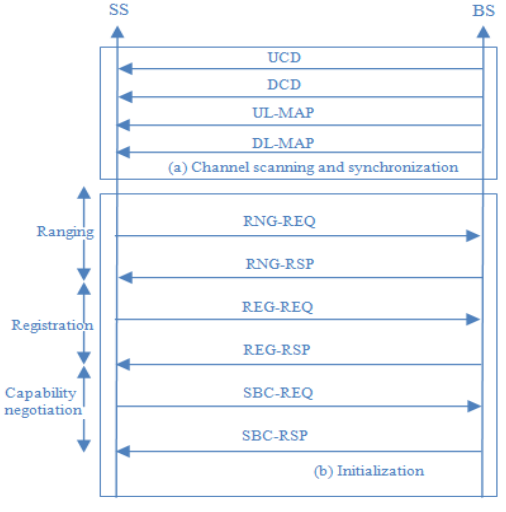


Fig. 2. Initial entry process

In its down link (DL) channel, the BS decides on allocating bandwidth to SSs per CID (Connection Identifier, connections are identified by a connection identifier. A 16-bit value that identifies a connection or an uplink/ downlink pair of associated management connections to equivalent peers in the medium access control layer of the BS and SS. The CID address space is common between DL and UL and partitioned among the different types of connections) basis and does not requires the SSs to be involved. The BS always reserve required bandwidth for ranging interval

which is indicated in Uplink-MAP (UL-MAP). To get bandwidth for ranging, an SS chooses an appropriate ranging code to transmit during the ranging interval. On successful reception of the code, the BS assigns proper bandwidth to the SS for ranging [13]. As Figure 2 shows, there is no authentication or authorization for granting bandwidth any SS can request bandwidth for ranging. This procedure provides a possibility of DDoS attack to BS and ASN-GW for malicious SSs to generate as many fraudulent requests as they intend to [4].

3.3 Attacks to BS and ASN GW

To connect to a network, an SS attempts to determine whether it is in the coverage of a suitable WiMAX network first. The SS stores a permanent list of all operational parameters of the connecting network, such as the DL (Downlink, the direction from the base station to the subscriber station) frequency used during the previous connection operation [12]. The MS first attempts to synchronize with the stored DL frequency if this fails, then scans other frequencies in an attempt to synchronize with the DL of the most suitable BS. During the DL synchronization, the MS listens for the DL frame preambles. When one is detected the MS can synchronize itself to the DL transmission of the BS. Once it obtains DL synchronization, the MS listens to the various control messages, such as DCD, UCD, DL-MAP, and UL-MAP, that follow the preamble to obtain the PHY- and MAC related parameters corresponding to the DL and UL (Uplink, the direction from a subscriber station to the base station) transmissions [13]. The initial entry process has 6 stages. Stage 1 and 2 that include synchronization and initial ranging are shown in figure 2. A RNG-REQ is transmitted by the SS at initialization periodically to determine network delay and to request power and/or DL burst profile change [12]. This message has a standard syntax that specifies type, size and the value of the request. An attacker may misuse RNG-REQ message, changing some fields randomly and send it to BS in large volume to waste the resources of the network.

4 Detection System

The goal of the proposed system is to provide an environment to analyze the network traffic under the DDoS attack more efficiently. In this section first statistical preliminaries will be introduced, and then an efficient traffic modeling for extracting features will be presented.

4.1 Statistical Preliminaries

To perform the DDoS attack, the attacker usually sends large amount of RNG-REQ messages to the BS changing the address field of the messages randomly. So we can exploit entropy concept to measure the dispersion of the addresses sent to the BS, because in the attack state the dispersion of the addresses is higher than the normal state.

The entropy of a random variable X measures the uncertainty of X , and is defined as:

$$E(X) = -\sum_{i=1}^{n} P(X = x_i) \log_2 P(X = x_i) \quad (1)$$

Where $x_1, x_{2,\ldots,} x_n$ are the values within the given range for X and $P(x_i)$ shows the probability that X takes the value x_i and is defined as:

$$P(X = x_i) = \frac{m_i}{m} \quad (2)$$

Where m_i is the number of the messages with x_i as the destination address and m represents the number of the total addresses within the epoch.
The normalized entropy is defined as:

$$E_n(X) = \frac{E(x)}{\log N_0} \quad (3)$$

Where N_0 represents the number of different addresses within the specified epoch and $\log N_0$ is the normalization factor.

In mobile WiMAX networks whenever an SS try to join the network, it should send a RNG-REQ message to BS; moreover, there are other situations in which the SSs send RNG-REQ messages toward the BS. So in detecting the attack we can consider just RNG-REQ messages sent in the initial network entry. The situations in which an SS also sends RNG-REQ messages to BS are as follow:

- Re-entry to network from idle mode: If an MS has some pending traffic or its security context is going to expire, it should perform re-entry to network from idle mode. Also after De-Registering from the network, the SS enters idle mode. Idle mode re-entry to network is the same as the initial network entry except that some procedures are omitted. In order to perform re-entry to network from idle mode, the MS should send RNG-REQ message to the BS, but in this case some parameters of the messages are different.
- Keep-alive check in sleep mode: In order for a BS to maintain supervision of MSs in sleep mode and to perform necessary adjustments, BS may implement a keep-alive check mechanism which includes sending RNG-REQ message to BS.
- Handover: When a mobile station tries to switch between two BSs and migrates from the air-interface provided by one base station to the air-interface provided by another BS, it should perform handover process. In order to perform handover it should send a RNG-REQ message to BS.
- Location update: There are two location update procedures, secure location update and unsecure location update. In secure location update, MS first sends a RNG-REQ message including the CMAC tuple to the BS.

In all the above situations since the MS has once joined the network, a CID has been assigned to it which is different from the CID assigned to the MS during the initial

network entry process. So RNG-REQ messages sent to BS in other situations except the initial network entry can be seprated using the conditional entropy. So we can use Conditional Entropy to get more information from the received messages. The conditional entropy quantifies the remaining entropy or uncertainty of a random variable Y given that the value of another random variable X is known.

The Conditional Entropy measures the uncertainty of the variable X considering the variable Y and is defined as:

$$H(X|Y) = -\sum_{i=1}^{n}\sum_{j=1}^{m}(P(X = x_i, Y = y_j).\log P(X = x_i \mid Y = y_j)) \quad (4)$$

Where $P(X|Y)$ represent the probability of X considering Y, and is defined as:

$$P(X|Y) = \frac{P(X \cap Y)}{P(Y)} \qquad P(Y) > 0 \quad (5)$$

Considering X as the RNG-REQ messages with x_i as the MAC address sent to BS and Y as the RNG-REQ messages sent to BS as none of the following purpose,

- Re-entry to network from idle mode
- Keep-alive check in sleep mode
- Handover
- Location update

We calculate the conditional entropy of the message to extract more efficient information from network traffic to detect the attack.

Another statistical concept which helps us in detecting the attack is called Mutual Information. The mutual information parameter I (X;Y) is defined as:

$$I(X;Y) = H(X) - H(X|Y) \quad (6)$$

Note that before considering Y, the uncertainty of X is H(x), after observing and considering it, this uncertainty goes down to H(x|y). Therefore, the mutual information measures the amount of information we learn about X by considering Y. In our analysis, we would like to estimate the value of X based on the observation of Y, which includes the real attack messages and the normal legitimate accessing messages. Recall that the conditional entropy measures how much uncertainty remains for X given considering Y.

4.2 Modeling the Network Traffic

The arrival process of packets to the network has been mostly modeled by Poisson process; however, recently it is shown that the self-similar model is more appropriate for heavy loaded network traffic [14]. The traffic of internet based protocols show that traffic variation exists in large time slots and the traffic in smaller slots is correlated to the traffic in larger slots. In fact, the traffic of heavy loaded networks can

be modeled by fractal time series which have the self-similarity and Long Range Dependency (LRD) properties [15]. Based on the LRD properties of the traffic, in the phase of extracting network traffic features, at the end of specified given epochs a vector is extracted. The more epochs are smaller, we have less delay, but we cannot choose the epochs smaller than a specified threshold, because in this case we may lose some useful data. The dataset is split in to non-overlapping epochs consisting of flows that completed within.

5 Experimental Results and Discussion

Most of the works done so far, assume that low volume of traffic has been used for DDoS attack, and the target network is not centralized. In contrast, the recent Mobile WiMAX networks are centralized and expected to handle considerable volume of data traffic, so that DDoS attack needs to be revisited. In our research, to analyze the attack traffic, first the Reference Network Model (RNM) is simulated by OPNet where some of the SSs generate the attack traffic. Our data set uses flow data captured in the OPNet simulation environment consisting of the traffic from normal and malicious SSs sent to the BS. In the following the entropy resulted from network traffic is presented.

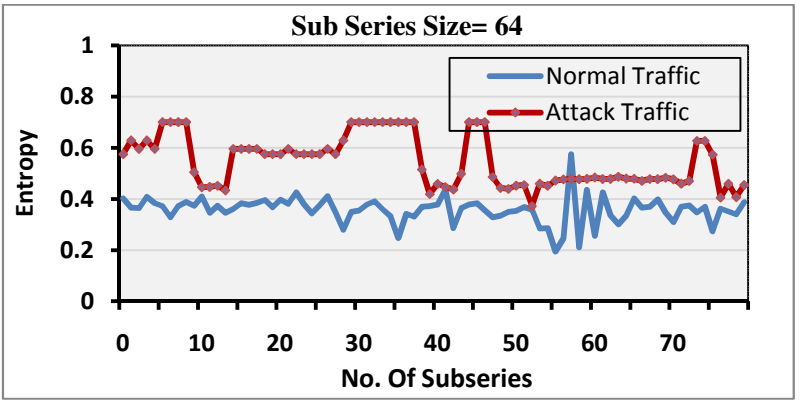


Fig. 3. The Entropy of the address of received messages, subseries=64

Table 1. Max, min and average value of the Entropy in figure 3

Entropy	Average	Min Value	Max Value
Normal Traffic	0.35797674	0.194652912	0.574149885
Attack Traffic	0.561667515	0.371713305	0.7

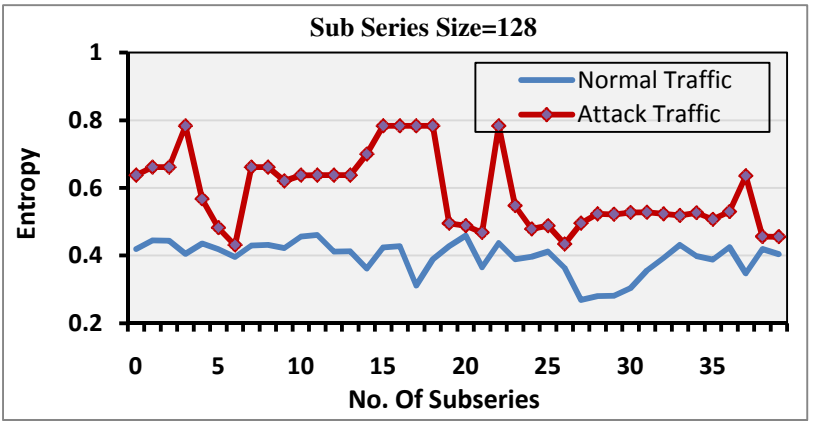


Fig. 4. The Entropy of the address of received messages, subseries=128

Table 2. Max, min and average value of the Entropy in figure4

Entropy	Average	Min Value	Max Value
Normal Traffic	0.403591572	0.311654769	0.460956424
Attack Traffic	0.626945912	0.43119818	0.783333333

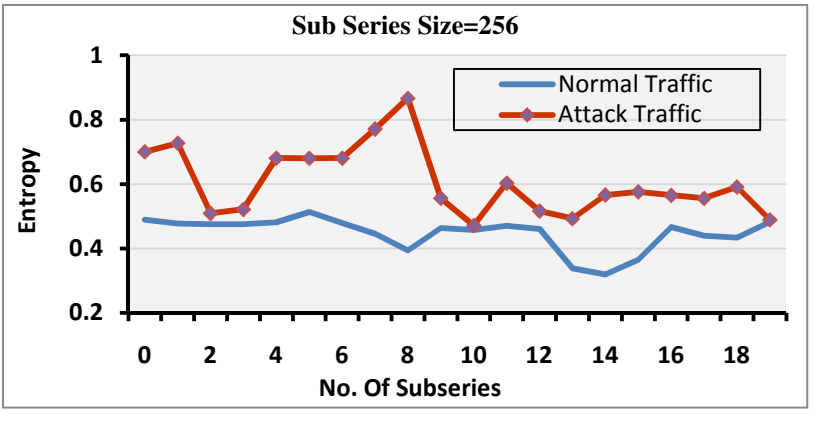


Fig. 5. The Entropy of the address of received messages, subseries=256

Table 3. Max, min and average value of the Entropy in figure5

Entropy	Average	Min Value	Max Value
Normal Traffic	0.480466913	0.475319062	0.512722339
Attack Traffic	0.636584255	0.509325773	0.727359224

Assuming the size of subseries to be 64, 128 and 256 respectively, the figures 5, 6 and 7 show the entropy value of the MAC address of the received messages in normal and attack state. The tables 1, 2 and 3 show the minimum, maximum and average value of the estimated entropy in these three cases. According to the irregular dispersion of the addresses in the attack state, the estimated entropy in the attack state is more than the estimated entropy of the normal state. On the other hand, estimating the entropy based on different subseries size show that the entropy value of the network traffic with subseries size of 128 varies between 0.3116 and 0.4609 in normal state and in the attack state it varies between 0.4311 and 0.7833. By making the subseries bigger and assuming the subseries size to be 256, it is seen that the entropy varies between 0.4753 and 0.5127 in the normal state and between 0.5093 and 0.7273 in the attack state. By choosing smaller subseries size, it is seen that the results are to some extent closer to results achieved by subseries size of 128. Comparing the results in three different condition shows that big subseries have less precision and give us less information about the network traffic in comparison with subseries of smaller size. Also it is shown that analyzing the network traffic with subseries size of 64 has more precision than the other two conditions, because there is a bigger difference between the entropy in the normal and attack state which help us to detect the attack more precisely. The more the difference between the entropy of the attack state and normal state is higher, the more precise the detection of the attack is. After calculating the entropy of the received messages, to get more information from the network traffic the conditional entropy of the network traffic is estimated. Then based on the entropy and conditional entropy, more precise information can be extracted from the network traffic to help us in detecting the attack which is called Mutual Information. In the following first the results achieved by calculating the conditional entropy will be presented then the mutual information of the entropy and conditional entropy will be presented.

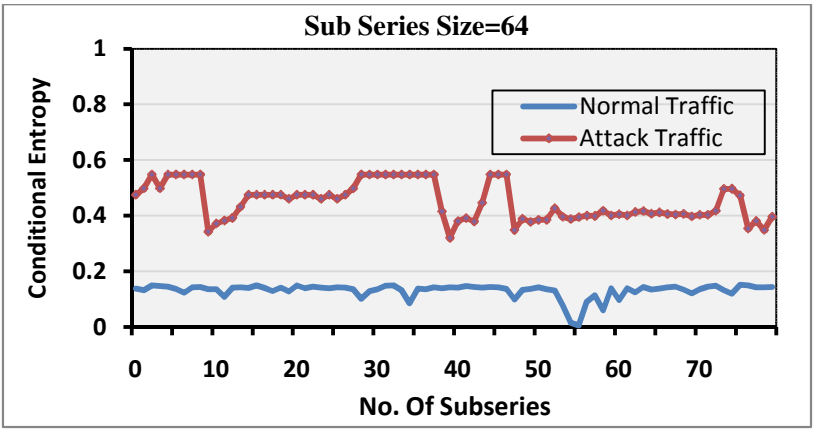


Fig. 6. The Conditional Entropy of the address of received messages, subseries=64

Table 4. Max, min and average value of the Conditional Entropy in figure6

Conditional Entropy	Average	Min Value	Max Value
Normal Traffic	0.12878809	0.00565246	0.150011
Attack Traffic	0.460910558	0.32036791	0.547494

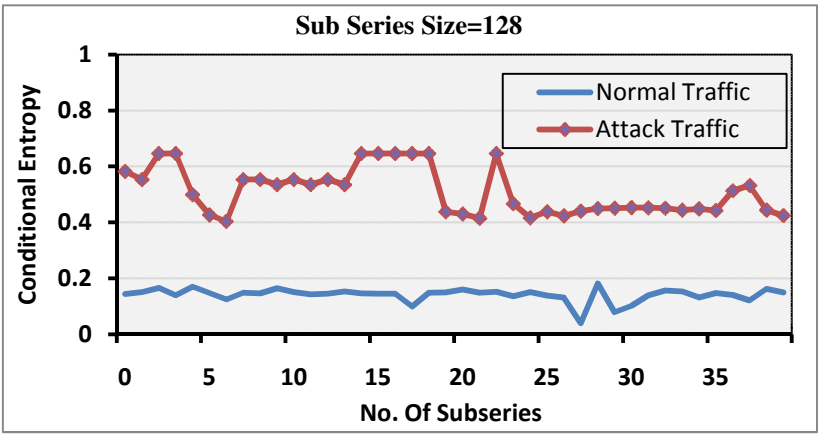


Fig. 7. The Conditional Entropy of the address of received messages, subseries=128

Table 5. Max, min and average value of the Conditional Entropy in figure7

Conditional Entropy	Average	Min Value	Max Value
Normal Traffic	0.14735965	0.099296019	0.170450176
Attack Traffic	0.54489437	0.40383335	0.646462667

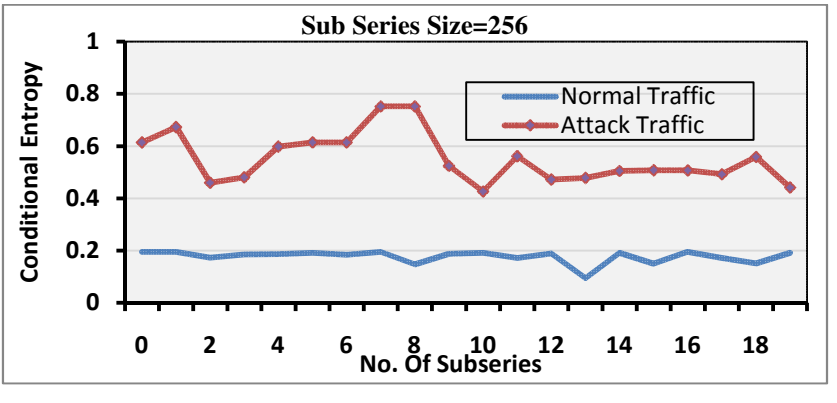


Fig. 8. The Conditional Entropy of the address of received messages, subseries=256

Table 6. Max, min and average value of the Conditional Entropy in figure8

Conditional Entropy	Average	Min Value	Max Value
Normal Traffic	0.186677367	0.172669932	0.195040818
Attack Traffic	0.579264891	0.459405771	0.673525469

Assuming the size of subseries to be 64, 128 and 256 respectively, the figures 8, 9 and 10 show the conditional entropy value of the MAC address of the received RNG-REQ messages considering the four states mentioned in the previous section, in normal and attack state. The tables 4, 5 and 6 show the minimum, maximum and average value of the estimated conditional entropy in these three cases. According to the irregular dispersion of the addresses in the attack state and the estimated conditional entropy, it is seen that the estimated conditional entropy in the attack state is more than the estimated value in the normal condition, but on the other hand by comparing it with the estimated entropy it is seen that it has gone down to some extent, because in this case some of the messages are ignored and are not considered. Moreover, comparing the resulted graphs with three different subseries show that with subseries size of 256 the conditional entropy varies between 0.172 and 0.1950 in the normal state and between 0.4594 and 0.6735 in the attack state, but by assuming the subseries size smaller, it is seen that in the case of subseries size of 64 it varies between 0.0056 and 0.15 in the normal state and between 0.3203 and 0.5474 in the attack state. So it can be concluded that smaller subseries size are more precise than bigger ones and give us more information about the network traffic.

Following graphs present the Mutual Information parameter information of the network traffic, achieved by the estimated Entropy and Conditional entropy of the network traffic.

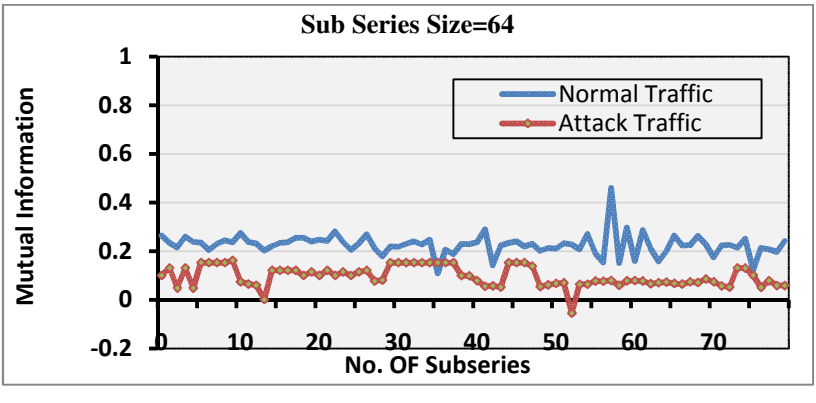


Fig. 9. The Mutual Information of the Entropy and Conditional entropy of received messages, subseries=64

Table 7. Max, min and average value of the Mutual Information parameter in figure9

Mutual Information	Average	Min Value	Max Value
Normal Traffic	0.229157	0.109292	0.46005
Attack Traffic	0.096786	-0.05491	0.161787

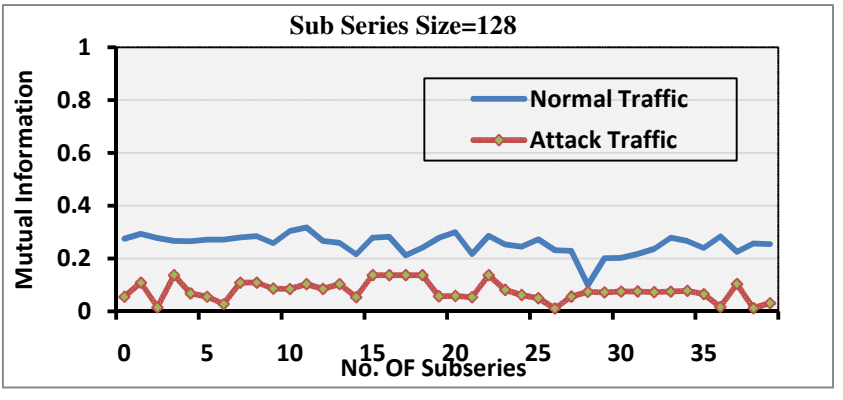


Fig. 10. The Mutual Information of the Entropy and Conditional entropy of received messages, subseries=128

Table 8. Max, min and average value of the Mutual Information parameter in figure10

Mutual Information	Average	Min Value	Max Value
Normal Traffic	0.255822	0.098504321	0.317333
Attack Traffic	0.081454	0.010591804	0.136871

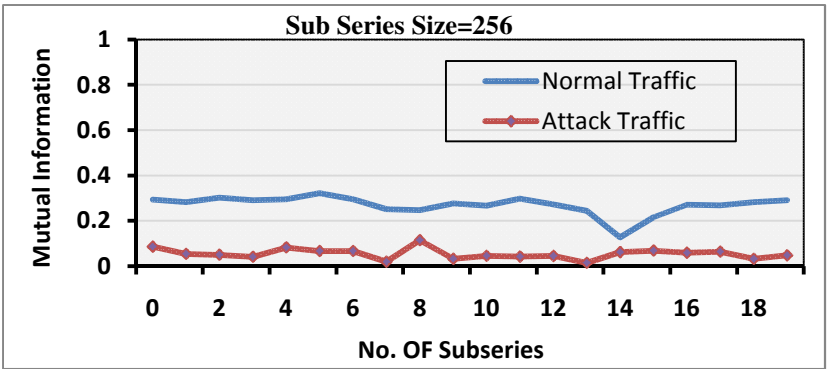


Fig. 11. The Mutual Information of the Entropy and Conditional entropy of received messages, subseries=256

Table 9. Max, min and average value of the Mutual Information parameter in figure11

Mutual Information	Average	Min Value	Max Value
Normal Traffic	0.297452	0.282255	0.32227
Attack Traffic	0.063164	0.040992	0.085665

Assuming the size of subseries to be 64, 128 and 256 respectively, the figures 11, 12 and 13 show the Mutual Information parameter value of the MAC address of the received RNG-REQ messages considering Entropy and Conditional Entropy value of the received traffic. The tables 7, 8, and 9 present the maximum, minimum and average value of the estimated Mutual Information parameter. The resulted Mutual Information parameter shows that the value of this parameter in normal state is higher than its value in attack state. Because in the normal state some of the RNG-REQ messages sent to BS may be sent as one of the four purposes mentioned in the previous section, so there is a difference between the resulted Entropy and Conditional Entropy, while in the attack state the ratio of these kinds of messages is low and there is no so much difference between the estimated Entropy and Conditional Entropy. According to the resulted graphs, in the attack state, the Mutual Information parameter varies between 0.2822 and 0.3222 when the size of subseries is considered to be 256 in the normal state and between0.0409 and 0.0856 in the attack state. By changing the subseries size to a smaller size, 64, the Mutual information parameter varies between 0.1092 and 0.96 in the normal state and between -0.0549 and 0.1617 in the attack state. So considering the estimated results and this fact that, the more the difference between Entropy and conditional Entropy is less the more information can be extracted from the network traffic, we conclude that totally the precision of the resulted parameters in the case of subseries size of256 is more than the other two cases.

6 Conclusion and Future Works

In this paper we presented an analytical model for WiMAX network traffic under DDoS attack and evaluated the Entropy, Conditional Entropy and Mutual Information parameters of the traffic in both normal and attack states. According to the simulation results, as the statistical properties of the attack traffic pattern differ from the ones for the normal traffic pattern, the attack can be detected using extracted statistical properties of the traffic. The difference between our proposed system and the existing system is that we had a different look at the arrival process of the messages to the BS. Since recently it is shown that the self-similar model is more appropriate for heavy loaded network traffic, we exploited this property and evaluated the network traffic with three different subseries size. For the future work, the results of this paper can be used to train an artificial neural network to detect DDoS attacks. Artificial Neural Networks in order to distinguish between the attack state and normal state need some feature to feed the network, so the results of this research will be used to train the network and identify the attack.

Acknowledgement. This work is supported by Iran Telecommunication Research Center (ITRC).

References

[1] Vafea, A.: Security of IEEE 802.16. Master of Information and Communication Systems Security, Department of Computer and Systems – Science Royal Institute of Technology (2006)
[2] Jamshed, H.: Security Issues of IEEE 802.16 (WiMAX), School of Computer and Information Science, Edith Cowan University, Australia (2006)
[3] Eren, E.: WiMAX Security Architecture – Analysis and Assessment. In: IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications Dortmund, Germany, September 6-8 (2007)
[4] Youngwook, K., Hyoung-Kyu, L., Saewoong, B.: Shared Authentication Information for Preventing DDoS attacks in Mobile WiMAX Networks. In: IT R&D program of MIC/IITA. IEEE, Korea (2007)
[5] Shon, T., Choi, W.: An Analysis of Mobile WiMAX Security: Vulnerabilities and Solutions. In: Enokido, T., Barolli, L., Takizawa, M. (eds.) NBiS 2007. LNCS, vol. 4658, pp. 88–97. Springer, Heidelberg (2007)
[6] Boom, D.: Denial of Service Vulnerabilities in IEEE 802.16 Wireless Networks. Master Thesis at Naval Postgraduate School Monterey. IEEE, California (2004)
[7] Nasreldin, M., Aslan, H., El-Hennawy, M., El-Hennawy, A.: WiMAX Security. In: 22nd International Conference on Advanced Information Networking and Applications, IEEE (2008)
[8] Lee, K., Kim, J., Kwon, K.H., Han, Y., Kim, S.: DDoS attack detection method using cluster analysis. ESWA 34, 1659–1665 (2008)
[9] George Nychis, V.S., Andersen, D.G., Kim, H., Zhang, H.: An Empirical Evaluation of Entropy-based Traffic Anomaly Detection. In: IMC 2008. ACM, Greece (2008)
[10] Zhou, W., Yu, S.: Entropy-Based Collaborative Detection of DDOS Attacks on Community Networks. In: Sixth Annual IEEE International Conference on Pervasive Computing a and Communication (2008)
[11] Kar, S., Sahoo, B.: An Anomaly Detection System for DDoS Attack in Grid Computing. International Journal of Computer Applications in Engineering, Technology and Sciences (ij-ca-ets) 1, 553 (2009)
[12] IEEE Standard 802.16-2009: Air Interface for Broadband Wireless Access Systems (May 2009)
[13] Taylor & Francis Group, WiMAX/MobileFi, Auerbach (2008) ISBN 978-1-4200-4351-8
[14] Karagiannis, T., Molle, M., Faloutsos, M.: Long Range Dependence. IEEE Computer Society (2004) 1089-7801/04/$20.00
[15] Millán, G., Lefranc, G.: Presentation of an Estimator for the Hurst parameter for a Self-similar Process Representing the Traffic in IEEE 802.3 Networks. Int. J. of Computers, Communications & Control IV(2), 137–147 (2009) ISSN 1841-9836, E-ISSN 1841-9844

A Mathematical Problem for Security Analysis of Hash Functions and Pseudorandom Generators

Koji Nuida[1], Takuro Abe[2], Shizuo Kaji[3],
Toshiaki Maeno[4], and Yasuhide Numata[5,6]

[1] Research Center for Information Security (RCIS), National Institute of Advanced Industrial Science and Technology (AIST), Ibaraki, Japan
k.nuida@aist.go.jp
[2] Department of Mechanical Engineering and Science, Kyoto University, Kyoto, Japan
abe.takuro.4c@kyoto-u.ac.jp
[3] Department of Mathematical Sciences, Faculty of Science, Yamaguchi University, Yamaguchi, Japan
skaji@yamaguchi-u.ac.jp
[4] Department of Electrical Engineering, Kyoto University, Kyoto, Japan
maeno@kuee.kyoto-u.ac.jp
[5] Department of Mathematical Informatics, The University of Tokyo, Tokyo, Japan
numata@stat.t.u-tokyo.ac.jp
[6] Japan Science and Technology Agency (JST), CREST, Japan

Abstract. The aim of this paper is to emphasize the significance of a certain mathematical problem in research on information security. We point out that the mathematical problem, which we refer to as "Function Density Problem," has connections to the following two major cryptographic topics; security analysis of hash functions in the real world (like SHA-1), and construction of pseudorandom generators with some enhanced security property. We also provide a first example to show how a study of Function Density Problem can contribute to the progress of the above-mentioned two topics. Other potential applications of Function Density Problem to information security are also discussed.

Keywords: Function Density Problem, hash function, pseudorandom generator, security evaluation.

1 Introduction

1.1 Background and Related Works

It is widely understood that some mathematical problems have been playing indispensable roles in research on cryptography and information security. For instance, the (expected) difficulty of integer factorization is the source of security of RSA cryptosystem [7], while the problem of solving multivariate quadratic (MQ)

T. Iwata and M. Nishigaki (Eds.): IWSEC 2011, LNCS 7038, pp. 144–160, 2011.

equations got attracting several studies after the development of Matsumoto-Imai cryptosystem [5] and its variants, whose constructions are closely related to MQ equations. It is expected that an interesting mathematical problem relevant to some cryptographic topics can contribute to the progress on such topics.

The aim of this paper is to emphasize the significance of a certain mathematical problem, which has connections to the following two major topics in information security; security analysis of hash functions in the real world (like MD5 and SHA-1), and construction of pseudorandom generators with some enhanced security property. First, we give some descriptions of these two topics.

Security Analysis of Hash Functions. Intuitively, a hash function is a function (mapping) $H\colon X \to Y$ from some (finite) set X to another (finite) set Y that possesses a certain desirable security property. When we concern efficiency or computability of H, we consider an algorithm that computes the function H (also denoted by H) and call it hash algorithm. A standard security requirement for hash functions is *collision resistance*, which informally means that it is difficult to find a collision pair (x_1, x_2) for H, i.e., $x_1 \neq x_2 \in X$ satisfying $H(x_1) = H(x_2)$. Hash functions have been playing central roles in various information security applications, and secure hash functions for real-life applications are usually expected to possess the collision resistance property.

However, most of the preceding successful studies for showing security of hash functions actually dealt with *keyed* hash functions (or hash *families*); intuitively, a family of hash functions H_k parameterized by a key k is called collision resistant if, for any (efficient) adversary, the attack to find a collision pair of H_k fails for a randomly chosen key k with high probability. Several constructions of keyed hash functions have been proposed so far (e.g., [2]). The above security notion of keyed hash functions can be interpreted as allowing one to (randomly) choose a concrete instance H_k of the hash family after an adversary is given. In contrast, in most of real-life applications, the concrete instance of hash algorithms is specified first, and then an adversary can try to attack the fixed hash algorithm. This reversal of order causes a crucial difficulty in guaranteeing (or even usefully formalizing) security of a non-keyed hash algorithm H, as (unless the trivial situation where the domain of H is not larger than the image of H) there *does* always exist a collision pair (x_1, x_2) for H and any adversary (existing in theory) who innately knows the pair (x_1, x_2) is obviously able to efficiently attack the H. In fact, even an instance of standardized or de facto standard hash algorithms has been suffered from feasible attacks (e.g., [9]). In this paper, we try to propose a theoretical, unified way to say (preferably affirmative) something about security of a concrete instance of hash algorithms.

Regarding related works, Rogaway [8] gave a detailed observation about the difference between "inexistence of effective attack algorithms" and "lack of knowledge on construction of effective attack algorithms" for non-keyed hash algorithms. He emphasized the difference of the two situations (by the term "human ignorance"), and discussed how to prove security of a cryptographic protocol by reducing the security into that of the hash algorithm which the protocol uses internally. However, he did not discuss how to evaluate security of non-keyed hash

algorithms themselves, which we study in this paper. On the other hand, in this paper we adopt the concrete security formulation rather than the asymptotic one; some observation for security of non-keyed hash algorithms in asymptotic security formulation is also given in Rogaway's paper.

Construction of Enhanced Pseudorandom Generators. A *pseudorandom generator* (*PRG*) is an algorithm $G\colon S \to X$ with (finite) set S of inputs (*seeds*) and (finite) output set X such that, when a seed $s \in S$ is chosen uniformly at random, the output $G(s) \in X$ of G is also "random" in some sense. Conventionally, the meaning of "randomness" here is formulated by using the notion of *distinguisher*, which is an algorithm $D\colon X \to \{0,1\}$ with 1-bit output and the input set being the output set X of G. In this paper we adopt concrete security formulation rather than asymptotic one, in which case a major security requirement for PRGs is (T,ε)-security; namely, G is called (T,ε)-*secure* if, for any distinguisher D for G with (time) complexity bounded by T, the statistical distance between the output distribution $D(G(U_S))$ of D with input given by G with uniformly random seed $s \in S$ (referred to as "pseudorandom input") and the output distribution $D(U_X)$ of D with uniformly random input $x \in X$ (referred to as "random input") is bounded by ε [4]. (Intuitively, any such D cannot distinguish the random element x of X and the pseudorandom element $G(s)$ with significant advantage.) There are a large number of constructions of PRGs, most of which are provably secure (possibly in asymptotic security formulation) under standard assumptions such as Factoring Assumption and Decisional Diffie-Hellman Assumption (e.g., [1,4]).

On the other hand, in a recent work of Dubrov and Ishai [3], an enhanced notion for PRGs, called *pseudorandom generators that fool non-boolean distinguishers* (*nb-PRGs* in short), was proposed. This notion is obtained by modifying the above-mentioned original security notion by allowing the distinguishers D to have larger output sets; namely, G is called (T,n,ε)-*secure* if, for any "non-boolean" distinguisher $D\colon X \to Y$ for G with (time) complexity bounded by T *and output set* Y *of size at most* n, the statistical distance between the output distributions of D with random and pseudorandom inputs is bounded by ε. In their paper Dubrov and Ishai showed interesting applications of nb-PRGs, e.g., secure pseudorandomization, without any restriction on computational complexity of the adversary's attack algorithm, of a certain kind of information-theoretically secure protocols.

However, constructing secure nb-PRGs seems much more difficult than the case of the usual PRGs. In fact, to the authors' best knowledge, the only constructions of nb-PRGs proposed so far are ones in the original paper of Dubrov and Ishai [3], which are based on certain non-standard computational assumption. Hence it will be fruitful if we can give some implication result such that any PRG (in the usual sense) with a certain parameter is also an nb-PRG with a (possibly different) parameter. In fact, a straightforward implication has been mentioned in the original paper, but this is far from being efficient (i.e., to obtain nb-PRGs with reasonable security parameters, the original PRGs are required

to have somewhat impractical security parameters). In this paper, we try to establish a more efficient implication result.

1.2 Our Contributions, and Organization of This Paper

In Section 2, we propose a mathematical problem, which we refer to as "Function Density Problem". Intuitively, this problem is to evaluate the possibility of close approximations of arbitrary functions by using some "easy" functions.

Then we introduce motivating applications of Function Density Problem to two topics in information security. First, in Section 3, we discuss theoretical analysis of collision resistance of non-keyed hash algorithms. We give an abstract "typical" attack strategy for a hash algorithm, in which one is supposed to make use of a close approximation of the target hash algorithm. We point out the relevance of Function Density Problem to this attack strategy.

Secondly, in Section 4, we discuss an enhanced security notion for PRGs (called nb-PRG) introduced by Dubrov and Ishai [3]. We give a new implication result (Theorem 1) showing that any secure PRG is also a secure nb-PRG, with somewhat modified security parameter. We point out that an application of Function Density Problem can contribute to make the implication more efficient.

Then in Section 5, we describe a concrete example of Function Density Problem, in order to arise some image or intuition of how a study of Function Density Problem can proceed. More detailed studies of this problem will be a future research topic.

Finally, in Section 6 we give a concluding remark, which includes a discussion on further possible applications of Function Density Problem in information security. We also include an appendix for a proof of a lemma in Section 3.

2 Function Density Problem

In this section, we introduce our proposed mathematical problem, which we call *Function Density Problem*. First we give a general description of the problem. Let $\mathcal{C}$ be a set of some functions, and let $\mathcal{C}'$ be a certain subset of $\mathcal{C}$. Moreover, suppose that to any pair of functions $f, g \in \mathcal{C}$, a "distance" denoted by $d(f, g)$ is associated in some manner. Then our problem is formulated as follows:

Definition 1 (Function Density Problem). *Under the above setting,* Function Density Problem *is the following problem: Determine (or give some estimate of) the quantity*

$$r(\mathcal{C}; \mathcal{C}') = \max\{d(f; \mathcal{C}') \mid f \in \mathcal{C}\} \tag{1}$$

(the symbol 'r' stands for "radius"), where, for each $f \in \mathcal{C}$, $d(f; \mathcal{C}') = \min\{d(f, g) \mid g \in \mathcal{C}'\}$ is the distance from f to $\mathcal{C}'$ induced by the function $d(\cdot, \cdot)$.

Remark 1. Although in a very general situation the maximum or minimum in the above definition may not exist (therefore we should use "sup" and "inf" instead of the "max" or "min"), these maximum and minimum indeed exist for

all situations discussed in this paper. For example, the maximum and minimum always exist when $\mathcal{C}$ is a finite set and the distance $d(f, g)$ is (as usual) a real number for any $f, g \in \mathcal{C}$.

Actually, the above formulation alone covers too wide situations to make some significant observation; we should suppose some concrete properties or structures of the sets $\mathcal{C}, \mathcal{C}'$ and the distance function $d(\cdot, \cdot)$ (which will depend on the concrete applications of this problem). In the remaining part of this paper, we adopt the following setting that is relevant to the applications discussed in later sections. Let $\mathcal{C}$ be the set of all functions $f\colon X \to Y$ from a given finite set X to a given set Y. Let $\mathcal{C}'$ be a set of functions $f \in \mathcal{C}$ which are "easy" in certain sense (specified in later arguments). Moreover, for any $f, g \in \mathcal{C}$, we define the distance between f and g by

$$d(f, g) = |\{x \in X \mid f(x) \neq g(x)\}| \ . \tag{2}$$

Note that, under identification of members of $\mathcal{C}$ with sequences of length $|X|$ over the alphabet Y, this definition coincides with the (generalized) Hamming distance. Intuitively, we regard a member g of $\mathcal{C}'$ as a close approximation of a given $f \in \mathcal{C}$ when the distance $d(f, g)$ is small, as such an f can be converted to g by changing its values at only a small number of points in the domain. Then the quantity $d(f; \mathcal{C}')$ would mean the potential of a function f by some "easy" function, and $r(\mathcal{C}; \mathcal{C}')$ would mean the potential of an *arbitrary* function $X \to Y$ by some "easy" function. In other words, it can be said that, the smaller the quantity $r(\mathcal{C}; \mathcal{C}')$ is, the more densely the "easy" functions are included in the set $\mathcal{C}$. How Function Density Problem can be related to some topics in information security is explained in the following sections.

Remark 2. It would be worth emphasizing that, even if the quantity $d(f; \mathcal{C}')$ is small in the above situation, it does *NOT* mean that a close approximation $g \in \mathcal{C}'$ of the f is efficiently computable. The fact that $d(f; \mathcal{C}')$ is small only guarantees the *existence* of such an approximation g, and how to construct it in practice is a different (possibly difficult) problem. (This would be also relevant to "human ignorance" observation in [8] mentioned in Section 1.1.)

3 Hash Functions and Function Density Problem

In this section, we point out a relation between security analysis of (non-keyed) hash functions and Function Density Problem introduced in Section 2. Let $H\colon X \to Y$ be a given concrete instance of hash algorithms, with input set X and output set Y being both assumed to be finite for simplicity. One of the goals of the cryptanalysis for H is to practically find a collision pair (x_1, x_2) for H (recall that (x_1, x_2) is called a collision pair for H if $x_1, x_2 \in X$, $x_1 \neq x_2$ and $H(x_1) = H(x_2)$). To motivate the relevance of Function Density Problem to this situation, first we informally describe an abstract "typical" strategy for finding a collision pair:

1. Construct a close approximation $H'\colon X \to Y$ of H such that collision pairs for H' can be found with reasonable computational time.
2. Find randomly a collision pair (x'_1, x'_2) for H'.
3. Construct from (x'_1, x'_2) a candidate (x_1, x_2) of a collision pair for H (in the simplest case we just set $(x_1, x_2) = (x'_1, x'_2)$).
4. Check if (x_1, x_2) is indeed a collision pair of H.
5. If (x_1, x_2) is not a collision pair for H, go back to Step 2 and repeat the process.

Intuitively, the number of iterations in the above strategy before finding a collision pair for H would be expected to be small if the approximation H' is sufficiently close to H (see Lemma 1 below for a quantitative expression of this expected tendency). Hence security of a hash algorithm H against such an attack strategy is related to the possibility of finding its close approximation.

More precisely, we set $(x'_1, x'_2) = (x_1, x_2)$ in the above strategy for simplicity. Let the set $\mathcal{C}$ in Function Density Problem be the set of all functions from X to Y. Let the distance $d(f, g)$ between $f, g \in \mathcal{C}$ be defined as in (2). Moreover, let $\mathcal{C}'$ be a subset of $\mathcal{C}$ such that any hash function H' in $\mathcal{C}'$ admits an efficient attack by a known attack strategy. In the above attack strategy, the approximation H' for H is supposed to be chosen from $\mathcal{C}'$. Now we have the following lemma:

Lemma 1. *Suppose that* $H, H'\colon X \to Y$ *with* $|Y| = n \geq 2$, *and* $d(H, H') = d$, $0 < d < |X|$. *Then the probability that a collision pair for* H' *chosen uniformly at random is also a collision pair for* H *is higher than or equal to*

$$\frac{2\alpha_0|X| - n(\alpha_0+1)\alpha_0 - 2d\alpha_0}{2\alpha_0|X| + 2d|X| - n(\alpha_0+1)\alpha_0 - 2d\alpha_0 - d^2 - d} \tag{3}$$

where $\alpha_0 = \lfloor (|X| - d - 1)/n \rfloor$. *Moreover, when* $|X| \geq d + (n-1)^2$, *the value in (3) is getting larger as* d *becomes smaller.*

Proof. See the Appendix below.

Now imagine the following situation: The subset $\mathcal{C}'$ of $\mathcal{C}$ consists of any hash function (from X to Y) for which collision pairs can be found in reasonable computational complexity by using some already known collision finding technique. In this situation, there are the following two cases:

1. If $r(\mathcal{C}; \mathcal{C}')$ is small, then *any* hash function H belonging to the set $\mathcal{C}$ can be *potentially* attacked by just finding a close approximation $H' \in \mathcal{C}'$ of H (and applying a known technique to this H'). This would suggest that, any hash function in $\mathcal{C}$ possesses a potential risk that a collision pair is found by someone with a combination of just known attack techniques and some expert's sixth sense to give an appropriate approximation.
2. If $r(\mathcal{C}; \mathcal{C}')$ is significantly large, then $\mathcal{C}$ contains at least one hash function for which the above attack strategy combined with any known collision finding technique will not succeed. This would suggest that, there is a hope to obtain, by searching within the set $\mathcal{C}$, a hash function for which one needs to develop a new attack technique in order to find a collision pair.

As a consequence, if two candidate sets $\mathcal{C}_1, \mathcal{C}_2$ for a new hash function and the corresponding subsets $\mathcal{C}'_1, \mathcal{C}'_2$ are given, and if $r(\mathcal{C}_1; \mathcal{C}'_1)$ is small while $r(\mathcal{C}_2; \mathcal{C}'_2)$ is significantly large, then this fact would give us a motivation to select a new hash function from $\mathcal{C}_2$ rather than $\mathcal{C}_1$. The authors hope that Function Density Problem can contribute to security analysis of non-keyed hash functions in such a way, though of course how to specify the subset $\mathcal{C}'$ is a big problem to be studied. (One may also feel that it seems infeasible to compute the quantity $r(\mathcal{C}; \mathcal{C}')$ for practical classes of hash functions; even if it is true, some result on a bound or tendency of $r(\mathcal{C}; \mathcal{C}')$ would still give us an insight into the security level of those hash functions.)

Remark 3. Here we notice that, although we have focused on the collision resistance in this paper, a similar argument would also be applicable to other security notions for hash functions, such as the (second) preimage resistance.

4 Pseudorandom Generators and Function Density Problem

As a second application of Function Density Problem, in this section we discuss some implication results of nb-PRGs from usual PRGs. First we recall the security notion for PRGs. Here we emphasize that we adopt concrete security formulation rather than asymptotic one. Now the definition is as follows, where U_X denotes the uniform probability distribution over a finite set X:

Definition 2 (see e.g., [4]). *Let $G\colon S \to X$ be an algorithm with finite input set S and finite output set X. Then G is called a* (T, ε)-secure pseudorandom generator *(*PRG*) if, for any algorithm (*distinguisher*) $D\colon X \to \{0, 1\}$ with time complexity bounded by T, we have $\mathsf{Adv}_D(G) \leq \varepsilon$ where $\mathsf{Adv}_D(G)$ is the* advantage *of D defined by*

$$\mathsf{Adv}_D(G) = |Pr[D(U_X) = 1] - Pr[D(G(U_S)) = 1]| \ . \tag{4}$$

Let $\Delta(P_1, P_2)$ denote the statistical distance of two probability distributions P_1, P_2 over the same finite set Z defined by

$$\Delta(P_1, P_2) = \frac{1}{2} \sum_{z \in Z} |Pr[P_1 = z] - Pr[P_2 = z]| \tag{5}$$

$$= \max_{E \subset Z} |Pr[P_1 \in E] - Pr[P_2 \in E]| \ . \tag{6}$$

Then the advantage $\mathsf{Adv}_D(G)$ of a distinguisher D defined above is equal to $\Delta(D(U_X), D(G(U_S)))$, as both $D(U_X)$ and $D(G(U_S))$ are probability distributions over the 1-bit set $\{0, 1\}$. This motivates the following enhancement of Definition 2 introduced by Dubrov and Ishai [3]:

Definition 3 ([3]). *Let $G\colon S \to X$ be an algorithm with finite input set S and finite output set X. Then G is called* (T, n, ε)-secure *if, for any distinguisher*

$D\colon X \to Y$ with time complexity bounded by T and output set Y of size bounded by n, we have $\mathsf{Adv}_D(G) \leq \varepsilon$ where we put $\mathsf{Adv}_D(G) = \Delta(D(U_X), D(G(U_S)))$. Such an algorithm G is called a pseudorandom generator that fools non-boolean distinguishers *(*nb-PRG*, in short).*

Note that, in this definition, the output set Y of D may be assumed to have the maximal size n without loss of generality. In the original paper [3], several applications of nb-PRGs are discussed; e.g., randomness used in some *information-theoretically secure* protocols (such as multi-party computation of certain types) can be replaced with outputs of nb-PRGs, without any restriction on computational complexity of the adversary's attack algorithm. However, despite the significance of nb-PRGs, it seems much more difficult to construct secure nb-PRGs than the case of usual PRGs. In fact, to the authors' best knowledge, the only constructions of nb-PRGs proposed so far are the ones by Dubrov and Ishai themselves [3], and their construction is based on certain non-standard computational assumption.

A hopeful solution for constructing nb-PRGs under standard assumptions is to convert usual PRGs secure under standard assumptions (which have been frequently proposed) into secure nb-PRGs. In fact, an implication relation such that any (T, ε)-secure PRG is also (T', n, ε')-secure with modified parameters T', ε' is mentioned (without proof) in [3]. The relation is derived from the first expression (5) of statistical distance, in the following manner (which is taken from [6]). For a subset Z' of a set Z, let $\chi_{Z'}\colon Z \to \{0,1\}$ denote the characteristic function of Z' defined by $\chi_{Z'}(x) = 1$ if $x \in Z'$ and $\chi_{Z'}(x) = 0$ if $x \in Z \setminus Z'$. We write $\chi_z = \chi_{\{z\}}$ for simplicity when $Z' = \{z\}$. Then for any PRG $G\colon S \to X$ and any non-boolean distinguisher $D\colon X \to Y$ with $|Y| \leq n$, we have

$$\begin{aligned}\Delta(D(U_X), D(G(U_S))) &= \frac{1}{2}\sum_{y\in Y} |Pr[D(U_X) = y] - Pr[D(G(U_S)) = y]| \\ &= \frac{1}{2}\sum_{y\in Y} |Pr[\chi_y \circ D(U_X) = 1] - Pr[\chi_y \circ D(G(U_S)) = 1]| \\ &= \frac{1}{2}\sum_{y\in Y} \mathsf{Adv}_{\chi_y \circ D}(G) \ . \end{aligned} \tag{7}$$

This implies that, to show that a (T', ε')-secure PRG G is also a (T, n, ε)-secure nb-PRG, it suffices to choose the parameters as $T' = T + \delta_1$ and $\varepsilon' = 2\varepsilon/n$, where δ_1 is the maximum of the overhead in computational complexity of composing some χ_y to D. In other words, we have the following proposition:

Proposition 1. *Any $(T + \delta_1, 2\varepsilon/n)$-secure PRG is also (T, n, ε)-secure, where the value δ_1 is defined as above.*

However, in practical applications n should be somewhat large, which makes the implication in Proposition 1 inefficient.

For another implication relation, here we use the second expression (6) of statistical distance instead of the first one (5) used above. Namely, we have

$$\begin{aligned}\Delta(D(U_X), D(G(U_S))) &= \max_{Y' \subset Y} |Pr[D(U_X) \in Y'] - Pr[D(G(U_S)) \in Y']| \\ &= \max_{Y' \subset Y} |Pr[\chi_{Y'} \circ D(U_X) = 1] - Pr[\chi_{Y'} \circ D(G(U_S)) = 1]| \\ &= \max_{Y' \subset Y} \mathsf{Adv}_{\chi_{Y'} \circ D}(G) \ . \end{aligned} \tag{8}$$

This implies the following result:

Proposition 2. *Any $(T + \delta_2, \varepsilon)$-secure PRG is also a (T, n, ε)-secure, where δ_2 is the maximum of the overhead in computational complexity of composing some $\chi_{Y'}$ to D.*

In contrast to Proposition 1, there is no overhead for bound of advantage ε in this second implication. However, the overhead δ_2 for bound of computational complexity is expected to be very large, as the set Y (of somewhat large size) may contain a very complicated subset Y', for which the function $\chi_{Y'}$ would be complicated as well.

From now, we try to improve the above trade-off between overheads for bounds of advantage and of computational complexity, by applying Function Density Problem. Let $\mathcal{C} = \mathcal{C}_Y$ be the set of characteristic functions $\chi_{Y'}$ for subsets $Y' \subset Y$, and let the distance $d(f, g)$ be defined as in (2) where Y is used instead of X. Then $d(\chi_{Y_1}, \chi_{Y_2})$ is equal to the size of the symmetric difference $Y_1 \ominus Y_2 = (Y_1 \setminus Y_2) \cup (Y_2 \setminus Y_1)$ of Y_1 and Y_2. Now we fix a subset $\mathcal{C}' = \mathcal{C}'_Y$ of $\mathcal{C}_Y$ for each Y. Let $\delta_{3,Y}$ be the maximum of the overhead in computational complexity of composing some $\chi_{Y'} \in \mathcal{C}'_Y$ to D. Moreover, we put $r_Y = r(\mathcal{C}_Y; \mathcal{C}'_Y)$ for simplicity. Then we have the following result:

Theorem 1. *In the above situation, suppose that $\delta_{3,Y} \leq \delta_3$ and $r_Y \leq r$ for every set Y of size n. Let δ_1 be as in Proposition 1. If $G\colon S \to X$ is $(T + \delta_1, \varepsilon_1)$-secure and $(T + \delta_3, \varepsilon_3)$-secure, then G is also $(T, n, r\varepsilon_1 + \varepsilon_3)$-secure.*

Proof. For each distinguisher $D\colon X \to Y$, let Y_0 be a subset of Y that attains the maximum of the second expression (6) of the statistical distance;

$$\Delta(D(U_X), D(G(U_S))) = |Pr[D(U_X) \in Y_0] - Pr[D(G(U_S)) \in Y_0]| \ . \tag{9}$$

Note that Y_0 can be chosen in such a way that

$$Pr[D(U_X) \in Y_0] - Pr[D(G(U_S)) \in Y_0] \geq 0 \tag{10}$$

(if this inequality fails, use $Y \setminus Y_0$ instead of Y_0). Moreover, as $r_Y \leq r$, there is a subset $Y_1 \subset Y$ such that $\chi_{Y_1} \in \mathcal{C}'_Y$ and $d(\chi_{Y_0}, \chi_{Y_1}) = |Y_0 \ominus Y_1| \leq r$. Now we have

$$\begin{aligned} &Pr[D(U_X) \in Y_0] - Pr[D(U_X) \in Y_1] \\ &= Pr[D(U_X) \in Y_0 \setminus Y_1] - Pr[D(U_X) \in Y_1 \setminus Y_0] \end{aligned} \tag{11}$$

and a similar equality holds for $Pr[D(G(U_S)) \in Y_0] - Pr[D(G(U_S)) \in Y_1]$. This implies that

$$\begin{aligned}
&(Pr[D(U_X) \in Y_0] - Pr[D(U_X) \in Y_1]) \\
&\quad - (Pr[D(G(U_S)) \in Y_0] - Pr[D(G(U_S)) \in Y_1]) \\
&= (Pr[D(U_X) \in Y_0 \setminus Y_1] - Pr[D(G(U_S)) \in Y_0 \setminus Y_1]) \\
&\quad - (Pr[D(U_X) \in Y_1 \setminus Y_0] - Pr[D(G(U_S)) \in Y_1 \setminus Y_0]) \\
&= \sum_{y \in Y_0 \setminus Y_1} (Pr[D(U_X) = y] - Pr[D(G(U_S)) = y]) \\
&\quad - \sum_{y \in Y_1 \setminus Y_0} (Pr[D(U_X) = y] - Pr[D(G(U_S)) = y]) \qquad (12) \\
&\leq \sum_{y \in Y_0 \ominus Y_1} |Pr[D(U_X) = y] - Pr[D(G(U_S)) = y]| \\
&= \sum_{y \in Y_0 \ominus Y_1} |Pr[\chi_y \circ D(U_X) = 1] - Pr[\chi_y \circ D(G(U_S)) = 1]| \\
&= \sum_{y \in Y_0 \ominus Y_1} \mathsf{Adv}_{\chi_y \circ D}(G) \ .
\end{aligned}$$

Now if D has computational complexity bounded by T, then the assumption on G and the definition of δ_1 imply that

$$\sum_{y \in Y_0 \ominus Y_1} \mathsf{Adv}_{\chi_y \circ D}(G) \leq \sum_{y \in Y_0 \ominus Y_1} \varepsilon_1 = |Y_0 \ominus Y_1| \cdot \varepsilon_1 \leq r\varepsilon_1 \ . \qquad (13)$$

Hence we have

$$\begin{aligned}
\Delta(D(U_X), D(G(U_S))) &\leq r\varepsilon_1 + (Pr[D(U_X) \in Y_1] - Pr[D(G(U_S)) \in Y_1]) \\
&\leq r\varepsilon_1 + |Pr[\chi_{Y_1} \circ D(U_X) = 1] - Pr[\chi_{Y_1} \circ D(G(U_S)) = 1]| \\
&= r\varepsilon_1 + \mathsf{Adv}_{\chi_{Y_1} \circ D}(G) \leq r\varepsilon_1 + \varepsilon_3 \ , \qquad (14)
\end{aligned}$$

as desired, therefore the claim follows.

Regarding the relation between parameters in Theorem 1, first note that it is natural to expect that $\delta_1 \leq \delta_3$ by the definitions, which allows us to suppose that $\varepsilon_1 \leq \varepsilon_3$. Now if we can find appropriate subsets $\mathcal{C}'_Y$ in such a way that every characteristic function $\chi_{Y'} \in \mathcal{C}'_Y$ has low computational complexity and the quantity $r(\mathcal{C}_Y; \mathcal{C}'_Y)$ is small, then both δ_3 and r can be small as well, which would make the implication relation in Theorem 1 more efficient than those in Propositions 1 and 2, improving the above-mentioned trade-off. Hence a study of Function Density Problem (of special case for functions with 1-bit output sets) will contribute to establish an efficient implication relation.

Remark 4. We mention that, for the above two applications of Function Density Problem, a kind of "risk-hedging" relation exists as follows. Namely, if we find that the quantity $r(\mathcal{C};\mathcal{C}')$ tends to be large, then it would support the argument in Section 3 to show security of the hash functions. On the other hand, if we find that the quantity $r(\mathcal{C};\mathcal{C}')$ tends to be small, then it would support the argument in Section 4 to show efficient implication relation of nb-PRGs from usual PRGs.

5 Example for Function Density Problem

In this section, we describe a concrete example of Function Density Problem. We emphasize that the example here is aimed at arising some image or intuition of how a study of the problem itself can proceed, rather than proposing some definition of the subsets $\mathcal{C}'$ which is useful from the viewpoints of above applications. The authors let the problem of specifying appropriate subsets $\mathcal{C}'$ be open as a future research topic.

In our example, we consider the case that the functions have n-bit inputs and 1-bit outputs; $X = \{0,1\}^n$ and $Y = \{0,1\}$ (which is relevant to the situation of Section 4). First note that, when we regard $\{0,1\}$ as the two-element finite field, each function $f\colon X \to Y$ can be expressed as an n-variable square-free polynomial;

$$f(x_1,\ldots,x_n) = \sum_{a_1,\ldots,a_n\in\{0,1\}} f(a_1,\ldots,a_n) \prod_{i;a_i=0} (1-x_i) \prod_{i;a_i=1} x_i \qquad (15)$$

(note that the term $\prod_{i;a_i=0}(1-x_i)\prod_{i;a_i=1} x_i$ for each summand in the right-hand side becomes 1 if $x_i = a_i$ for every i, and 0 otherwise). For example, when $n = 2$ we have

$$\begin{aligned} &f(x_1,x_2) \\ &= f(0,0)(1-x_1)(1-x_2) + f(0,1)(1-x_1)x_2 + f(1,0)x_1(1-x_2) + f(1,1)x_1x_2 \ . \end{aligned} \qquad (16)$$

Now we set $\mathcal{C}' = \mathcal{C}'_k$ to be the subset of $\mathcal{C}$ consisting of functions that can be expressed as a square-free polynomial of degree at most k. For example, $\mathcal{C}'_0$ is the set of constant functions, and $\mathcal{C}'_1$ is the set of affine functions. (The choice of $\mathcal{C}'$ here as a set of "easily breakable" hash functions would be more reasonable when we consider the (second) preimage resistance rather than the collision resistance; see Remark 3.) The distance $d(f,g)$ is defined as in (2). In this situation, we have the following result on upper and lower bounds for the quantity $r(\mathcal{C};\mathcal{C}'_k)$:

Proposition 3. *Let* $u_{n,k} = \sum_{i=k+1}^{n} \binom{n}{i}$, *and let* $\ell_{n,k}$ *be the minimum integer* ℓ *such that* $2^{u_{n,k}} \le \sum_{i=0}^{\ell} \binom{2^n}{i}$. *Then we have*

$$\ell_{n,k} \le r(\mathcal{C};\mathcal{C}'_k) \le \min\{u_{n,k}, 2^{n-1}\} \ . \qquad (17)$$

Proof. First we prove the lower bound. For a function $f\colon \{0,1\}^n \to \{0,1\}$, let $f_{>k}$ denote the sum of monomials of degree larger than k appearing in the polynomial expression of f. Note that $f_{>k} = 0$ for every $f \in \mathcal{C}'_k$. Note also that $d(f,g) \le d$ means that $f = g + \chi_Z$ for some subset $Z \subset \{0,1\}^n$ of size at most d. Now if $r(\mathcal{C}; \mathcal{C}'_k) = r$, then for each $f \in \mathcal{C}$, there is a $g \in \mathcal{C}'_k$ such that $f = g + \chi_Z$, hence $f_{>k} = (\chi_Z)_{>k}$, for some $Z \subset \{0,1\}^n$ of size at most r. The number of possibilities of $f_{>k}$ is $2^{u_{n,k}}$, as there are $u_{n,k}$ monomials of degree larger than k. On the other hand, the number of possibilities of $(\chi_Z)_{>k}$ is at most the number of subsets of $\{0,1\}^n$ of size at most r, which is $\sum_{i=0}^{r} \binom{2^n}{i}$. Hence it must hold that $2^{u_{n,k}} \le \sum_{i=0}^{r} \binom{2^n}{i}$, therefore $r \ge \ell_{n,k}$ by the definition of $\ell_{n,k}$.

Secondly, we prove the upper bound. Note that $r(\mathcal{C}; \mathcal{C}_k) \le 2^{n-1}$, as any function $f \in \mathcal{C}$ can be converted into a constant function by changing the value $f(x)$ at every point $x \in \{0,1\}^n$ such that $f(x)$ is in the minority among the 2^n values of f (the number of such points is at most 2^{n-1}). From now, we show that $r(\mathcal{C}; \mathcal{C}'_k) \le u_{n,k}$. Now note that, for each square-free monomial $x_{i_1} x_{i_2} \cdots x_{i_{k+1}}$ of degree $k+1$, there exists a (unique) point $a \in \{0,1\}^n$ such that the characteristic function χ_a (in the polynomial form) contains no monomials of degree at most k and a unique monomial of degree $k+1$. Indeed, the point $a = (a_1, \ldots, a_n)$ defined as $a_j = 1$ if $j = i_h$ for some $1 \le h \le k+1$ and $a_j = 0$ otherwise satisfies the condition. This implies that, for any $f \in \mathcal{C}$, by adding at most the same number (i.e., $\binom{n}{k+1}$) of χ_a as the square-free monomials of degree $k+1$, the monomials of degree $k+1$ in f can be cancelled without changes of monomials of degree lower than or equal to k. Iterating the process also for higher degrees, all the monomials of degree at least $k+1$ can be cancelled (hence a function in $\mathcal{C}'_k$ is obtained) by adding at most $u_{n,k}$ functions χ_a. This implies that $r(\mathcal{C}; \mathcal{C}'_k) \le u_{n,k}$, therefore the claim of Proposition 3 holds.

Remark 5. We notice that a part $r(\mathcal{C}; \mathcal{C}'_k) \le u_{n,k}$ of the above inequality can be generalized to an arbitrary linear subspace $\mathcal{C}'_k$ of $\mathcal{C}$, in which case $u_{n,k}$ will be replaced with $\dim(\mathcal{C}) - \dim(\mathcal{C}'_k)$ where dim denotes the dimension as a linear space over the two-element field. See the forthcoming full version of this paper.

We give the precise values of $\ell_{n,k}$ for some smaller cases, in Table 1.

Table 1. The values of $\ell_{n,k}$ for some small parameters

	$n-k$							
n	1	2	3	4	5	6	7	8
2	1	2						
3	1	2	4					
4	1	2	4	8				
5	1	2	5	10	16			
6	1	2	5	13	22	32		
7	1	2	6	16	31	49	64	
8	1	2	6	19	43	75	105	128

The next result shows how the lower and upper bounds in Proposition 3 are close to each other:

Proposition 4. *In the setting of Proposition 3, we have*

$$\ell_{n,k} \leq u_{n,k} \leq n\ell_{n,k} \ . \tag{18}$$

Proof. It suffices to prove the second inequality, or equivalently, the inequality $u_{n,k} < n\ell_{n,k}+1$. As $2^{u_{n,k}} \leq \sum_{i=0}^{\ell_{n,k}} \binom{2^n}{i}$ by the definition of $\ell_{n,k}$, it suffices to show that $\sum_{i=0}^{\ell_{n,k}} \binom{2^n}{i} < 2^{n\ell_{n,k}+1}$, or more generally, $\sum_{i=0}^{m} \binom{N}{i} < 2N^m$ for all integers $N > m \geq 0$ (then apply it to $N = 2^n$, $m = \ell_{n,k}$; note that $\ell_{n,k} \leq u_{n,k} < 2^n$ by the definition of $u_{n,k}$). We use induction on m. The case $m = 0$ is trivial. For the case $m \geq 1$, we have $N \geq m+1 \geq 2$ and

$$\begin{aligned} \sum_{i=0}^{m} \binom{N}{i} &= \sum_{i=0}^{m-1} \binom{N}{i} + \binom{N}{m} \\ &< 2N^{m-1} + \binom{N}{m} \qquad \text{(induction hypothesis)} \\ &= 2N^{m-1} + \frac{N(N-1)\cdots(N-m+1)}{m!} \\ &\leq N \cdot N^{m-1} + N^m = N^m + N^m = 2N^m \ . \end{aligned} \tag{19}$$

Hence the claim holds.

This proposition shows that the lower bound and the upper bound in Proposition 3 have almost the same order. For further evaluation of the quantity $r(\mathcal{C}; \mathcal{C}'_k)$, we can introduce other formulations of the problem in terms of some mathematical tools such as simplicial complexes, linear algebra, and Gröbner bases. In this paper the authors do not want to go into details, which will be discussed in the full version of this paper.

6 Concluding Remarks

In this paper, we first introduced a mathematically formulated problem, Function Density Problem. Then we proposed two applications of the problem to the area of information security; collision-resistance analysis of non-keyed hash algorithms, and efficient implication of nb-PRGs from usual PRGs. The authors hope that these applications can contribute to security evaluation of real-life hash algorithms such as the forthcoming SHA-3, and to secure implementation of information-theoretically secure protocols by using nb-PRGs. We also showed an example of how a study of Function Density Problem may proceed.

To conclude this paper, we mention some other potential applications of Function Density Problem. There are some cryptographic protocols for which the constructions are motivated by some NP-complete/NP-hard problems, but actually the distributions of the problem instances in the protocols are somewhat biased, therefore it has not succeeded to prove the security of the protocols directly from the hardness of the underlying problems (e.g., McEliece cryptosystem

and other code-based protocols relevant to decoding problem for random linear codes; knapsack cryptosystem relevant to Subset Sum Problem; etc.). We hope that the idea of Function Density Problem can be applied to measure the closeness of the approximations of the underlying hard problems in those protocols. On the other hand, it would also be an interesting future work to find other applications of Function Density Problem.

Acknowledgements. The authors would like to thank Dr. Goichiro Hanaoka for his precious comment on potential applications of the subject of this paper discussed in Section 6. The authors would also like to thank the anonymous referees for their kind reviews and comments.

References

1. Blum, L., Blum, M., Shub, M.: A simple unpredictable pseudo-random number generator. SIAM J. Comput. 15, 364–383 (1986)
2. Carter, J.L., Wegman, M.N.: Universal classes of hash functions (extended abstract). In: Proc. STOC 1977, pp. 106–112 (1977)
3. Dubrov, B., Ishai, Y.: On the randomness complexity of efficient sampling. In: Proc. STOC 2006, pp. 711–720 (2006)
4. Farashahi, R.R., Schoenmakers, B., Sidorenko, A.: Efficient Pseudorandom Generators Based on the DDH Assumption. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 426–441. Springer, Heidelberg (2007)
5. Matsumoto, T., Imai, H.: Public Quadratic Polynomial-Tuples for Efficient Signature-Verification and Message-Encryption. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 419–453. Springer, Heidelberg (1988)
6. Nuida, K., Hanaoka, G.: On the Security of Pseudorandomized Information-Theoretically Secure Schemes. In: Kurosawa, K. (ed.) ICITS 2009. LNCS, vol. 5973, pp. 56–73. Springer, Heidelberg (2010)
7. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21(2), 120–126 (1978)
8. Rogaway, P.: Formalizing human ignorance – collision-resistant hashing without the keys. In: Nguyên, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 211–228. Springer, Heidelberg (2006)
9. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)

Appendix: Proof of Lemma 1

Here we give a proof of Lemma 1 in Section 3. Put $Y = \{y_1, \ldots, y_n\}$. For $1 \leq i \leq n$, put

$$a_i = |\{x \in X \mid H'(x) = y_i\}|\,,\ b_i = |\{x \in X \mid H(x) \neq H'(x) = y_i\}|\ . \tag{20}$$

Put

$$\begin{aligned} \varphi_1(a_1, \ldots, a_n; b_1, \ldots, b_n) &= \sum_{i=1}^{n} a_i(a_i - 1)\,, \\ \varphi_2(a_1, \ldots, a_n; b_1, \ldots, b_n) &= \sum_{i=1}^{n} (a_i - b_i)(a_i - b_i - 1)\ . \end{aligned} \tag{21}$$

Then the number of collision pairs for H' is $\varphi_1(a_1,\ldots,a_n;b_1,\ldots,b_n)$, while the number of collision pairs for H is at least $\varphi_2(a_1,\ldots,a_n;b_1,\ldots,b_n)$. Therefore the probability specified in the statement of Lemma 1 is at least

$$\varphi(a_1,\ldots,a_n;b_1,\ldots,b_n) = \frac{\varphi_2(a_1,\ldots,a_n;b_1,\ldots,b_n)}{\varphi_1(a_1,\ldots,a_n;b_1,\ldots,b_n)} \ . \tag{22}$$

From now, we give a lower bound for the values of φ under the obvious conditions $0 \le b_i \le a_i$, $\sum_{i=1}^n a_i = |X|$ and $\sum_{i=1}^n b_i = d$.

Step 1: If the minimum of the function φ is attained by $(a_i)_{i=1}^n$ and $(b_i)_{i=1}^n$, then we have $b_i > 0$ for a unique index i, and $a_i - b_i \ge a_j$ for any index $j \ne i$.

(Proof of Step 1) If $i \ne j$ and $b_i, b_j > 0$, and we suppose $a_i \le a_j$ by symmetry, then we have

$$\big((a_i-1)(a_i-2)+(a_j+1)a_j\big)-\big(a_i(a_i-1)+a_j(a_j-1)\big) = 2(a_j-a_i+1) > 0 \ , \tag{23}$$

therefore the value of φ_1 increases when a_i, a_j, b_i, b_j are replaced with $a_i-1, a_j+1, b_i-1, b_j+1$, respectively. On the other hand, the value of φ_2 is not changed by this replacement. Therefore the value of φ is decreased by this replacement, contradicting the assumption on $(a_i)_{i=1}^n$ and $(b_i)_{i=1}^n$. Hence an index i with $b_i > 0$ is unique, therefore $b_i = d$. Similarly, if $j \ne i$ and $a_i - b_i < a_j$, then we have

$$\begin{aligned}\big((a_i-b_i+1)(a_i-b_i)+(a_j-1)(a_j-2)\big)-\big((a_i-b_i)(a_i-b_i-1)+a_j(a_j-1)\big)\\ = 2(a_i-b_i-a_j+1) \le 0 \ ,\end{aligned} \tag{24}$$

with equality holding if and only if $a_i - b_i = a_j - 1$. This implies that the value of φ at the $(a_i)_{i=1}^n$ and $(b_i)_{i=1}^n$ is larger than or equal to the value of φ with b_i and b_j $(= 0)$ being replaced with $b_i - 1$ and 1, respectively, where the equality holds if and only if $a_i - b_i = a_j - 1$. As the former value is the minimum, the equality condition $a_i - b_i = a_j - 1$ should hold. Moreover, if $b_i - 1 > 0$ then the latter value of φ (which is now equal to the former value) cannot be the minimum by the above argument, which also leads to a contradiction. Hence we have $b_i = 1$ (therefore $d = 1$) and $a_i = a_j$. Now we have

$$\big((a_i+1)a_i+(a_j-1)(a_j-2)\big)-\big(a_i(a_i-1)+a_j(a_j-1)\big) = 2(a_i-a_j+1) > 0 \ . \tag{25}$$

This implies that the value of φ will decrease when a_i and a_j are replaced with $a_i + 1$ and $a_j - 1$, respectively, contradicting the assumption that the former value is the minimum. Hence we have $a_i - b_i \ge a_j$ for every $j \ne i$, concluding the proof of Step 1.

Step 2: If the minimum of the function φ is attained by $(a_i)_{i=1}^n$ and $(b_i)_{i=1}^n$, then we have $|a_i - a_j| \le 1$ for any pair of indices $i \ne j$ such that $b_i = b_j = 0$.

(Proof of Step 2) Assume contrary that $a_i - a_j \ge 2$ for such a pair of indices $i \ne j$. For $\ell = 1, 2$, let α_ℓ denote the value of φ_ℓ at the $(a_i)_{i=1}^n$ and $(b_i)_{i=1}^n$, and

let β_ℓ denote the value of φ_ℓ with a_i and a_j being replaced with $a_i - 1$ and $a_j + 1$, respectively. Then we have $\beta_1 - \alpha_1 = \beta_2 - \alpha_2 = 2(a_j - a_i + 1) < 0$. On the other hand, for the index i' with $b_{i'} > 0$, we have $a_{i'} \geq b_{i'} + a_i \geq b_{i'} + a_j + 2 \geq 2$ by the assumption and Step 1, therefore $\alpha_1 > \alpha_2$. Now we present the following lemma, which is proven by an easy calculation:

Lemma 2. *If $p > q \geq 0$ and $r > 0$, then $q/p < (q + r)/(p + r)$.*

By using this lemma, we have

$$\frac{\alpha_2}{\alpha_1} = \frac{\beta_2 - 2(a_j - a_i + 1)}{\beta_1 - 2(a_j - a_i + 1)} > \frac{\beta_2}{\beta_1} \ , \tag{26}$$

contradicting the assumption that α_2/α_1 is the minimum of the value of φ. Hence the claim of Step 2 holds.

By Step 1 and Step 2, the values $(a_i)_{i=1}^n$ and $(b_i)_{i=1}^n$ that attain the minimum of φ satisfy the following conditions: $b_i > 0$ for a unique i, and there is an integer α such that $a_i - b_i \geq \alpha + 1$ and $a_j \in \{\alpha, \alpha + 1\}$ for every $j \neq i$. Note that this α can be taken as $\alpha \geq 0$; this is obvious if some a_j with $j \neq i$ is positive, while if $a_j = 0$ for every $j \neq i$, then we can choose $\alpha = 0$, as $a_i = |X| > d = b_i$ and $a_i - b_i \geq 1$. Let k be the number of indices $j \neq i$ such that $a_j = \alpha + 1$, therefore $0 \leq k \leq n - 1$. Then we have $a_i = |X| - (n - 1)\alpha - k$, while $b_i = d$, therefore the condition $a_i - b_i \geq \alpha + 1$ implies that $k \leq |X| - n\alpha - d - 1$. Now we write the values of φ_1 and φ_2 in this case as $\varphi_1(\alpha, k)$ and $\varphi_2(\alpha, k)$, respectively. Then we have

$$\begin{aligned} \varphi_1(\alpha, k) &= k(\alpha + 1)\alpha + (n - 1 - k)\alpha(\alpha - 1) + a_i(a_i - 1) \ , \\ \varphi_2(\alpha, k) &= k(\alpha + 1)\alpha + (n - 1 - k)\alpha(\alpha - 1) + (a_i - d)(a_i - d - 1) \ , \end{aligned} \tag{27}$$

therefore $\varphi_1(\alpha, k) - \varphi_2(\alpha, k) = 2da_i - d^2 - d$. Now by Lemma 2, we have

$$\begin{aligned} 1 - \frac{\varphi_2(\alpha, k)}{\varphi_1(\alpha, k)} &= \frac{2da_i - d^2 - d}{\varphi_1(\alpha, k)} \leq \frac{2da_i - d^2 - d + 2d((n - 1)\alpha + k)}{\varphi_1(\alpha, k) + 2d((n - 1)\alpha + k)} \\ &= \frac{2d|X| - d^2 - d}{k(\alpha + 1)\alpha + (n - 1 - k)\alpha(\alpha - 1) + a_i(a_i - 1) + 2d(n - 1)\alpha + 2dk} \end{aligned} \tag{28}$$

(note that $2d((n - 1)\alpha + k) \geq 0$, as $\alpha \geq 0$). Let $\psi(\alpha, k)$ denote the denominator of the right-hand side. Then, by using the property $\frac{\partial}{\partial k} a_i = -1$, we have

$$\frac{\partial}{\partial k}\psi(\alpha, k) = (\alpha + 1)\alpha - \alpha(\alpha - 1) - (2a_i - 1) + 2d = 2\alpha - 2a_i + 1 + 2d < 0 \tag{29}$$

(note that $a_i - d \geq \alpha + 1$), therefore $\psi(\alpha, k)$ is decreasing as k is increasing. On the other hand, we have $\psi(\alpha, n - 1) = \psi(\alpha + 1, 0)$. Now note that $\alpha \leq (|X| - d - 1)/n$, as $0 \leq k \leq |X| - n\alpha - d - 1$. This implies that $\psi(\alpha, k)$ takes the minimum value at $\alpha = \alpha_0 = \lfloor (|X| - d - 1)/n \rfloor$ and $k = k_0 = |X| - n\alpha_0 - d - 1$ (note that

$k_0 \leq n-1$). Moreover, we have $a_i = \alpha_0 + d + 1$ if $\alpha = \alpha_0$ and $k = k_0$. Hence a straightforward calculation shows that

$$\begin{aligned} 1 - \frac{\varphi_2(\alpha,k)}{\varphi_1(\alpha,k)} &\leq \frac{2d|X| - d^2 - d}{\psi(\alpha_0,k_0)} \\ &= \frac{2d|X| - d^2 - d}{2\alpha_0|X| + 2d|X| - n(\alpha_0+1)\alpha_0 - 2d\alpha_0 - d^2 - d} \ , \end{aligned} \tag{30}$$

therefore

$$\begin{aligned} \frac{\varphi_2(\alpha,k)}{\varphi_1(\alpha,k)} &\geq 1 - \frac{2d|X| - d^2 - d}{2\alpha_0|X| + 2d|X| - n(\alpha_0+1)\alpha_0 - 2d\alpha_0 - d^2 - d} \\ &= \frac{2\alpha_0|X| - n(\alpha_0+1)\alpha_0 - 2d\alpha_0}{2\alpha_0|X| + 2d|X| - n(\alpha_0+1)\alpha_0 - 2d\alpha_0 - d^2 - d} \ , \end{aligned} \tag{31}$$

which proves the lower bound (3) in the statement of Lemma 1.

Finally, suppose that $d \geq 2$, and let $\eta_1(d)$ and $\eta_2(d)$ denote the denominator and the numerator in (3), respectively. For any value x depending on d, let $\Delta[x]$ temporarily denote the value of x at $d-1$ minus the value of x at d. Then we have $\Delta(-d^2 - d) = 2d$, therefore

$$\begin{aligned} \Delta[\eta_2(d)] &= \Delta[2\alpha_0|X| - n(\alpha_0+1)\alpha_0 - 2d\alpha_0] \ , \\ \Delta[\eta_1(d)] &= \Delta[2\alpha_0|X| - n(\alpha_0+1)\alpha_0 - 2d\alpha_0] - 2|X| + 2d < \Delta[\eta_2(d)] \ . \end{aligned} \tag{32}$$

Moreover, we have $\Delta[\alpha_0] \in \{0,1\}$, and if $\Delta[\alpha_0] = 0$, then $\Delta[\eta_2(d)] = 2d\alpha_0 > 0$. On the other hand, if $\Delta[\alpha_0] = 1$, then we have

$$\begin{aligned} \Delta[(\alpha_0+1)\alpha_0] &= (\alpha_0+2)(\alpha_0+1) - (\alpha_0+1)\alpha_0 = 2(\alpha_0+1) \ , \\ \Delta[2d\alpha_0] &= 2(d-1)(\alpha_0+1) - 2d\alpha_0 = 2d - 2\alpha_0 - 2 \ , \end{aligned} \tag{33}$$

therefore

$$\begin{aligned} &\Delta[\eta_2(d)] = 2|X| - 2n(\alpha_0+1) - 2d + 2\alpha_0 + 2 \\ &= 2|X| - 2(n-1)\alpha_0 - 2n - 2d + 2 \\ &\geq 2|X| - 2(n-1)\frac{|X| - d - 1}{n} - 2n - 2d + 2 = \frac{2}{n}\big(|X| - d + 2n - 1 - n^2\big) \geq 0 \end{aligned} \tag{34}$$

(where we used the assumption $|X| \geq d + (n-1)^2$). Now by Lemma 2, we have

$$\frac{\eta_2(d-1)}{\eta_1(d-1)} = \frac{\eta_2(d) + \Delta[\eta_2(d)]}{\eta_1(d) + \Delta[\eta_1(d)]} \geq \frac{\eta_2(d)}{\eta_1(d) + \Delta[\eta_1(d)] - \Delta[\eta_2(d)]} > \frac{\eta_2(d)}{\eta_1(d)} \ . \tag{35}$$

Hence the proof of Lemma 1 is concluded.

A Theoretical Analysis of the Structure of HC-128

Goutam Paul[1], Subhamoy Maitra[2], and Shashwat Raizada[2]

[1] Department of Computer Science and Engineering,
Jadavpur University, Kolkata 700 032, India
goutam.paul@ieee.org
[2] Applied Statistics Unit,
Indian Statistical Institute, Kolkata 700 108, India
subho@isical.ac.in, shashwat.raizada@gmail.com

Abstract. HC-128 is an eSTREAM finalist and no practical attack on this cipher is known. We show that the knowledge of any one of the two internal state arrays of HC-128 along with the knowledge of 2048 keystream words is sufficient to construct the other state array completely in 2^{42} time complexity. Though our analysis does not lead to any attack on HC-128, it reveals a structural insight into the cipher. In the process, we theoretically establish certain combinatorial properties of HC-128 keystream generation algorithm. Our work may be considered as the first step towards a possible state recovery of HC-128. We also suggest a modification to HC-128 that takes care of the recently known cryptanalytic results with little reduction in speed.

Keywords: Cryptography, eSTREAM, HC-128, Keystream, State Recovery, Stream Cipher.

1 Introduction

The stream cipher HC-128 [16] is part of the eSTREAM [4] Portfolio (revision 1, September 2008) in the Software category. The designer, Wu, performed some security analysis of the cipher [16, Sections 3,4]. Another observation by Dunkelman [3] in the eSTREAM discussion forum shows that the keystream words of HC-128 leak information regarding secret states. A generalization of these results has been studied in [10]. In [9], it has been shown that the key and IV setup algorithm can be reversed if both the P, Q arrays are available after the key scheduling. Recently, a fault analysis on HC-128 has been presented in [7]. None of the existing results on HC-128 disproves the security conjectures of the designer and frequent use of this cipher in commercial domain is expected.

There are two internal state arrays of HC-128, P and Q, each containing 512 many 32-bit words. The keystream is generated in blocks of 512 words. Within a block, one of these arrays gets updated and the keystream word is produced by XOR-ing the updated entry with the sum of two words from the other array.

T. Iwata and M. Nishigaki (Eds.): IWSEC 2011, LNCS 7038, pp. 161–177, 2011.

The role of the two arrays is reversed after every block of 512 keystream words generation. In this paper, we show that the knowledge of one internal state array of HC-128 reveals the other.

1.1 Outline of the Contribution

Without loss of generality, we consider four consecutive blocks B_1, B_2, B_3 and B_4 of keystream generation such that Q is updated in blocks B_1 and B_3 and P is updated in blocks B_2 and B_4. Suppose the keystream words corresponding to all of these four blocks are known. Henceforth, by the symbols P and Q, we would denote the arrays after the completion of block B_1 and before the start of block B_2. After the completion of block B_2, Q remains unaltered and P is updated to, say, P_N. After the completion of block B_3, Q would again be updated to, say, Q_N.

Block B_1**:**	**Block** B_2**:**	**Block** B_3**:**
P unchanged,	P updated to P_N,	P_N unchanged,
Q updated.	Q unchanged.	Q updated to Q_N.
(Q denotes the updated array)		

Block B_4, that is not shown in the diagram, would only be used for verifying if our reconstruction is correct or not.

In Section 3, we present Algorithm 4 (called *ReconstructState*), that takes as input the 512 words of the array P and (assuming that the 2048 keystream words corresponding to the four blocks B_1, B_2, B_3 and B_4 are known) produces as output 512 words of the array Q_N. Since the update of an array depends only on itself, it turns out that from block B_3 onwards the complete state becomes known. The proof of correctness of the algorithm is established through Lemma 2 and Theorems 1, 2 and 3 and the data and time complexity requirements are analyzed in Theorem 4.

Our analysis shows some combinatorial properties of the internal state evolution of HC-128. At present, this does not lead to any immediate attack on HC-128.

In Section 4, we propose little modification to the existing HC-128 that escapes the structural analysis of this paper and all currently known cryptanalytic results over HC-128. We also present performance comparisons with the existing design of HC-128.

1.2 Connection to Previous Works and Motivation

Liu and Qin [9] showed that the initialization (i.e., Key and IV setup) algorithm of HC-128 is reversible and hence once all the 1024 (32-bit) words of the internal

state consisting of two tables P and Q, each containing 512 words are known then one can get the Key and IV. Referring to [9], denote the tables P and Q after $1024k$ steps (one "big step" [9]) by P_k and Q_k. It is shown in [9] that $P_k||Q_k$ can be reconstructed from $P_{k+1}||Q_{k+1}$ in two steps: first compute Q_k using $P_{k+1}||Q_{k+1}$, then compute P_k using Q_k and P_{k+1}. That is, computing P_k only needs knowing Q_k together with P_{k+1}, and similarly, computing Q_k only needs knowing P_k together with Q_{k+1}. Our work shows that the assumption that "adversary needs P_k and Q_{k+1} to reconstruct Q_k" (according to [9]) can be replaced with the assumption that "adversary needs P_k and 2048 words of the keystream to reconstruct Q_k", albeit with an extra effort of 2^{42} computations. As a consequence, information of only half of the internal state can reveal the complete secret key.

Use of partial state information in mounting a full state recovery attack is not new in the domain of stream ciphers. In case of RC4, for instance, Knudsen et al.'s state recovery attack [8] reconstructs the complete state in a complexity of 2^{779}. However, the work [15] showed that the above attack requires only 2^{220} search complexity if 112 entries of the permutation are known and presents an improvement whereby state recovery with the same complexity requires prior knowledge of only 73 permutation entries in certain cases. The work [11] is currently the best known state recovery attack on RC4 with complexity 2^{241}. In [11, Appendix C], the complexity is formulated as a function of L, where L is the number of previously known permutation entries and so the complexity can be reduced below 2^{241} if certain permutation entries are available to the attacker. Though there are many differences between RC4 and HC-128 (e.g., in terms of key size) and a particular attack on one may not be readily applicable to the other, it gives motivation to study the effect of partial state exposures in the cryptanalysis of other stream ciphers. If by side channel information or some fault model, half of the state of HC-128 becomes revealed, then that can be cascaded with our analysis to reveal the full state efficiently.

HC-128 is a relatively new cipher and more cryptanalytic research on this cipher is expected. We believe our work is just a first step towards state recovery of HC-128 and would generate interest in the community for further investigation in this direction.

2 Description of HC-128

This is adapted from [16, Section 2]. The following operations are used in HC-128:

$+ : x + y$ means $(x + y) \bmod 2^{32}$, where $0 \le x < 2^{32}$ and $0 \le y < 2^{32}$.
$\boxminus : x \boxminus y$ means $(x - y) \bmod 512$.
$\oplus$: bit-wise exclusive OR.
$\|$: concatenation.
$\gg$: right shift operator. $x \gg n$ means x being right shifted n bits.
$\ll$: left shift operator. $x \ll n$ means x being left shifted n bits.
$\ggg$: right rotation operator. $x \ggg n$ means
$((x \gg n) \oplus (x \ll (32 - n))$, where $0 \le n < 32$, $0 \le x < 2^{32}$.
$\lll$: left rotation operator. $x \lll n$ means
$((x \ll n) \oplus (x \gg (32 - n))$, where $0 \le n < 32$, $0 \le x < 2^{32}$.

Two tables P and Q, each with 512 many 32-bit elements are used as internal states of HC-128. A 128-bit key array $K[0, \ldots, 3]$ and a 128-bit initialization vector $IV[0, \ldots, 3]$ are used, where each entry of the array is a 32-bit element. Let s_t denote the keystream word generated at the t-th step, $t = 0, 1, 2, \ldots$.

The following six functions are used in HC-128:

$f_1(x) = (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3)$,
$f_2(x) = (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10)$,
$g_1(x, y, z) = ((x \ggg 10) \oplus (z \ggg 23)) + (y \ggg 8)$,
$g_2(x, y, z) = ((x \lll 10) \oplus (z \lll 23)) + (y \lll 8)$,
$h_1(x) = Q[x^{(0)}] + Q[256 + x^{(2)}]$,
$h_2(x) = P[x^{(0)}] + P[256 + x^{(2)}]$.

Here $x = x^{(3)} \| x^{(2)} \| x^{(1)} \| x^{(0)}$, x is a 32-bit word and $x^{(0)}$ (least significant byte), $x^{(1)}$, $x^{(2)}$ and $x^{(3)}$ (most significant byte) are four bytes.

The key scheduling of HC-128 is as follows.

Key and IV Setup:

1. Let $K[0, \ldots, 3]$ be the secret key and $IV[0, \ldots, 3]$ be the initialization vector. Let $K[i + 4] = K[i]$ and $IV[i + 4] = IV[i]$ for $0 \le i \le 3$.
2. The key and IV are expanded into an array $W[0, \ldots, 1279]$ as follows.

$$\begin{aligned} W[i] &= K[i], && \text{for } 0 \le i \le 7; \\ &= IV[i - 8], && \text{for } 8 \le i \le 15; \\ &= f_2(W[i - 2]) + W[i - 7] \\ &\quad + f_1(W[i - 15]) + W[i - 16] + i, && \text{for } 16 \le i \le 1279. \end{aligned}$$

3. Update the tables P and Q with the array W as follows.

$P[i] = W[i + 256]$, for $0 \le i \le 511$
$Q[i] = W[i + 768]$, for $0 \le i \le 511$

4. Run the cipher 1024 steps and use the outputs to replace the table elements as follows.

For $i = 0$ to 511, do
$P[i] = (P[i] + g_1(P[i \boxminus 3], P[i \boxminus 10], P[i \boxminus 511])) \oplus h_1(P[i \boxminus 12])$;
For $i = 0$ to 511, do
$Q[i] = (Q[i] + g_2(Q[i \boxminus 3], Q[i \boxminus 10], Q[i \boxminus 511])) \oplus h_2(Q[i \boxminus 12])$;

The keystream is generated using the following steps.

The Keystream Generation Algorithm:

$i = 0$;
repeat until enough keystream bits are generated
{
  $j = i \bmod 512$;
  if $(i \bmod 1024) < 512$
  {
    $P[j] = P[j] + g_1(P[j \boxminus 3], P[j \boxminus 10], P[j \boxminus 511])$;
    $s_i = h_1(P[j \boxminus 12]) \oplus P[j]$;
  }
  else
  {
    $Q[j] = Q[j] + g_2(Q[j \boxminus 3], Q[j \boxminus 10], Q[j \boxminus 511])$;
    $s_i = h_2(Q[j \boxminus 12]) \oplus Q[j]$;
  }
  end-if
  $i = i + 1$;
}
end-repeat

3 Reconstruction of One Internal Array from Another

We introduce a few notations for the ease of analysis. We describe them first and then move on to the actual strategy.

As discussed in Section 1, we consider four consecutive blocks B_1, B_2, B_3 and B_4. In B_1 and B_3, Q is updated. Let Q denote the updated array after the completion of block B_1 and let Q_N be the new array after Q is updated in block B_3. In B_1, P remains unchanged and in B_2, it is updated to P_N. Let $s_{b,i}$ denote the i-th keystream word produced in block B_b, $1 \leq b \leq 4$, $0 \leq i \leq 511$.

Consider that the keystream words $s_{b,i}$, $1 \leq b \leq 4$, $0 \leq i \leq 511$, are observable. We formulate a special state reconstruction problem as follows.

Given the partial state information $P[0 \ldots 511]$,
reconstruct the complete state $(P_N[0 \ldots 511], Q_N[0 \ldots 511])$.

Since the update of each of P and Q depends only on P and Q respectively, once we determine P_N and Q_N, we essentially recover the complete state information for all subsequent steps.

Our state reconstruction proceeds in five phases, described in next five subsections.

3.1 First Phase: Complete P_N from P

Observe that the update of P (or Q) depends only on itself, i.e.,

$$P_N[i] = \begin{cases} P[i] + g_1(P[509+i], P[502+i], P[i+1]), & \text{for } 0 \le i \le 2; \\ P[i] + g_1(P_N[i-3], P[502+i], P[i+1]), & \text{for } 3 \le i \le 9; \\ P[i] + g_1(P_N[i-3], P_N[i-10], P[i+1]), & \text{for } 10 \le i \le 510; \\ P[i] + g_1(P_N[i-3], P_N[i-10], P_N[i-511]), & \text{for } i = 511. \end{cases} \quad (1)$$

Thus, if one knows the 512 words of P (or Q) corresponding to any one block, then one can easily derive the complete P (or Q) array corresponding to any subsequent block.

The **First Phase** consists of determining P_N from P using Equation (1).

3.2 Second Phase: Part of Q from P_N

The keystream generation of block B_2 follows the equation

$$s_{2,i} = \begin{cases} h_1(P[500+i]) \oplus P_N[i], & \text{for } 0 \le i \le 11; \\ h_1(P_N[i-12]) \oplus P_N[i], & \text{for } 12 \le i \le 511. \end{cases} \quad (2)$$

Since $h_1(x) = Q[x^{(0)}] + Q[256 + x^{(2)}]$, we can rewrite Equation (2) as

$$Q[l_i] + Q[u_i] = s_{2,i} \oplus P_N[i] \quad (3)$$

$$\left.\begin{array}{l} \text{where for } 0 \le i \le 11,\ l_i = (P[500+i])^{(0)} \text{ and } u_i = 256 + (P[500+i])^{(2)} \\ \text{and for } 12 \le i \le 511,\ l_i = (P_N[i-12])^{(0)} \text{ and } u_i = 256 + (P_N[i-12])^{(2)}. \end{array}\right\} \quad (4)$$

Note that l_i, u_i and the right hand side $s_{2,i} \oplus P_N[i]$ of System (3) of equations are known for all $i = 0, 1, \ldots, 511$. Thus, there are 512 equations in ≤ 512 unknowns and so applying Gauss elimination would require a complexity of $512^3 = 2^{27}$. However, we show in Lemma 1 that a unique solution does not exist for any such system and hence we have to take a different approach to solve the system. Though the proof of Lemma 1 is simple, we include here for easy reference.

Lemma 1. *Suppose $r + s$ linear equations are formed using variables from the set $\{x_1, x_2, \ldots, x_r, y_1, y_2, \ldots, y_s\}$, $r \ge 1$, $s \ge 1$. If each equation is of the form $x_i + y_j = b_{ij}$ for some i in $[1, r]$ and some j in $[1, s]$, where b_{ij}'s are all known, then such a system does not have a unique solution.*

Proof. Consider the $(r + s) \times (r + s)$ coefficient matrix A of the system with the columns denoted by $C_1, \ldots, C_{r+s}$, such that the first r columns $C_1, \ldots, C_r$ correspond to the variables $x_1, \ldots, x_r$ and the last s columns $C_{r+1}, \ldots, C_{r+s}$ correspond to the variables $y_1, \ldots, y_s$. Every row of A has the entry 1 in exactly two places and the entry 0 elsewhere. The first 1 in each row appears in one of the columns $C_1, \ldots, C_r$ and the second 1 in one of the columns $C_{r+1}, \ldots, C_{r+s}$. After the elementary column transformations $C_1 \leftarrow C_1 + \ldots + C_r$ and $C_{r+1} \leftarrow C_{r+1} + \ldots + C_{r+s}$, the two columns C_1 and C_{r+1} has 1's in all the rows and hence become identical. This implies that the matrix is not of full rank and hence unique solution does not exist for the system. □

The left hand side of every equation in System (3) is of the form $Q[l_i] + Q[u_i]$, where $0 \leq l_i \leq 255$ and $256 \leq u_i \leq 511$. Taking $r = s = 256$, $x_i = Q[i-1]$, $1 \leq i \leq 256$ and $y_j = Q[255+j]$, $1 \leq j \leq 256$, we see that Lemma 1 directly applies to this system, establishing the non-existence of a unique solution. At this stage, one could remove the redundant rows to find a linear space which contains the solution. However, it is not clear how many variables need to be guessed to arrive at the final solution. Below we formulate a graph theoretic approach to derive the entries of the array Q efficiently, by guessing the value of only a single variable.

Definition 1. *System (3) of 512 equations can be represented in the form of a bipartite graph $G = (V_1, V_2, E)$, where $V_1 = \{0, \ldots, 255\}$, $V_2 = \{256, \ldots, 511\}$ and for each term $Q[l_i] + Q[u_i]$ of System (3) of equations, there is an edge $\{l_i, u_i\} \in E$, $l_i \in V_1$ and $u_i \in V_2$. Thus, $|E| = 512$ (counting repeated edges, if any). We call such a graph G with the vertices as the indices of one internal array of HC-128 the* index graph *of the state of HC-128.*

Lemma 2. *Let M be the size of the largest connected component of the index graph G corresponding to block B_2. Then M out of 512 words of the array Q can be derived in 2^{32} search complexity.*

Proof. Consider any one of the 512 equations of System (3). Since the sum $Q[l_i] + Q[u_i]$ is known, knowledge of one of $Q[l_i]$, $Q[u_i]$ reveals the other. Thus, if we know one word of Q at any index of a connected component, we can immediately derive the words of Q at all the indices of the same component. Since this holds for each connected component, we can guess any one 32-bit word in the largest connected component correctly in 2^{32} attempts and thereby the result follows. □

The arrays P, Q and the keystream of HC-128 can be assumed to be random. Hence our *index graph* G can be considered to be a random bipartite graph. Analysis of the size distribution of the connected components of random finite graphs is a vast area of research in applied probability and there have been several works [6,14,12,5,2] in this direction under different graph models. In [14], the model considered is a bipartite graph $G(n_1, n_2, T)$ with n_1 vertices in the first part, n_2 vertices in the second one and the graph is constructed by T independent trials, each of them consists of drawing an edge which joins two vertices chosen independently of each other from distinct parts. This coincides with our index graph model of Definition 1 with $n_1 = |V_1|$, $n_2 = |V_2|$ and $T = |E|$.

In general, let $n_1 \geq n_2$, $\alpha = \frac{n_2}{n_1}$, $\beta = (1-\alpha)\ln n_1$, $n = n_1 + n_2$. Let $\xi_{n_1,n_2,T}$ and $\chi_{n_1,n_2,T}$ respectively denote the number of isolated vertices and the number of connected components in $G(n_1, n_2, T)$. We have the following result from [14].

Proposition 1. *If $n \to \infty$ and $(1+\alpha)T = n \ln n + Xn + o(n)$, where X is a fixed number, then Prob $(\chi_{n_1,n_2,T} = \xi_{n_1,n_2,T} + 1) \to 1$ and for any $k = 0, 1, 2, \ldots$, Prob $(\xi_{n_1,n_2,T} = k) - \frac{\lambda^k e^{-\lambda}}{k!} \to 0$, where $\lambda = \frac{e^{-X}(1+e^{-\beta})}{1+\alpha}$.*

In other words, if n is sufficiently large and n_1, n_2, T are related by $(1+\alpha)T = n \ln n + Xn + o(n)$, then the graph contains one giant connected component and some isolated vertices whose number follows a Poisson distribution with parameter λ given above.

Corollary 1. *If M is the size of the largest component of the index graph G, then the mean and standard deviation of M is respectively given by $E(M) \approx 442.59$ and $sd(M) \approx 8.33$.*

Proof. For our index graph, $n_1 = n_2 = 256$, $n = n_1 + n_2 = 512$, $T = 512$, $\alpha = \frac{n_2}{n_1} = 1$, $\beta = (1-\alpha)\ln n_1 = 0$. The relation $(1+\alpha)T = n \ln n + Xn + o(n)$ is equivalent to $\frac{(1+\alpha)}{n}T = \ln n + X + \frac{o(n)}{n}$. As $n \to \infty$, the ratio $\frac{o(n)}{n} \to 0$ and hence $X \to \frac{(1+\alpha)}{n}T - \ln n$. Substituting $\alpha = 1$, $T = 512$ and $n = 512$, we get $X \approx -4.24$. By Proposition 1, the limiting distribution of the random variable $\xi_{n_1,n_2,T}$ is Poisson with mean (as well as variance) $\lambda = \frac{e^{-X}(1+e^{-\beta})}{1+\alpha} \approx e^{4.24} \approx 69.41$. Moreover, in the limit, $\chi_{n_1,n_2,T} = \xi_{n_1,n_2,T} + 1$ and this implies that all the vertices except the isolated ones would be in a single giant component. So $M = n - \xi_{n_1,n_2,T}$ and the expectation $E(M) = n - E(\xi_{n_1,n_2,T}) = n - \lambda \approx 512 - 69.41 = 442.59$. Again, the variance $Var(M) = Var(n - \xi_{n_1,n_2,T}) = Var(\xi_{n_1,n_2,T}) = \lambda$, giving $sd(M) = sd(\xi_{n_1,n_2,T}) = \sqrt{\lambda} \approx 8.33$. □

We simulate 10 million trials of HC-128, each time generating 1024 consecutive words of keystream corresponding to the complete arrays P and Q. In our experiments, the average of the number $\xi_{n_1,n_2,T}$ of isolated vertices of the index graph of the state of HC-128 was found to be 69.02 with a standard deviation of 6.41. These values closely match with the theoretical estimates of the mean $\lambda \approx 69.41$ and standard deviation $\sqrt{\lambda} \approx 8.33$ of $\xi_{n_1,n_2,T}$ derived in Corollary 1.

Again from Corollary 1, theoretical estimates of the mean and standard deviation of the size M of the largest component is 442.59 and 8.33 respectively. From the same simulation described above, the empirical average and standard deviation of M are found to be $407.91 \approx 408$ and 9.17 respectively.

Observe that there is a small gap between the theoretical estimate and the empirical result of the mean value of M. It arises from the fact that the theory takes n as infinity, but its value in the context of HC-128 is 512. In the limit when $n \to \infty$, each vertex is either an isolated one or it belongs to the single giant component. In practice, on the other hand, except the isolated vertices (≈ 69 in number) and the vertices of the giant component (≈ 408 in number), the remaining few ($\approx 512 - 69 - 408 = 35$ in number) vertices form some small components. However, the low (9.17) empirical standard deviation of M implies the robustness of the empirical estimate 408 of $E(M)$. Later we show that as a consequence of Theorem 2, any $M > 200$ is sufficient for our purpose.

Suppose $C = \{y_1, y_2, \ldots, y_M\}$ is the largest component of G. Algorithm 1 summarizes the steps to obtain the set of indices $y_1, y_2, \ldots, y_M$ from the System (3) of equations.

Input: System (3) and (4) of equations.
Output: Set $\{y_1, y_2, \ldots, y_M\}$ corresponding to the largest component of the index graph G.

Form a bipartite graph $G = (V_1, V_2, E)$ as follows;
$V_1 \leftarrow \{0, \ldots, 255\}$; $V_2 \leftarrow \{256, \ldots, 511\}$; $E \leftarrow \emptyset$;
for $i \leftarrow 0$ **to** 511 **do**
    Determine l_i and u_i using Equation (4);
    $E \leftarrow E \cup \{l_i, u_i\}$;
end
Find all connected components of G;
Let $C = \{y_1, y_2, \ldots, y_M\}$ be the largest component with size M;

Algorithm 1. FindMaxCC

After applying algorithm *FindMaxCC*, we can guess the word corresponding to any fixed index, say y_1. As explained in the proof of Lemma 2, each guess of $Q[y_1]$ uniquely determines the values of $Q[y_2], \ldots, Q[y_M]$. According to Corollary 1 and the discussion following it, we can guess around 408 words of Q in this method. This is the **Second Phase** of our solution.

3.3 Third Phase: Tail of Q from Its Parts

We use the following result, which we call *Propagation Theorem*, to determine the remaining unknown words.

Theorem 1 (Propagation Theorem). *If $Q[y]$ is known for some y in $[0, 499]$, then $m = \lfloor \frac{511-y}{12} \rfloor$ more words of Q, namely, $Q[y+12], Q[y+24], \ldots, Q[y+12m]$, can all be determined from $Q[y]$ in a time complexity that is linear in the size of Q.*

Proof. Consider block B_1. Following our notation in the beginning of Section 3, the equation for keystream generation is

$s_{1,i} = h_2(Q[i-12]) \oplus Q[i]$, for $12 \le i \le 511$.

Written in another way, it becomes

$Q[i] = s_{1,i} \oplus \left(P\left[(Q[i-12])^{(0)}\right] + P\left[256 + (Q[i-12])^{(2)}\right]\right)$.

Setting $y = i - 12$, we have, for $0 \le y \le 499$,

$$Q[y+12] = s_{1,y+12} \oplus \left(P\left([Q[y])^{(0)}\right] + P\left[256 + (Q[y])^{(2)}\right]\right) \tag{5}$$

This is a recursive equation, in which all s_1 values and the array P are completely known. Clearly, if we know one $Q[y]$, we know all subsequent $Q[y + 12k]$, for $k = 1, 2, \ldots$, as long as $y + 12k \le 511$. This means $k \le \frac{511-y}{12}$. The number m of words of Q that can be determined is then the maximum allowable value of k, i.e., $m = \lfloor \frac{511-y}{12} \rfloor$. □

We recursively apply Equation (5) to the words of the Q array determined from the maximum size connected component of the index graph and derive many

of around $104(= 512 - 408)$ unknown words in the array. This is the **Third Phase** of our solution. If we imagine the words initially marked as 'known' or 'unknown', then this step can be visualized as propagation of the 'known' labels in the forward direction. We represent this procedure in the form of Algorithm 2, called *PropagateQ*.

Input: $Q[0 \dots 511]$ with known entries in indices $y_1, y_2, \dots, y_M$.
Output: $Q[0 \dots 511]$ with fewer unknowns.
for $j \leftarrow 1$ **to** M **do**
  $y \leftarrow y_j$;
  while $y \leq 499$ **do**
    if $Q[y+12]$ *is still unknown* **then**
      $Q[y+12] \leftarrow s_{1,y+12} \oplus \left(P\left[(Q[y])^{(0)}\right] + P\left[256 + (Q[y])^{(2)}\right]\right)$;
    end
    $y \leftarrow y + 12$;
  end
end

Algorithm 2. PropagateQ

Even after this phase, some words of Q remain unknown. However, as Theorem 2 implies, we observe that through this propagation, all the words $Q[500]$, $Q[501]$, ..., $Q[511]$ can be 'known' with probability almost 1.

Theorem 2. *After the execution of Algorithm* PropagateQ*, the expected number of unknown words amongst* $Q[500]$, $Q[501]$, ..., $Q[511]$ *is approximately* $8 \cdot (1 - \frac{43}{512})^M + 4 \cdot (1 - \frac{42}{512})^M$*, where* M *is the size of the largest component of the index graph* G*.*

Proof. After the Second Phase, exactly M words $Q[y_1]$, $Q[y_2]$, ..., $Q[y_M]$ are known corresponding to the distinct indices y_1, y_2, ..., y_M in the largest component C of size M in G. Since G is a random bipartite graph, each of indices $y_1, y_2, \dots y_M$ can be considered to be drawn from the set $\{0, 1, \dots, 511\}$ uniformly at random (without replacement). We partition this sample space into 12 disjoint residue classes modulo 12, denoted by, $[0], [1], \dots, [11]$. Then, each of the indices $y_1, y_2, \dots, y_M$ can be considered to be drawn from the set $\{[0], [1], \dots, [11]\}$ (this time with replacement; this is a reasonable approximation because $M \gg 12$) with probabilities proportional to the sizes of the residue classes. Thus, for $1 \leq j \leq M$, $Prob(y_j \in [r]) = \frac{43}{512}$ if $0 \leq r \leq 7$ and $\frac{42}{512}$ if $8 \leq r \leq 11$.

Let $m_r = 1$, if none of $y_1, y_2, \dots, y_M$ are from $[r]$; otherwise, let $m_r = 0$. Hence, the total number of residue classes from which no index is selected is $Y = \sum_{r=0}^{11} m_r$. Now, in the Third Phase, we propagate the known labels in the

forward direction using Equation (5) (see Theorem 1, the Propagation Theorem). The indices $\{500, 501, \ldots, 511\}$ are to the extreme right end of the array Q and hence they also form the set of "last" indices where the propagation eventually stops. Further, each index in the set $\{500, 501, \ldots, 511\}$ belongs to exactly one of the sets $[r]$. Hence, the number of unknown words amongst $Q[500]$, $Q[501]$, ..., $Q[511]$ is also given by Y.

We have,

$$E(m_r) = Prob(m_r = 1) = \begin{cases} (1 - \frac{43}{512})^M \text{ for } 0 \leq r \leq 7; \\ (1 - \frac{42}{512})^M \text{ for } 8 \leq r \leq 11. \end{cases}$$

Thus, $E(Y) = \sum_{r=0}^{11} E(m_r) = 8 \cdot (1 - \frac{43}{512})^M + 4 \cdot (1 - \frac{42}{512})^M$. □

Substituting M by its theoretical mean estimate 443 as well as by its empirical mean estimate 408 yields $E(Y) \approx 0$. In fact, for any $M > 200$, the expression $(1 - \frac{43}{512})^M + 4 \cdot (1 - \frac{42}{512})^M$ for $E(Y)$ becomes vanishingly small. Our experimental data also supports that in every instance, none of the words $Q[500], Q[501], \ldots, Q[511]$ remains unknown.

Remarks

1. The probability that one particular 8-bit pattern is missing from 512 many randomly selected 8-bit segments of the P array, that are used to form the indices u_i's in System (3) of equations, is $(1 - 2^{-8})^{512} \approx 0.13$. Assuming that the missing unknown is equally likely to be one of $\{Q[256], \ldots, Q[511]\}$, the probability that there is one missing unknown and it is in $\{Q[500], \ldots, Q[511]\}$ is $\approx 0.13 \cdot \frac{12}{256} \approx 0.0061$. Thus, one may be tempted to conclude that the probability that at least one of $\{Q[500], \ldots, Q[511]\}$ remains unknown is non-negligible.

   However, we like to point out that the analysis in the above paragraph corresponds to the unknown values in System (3) of equations, i.e., after the First Phase of the solution. On the other hand, the analysis in Theorem 2 corresponds to the unknown values after we have propagated the known indices using the Propagation Theorem, i.e., after the Third Phase of the solution.
2. Changing bytes 1 or 3 of $Q[y]$ yields no change in Equation (5). Combining this with the Second Phase, we could form a new set of equations and attempt to solve them. However, as Theorem 2 establishes, this is not required; propagation of known $Q[y]$ values in steps of 12 covers all the unknowns.

3.4 Fourth Phase: Complete Q_N from Tail of Q

Next, we use the following result to determine the entire Q_N array.

Theorem 3. *Suppose the complete array P_N and the 12 words $Q[500]$, $Q[501]$, ..., $Q[511]$ from the array Q are known. Then the entire Q_N array can be reconstructed in a time complexity linear in the size of Q.*

Proof. Following our notation in the beginning of Section 3, the equation for the keystream generation of the first 12 steps of block B_3 is $s_{3,i} = h_2(Q[500+i]) \oplus Q_N[i]$, $0 \le i \le 11$. Expanding $h_2(.)$, we get, for $0 \le i \le 11$,

$$Q_N[i] = s_{3,i} \oplus \left(P_N \left[(Q[500+i])^{(0)} \right] + P_N \left[256 + (Q[500+i])^{(2)} \right] \right).$$

Thus, we can determine $Q_N[0], Q_N[1], \ldots Q_N[11]$ from $Q[500], Q[501], \ldots Q[511]$. Now, applying Theorem 1 on these first 12 words of Q_N, we can determine all the words of Q_N in linear time (in size of Q). □

Due to Theorem 3, we can devise an algorithm called *PropagateQTail_to_QNext*. Applying this algorithm constitutes the **Fourth Phase** of our solution.

Input: $Q[0 \ldots 511]$ with known entries in indices $500, \ldots, 511$.
Output: $Q_N[0 \ldots 511]$ with all entries known.
for $i \leftarrow 0$ **to** 11 **do**
  $Q_N[i] \leftarrow s_{3,i} \oplus \left(P_N \left[(Q[500+i])^{(0)} \right] + P_N \left[256 + (Q[500+i])^{(2)} \right] \right)$;
  $y \leftarrow i$;
  while $y \le 499$ **do**
    $Q_N[y+12] \leftarrow s_{3,y+12} \oplus \left(P_N \left[(Q_N[y])^{(0)} \right] + P_N \left[256 + (Q_N[y])^{(2)} \right] \right)$;
    $y \leftarrow y + 12$;
  end
end

Algorithm 3. PropagateQTail_to_QNext

3.5 Fifth Phase: Verification

After Q_N is derived, we need to verify its correctness. For this, we update P_N as it would be updated in block B_4 and generate 512 keystream words with this P_N and the derived Q_N. If the generated keystream words entirely match with the observed keystream words $\{s_{4,0}, s_{4,1}, \ldots, s_{4,511}\}$ of block B_4, then our guess is correct. This verification is the **Fifth** (and final) **Phase** of our algorithm. If we find a mismatch, then we repeat the procedure with the next guess, i.e., with another possible value in $[0, 2^{32} - 1]$ of the word $Q[y_1]$.

Once Q_N is correctly determined, the words of the Q array for all the succeeding blocks can be deterministically computed from the update rule for Q.

3.6 Complete State Reconstruction Algorithm and Total Complexity

The above discussion is formalized in Algorithm 4, called *ReconstructState*.

Input: $P[0 \dots 511]$.
Output: $P_N[0 \dots 511], Q_N[0 \dots 511]$.
1 **for** $i \leftarrow 0$ **to** 511 **do**
2 | Determine $P_N[i]$ using Equation (1);
end
3 Call *FindMaxCC* to find out $y_1, y_2, \dots, y_M$;
4 **for** *Each possible value of* $Q[y_1]$ **do**
5 | Determine $Q[y_2], \dots, Q[y_M]$ from System (3);
6 | Call *PropagateQ*;
7 | Call *PropagateQTail_to_QNext*;
8 | With the new Q_N, generate 512 keystream words by updating P_N;
9 | Verify correctness of the guess in Step 4 by matching these keystream words with the observed keystream words of block B_4;
end

Algorithm 4. ReconstructState

Theorem 4. *The data complexity of Algorithm 4 is* 2^{16} *and its time complexity is* 2^{42}.

Proof. For the First Phase, we do not need any keystream word. For each of the Second, Third, Fourth and Fifth Phases, we need a separate block of 512 keystream words. Thus, the required amount of data is $4 \cdot 512 = 2^{11}$ no. of 32 $(= 2^5)$-bit keystream words.

Step 1 of Algorithm 4 takes time linear in the size of P, i.e., of complexity 2^9. Then Step 3 of Algorithm 4 calls the *FindMaxCC* subroutine.

For all the steps inside *FindMaxCC* up to Step 5, the total time required is linear in the size of Q, i.e., of complexity 2^9. Step 6 of *FindMaxCC* can be performed through depth-first search which requires $O(|V_1| + |V_2| + |E|)$ time complexity. For $|V_1| = 256$, $|V_2| = 256$ and $|E| = 512$, the value turns out to be 2^{10}.

After returning from *FindMaxCC*, the For loop at Step 4 of Algorithm 4 needs to be iterated 2^{32} times and for each iteration, the statements inside take time that is linear in the size of the array Q, i.e., of complexity 2^9. Thus, the total time required is $2^9 + 2^{10} + 2^{32} \cdot 2^9 < 2^{42}$. □

Note that for System (3) of equations, one must verify the solution by first generating some keystream words and then matching them with the observed keystream, as is done in Algorithm 4. During Step 4 (i.e., in the Second Phase), one may exploit the cycles of the largest component to verify correctness of the guess. If the guessed value of a variable in a cycle does not match with the value of the variable derived when the cycle is closed, we can discard that guess. However, in the worst case, all the 2^{32} guesses have to be tried and if there is no conflict in a cycle, the guess has to be verified by keystream matching. Thus, it is not clear if there is any significant advantage by detecting and exploiting the cycles and so we have not considered this in the description of the algorithm.

4 Design Modification with Respect to Known Observations

Here we propose a modification of the HC-128 cipher. We have two design goals:

- to guard against the available analysis in literature and
- not to sacrifice the speed in the process.

Thus, we attempt to keep the same structure as the original HC-128 with minimal changes.

Apart from the present paper, we are aware of three other works on the analysis of keystream generation algorithm of HC-128, one by the designer Wu [16], another in [10] and the most recent one from [7].

The works [16,10] exploit the fact that $h_1(.)$ as well as $h_2(.)$ makes use of only 16 bits from the 32-bit input. Our current work also uses this fact to form System (3) of equations, that eventually leads to reconstruction of the state. Thus, all of these results indicate that the form of $h_1(.), h_2(.)$ need to be modified so as to incorporate all the 32 bits of their inputs. In our new versions of these functions (Equation (6)), we suggest XOR-ing the entire input with the existing output (sum of two array entries). However, certain precautions may need to be taken so that other security threats do not come into play.

We replace h_1 and h_2 as follows.

$$\left.\begin{aligned} h_{N1}(x) &= (Q[x^{(0)}] + Q[256 + x^{(2)}]) \oplus x. \\ h_{N2}(x) &= (P[x^{(0)}] + P[256 + x^{(2)}]) \oplus x. \end{aligned}\right\} \tag{6}$$

We need to modify the update functions g_1 and g_2 with the twin motivation of preserving the internal state as well as making sure that the randomness of the keystream is ensured. We propose the following:

$$\left.\begin{aligned} g_{N1}(x, y, z) &= \big((x \ggg 10) \oplus (z \ggg 23)\big) + Q[(y \gg 7) \wedge 1FF]. \\ g_{N2}(x, y, z) &= \big((x \lll 10) \oplus (z \lll 23)\big) + P[(y \gg 7) \wedge 1FF]. \end{aligned}\right\} \tag{7}$$

We keep f_1 and f_2 the same as in original HC-128.

We include a randomly chosen word from the Q array in the update of P array elements and a randomly chosen word from the P array while updating the Q array elements. This would ensure that each new block of P (or Q) array is dependent on the previous block of Q(or P) array. Thus, our analysis of Section 3 would not apply and the internal state would be preserved even if half the internal state elements are known.

Likewise, in the equation of the distinguisher proposed by the designer [16, Section 4], the term $P[i \boxminus 10]$ will get replaced by some random term of Q array. With this replacement, it is not obvious how a similar distinguishing attack can be mounted. The similar situation will happen for the distinguishers proposed in [10].

Now let us concentrate on the fault attack presented in [7]. The fault analysis in [7] assumes that if a fault occurs at $Q[f]$ in the block in which P is updated,

then $Q[f]$ is not referenced until step $f-1$ of the next block (in which Q would be updated). This assumption does not hold for our design due to the nesting use of P and Q in the updates of one another (Equation (7)). Thus, on our modified design, the fault position recovery algorithm given in [7, Section 4.2] would not work immediately. In particular, Lemma 1 and Lemma 2 of [7] would not hold on our modified cipher.

The security of any stream cipher is always a conjecture. We have tried to circumvent the known weaknesses of HC-128. The way we have modified the design, it appears that no new security holes are introduced. However, the new design is open to the community for further analysis.

4.1 Performance Evaluation

We evaluated the performance of our new design using the eSTREAM testing framework [1]. The C-implementation of the testing framework was installed in a machine with Intel(R) Pentium(R) D CPU, 2.8 GHz Processor Clock, 2048 KB Cache Size, 1 GB DDR RAM on Ubuntu 7.04 (Linux 2.6.20-17-generic) OS. A benchmark implementation of HC-128 and HC-256 [17] is available within the test suite. We implemented our modified version of HC-128, maintaining the API compliance of the suite. Test vectors were generated in the NESSIE [13] format. The results presented below correspond to tests with null IV using the gcc-3.4_prescott_O3-ofp compiler.

	HC-128	Our Proposal	HC-256
Stream Encryption (cycles/byte)	4.13	4.29	4.88

The encryption speed of our proposed design is of the same order as that of original HC-128. We also observe that the extra array element access in the new update rules (Equation (7)) as compared to the original update rules does not affect the performance much. HC-128 was designed as a lightweight version of HC-256. The idea of cross-referencing each other in the update rules of P and Q has also been used in the design of HC-256 and that is why the half state exposure does not reveal the full state in case of HC-256. However, our modification to HC-128 removes the known weaknesses of HC-128 but keeps the speed much better than HC-256, with only little reduction in speed compared to HC-128.

5 Conclusion

The eSTREAM candidate HC-128 uses two internal arrays, each containing 512 many 32-bit words. In this paper, we show that one of the two arrays is redundant

in the sense that if one knows only one array completely and has access to 2048 consecutive keystream words then the other array can be completely reconstructed in 2^{42} time complexity. This does not immediately pose a threat to the security of the cipher. However, this is a structural weakness which has not been studied before the present work. We also propose a design modification of HC-128 that avoids the half state exposure analysis as well as is free from the known weaknesses in the literature. We evaluate the performance of our proposal in the eSTREAM testing framework and compare the speed with that of HC-128 and HC-256.

References

1. Cannière, C.D.: eSTREAM testing framework, http://www.ecrypt.eu.org/stream/perf (last accessed on September 12, 2011)
2. Cooper, C., Frieze, A.: The Size of the Largest Strongly Connected Component of a Random Digraph with a Given Degree Sequence. Combinatorics, Probability and Computing 13(3), 319–337 (2004)
3. Dunkelman, O.: A small observation on HC-128, http://www.ecrypt.eu.org/stream/phorum/read.php?1,1143 (November 14, 2007) (last accessed on September 12, 2011)
4. http://www.ecrypt.eu.org/stream/ (last accessed on September 12, 2011)
5. Hansen, J., Jaworski, J.: Large components of bipartite random mappings. Random Structures & Algorithms 17(3-4), 317–342 (2000)
6. Kalugin, I.B.: The number of components in a random bipartite graph. Diskretnaya Matematika 1(3), 62–70 (1989)
7. Kircanski, A., Youssef, A.M.: Differential Fault Analysis of HC-128. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 261–278. Springer, Heidelberg (2010)
8. Knudsen, L.R., Meier, W., Preneel, 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)
9. Liu, Y., Qin, T.: The key and IV setup of the stream ciphers HC-256 and HC-128. In: International Conference on Networks Security, Wireless Communications and Trusted Computing, pp. 430–433 (2009)
10. Maitra, S., Paul, G., Raizada, S., Sen, S., Sengupta, R.: Some Observations on HC- 128. Designs, Codes and Cryptography 59(1-3), 231–245 (2011); This is a revised and extended version of the paper with the same title, written by the first three authors, that appeared in Pre-Proceedings of the International Workshop on Coding and Cryptography (WCC), Ullensvang, Norway, May 10-15, pp. 527–539 (2009)
11. 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)
12. Molloy, M., Reed, B.: The Size of the Giant Component of a Random Graph with a Given Degree Sequence. Combinatorics, Probability and Computing 7, 295–305 (1998)
13. New European Schemes for Signatures, Integrity, and Encryption, https://www.cosic.esat.kuleuven.be/nessie (last accessed on September 12, 2011)

14. Saltykov, A.I.: The number of components in a random bipartite graph. Diskretnaya Matematika 7(4), 86–94 (1995)
15. Shiraishi, Y., Ohigashi, T., Morii, M.: An Improved Internal-state Reconstruction Method of a Stream Cipher RC4. In: Hamza, M.H. (ed.) Proceedings of Communication, Network, and Information Security, Track 440-088, New York, USA, December 10-12 (2003)
16. Wu, H.: The Stream Cipher HC-128, http://www.ecrypt.eu.org/stream/hcp3.html (last accessed on September 12, 2011)
17. Wu, H.: A New Stream Cipher HC-256. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 226–244. Springer, Heidelberg (2004) The full version is, http://eprint.iacr.org/2004/092.pdf (last accessed on September 12, 2011)

Experimental Verification of Super-Sbox Analysis — Confirmation of Detailed Attack Complexity

Yu Sasaki[1], Naoyuki Takayanagi[2], Kazuo Sakiyama[2], and Kazuo Ohta[2]

[1] NTT Information Sharing Platform Laboratories, NTT Corporation
3-9-11 Midori-cho, Musashino-shi, Tokyo 180-8585 Japan
sasaki.yu@lab.ntt.co.jp
[2] The University of Electro-Communications
1-5-1 Choufugaoka, Choufu-shi, Tokyo, 182-8585 Japan
{saki,ota}@inf.uec.ac.jp

Abstract. This paper implements the super-sbox analysis on 8-round AES proposed by Gilbert and Peyrin in order to verify its correctness and the attack cost. The attack consists of three parts; the first outbound phase, inbound phase with a super-sbox technique, and the second outbound phase. Gilbert and Peyrin estimated that the attack would require 2^{48} computational cost and 2^{32} memory, which could be feasible but not easy to practically implement. In this research, we first analyze the relationship among memory, computational cost, and the number of solutions in the inbound phase, and then show that the tradeoff exists for the super-sbox analysis. With this tradeoff, we implement the attack for each of the outbound phase independently so that the cost for the entire attack can be estimated by the experiments. As a result of our experiment, we show that the computational cost to obtain a pair of values satisfying the inbound phase is approximately 4 times higher and the freedom degrees are 4 times smaller than the previous estimation, which indicates that applying the super-sbox analysis is harder than expected.

Keywords: super-sbox analysis, time-memory tradeoff, AES, AES based hash.

1 Introduction

Hash functions are used in the wide range of cryptographic applications. Since the break of MD5 and SHA-1, [1,2], cryptographers have been seeking secure and efficient hash function constructions. Currently, various types of hash functions can be seen, especially in the SHA-3 competition [3].

One of the most solid approaches for designing hash functions is the construction based on AES [4,5]. For the security evaluation, the following two factors are important; 1) provable security and 2) enough amount of dedicated analysis. From these viewpoints, AES based hash functions have the following strong points; 1) AES round function, which consists of S-box and MDS matrix, gives

T. Iwata and M. Nishigaki (Eds.): IWSEC 2011, LNCS 7038, pp. 178–192, 2011.

the lower bound of the number of active S-boxes in the differential trail. Thus the upperbound of the differential and linear characteristics can be evaluated. 2) Regardless of the fact that AES, as a block-cipher, has been analyzed for more than 10 years, no significant weakness is detected in a single-key model. This gives the trust about the security of the AES based structure. In fact, there are many hash functions that are designed based on AES, for example Whirlpool [6], Grøstl [7], ECHO [8], and SHAvite-3 [9].

Recently, an outstanding progress in the known-key attack [10] on AES and cryptanalysis against AES based hash functions or permutations has been made. Specifically, the rebound attack proposed by Mendel *et al.* at FSE 2009 [11] works efficiently against AES based structures. After its publication, many improvements upon the rebound attack have been proposed, such as start-from-the-middle attack [12], super-sbox analysis on the rebound attack [13,14], rebound attack with multiple inbound phases [15], and scrutinized rebound attack [16]. Furthermore, several techniques were proposed as applications of the super-sbox analysis, for example, hyper-sbox analysis [17] and non-full-active super-sbox analysis [18].

Different from the rebound attack, super-sbox analysis requires expensive attack cost especially for memory, e.g. 2^{32} computational cost and 2^{32} memory for AES, or 2^{64} computational cost and 2^{64} memory for Grøstl and Whirlpool. In [18], Sasaki *et al.* changed the differential path so that several bytes in the super-sbox part are not activated. This could reduce the attack cost to a practical range, but instead, the attacker's advantage in their distinguisher was also reduced. As far as we know, any method to reduce the attack cost especially for memory with the original differential path (activating all bytes) is not proposed.

Regarding the rebound attack (without the super-sbox), the attack complexity is strictly evaluated by considering the details of the differential distribution table of the S-box and the property of the MDS matrix. On the other hand, regarding the super-sbox, which consists of the mixture of two S-box layers and one linear diffusion layer, its behavior is hard to strictly evaluate and thus has not been discussed deeply so far. While there is a consensus that the complexity evaluation of the super-sbox analysis in [13] is reasonable, the analysis still lacks the theoretical and precise explanation, and as far as we know there is no experimental result showing the cost of the super-sbox analysis. Therefore, there could be a constant-time difference between the widely believed complexity and the actual attack cost.

In several previous analyses, the available freedom degrees is really tight, e.g. distinguishers for the old version of the full Grøstl compression function [19,20] has 2^z freedom degrees to satisfy the differential characteristic with a probability of 2^{-z}. In such a case, a constant-time increase of the attack cost, and thus, a constant-time decrease of freedom degrees will give a big impact on the success probability of the attack. In addition, by taking into account the fact that several applications of the super-sbox analysis were proposed, verifying the cost of the super-sbox analysis by implementing the attack is useful.

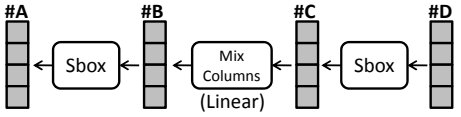


Fig. 1. Internal structure of the super-sbox analysis against AES

An Example of Previous Optimistic Complexity Evaluation

The detailed structure of the super-sbox against AES is described in Fig. 1. All bytes in all states must have a difference. First, the difference of State #D is randomly chosen. Then, for all possible 2^{32} paired values at State #D, the corresponding difference and values at State #A is computed. Previous work [13] assumed that all 2^{32} computations make State #A full active. However, this is not true because the differences in several bytes may be cancelled in the middle (at State #B). Hence, the number of solutions of the super-sbox analysis is obviously less than the previous estimation. The impact of the loss only from this observation is small, but this motivates us to do the precise complexity evaluation of the super-sbox analysis.

Our Contributions

In this paper, we implement the super-sbox analysis against 8-round AES [13]. The attack in [13] requires 2^{48} 8-round AES computations and a memory for storing 2^{32} AES state, which is 2^{36} bytes, namely 64 GBytes.

First of all, we consider the time-memory tradeoff for the super-sbox analysis, namely, obtaining the same number of solutions of the inbound phase with a reduced memory and an increased computational cost. Developing the time-memory tradeoff gives us several benefits.

1. The super-sbox analysis is mainly useful for the evaluation in the hash function setting. Many AES based hash functions use a larger internal state size. For example, the super-sbox analysis against Whirlpool and Grøstl requires 2^{64} computational cost and a memory for 2^{64} internal state. If the time-memory tradeoff is available, this can be 2^{80} computational cost and 2^{48} memory. Hence, the time-memory tradeoff will be useful in the future.
2. Developing the tradeoff is a usual way to proceed in estimating the efficiency of the attack.
3. The attack with 64-GByte memory can be implemented with a machine whose price is about 10,000 dollars. If the memory size is much smaller, the attack can be implemented with cheaper price.

Let us denote the computational cost for computing each super-sbox by T, the memory size for each super-sbox by M, the number of iterations of the inbound phase by N, and the number of obtained solutions of the inbound phase by D, We show that the following tradeoff exists in the super-sbox analysis;

$$D = N \cdot M \cdot (T/2^{32})^4.$$

Therefore, the same results as previous super-sbox analysis can be obtained by reducing the memory size M and increasing the computational cost coming from N by the same factor.

We then separate the differential path into the first half and the last half, and implement the attack for these two parts independently. Separating the path makes the cost for the experiment to be 2^{24} 4-round AES computations in time and 2^{32} AES state in memory. We then apply the tradeoff to change the costs into 2^{32} 4-round AES computations in time and 2^{24} AES state in memory. Finally, we can implement the attack with a PC which is much cheaper than the one with more than 64-GByte memory. We assume that the number of solutions D for the super-sbox (with 2^{32} AES state in memory) is 2^8 times of the one for our memory-reduced version. Based on this assumption, we estimate the attack cost for the original super-sbox analysis.

Our experimental results show that the computational cost of the super-sbox analysis is approximately 4 times higher than the previous estimation and the freedom degrees are 4 times smaller than the previous estimation. Namely, the cost of the previous super-sbox is approximately 2^{50} 8-round AES computations instead of 2^{48}, and the available freedom degrees are approximately 2^{62} rather than 2^{64}, which indicates that applying the super-sbox analysis is harder than expected.

2 Specifications

Advanced Encryption Standard (AES) [4,5] is a 128-bit block cipher supporting three different key sizes; 128, 192, and 256 bits. The AES encryption and decryption perform 10, 12, and 14 rounds for AES-128, -192, and -256, respectively.

By using the key schedule function, round keys are generated from the original secret key. We omit its description because the super-sbox analysis regards round keys as given constant numbers.

When the data is processed, the internal state is represented by a $4 * 4$ byte array. At the first, the original secret key is XORed to the plaintext, and then, a round function consisting of the following four operations is iteratively applied.

- SubBytes(SB): substitute each byte according to an S-box table.
- ShiftRows(SR): apply the j-byte left rotation to each byte at row j.
- MixColumns(MC): multiply each column by an MDS matrix. MDS guarantees that the sum of active bytes in the input and output of the MixColumns operation is at least 5 unless all bytes are non-active.
- AddRoundKey(AK): apply a bit-wise exclusive-or with a round key.

Note that the MixColumns operation is not computed for the last round.

In this paper, we count the round number of AES from 1. Namely, the computation for AES-128 starts from round 1 and ends with round 10. We denote the initial state in round x by $\#x^I$. Then, states immediately after SubBytes, ShiftRows, MixColumns, and AddRoundKey in round x are denoted by $\#x^{SB}$, $\#x^{SR}$, $\#x^{MC}$, and $\#x^{AK}$, respectively. Obviously, $\#x^{AK}$ is identical with $\#(x+1)^I$.

Table 1. An enumeration for byte positions in a state

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

We also denote byte positions in a state S by an enumeration $\{0, 1, 2, \ldots, 15\}$, where the byte $4j + i$ corresponds to the byte in the i-th row and j-th column of S, and is denoted by $S[4j + i]$. We often denote several bytes of state S by $S[A, B, \ldots]$, e.g. 4 bytes in the right most column are denoted by $S[12, 13, 14, 15]$. To make it clear, we show the enumeration in Table 1.

3 Previous Work

3.1 Known-Key Attack on 8-Round AES

In 2010, Gilbert and Peyrin applied the super-sbox analysis upon the rebound attack, and obtained the known-key distinguisher on 8-round AES [13]. The goal of this attack is to find a pair of plaintexts such that the paired plaintexts have differences in byte positions 0, 5, 10, and 15, and the corresponding ciphertexts have differences in byte positions 0, 7, 10, and 13. The sketch of the attack is as follows.

1. The attacker prepares the differential path. Fig. 2 is the one used in [13]. Then, the differential path is divided into two phases; inbound and outbound.
2. Many paired values which satisfy the inbound differential path are generated efficiently by using the super-sbox technique.
3. These pairs are tested whether or not they can also satisfy the outbound differential path.

In the analysis by [13], one paired values satisfying the inbound differential path can be generated with a complexity of 1 on average. Because the probability of the differential propagation for the outbound phase is 2^{-48}, the attacker finds the paired values satisfying the whole path with a computational cost of 2^{48} 8-round AES computations. Note that finding paired values which have the same property for a random permutation requires 2^{64} queries with a limited birthday attack [13], and thus 8-round AES can be distinguished from a random permutation.

The outbound phase is just testing the differential propagation of the given paired values. Therefore, the essence of the attack lies in the inbound phase, namely, how to efficiently find many paired values satisfying the inbound differential path. We explain the details of the inbound and outbound phases in Sect. 3.2 and Sect. 3.3, respectively.

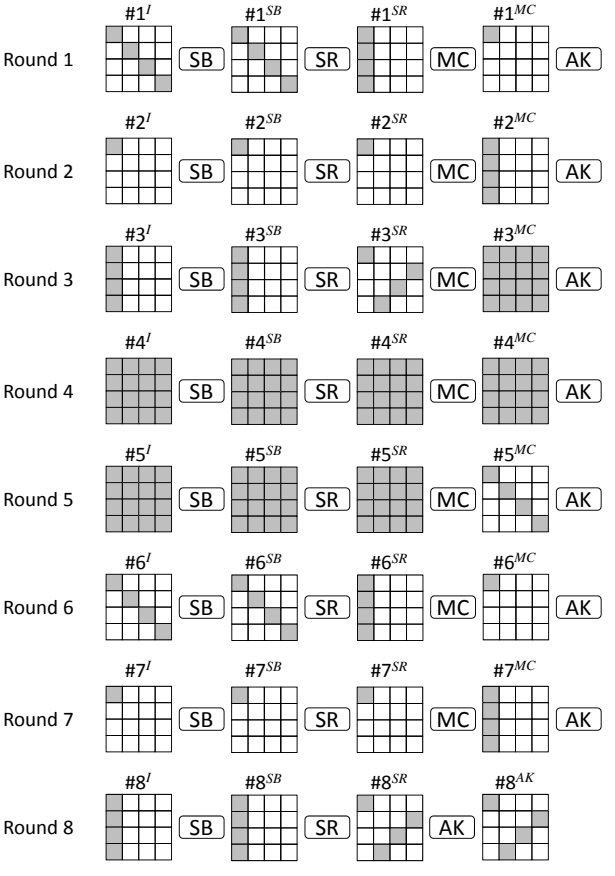


Fig. 2. Differential path of 8-round known-key distinguisher on AES. Active bytes are in grey.

3.2 Inbound Phase by Using the Super-Sbox

The goal of the inbound phase is finding solutions satisfying the differential path through 3rd round to 5th round, which is shown in Fig. 3. The previous super-sbox analysis finds 2^{32} pairs satisfying the differential path with 2^{32} memory and 2^{32} computational cost. The detailed algorithm is described in Alg. 3.1.

Gilbert and Peyrin [13] and several following work assume that by the 2^{32} iterations of step 3.1, each of 2^{32} entries in T will have one solution on average for one super-sbox. Therefore, at step 3.1, for each entry of T, each super-sbox will produce one solution on average. In the end, the algorithm in Alg. 3.1 can produces 2^{32} solutions of the inbound differential path with 2^{32} memory for step 3.1 and 2^{32} computational cost for step 3.1. Note that, this algorithm can be iterated by changing the chosen value at step 3.1. Hence, the attack can produce up to $2^{32} \cdot 2^{32} = 2^{64}$ solutions by iterating the super-sbox part for 2^{32} times.

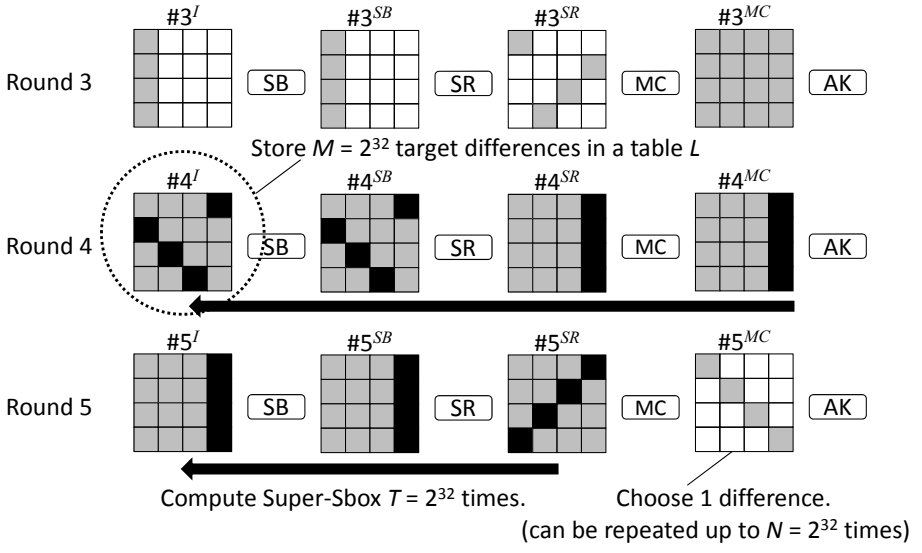


Fig. 3. Differential path for inbound phase. Black cells denote bytes involved in one of the super-sboxes.

Algorithm 3.1. Previous algorithm for the super-sbox analysis by [13].

Input: the differential path and a randomly generated subkey for round 4.
Output: 2^{24} paired values satisfying the differential path of Fig. 3.

1. For all possible 2^{32} differences at state $\#3^{SB}$, compute the corresponding difference at state $\#4^{I}$, and store the results in a table L with the size of 2^{32} AES state.
2. Choose a difference at state $\#5^{MC}$ and compute the corresponding difference at state $\#5^{SR}$.
3. For each of 2^{32} possible 4-byte values of $\#5^{SR}[3, 6, 9, 12]$, compute the super-sbox, namely, compute the corresponding difference at $\#4^{I}[12, 1, 6, 11]$. If this difference matches one of the entries of T generated at step 1, store these values in the same entry of L. (2^{32} computational cost is required for this step.)
4. Repeat step 3.1 for the other three super-sboxes (starting from $\#5^{SR}[0, 7, 10, 13]$, $\#5^{SR}[1, 4, 11, 14]$, and $\#5^{SR}[2, 5, 8, 15]$.)
5. For each entry of L, if the difference can be produced for all super-sboxes, output all possible combinations of solutions for all super-sboxes.

3.3 Outbound Phase

In the outbound phase, we check the differential propagation of the paired values sent from the inbound phase. Namely, we check whether or not the pair satisfies the differential paths from $\#2^{MC}$ to $\#2^{SR}$ in round 2 and from $\#6^{SR}$ to $\#6^{MC}$ in round 6. If both propagations are satisfied, we compute the corresponding values of state $\#1^{I}$ and $\#8^{AK}$. The both differential propagations from $\#2^{MC}$ to $\#2^{SR}$ and $\#6^{SR}$ to $\#6^{MC}$ have a success probability of $(2^{-8})^3 = 2^{-24}$, and

thus, the total success probability is 2^{-48}. Therefore, by generating 2^{48} solutions in the inbound phase, we can find a pair which satisfies the whole differential path.

The total attack cost is a sum of the one for the inbound phase and the one for the outbound phase. For one iteration of the inbound phase in Alg. 3.1, the inbound phase requires 2^{32} in time and a memory for 2^{32} AES state. the outbound phase requires 2^{32} in time and negligible memory. Because the inbound phase is iterated 2^{16} times to produce 2^{48} solutions, the final complexity is $2^{16} \cdot 2^{32} = 2^{48}$ 8-round AES computations in time and 2^{32} AES state in memory.

4 Tradeoff in Super-Sbox Analysis

In this section, we analyze the relation among the computational cost (T), the required memory (M), and the number of obtained solutions (D), and show that the time-memory tradeoff exists in the super-sbox analysis.

4.1 Algorithms for Super-Sbox Analysis with Reduced Memory

First of all, we clarify the procedure for the super-sbox analysis with reduced time and memory. In this attack, compared to Alg. 3.1, we only use the memory of size M for step 3.1, T computational cost for step 3.1, and obtain D solutions of the inbound phase. The algorithm is described in Alg. 4.1. Note that steps 3.1, 3.1, and 3.1 are exactly the same as the ones in Alg. 3.1.

Algorithm 4.1. Super-Sbox analysis with a Time-Memory tradeoff.

Input: the different path and a randomly generated subkey for round 4
Output: D paired values satisfying the differential path of Fig. 3
Parameters: T, M, D

1. Choose M differences out of 2^{32} choices at state $\#3^{SB}$ and compute the corresponding difference at state $\#4^{I}$. Store the results in a table L with the size of M AES state.
2. Choose a difference at state $\#5^{MC}$ and compute the corresponding difference at state $\#5^{SR}$.
3. For T out of 2^{32} possible 4-byte values of $\#5^{SR}[3, 6, 9, 12]$, compute the super-sbox, namely, compute the corresponding difference at $\#4^{I}[12, 1, 6, 11]$. If this difference matches one of the entries of L generated at step 1, store these values in the same entry of L. If the match does not occur, discard the pair. (T computational cost is required for this step.) After T trials, eliminate entries from L which do not have solutions for this super-sbox.
4. Repeat step 4.1 for the other three super-sboxes (starting from $\#5^{SR}[0, 7, 10, 13]$, $\#5^{SR}[1, 4, 11, 14]$, and $\#5^{SR}[2, 5, 8, 15]$.)
5. For each entry of L, if the difference can be produced for all super-sboxes, output all possible combinations of solutions for all super-sboxes.

4.2 Relationship among T, M, and D

We analyze the value of D with T in time and M in memory. For simplicity, let us firstly discuss the cases where we only reduce either T or M.

Reducing the Memory Size M. We estimate the number of solutions D for the parameters $T = 2^{32}$ and $M < 2^{32}$. In step 4.1, M differences are stored at state $\#4^I$. We call these differences "target differences." That is to say, we have M target differences in a list L. Then in step 4.1, 2^{32} differences at state $\#4^I[12, 1, 6, 11]$ are computed. If we assume that the resulting differences are uniformly distributed, all differences at state $\#4^I[12, 1, 6, 11]$ are reached exactly once. Therefore, all target differences in L will have one solution for this super-sbox. In step 4.1, the same procedure is repeated for the other three super-sboxes, and thus all target differences in L will have one solution for all super-sboxes. Finally, in step 4.1, all M target differences in L will produce one combination of solutions of four super-sboxes. Therefore, the value of D is equal to M.

Reducing the Computational Cost T. We estimate the number of solutions D for the parameters $T < 2^{32}$ and $M = 2^{32}$. In step 4.1, we have 2^{32} target differences (all possibilities) in a list L. Then in step 4.1, T differences at state $\#4^I[12, 1, 6, 11]$ are computed. Under the same assumption, only T differences out of 2^{32} possibilities can be reached and has one solution for this super-sbox. Hence, the number of the target differences in L will be $2^{32} \cdot T/2^{32}$ (decrease by a factor of $T/2^{32}$). In step 4.1, after the repetition for the other three super-sboxes, the number of the target differences in L will be $2^{32} \cdot (T/2^{32})^4$, and each of them has only solution for all the super-sboxes. Finally, in step 4.1, $2^{32} \cdot (T/2^{32})^4$ differences in L will produce one combination of solutions of four super-sboxes. Therefore, the value of D is $2^{32} \cdot (T/2^{32})^4$.

Generic Relationship among T, M, and D. By combining the previous two analyses, we can express D by reduced M and T. First, in step 4.1, M target differences are stored in a list L. Next, in steps 4.1 and 4.1, the size of the target differences will decrease by the factor of $(T/2^{32})^4$, and each remaining target difference have one solution for all the super-sboxes. Finally, the relations among T, M, and D is described as

$$D = M \cdot (T/2^{32})^4. \tag{1}$$

Note that the limitation exists for the range of M and T. During one choice of the difference at step 4.1 in Alg. 4.1, no advantage can be obtained if the attacker spends more than 2^{32} computational cost and memory. Hence, the range of M and T are expressed as

$$0 < T \leq 2^{32}, \qquad 0 < M \leq 2^{32}. \tag{2}$$

Equation (1) can be used for optimizing the attack depending on the attack scenario. It indicates that if we reduce T and keep M unchanged, D, which is

the number of solutions we can obtain, decreases by the speed of the power 4. On the other hand, if we reduce M and keep T unchanged, D decreases only linearly. Hence, reducing T makes the super-sbox analysis very inefficient, while reducing M can be a possible variation of the analysis.

4.3 Time-Memory Tradeoff for Entire Inbound Phase

The relation in Eq.(1) is the tradeoff inside the one iteration (one choice of the difference at step 4.1) of the algorithm in Alg. 4.1. we should consider the fact that algorithm in Alg. 4.1 can be iterated up to 2^{32} times with changing the chosen difference at step 4.1. Let us denote the number of repetitions of Alg. 4.1 by N. Then, the relation for M, N, T, and D can be written as

$$D = N \cdot M \cdot (T/2^{32})^4. \tag{3}$$

Note that by repeating the attack N times, the total computational cost becomes NT and the total memory size keeps unchanged from M.

As discussed in Sect. 4.2, reducing T makes the analysis very inefficient. Therefore, keeping $T = 2^{32}$ is a reasonable strategy. Under this condition, the equation becomes

$$NM = D, \tag{4}$$

where, the total attack cost is $N \cdot 2^{32}$. In most of the case including the super-sbox analysis, the number of required solutions of the inbound phase is a fixed constant as soon as the attack target and the attack model are determined. Hence, regarding D as a constant number would be useful. In fact, D is 2^{48} for the super-sbox analysis. Finally, the equation becomes

$$NM = Constant. \tag{5}$$

It is obvious that if M is reduced, N must be increased by the same factor for achieving the attack, and this increases the computational cost by the same factor. Finally, under the condition that T is fixed, we can conclude that the time-memory tradeoff exists for the entire super-sbox analysis, which $Time \times Memory = Constant$. This answers the question we raised in the beginning of Sect. 4, namely, for example, we can perform the super-sbox analysis with 2^{40} in time and 2^{24} in memory.

Note that the limitation also exists for N. At step 4.1 in Alg. 4.1, the number of possible differences we can choose is 2^{32}. Hence, the limitation for all of the parameters are expressed as

$$0 < T \leq 2^{32}, \qquad 0 < M \leq 2^{32}, \qquad 0 < N \leq 2^{32}. \tag{6}$$

5 Experiment

In this section, we verify the correctness and the complexity of the super-sbox analysis on 8-round AES-128 by implementing the attack and carrying out some

experiments. Note that the complexity of previous super-sbox and ours in Sect. 4 were roughly estimated. There may exist some difference between the theoretical complexity estimation and the cost in practice. This is mainly because of the optimistic assumption on the super-sbox behavior in the theoretical estimation; when we try 2^{32} values for step 3.1 in Alg. 3.1, the output of the super-sbox is uniformly distributed. Our experiment actually shows that the required cost is approximately 4 times higher than the theoretical estimation.

5.1 Implementation Technique

We implement the super-sbox analysis on a 32-bit OS personal computer. We introduce two ideas to carry out the verification with a small computational cost and memory.

Separating differential path. We separate the differential path into two parts; from state $\#1^I$ to $\#5^{MC}$ and from $\#6^I$ to state $\#8^{AK}$. We then try to satisfy each differential path independently. Because it includes only 1 probabilistic differential propagation with a success probability of 2^{-24}, each path can be satisfied with only 2^{24} solutions of the inbound phase. Therefore, we can do the verification with the less computational cost than usual.

Time-Memory tradeoff. Previous super-sbox analysis requires 2^{32} computational cost and 2^{32} memory. Note that attacks using 2^{32} memory is hard to implement, and thus we need to reduce the memory. By using the separated differential path, the outbound phase has a success probability of 2^{-24}. Therefore, the number of solutions we need is reduced from 2^{32} of previous work. In this case, we can apply the tradeoff discussed in Sect. 4. According to Eq. (1), if D is reduced, M can be reduced by the same factor. Therefore, we can implement the attack for $D = 2^{24}$ with parameters $T = 2^{32}$ and $M = 2^{24}$ in Alg. 4.1.

In the end, by verifying both half-paths with a computational cost of $2 \cdot 2^{32}$ half-AES $= 2^{32}$ full-AES computations and 2^{24} memory, we can expect that 8-round known-key distinguisher would correctly work with a computational cost of 2^{48} and 2^{32} memory.

5.2 Experimental Results

Parameters for the Experiment. We generate solutions for the inbound phase by the algorithm in Alg. 4.1 with parameters $T = 2^{32}$ and $M = 2^{24}$. We expect that 2^{24} solutions for the inbound phase can be obtained, and for each solution, we check the half-differential path for the outbound phase. If any solution for the inbound phase cannot satisfy the outbound phase, we repeat the algorithm by changing the difference at step 4.1, in other words, we increase N.

In our experiment, for step 4.1 of the algorithm in Alg. 4.1, we fixed the difference of $\#3^{SB}[0]$ to a unique value and considered all possible 2^{24} differences for $\#3^{SB}[1, 2, 3]$. One may suspect that fixing the difference of $\#3^{SB}[0]$ causes

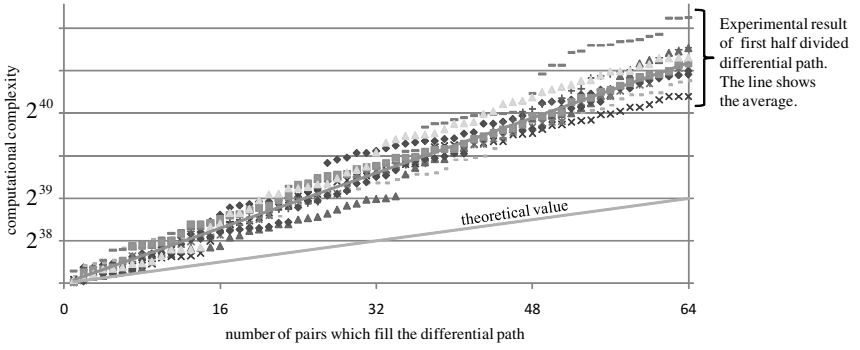


Fig. 4. Experimental results of the first half of the separated differential path. The unit for the vertical axis is 2^{38}. "theoretical value" describes the double of the complexity estimation of [13].

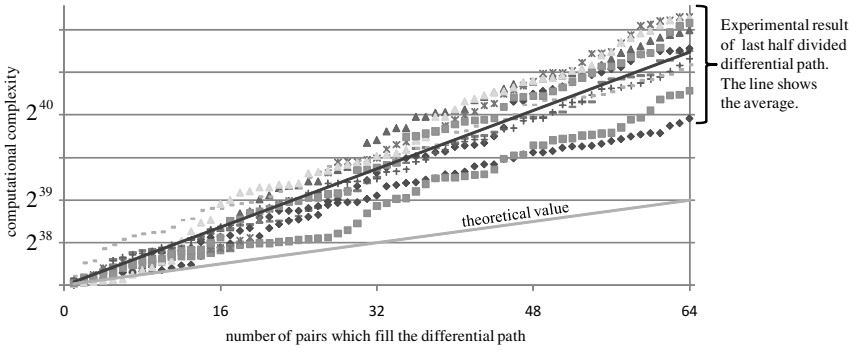


Fig. 5. Experimental results of the last half of the separated differential path. The unit for the vertical axis is 2^{38}. "theoretical value" describes the double of the complexity estimation of [13].

the biased distribution, and thus generates biased experimental results. In the remarks at the end of this section, we show that fixing the difference of $\#3^{SB}[0]$ does not generate biased results. In order to collect the data for Figs. 4 and 5, we repeated the inbound phase producing 2^{24} solutions many times. For all trials, we randomly determine the difference at state $\#5^{MC}$ in step 2 of the algorithm.

Results. The results of the experiment which obtains the pairs satisfying the differential path from state $\#1^{I}$ to state $\#5^{MC}$ is shown in Fig. 4.

The figure shows the computational cost for satisfying the outbound path several times. First, we point out that we simply need to spend the doubled cost compared to [13]. This is because that among 2^{24} pairs generated in the inbound phase by combining all possibilities of 4 super-sboxes at step 3.1 in Alg. 3.1,

duplicated pairs such as (a, b) and (b, a) are included. These pairs always result in the same difference through any linear operation including MixColumns. Hence, we can use only 2^{23} unique pairs in this experiment, and thus to obtain 2^{24} pairs we need double of the computational cost of [13]. Due to this property, in Figs. 4 and 5, we denote the double complexity of [13] by "theoretical value".

Our experimental results in Figs. 4 and 5 indicate that the cost for the outbound phase is even higher than the doubled theoretical estimation. We confirmed that the probability for the outbound path is 2^{-24}, and thus it can be satisfied with a high probability if 2^{24}-2^{25} solutions of the inbound phase are generated[1]. Therefore, this fact indicates that we cannot obtain $D = 2^{24}$ solutions for the inbound phase with $T = 2^{32}$ and $M = 2^{24}$, which was estimated based on the assumption of the uniform output of the super-sbox. In practice, it outputs only $2^{21.7}$ solutions, and to cover the loss of the factor of $2^{2.3}$, the attack is repeated $N = 2^{2.3}$ times.

From the same reason, we get to know that the original super-sbox analysis which uses the parameters $T = 2^{32}$ and $M = 2^{32}$ can produce only $D = 2^{32-2.3} = 2^{29.7}$ solutions for the inbound phase, and to obtain 2^{32} solutions, the attack must be repeated $N = 2^{2.3}$ times. This raises the computational cost of previous super-sbox [13] from 2^{48} to $2^{50.3}$ and reduces the maximum freedom degrees from 2^{64} to $2^{61.7}$.

Remarks. Considering that $2^8 - 1$ differential patterns exist in $\#2^{SR}[0]$, $2^8 - 1$ differential patterns of $\#2^{MC}[0, 1, 2, 3]$ can satisfy the differential path, where the differential patterns of $\#2^{MC}[0]$ varies from 1 to $2^8 - 1$. However, the fixed difference of $\#3^{SB}[0]$ only produces 127 differential patterns in $\#2^{MC}[0]$, and thus the probability of satisfying the differential path seems $2^{-25}(= 2^{-32} \cdot 2^7)$ rather than $2^{-24}(= 2^{-32} \cdot 2^8)$. We show that the above argument is not correct.

Let us explain why the probability of the differential propagation from $\#2^{MC}$ to $\#2^{SR}$ is roughly 2^{-24} with the original super-sbox analysis (with 2^{32} memory). We count the number of valid paired values in each state. For $\#2^{SR}[0, 1, 2, 3]$, the number of valid paired values is $2^{40}(= 2^{32} \cdot 2^8)$. Therefore, the number of valid paired values at $\#3^{SB}[0, 1, 2, 3]$ is also 2^{40}. On the other hand, solutions of the inbound phase will have a random value and difference at $\#3^{SB}[0, 1, 2, 3]$, and thus, it takes $2^{64}(= 2^{32} \cdot 2^{32})$ possibilities. Finally, the probability that a solution of the inbound phase reaches $\#2^{SR}$ is $2^{40}/2^{64} = 2^{-24}$.

Similarly, we evaluate the probability in our memory-reduced algorithm (the difference of $\#3^{SB}[0]$ is fixed). For $\#2^{SR}[0, 1, 2, 3]$, the number of valid paired values is $2^{40}(= 2^{32} \cdot 2^8)$. Among 2^{40} paired values, they are valid only if the corresponding difference at $\#3^{SB}[0]$ matches the fixed value. The probability of the match is 2^{-8}. Hence, the number of valid paired values at $\#3^{SB}$ is $2^{32}(= 2^{40} \cdot 2^{-8})$. On the other hand, solutions of the inbound phase will have 2^{32} possibilities on the value and 2^{24} possibilities on the difference, and thus it takes

[1] The probability for the outbound path comes from a simple differential propagation through MixColumns or InverseMixColumns. The correctness of this probability has been confirmed by much previous researches on AES.

2^{56} possibilities. Finally, the probability that a solution of the inbound phase reaches $\#2^{SR}$ is $2^{32}/2^{56} = 2^{-24}$, which is the same as the original super-sbox analysis.

6 Concluding Remarks and Future Work

In this paper, we analyzed the relations among memory, computational cost, and the number of solutions for the super-sbox analysis on 8-round AES, and discovered the tradeoff among these parameters. We then implemented the super-sbox analysis in order to verify the attack complexity. For this purpose, we introduced two ideas to make the implementation feasible. As a result of the experiments, we found that the cost for the super-sbox analysis was more than the previous estimation, and the available freedom degrees was smaller than the previous estimation.

Future Work

- We experimentally confirmed the details of the approximated attack complexity for the super-sbox analysis. Giving a theoretical reasoning is a remaining work.
- To investigate the super-sbox behavior deeply, making the differential distribution table (DDT) seems helpful. However, making a DDT for 32-bit S-box requires 2^{64} memory and thus infeasible. The remaining work is doing the simulation by reducing the byte size e.g. 4 bits in one byte. Comparison of the super-sbox behavior with changing the byte size seems interesting.
- One of the most important work is investigating the impact of the reconsidered complexity on the previously published super-sbox analyses. For example, in the distinguisher for the old version of the full Grøstl compression function [19,20], the attacker only has the freedom degrees of 2^z to satisfy the differential path with the probability of 2^{-z}. In such a case, due to the reduced freedom degrees, the success probability of the attack will decrease by a non-negligible factor. Another interesting target is the analysis on ECHO [18]. Because ECHO uses the AES round structure recursively, the impact of the change of the super-sbox complexity will be increase, and thus the strict complexity estimation is important.

References

1. 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)
2. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
3. U.S. Department of Commerce, National Institute of Standards and Technology: Federal Register Vol. 72(212) (November 2, 2007) Notices (2007) http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf

4. U.S. Department of Commerce, National Institute of Standards and Technology: Specification for the ADVANCED ENCRYPTION STANDARD (AES) (Federal Information Processing Standards Publication 197) (2001)
5. Daemen, J., Rijmen, V.: The design of Rijndeal: AES – the Advanced Encryption Standard (AES). Springer, Heidelberg (2002)
6. Rijmen, V., Barreto, P.S.L.M.: The WHIRLPOOL hashing function. Submitted to NISSIE (September 2000)
7. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl addendum. Submission to NIST (updated) (2009)
8. Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., Seurin, Y.: SHA-3 proposal: ECHO. Submission to NIST (updated) (2009)
9. Biham, E., Dunkelman, O.: The SHAvite-3 hash function. Submission to NIST (Round 2) (2009)
10. Knudsen, L.R., Rijmen, V.: Known-key distinguishers for some block ciphers. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 315–324. Springer, Heidelberg (2007)
11. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The rebound attack: Cryptanalysis of reduced whirlpool and Grøstl. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 260–276. Springer, Heidelberg (2009)
12. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved cryptanalysis of the reduced Grøstl compression function, ECHO permutation and AES block cipher. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 16–35. Springer, Heidelberg (2009)
13. Gilbert, H., Peyrin, T.: Super-sbox cryptanalysis: Improved attacks for AES-like permutations. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 365–383. Springer, Heidelberg (2010)
14. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound distinguishers: Results on the full whirlpool compression function. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 126–143. Springer, Heidelberg (2009)
15. Matusiewicz, K., Naya-Plasencia, M., Nikolić, I., Sasaki, Y., Schläffer, M.: Rebound attack on the full LANE compression function. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 106–125. Springer, Heidelberg (2009)
16. Naya-Plasencia, M.: Scrutinizing rebound attacks: new algorithms for improving the complexities. Cryptology ePrint Archive, Report 2010/607 (2010), http://eprint.iacr.org/2010/607
17. Wu, S., Feng, D., Wu, W., Su, B.: Hyper-sbox view of AES-like permutations: A generalized distinguisher. In: Lai, X., Yung, M., Lin, D. (eds.) Inscrypt 2010. LNCS, vol. 6584, pp. 155–168. Springer, Heidelberg (2011)
18. Sasaki, Y., Li, Y., Wang, L., Sakiyama, K., Ohta, K.: Non-full-active super-sbox analysis: Applications to ECHO and grøstl. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 38–55. Springer, Heidelberg (2010)
19. Peyrin, T.: Improved cryptanalysis of ECHO and Grøstl. Cryptology ePrint Archive, Report 2010/223 (2010), http://eprint.iacr.org/2010/223 Full version of CRYPTO 2010
20. Peyrin, T.: Improved differential attacks for ECHO and Grøstl. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 370–392. Springer, Heidelberg (2010)

Towards Restricting Plaintext Space in Public Key Encryption

Yusuke Sakai[1], Keita Emura[2], Goichiro Hanaoka[3],
Yutaka Kawai[4], and Kazumasa Omote[2]

[1] The University of Electro-Communications, Japan
y-sakai@ice.uec.ac.jp
[2] Japan Advanced Institute of Science and Technology, Japan
k-emura@jaist.ac.jp, omote@jaist.ac.jp
[3] National Institute of Advanced Industrial Science and Technology, Japan
hanaoka-goichiro@aist.go.jp
[4] The University of Tokyo, Japan
kawai@it.k.u-tokyo.ac.jp

Abstract. This paper investigates methods that allow a third-party authority to control contents transmitted using a public key infrastructure. Since public key encryption schemes are normally designed not to leak even partial information of plaintext, traditional public key encryption schemes do not allow such controlling by an authority. In the proposed schemes, an authority specifies some set of forbidden messages, and anyone can detect a ciphertext that encrypts one of the forbidden messages. The syntax of public key encryption with such a functionality (restrictive public key encryption), formal definitions of security requirement for restrictive public key encryption schemes, and an efficient construction of restrictive public key encryption are given.

In principle, restrictive public key encryption schemes can be constructed by adding an NIZK proof that proves whether the encrypted messages are not prohibited. However if one uses the general NIZK technique to construct such a non-interactive proof, the scheme becomes extremely inefficient. In order to avoid such an inefficient construction, the construction given in this paper uses techniques of Teranishi et al., Boudot, and Nakanishi et al.

One of the possible applications of restrictive public key encryption is protecting a public key infrastructure from abuse by terrorists by disallowing encryption of crime-related keywords. Another example is to perform format-check of a ballot in an electronic voting, by disallowing encryption of irregular format voting.

1 Introduction

Background and Motivation. Public key encryption schemes are required to hide even partial information of plaintexts. This strong requirement is formalized as the notion of semantic security [14], and it is currently considered as even one of the lowest requirement for encryption scheme.

As a consequence of the strong secrecy requirement of semantic security, no one can detect the ciphertext which encrypts some particular plaintexts. This paper considers how to add such a functionality to public key encryption without losing reasonable

T. Iwata and M. Nishigaki (Eds.): IWSEC 2011, LNCS 7038, pp. 193–209, 2011.

secrecy of encrypted plaintexts. To formally treat this functionality, we define the notion of *restrictive public key encryption (RPKE)*. Restrictive public key encryption allows a trusted third party to specify a set of *prohibited messages*, and anyone can detect a ciphertext which encrypts one of the prohibited messages. Moreover, this verification process is required not to leak information about the encrypted plaintext except whether it is a prohibited messages or not.

Such a functionality can be realized using well-known NIZK technique, like a general NIZK through the Hamilton path problem. However, this realization is extremely inefficient. Even an OR-proof based NIZK does not suffice for an efficient construction, this construction requires the very large ciphertext, whose length linearly depends on the number of allowed messages. In this paper, we explore further efficient construction, and proposes a scheme that achieves shorter ciphertext whose length does not increase as the number of allowed messages increase.

One of the application of restrictive public key encryption is a countermeasure against abuse of a public key infrastructure by terrorists. This is achieved by disallowing encryption of crime-related messages, and forbid terrorists from using a public key infrastructure to planning terrorism or sending instruction for terrorist activities. Another application may be a format-checking in electronic voting, by disallowing encryption of irregular format ballots. In this application, only encryptions of correctly-formatted voting is allowed, and gateways can dispose any encrypted ballot of irregular format without violating privacy of voters.

Contribution. In this paper, we give a formal definition of restrictive public key encryption. We also give an efficient construction of restrictive public key encryption. The definition even captures very strong security of chosen-ciphertext security. The construction utilizes the techniques of Teranishi et al. [18], Boudot [6], and Nakanishi et al. [15], in order to obtain an efficient construction. The construction also has a capability of updating the message space specified by the authority. We again emphasize that the construction given in this paper is quite more efficient than the trivial construction employing the general NIZK technique through the Hamilton path problem and even more efficient than a simple OR-proof based construction. More concretely, the encryption cost, the verification cost, and the ciphertext length is constant (independent from the number of allowed messages and the number of prohibited messages), whereas in the OR-proof construction they all linearly increase as allowed messages increase. This efficiency is achieved by the use of techniques of Teranishi et al. [18], Boudot [6], and Nakanishi et al. [15]. The proposed construction uses the BB signature [2] and the BBS+ signature [1,3,13] and further uses non-interactive proofs proving possession of the signature. This non-interactive proofs are constructed from novel algebraic properties of the BB signature and the BBS+ signature.

Related Works. Verifiable encryption [7,8,17] is one of the most widely known ways to restrict contents under the secret channel and enables anyone to verify whether encrypted messages satisfy certain restrictions or not without leaking other information about plaintexts. However, verifiable encryption does not have the capability of disallowing to encrypt some specified messages. Fuchsbauer and Pointcheval [12] proposed a techniques to verify whether an encrypted plaintext satisfies some pairing-product

equation, but it also lacks a capability of restricting the message space. Searchable encryption [4] seems to be a promising technique to construct restrictive PKE, that is, once the authority publicizes trapdoor information corresponding to some prohibited keyword, anyone can detect ciphertexts that encrypt one of the prohibited keyword. However, in order to update the message space publicized by the authority, this approach requires to revoke the trapdoor previously publicized. Since all known searchable encryption schemes does not have such a capability, searchable encryption cannot be directly adopted to the restrictive PKE context. Another approach is publicizing all trapdoors for allowed messages, in order to recover encrypted message by using this trapdoor information and detecting prohibited messages. However, this approach is also inappropriate, because the information of encrypted message is completely leaked due to the trapdoor information publicized by the authority.

2 Preliminary

In this section, we define building tools used in our generic RPKE construction.

Public Key Encryption. A public key encryption (PKE) scheme consists of three algorithms, $\mathsf{PKE.KeyGen}$, $\mathsf{PKE.Enc}$ and $\mathsf{PKE.Dec}$. The public key pk and the secret key sk are given by executing $\mathsf{PKE.KeyGen}(1^\kappa)$, where κ is the security parameter. For a message $M \in \mathcal{M}_{\mathrm{PKE}}$, where $\mathcal{M}_{\mathrm{PKE}}$ denotes the message space of PKE, a user runs $\mathsf{PKE.Enc}(pk, M)$ and obtains a ciphertext c. The message M is recovered by executing $\mathsf{PKE.Dec}(sk, c)$. In our construction, we need to explicitly describe a random coin u which is used for encryption. We denote it $c = \mathsf{PKE.Enc}(pk, M; u)$. We sometimes omit pk and describe this $\mathsf{PKE.Enc}(M; u)$.

Definition 1. *A PKE scheme* $(\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *is said to be IND-CCA secure when for any probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *which does not query the decryption oracle* $\mathsf{PKE.Enc}(sk, \cdot)$ *with the challenge ciphertext* c^* *after receiving it, the advantage* $Adv^{IND\text{-}CCA}_{\mathcal{A}}(1^\kappa) = \Pr[(pk, sk) \leftarrow \mathsf{PKE.KeyGen}(1^\kappa); (M_0^*, M_1^*, State) \leftarrow \mathcal{A}^{\mathsf{PKE.Dec}(sk,\cdot)}(pk); b \xleftarrow{\$} \{0,1\}; c^* \leftarrow \mathsf{PKE.Enc}(pk, M_b^*); b' \leftarrow \mathcal{A}^{\mathsf{PKE.Dec}(sk,\cdot)}(c^*, State) : b = b'] - \frac{1}{2}$ *is negligible.*

Signature. A signature scheme consists of three algorithms, $\mathsf{Sig.KeyGen}$, Sign, and Verify. $\mathsf{Sig.KeyGen}$ is a probabilistic algorithm which outputs a signing/verification key pair (K_s, K_v). $\mathsf{Sign}(K_s, M)$ is a probabilistic algorithm which outputs a signature σ from K_s and a message $M \in \mathcal{M}_{Sig}$, where $\mathcal{M}_{Sig}$ is the message space. $\mathsf{Verify}(K_v, M, \sigma)$ is a deterministic algorithm which outputs 1 if σ is a valid signature, and 0 otherwise.

Definition 2. *A signature scheme* $(\mathsf{Sig.KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is said to be EUF-CMA secure when for any PPT adversary* $\mathcal{A}$ *the advantage* $Adv^{EUF\text{-}CMA}_{\mathcal{A}}(1^\kappa) = \Pr[(K_s, K_v) \leftarrow \mathsf{Sig.KeyGen}(1^\kappa); (M^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(K_s,\cdot)}(K_v) : M^*$ *was not queried to* $\mathsf{Sign}(K_s, \cdot) \wedge \mathsf{Verify}(K_v, M^*, \sigma^*) = 1]$ *is negligible.*

In addition, a signature scheme is said to be EUF-wCMA (weak CMA) [2] if $\mathcal{A}$ gives the set of signing query before the challenger sends K_v to $\mathcal{A}$.

Σ-protocol [10,9]. Let $R \subset \{0,1\}^* \times \{0,1\}^*$ be a binary relation. For $(x, \omega) \in R$, we call ω is a witness of x. We assume that the following 3-round form, where x is common input of a prover P and a verifier V, and ω (such that $(x, \omega) \in R$)) is private input to P. First, P sends a message a to V. V sends a random bit string e'. Finally, P sends a reply z, and V decides whether the proof is accepted or not. We say that a 3-round protocol $\langle P, V \rangle$ is a Σ-protocol for relation R if the following hold:

Completeness: If P and V follow the protocol, then V always accepts.
Special soundness: From any common input x and any pair of accepting conversations on input x, (a, e', z) and (a, e'', z'') where $e' \neq e''$, one can efficiently compute ω such that $(x, \omega) \in R$.
Special honest verifier zero-knowledge: There exists a polynomial-time simulator, which on input x and a random challenge string e', outputs an accepting conversation of the form (a, e', z), with the same probability distribution as conversations between the honest P, V on input x.

In our RPKE construction, we convert the underlying Σ-protocol into NIZK proof of knowledge by applying Fiat-Shamir heuristic [11]. Therefore, we require random oracles in our construction. We denote such a converted proof as $NIZK\{\omega : (x, \omega) \in R\}$ where x is an instance of the language and ω is its witness.

3 Restrictive Public Key Encryption

3.1 Motivating Discussion

First, to control the contents under secure communications, we consider a scenario as follows. Let us consider four entities: a message restriction authority (MRA), a verifier, a sender, and a receiver. The MRA indicates a restricted message space MS (a set of allowed messages), and publicizes the corresponding public verification key to verifiers and senders. A verifier (which is assumed to be a gateway) inspects whether a ciphertext sent by the sender is an encryption of a value belonging to the message space MS. We require that any information about a plaintext is not revealed from a ciphertext, except the above information. In addition, a verifier inspects all ciphertext, and disposes it if it does not pass the verification process. In this scenario, the MRA can control the encrypted contents without compromising user privacy.

3.2 Formal Definitions

Definition 3. *A restrictive public key encryption (RPKE) consists of six algorithms* (MRASetup, RKeyGen, MSSetup, REnc, VerifyMS, RDec) *such that:*

MRASetup**:** *The key generation algorithm for the MRA takes as input a security parameter $\kappa \in \mathbb{N}$, and returns a public key pk_{MRA} and a private key sk_{MRA}.*
RKeyGen**:** *The receiver key generation algorithm takes as input pk_{MRA}, and returns a public key pk_d and a private key sk_d.*
MSSetup**:** *The public verification key generation algorithm takes as inputs pk_{MRA}, sk_{MRA}, and MS, and returns the public verification key pk_{MS}.*

RKeyGen: *The key generation algorithm takes as input* pk_{MRA}*, and returns a key pair* (pk_d, sk_d)*.*

REnc: *The encryption algorithm takes as inputs* pk_{MRA}*,* pk_d*,* MS*,* pk_{MS}*, and a message* M*, and returns a ciphertext* C*. If* $M \notin MS$*, then the algorithm returns* $\perp$*.*

VerifyMS: *The public verification algorithm takes as inputs* pk_{MRA}*,* pk_d*,* pk_{MS}*, and* C*, and returns a bit* 1 *or* 0*.*

RDec: *The decryption algorithm takes as inputs* pk_{MRA}*,* pk_d*,* sk_d*, and* C*, and returns* M *or* $\perp$*.*

As a correctness, a restrictive public key encryption scheme has to satisfy that for any $\kappa \in \mathbb{N}$*, any restrictive message space* MS*, any message* $M \in MS$*,* $(pk_{MRA}, sk_{MRA}) \leftarrow \mathsf{MRASetup}(1^\kappa)$*,* $(pk_d, sk_d) \leftarrow \mathsf{RKeyGen}(pk_{MRA})$*,* $pk_{MS} \leftarrow \mathsf{MSSetup}(pk_{MRA}, sk_{MRA}, MS)$*, and* $C \leftarrow \mathsf{REnc}(pk_{MRA}, pk_d, MS, pk_{MS}, M)$*, it holds that* $\mathsf{RDec}(pk_{MRA}, pk_d, sk_d, C) = M$ *and* $\mathsf{VerifyMS}(pk_{MRA}, pk_d, pk_{MS}, C) = 1$*.*

Here, we describe an implementation of a scenario to control the contents under secure communications (presented in Sect. 3.1) using restrictive PKE notations. The MRA runs $\mathsf{MRASetup}(1^\kappa)$, publicizes its public key pk_{MRA}, and keeps its secret key sk_{MRA}. The MRA indicates an allowed message space MS, runs $\mathsf{MSSetup}(pk_{MRA}, sk_{MRA}, MS)$, and publicizes pk_{MS}. A receiver runs $\mathsf{RKeyGen}(pk_{MRA})$ and publicizes its public key pk_d, and keeps its corresponding secret key sk_d. For a plaintext M, a sender computes a ciphertext C by running $\mathsf{REnc}(pk_{MRA}, pk_d, MS, pk_{MS}, M)$, and sends C to a verifier (which is assumed to be gateway). By using only public values pk_{MRA}, pk_d, and pk_{MS}, a verifier checks whether $M \in MS$ or not *without decrypting* C. In addition, this procedure should be done without any interaction with other entities. If $\mathsf{VerifyMS}(pk_{MRA}, pk_d, pk_{MS^*}, C) = 1$ (i.e., $M \in MS$), then the verifier forwards C to the corresponding receiver. Otherwise, if $\mathsf{VerifyMS}(pk_{MRA}, pk_d, pk_{MS^*}, C) = 0$ (i.e., $M \notin MS$ or C is an ill-formed value), the verifier disposes C. The receiver runs $\mathsf{RDec}(pk_{MRA}, pk_d, sk_d, C)$, and obtains M. The receiver does not have to consider MS for decrypting C. In the above scenario, The MRA and the verifier cannot obtain any information about a plaintext M from a ciphertext, except whether $M \in MS$ or not. If MS is changed updated to MS', then the MRA runs $\mathsf{MSSetup}(pk_{MRA}, sk_{MRA}, MS')$, and publicizes $pk_{MS'}$ again. And then $pk_{MS'}$ is broadcasted to all users.

Here we define verification soundness, which requires that all dishonestly-generated ciphertext never passes the verification process of VerifyMS. Furthermore, this notion requires even a ciphertext which is honestly-generated with MS not to pass the verification process with a different message space MS'. The latter prevents a sender from reusing a previous public verification key pk_{MS}. To guarantee that even a receiver cannot produce such a invalid (dishonestly-generated but passing the verification) ciphertext, we allow $\mathcal{A}$ to generate (pk_d, sk_d).

Definition 4. *A restrictive PKE is said to satisfy verification soundness if the advantage* $\Pr[(pk_{MRA}, sk_{MRA}) \leftarrow \mathsf{MRASetup}(1^\kappa); (pk_d, sk_d, C^*, MS^*) \leftarrow \mathcal{A}^{\mathsf{MSSetup}(pk_{MRA}, sk_{MRA}, \cdot)}(pk_{MRA}); pk_{MS^*} \leftarrow \mathsf{MSSetup}(pk_{MRA}, sk_{MRA}, MS^*) : \mathsf{VerifyMS}(pk_{MRA}, pk_d, pk_{MS^*}, C^*) = 1 \wedge \mathsf{RDec}(pk_{MRA}, pk_d, sk_d, C^*) \notin MS^*]$ *is negligible for any PPT adversary* $\mathcal{A}$*.*

Next, we define indistinguishability with restrictive message space under chosen ciphertext attack (IND-MSR-CCA). To guarantee that even the MRA cannot decrypt a ciphertext, we assume that $\mathcal{A}$ can generate (pk_{MRA}, sk_{MRA}).

Definition 5. *A restrictive PKE is said to satisfy IND-MSR-CCA if the advantage* $\Pr[(pk_{MRA}, sk_{MRA}, s) \leftarrow \mathcal{A}(1^\kappa); (pk_d, sk_d) \leftarrow \mathsf{RKeyGen}(pk_{MRA}); (M_0^*, M_1^*, MS^*, pk_{MS^*}, s') \leftarrow \mathcal{A}^{\mathsf{RDec}(pk_{MRA}, pk_d, sk_d, \cdot)}(pk_d, s); b \xleftarrow{\$} \{0,1\}; C^* \leftarrow \mathsf{REnc}(pk_{MRA}, pk_d, MS, pk_{MS}, M_b); b' \leftarrow \mathcal{A}^{\mathsf{RDec}(pk_{MRA}, pk_d, sk_d, \cdot)}(C^*, s') : b = b'] - 1/2$ *is negligible for all PPT adversary* $\mathcal{A}$ *which satisfies the following conditions: (1) The adversary* $\mathcal{A}$ *doesn't query the decryption oracle* $\mathsf{RDec}(pk_{MRA}, pk_d, sk_d, \cdot)$ *with the challenge ciphertext* C^* *after receiving it and (2)* M_0^*, M_1^*, *and* MS^* *output by the adversary* $\mathcal{A}$ *always satisfy that* $M_0^*, M_1^* \in MS^*$.

4 Constructions

In this section we show a generic construction of a restrictive PKE scheme from an IND-CCA secure PKE, an EUF-CMA signature, and a non-interactive proof obtained from techniques of Boudot [6], Nakanishi et al. [15], and Teranishi et al. [6,15,18]. We further show that a concrete instantiation of the generic construction.

Let $[1, N] = \{1, \ldots, N\}$ be a set of all possible messages (may or may not be prohibited) and r be the number of all prohibited messages. We say that the sequence $(m_1, \ldots, m_r)$ is the consecutive prohibited messages of MS when $\{m_1, \ldots, m_r\}$ is the all prohibited messages of MS and it holds that $m_1 < \cdots < m_r$. Later $(m_1, \ldots, m_r)$ denotes the consecutive prohibited messages of MS, where MS is the allowed message space implicit in the context.

4.1 High Level Overview

From the highest perspective, the proposed construction is to encrypt a plaintext M by computing $c = \mathsf{PKE.Enc}(pk_d, M; u)$ and adding a non-interactive proof π that proves that $M \in MS$, and constitute a whole ciphertext (c, π) as $c = \mathsf{PKE.Enc}(pk_d, M; u)$ and

$$\pi = NIZK\{(M, u) : c = \mathsf{PKE.Enc}(pk_d, M; u) \wedge M \in MS\}.$$

For example, π is a OR-proof (through Fiat-Shamir heuristics) as $NIZK\{(M, u) : (M = M_1) \vee \cdots \vee (M = M_{N-r})\}$, however, the efficiency might linearly depend on the number of allowed messages $N - r$. Or, when using inequality proof with OR-proof to construct a proof π, the efficiency increases linearly depends on the number of prohibited messages r. These construction does not provide satisfiable efficiency. To improve the efficiency of the construction we first apply Teranishi et al. technique.

Teranishi et al. Technique [18]. Using the technique of Teranishi, Furukawa, and Sako [18], we can reduce the computational complexity just mentioned above. Briefly speaking, Teranishi et al. technique is an NIZK proof of knowledge that proves a secret knowledge ω is in the interval $[1, N]$. This technique involves a signature scheme $(\mathsf{Sig.KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, and its NIZK proof has the form of $NIZK\{(S, \omega) : S = \mathsf{Sign}(\omega)\}$ (where the witness is (S, ω)). This proof system can be efficiently constructed by using an appropriate signature scheme (the BB signature [2] is used indeed) and its algebraic property.

When applying this technique to the restrictive PKE construction, we get the following improvement: In the setup, the MRA generates a verification/signing key pair and secretly possesses the signing key. The MRA then publicizes signatures for all allowed messages. To encrypt a message M, a sender uses the signature S publicized by the MRA and compute a ciphertext $c = \mathsf{PKE.Enc}(pk_d, M; u)$ and a proof of knowledge

$$\pi = NIZK\{\,(M, u, S) : c = \mathsf{PKE.Enc}(pk_d, M; u) \wedge S = \mathsf{Sign}(M)\,\}, \tag{1}$$

which is attached to the ciphertext c. In this way, if a sender wants to encrypt an allowed message $M \in MS$, he can generate an acceptable π using $\mathsf{Sign}(M)$ publicized by the MRA. In contrast, if a sender wants to encrypt a prohibited message $M' \notin MS$, he cannot generate an acceptable proof π of the form above. This is because the MRA does not publicize $\mathsf{Sign}(M')$, nor the sender cannot generate $\mathsf{Sign}(M')$ by himself (due to the unforgeability of the signature), and thus the sender cannot generate the proof of knowledge $NIZK\{(M, S) : S = \mathsf{Sign}(M')\}$ due to the lack of the knowledge needed.

Furthermore, the computational cost of verification of the proof does not depend on the number of allowed messages, because the proof of knowledge $NIZK\{(M, u, S) : c = \mathsf{PKE} : \mathsf{Enc}(pk_d, M; u) \wedge S = \mathsf{Sign}(M)\}$ used here does not depend on the number of allowed messages.

Boudot Technique [6]. The construction discussed above requires the MRA to publicize $|MS|$ signatures. Here we show how the Boudot technique [6] reduces the size of this large public parameter. This technique exploits the fact that any natural number $\omega \in [1, N]$ can be written as $\omega = \omega_1^2 + \omega_2$ using $\omega_1 \in [1, N_1]$, the greatest square less than ω, and $\omega_2 \in [0, N_2]$, where $N_1 = \lfloor \sqrt{N} \rfloor$ and $N_2 = \lfloor 2\sqrt{N} \rfloor$. More concretely, this technique proves knowledge of the signature using zero-knowledge proof of knowledge as $NIZK\{(S_1, S_2, \omega, \omega_1^2, \omega_2) : S_1 = \mathsf{Sign}(\omega_1) \wedge S_2 = \mathsf{Sign}(\omega_2) \wedge \omega = \omega_1^2 + \omega_2\}$, instead of $NIZK\{(S, \omega) : S = \mathsf{Sign}(\omega)\}$ as in the Teranishi et al. technique. The technique also relies on the algebraic property of the BB signature [2] to construct an efficient NIZK proof, instead of relying on an inefficient general NIZK proof.

This technique seems to reduce the size of the public parameter in the proposed construction, however, straightforward application of this technique does not correctly work. To show this, let us consider the situation that $M_1 = \omega_{1,1}^2 + \omega_{1,2}$ and $M_2 = \omega_{2,1}^2 + \omega_{2,2}$ is allowed, $M_3 = \omega_{3,1}^2 + \omega_{3,2}$ is prohibited, and $\omega_{1,1} = \omega_{3,1}$ and $\omega_{2,2} = \omega_{3,2}$ hold. In this situation, the MRA publicizes at least four signatures $\mathsf{Sign}(\omega_{1,1})$, $\mathsf{Sign}(\omega_{2,1})$, $\mathsf{Sign}'(\omega_{1,2})$, $\mathsf{Sign}'(\omega_{2,2})$, for senders to produce a proof that the encrypted message is allowed. This in turn causes malicious senders to be able to produce a proof for a prohibited message $M_3 = \omega_{3,1}^2 + \omega_{3,2}$ by picking up $\mathsf{Sign}(\omega_{1,1})$ and $\mathsf{Sign}'(\omega_{2,2})$ from the public verification key. This drawback are solved by applying Nakanishi et al. technique as follows.

Nakanishi et al. Technique [15]. Adopting Nakanishi et al. technique solves the above problem and moreover it enables further efficiency improvement. This improvement relies on the fact that if all the prohibited messages are denoted as $m_1, \ldots, m_r$, and $m_1 < \cdots < m_r$ holds, then any allowed message $M \in MS$ has a unique "position" j such that $m_j < M < m_{j+1}$ holds, and any prohibited message $M \notin MS$ has no such

position. Another fact that the improvement relies on is that when $N < p/2$ holds, $y > x$ is logically equivalent to $y - x \mod p \in [1, N]$ for any $x, y \in [1, N]$. Using this technique, one can prove the fact $M \in MS$ by proving the existence of j such that $m_j < M < m_{j+1}$ instead, and prove $m_j < M < m_{j+1}$ itself by proving $M - m_j \in [1, N] \wedge m_{j+1} - M \in [1, N]$. To prove $M - m_j \in [1, N]$, one can further apply the Boudot technique as proving knowledge of signatures $\mathsf{Sign}(\delta_{1,1})$ and $\mathsf{Sign}(\delta_{1,2})$ such that $M - m_j = \delta_{1,1}^2 + \delta_{1,2}$ to reduce the size of the public parameter that the MRA has to prepare. More precisely, in order to ensure that m_j and m_{j+1} used to prove $M - m_j \in [1, N]$ are the prohibited messages, a sender also proves knowledge of signatures $\mathsf{Sign}(m_j, m_{j+1})$. This technique is originally developed by Nakanishi et al. [15] in the context of revocable group signature. We further apply the technique of Nakanishi et al., in order to construct an efficient restrictive PKE scheme.

Putting all together, the ciphertext of the proposed construction has the form of (c, π), and each of components are computed as

$$c = \mathsf{PKE.Enc}(pk_d, M; u),$$
$$\pi = NIZK \left\{ \begin{array}{l} (M, u, S'', S_1, S_1', S_2, S_2', \delta_{1,1}, \delta_{1,2}, \delta_{2,1}, \delta_{2,2}, m_j, m_{j+1}) \\ : \mathsf{Verify}((m_j, m_{j+1}), S'') = 1 \wedge \mathsf{Verify}(\delta_{1,1}, S_1) = 1 \wedge \mathsf{Verify}'(\delta_{1,2}, S_2) = 1 \\ \wedge\ \mathsf{Verify}(\delta_{2,1}, S_1') = 1 \wedge \mathsf{Verify}'(\delta_{2,2}, S_2') = 1 \\ \wedge\ M - m_j = \delta_{1,1}^2 + \delta_{1,2} \bmod p \wedge m_{j+1} - M = \delta_{2,1}^2 + \delta_{2,2} \bmod p \\ \wedge\ c = \mathsf{PKE.Enc}(pk_d, M; u) \end{array} \right\}. \tag{2}$$

We again emphasize that π, a non-interactive proof of knowledge, can be instantiated efficiently, which is obtained from algebraic properties of the involved public key encryption scheme and signature scheme (More concretely, algebraic property of the BB signature [2] and the BBS+ signature [1,3,13] is used, and for the detailed description of these two signature scheme see Sect. 4.4).

Updating pk_{MS}. The above idea does not provide the functionality of updating the message space, but a simple modification (which will be explained below) enables us to obtain such a functionality. To update the message space specified by the MRA from MS to MS' where the prohibited messages of MS and MS' are $\{m_1, \ldots, m_r\}$ and $\{m_1', \ldots, m_{r'}'\}$ respectively, one may think that just re-publicizing signatures $\mathsf{Sign}''(m_i', m_{i+1}')$ for all $i \in \{0, \ldots, r'\}$ is suffice to do that (where $m_0' = 0$ and $m_{r'+1}' = N + 1$ as in the construction). However, in this way, a malicious sender will re-use some old signature $\mathsf{Sign}''(m_i, m_{i+1})$ and try to fool the verification process, which inspects whether a ciphertext encrypts a value belonging to a new message space MS'. A simple way to avoid the above attack is to publicize signatures $\mathsf{Sign}(t, m_i, m_{i+1})$ where t is a serial number, instead of $\mathsf{Sign}(m_i, m_{i+1})$. In this case a malicious sender is no longer able to re-use old signatures to fool the verification process.

4.2 Generic Construction

In this section, using the ideas we mentioned above, we show that a generic construction of restrictive PKE from an IND-CCA secure PKE, an EUF-CMA secure signature, and a

Σ-protocol for Eq. (2). In the following, let $[1, N] = \{1, \ldots, N\}$ be a set of whole possible messages (they may or may not be prohibited), r be the number of prohibited messages. Let (PKE.KeyGen, PKE.Enc, PKE.Dec) be an IND-CCA secure public key encryption scheme, (Sig.KeyGen, Sign, Verify), (Sig.KeyGen′, Sign′, Verify′), and (Sig.KeyGen″, Sign″, Verify″) be EUF-CMA secure signature schemes. The construction is as follows:

MRASetup(1^κ): Run $(K_s, K_v) \leftarrow \mathsf{Sig.KeyGen}(1^\kappa)$, $(K'_s, K'_v) \leftarrow \mathsf{Sig.KeyGen'}(1^\kappa)$, and $(K''_s, K''_v) \leftarrow \mathsf{Sig.KeyGen''}(1^\kappa)$. For $k \in [1, \lfloor \sqrt{N} \rfloor]$, compute $\sigma_{1,k} \leftarrow \mathsf{Sign'}(K'_s, k)$. For $k \in [0, \lfloor 2\sqrt{N} \rfloor]$, compute $\sigma_{2,k} \leftarrow \mathsf{Sign''}(K''_s, k)$. Output $pk_{MRA} = (K_v, K'_v, K''_v, \{\sigma_{1,k}\}_{k=1}^{\lfloor \sqrt{N} \rfloor}, \{\sigma_{2,k}\}_{k=0}^{\lfloor 2\sqrt{N} \rfloor})$ and $sk_{MRA} = (K_s, K'_s, K''_s)$.

RKeyGen(pk_{MRA}): Run $(pk, sk) \leftarrow \mathsf{PKE.KeyGen}(1^\kappa)$, and output $pk_d = pk$ and $sk_d = sk$.

MSSetup(pk_{MRA}, sk_{MRA}, MS): Let $MS = [1, N] \setminus \{m_1, \ldots, m_r\}$ where $m_1 < \cdots < m_r$, $m_0 = 0$, and $m_{r+1} = N + 1$. Choose a current serial number $t \in \mathbb{Z}_p$. For $\ell \in [0, r]$, compute $\sigma_\ell \leftarrow \mathsf{Sign}(K_s, t, m_\ell, m_{\ell+1})$. Output $pk_{MS} = (t, \{\sigma_\ell\}_{\ell=0}^{r})$.

REnc($pk_{MRA}, pk_d, MS, pk_{MS}, M$): For $M \in MS$, find the position j such that $m_j < M < m_{j+1}$. If there is no such m_j (which means $M \notin MS$), output $\perp$. Otherwise, find σ_j from pk_{MS}, compute $\delta_{1,1}, \delta_{2,1} \in [1, \lfloor \sqrt{N} \rfloor]$, and $\delta_{1,2}, \delta_{2,2} \in [0, \lfloor 2\sqrt{N} \rfloor]$, where $M - m_j = \delta_{1,1}^2 + \delta_{1,2}$ and $m_{j+1} - M = \delta_{2,1}^2 + \delta_{2,2}$ and find $\sigma_{1,\delta_{1,1}}, \sigma_{2,\delta_{1,2}}, \sigma_{1,\delta_{2,1}}$, and $\sigma_{2,\delta_{2,2}}$ from pk_{MRA}. Compute $c = \mathsf{PKE.Enc}(pk_d, M; u)$, and π of the following relations:

$$NIZK\left\{\begin{array}{l}(M, u, \sigma_j, \sigma_{1,\delta_{1,1}}, \sigma_{2,\delta_{1,2}}, \sigma_{1,\delta_{2,1}}, \sigma_{2,\delta_{2,2}}, \delta_{1,1}, \delta_{1,2}, \delta_{2,1}, \delta_{2,2}, m_j, m_{j+1}) \\ : \sigma_j = \mathsf{Sign}(t, m_j, m_{j+1}) \\ \wedge\ \sigma_{1,\delta_{1,1}} = \mathsf{Sign'}(\delta_{1,1}) \wedge \sigma_{2,\delta_{1,2}} = \mathsf{Sign''}(\delta_{1,2}) \\ \wedge\ \sigma_{1,\delta_{2,1}} = \mathsf{Sign'}(\delta_{2,1}) \wedge \sigma_{2,\delta_{2,2}} = \mathsf{Sign''}(\delta_{2,2}) \\ \wedge\ M - m_j = \delta_{1,1}^2 + \delta_{1,2} \bmod p \\ \wedge\ m_{j+1} - M = \delta_{2,1}^2 + \delta_{2,2} \bmod p \\ \wedge\ c = \mathsf{PKE.Enc}(pk_d, M; u)\end{array}\right\}$$

Finally, output $C = (c, \pi)$.

VerifyMS($pk_{MRA}, pk_d, pk_{MS}, C$): Output 1 if π is a valid proof, and 0, otherwise.

RDec(pk_{MRA}, pk_d, sk_d, C): Output $\mathsf{PKE.Dec}(sk_d, C)$.

The above construction, especially the zero-knowledge proof of Eq. (1), can be quite efficiently instantiated when one adopts appropriate digital signatures and Teranishi et al., Boudot, and Nakanishi et al. techniques. More concretely, the BBS+ signature [1,3,13] is applied for Sign, and two instance of the BB signature [2] are applied for Sign′ and Sign″. When applying the BBS+ signature and the BB signature, adopting the techniques of [3,15], the zero-knowledge proof of Eq. (1) is efficiently constructed, and the entire restrictive PKE construction becomes drastically more efficient than the construction employing the general NIZK technique.

However, a drawback of this construction is that the plaintext space has to be small. More concretely, $[1, N]$, the set of all possible (prohibited or allowed) messages, has to be just a polynomially (not exponentially) large. Due to the construction of NIZK proof, a message $M \in [1, N]$ have to be encoded into the underlying group as g^M, and hence a receiver must compute a discrete logarithm of g^M in order to recover the message

M. This constraint causes an inefficient decryption. However, it can be bypassed by restricting N to be sufficiently small to compute a discrete logarithm efficiently. When one can use Pollard's lambda method, M can be recovered from g^M in $O(\sqrt{N})$ computation time.

4.3 Security Analysis

The above construction satisfies the security requirement of verification soundness and IND-MSR-CCA security. Due to the limitation of pages, the proof is given in the appendix.

Theorem 1. *The construction given above satisfies verification soundness if the underlying signature scheme* (Sig.KeyGen, Sign, Verify) *is EUF-CMA secure, signatures* (Sig.KeyGen′, Sign′, Verify′) *and* (Sig.KeyGen″, Sign″, Verify″) *are EUF-wCMA secure, and the NIZK proof is constructed from Σ-protocol by using the Fiat-Shamir heuristics.*

Theorem 2. *The construction given above is IND-MSR-CCA secure if the underlying PKE scheme is IND-CCA secure and the NIZK proof is constructed from Σ-protocol by using Fiat-Shamir heuristics.*

4.4 Concrete Instantiation

BB and BBS+ Signatures. Here we describe the BB signature, the BBS+ signature, and their related definitions required in this paper.

Definition 6. *Bilinear groups are a tuple* $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ *such that* $\mathbb{G}$ *and* $\mathbb{G}_T$ *are cyclic groups of prime order* p, $g \in \mathbb{G}$ *is a generator of* $\mathbb{G}$, *and* e *is an efficiently computable bilinear map* $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ *with the following properties: for all* g, g', h, $h' \in \mathbb{G}$, $e(gg', h) = e(g, h)e(g', h)$ *and* $e(g, hh') = e(g, h)e(g, h')$, *and* $e(g, g)$ *is not the unit of* $\mathbb{G}_T$.

The description of the BB signature [2] is as follows:

$\mathsf{KeyGen}_{\mathrm{BB}}(1^\kappa)$**:** Choose $X \in_R \mathbb{Z}_p$ and $\tilde{g} \in \mathbb{G}$, computes $Y = \tilde{g}^X$, and outputs a verification key $vk = (\tilde{g}, Y) \in \mathbb{G}^2$ and a private signing key $sigk = X$.

$\mathsf{Sign}_{\mathrm{BB}}(pk, sk, M)$**:** Output a signature $F = \tilde{g}^{\frac{1}{X+M}}$.

$\mathsf{Verify}_{\mathrm{BB}}(pk, F)$**:** Check whether $e(F, Y\tilde{g}^M) \stackrel{?}{=} e(g, g)$.

In our RPKE scheme, to prove $M \in [1, N]$ we apply the BB signature whose signatures are represented as $\mathsf{Sign}_{\mathrm{BB}}(1)$, $\mathsf{Sign}_{\mathrm{BB}}(2)$, …, $\mathsf{Sign}_{\mathrm{BB}}(N)$. To prove the knowledge of a BB signature $F_k = \mathsf{Sign}_{\mathrm{BB}}(k)$ is as follows: Let $g_5 \in \mathbb{G}$ (we use g_5 for the same purpose in our RPKE, and therefore for the sake of clarity we use it here). Choose $\beta \in \mathbb{Z}_p$ and compute $C = F_k g_5^\beta$. Then, C satisfies the relation: $e(C, Y)/e(\tilde{g}, g) = e(g_5, Y)^\beta e(g_5, g)^\theta / e(C, g)^k$ where $\theta = \beta k$. Therefore, we prove the knowledge of DLs β and θ to prove the knowledge of $\mathsf{Sign}_{\mathrm{BB}}(k)$. This relation is appeared in the REnc algorithm of our RPKE scheme for the relations of C_2, C_3, C_4, and C_5. Note that, for C_3 and C_5, we use the notation $\dot{g}$ instead of $\tilde{g}$ in our RPKE scheme.

The BBS+ Signature [1,3,13] can be described as follows:

$\mathsf{KeyGen}_{\mathrm{BBS+}}(1^\kappa, L)$ Choose $X \in_R \mathbb{Z}_p$ and $g, g_1, \ldots, g_{L+1} \in \mathbb{G}$, where L is the length of messages. Computes $Y = g^X$, and outputs a verification key $vk = (g, g_1, \ldots, g_{L+1}, Y) \in \mathbb{G}^{L+3}$ and a private signing key $sigk = X$.

$\mathsf{Sign}_{\mathrm{BBS+}}(pk, sk, (m_1, \ldots, m_L))$ Choose $y, z \in_R \mathbb{Z}_p$, compute $B = (g_1^{m_1} \cdots g_L^{m_L} g_{L+1}^{y} g)^{\frac{1}{X+z}}$, and output a signature (B, y, z).

$\mathsf{Verify}_{\mathrm{BBS+}}(pk, (B, y, z))$ Check whether $e(B, Yg^z) \stackrel{?}{=} e(g_1^{m_1} \cdots g_L^{m_L} g_{L+1}^{y}, g)$.

In our RPKE scheme, to restrictive message space, we apply BBS+ signature with $L = 3$ whose signatures are represented as $\mathsf{Sign}_{\mathrm{BBS+}}(t, m_0, m_1)$, $\mathsf{Sign}_{\mathrm{BBS+}}(t, m_1, m_2)$, …, $\mathsf{Sign}_{\mathrm{BBS+}}(t, m_r, m_{r+1})$, where $(m_1, m_2, \ldots, m_r)$ are prohibited messages, $(m_0, m_{r+1}) = (0, N+1)$, and t is the serial number. To prove the knowledge of a BBS+ signature $\mathsf{Sign}_{\mathrm{BBS+}}(t, m_j, m_{j+1}) := (B_j, y_j, z_j)$ is as follows: Let $g_5 \in \mathbb{G}$. Choose $\alpha \in \mathbb{Z}_p$ and compute $C = B_j g_5^{\alpha}$. Then, C satisfies the relation:

$$e(C, Y)/e(g, g) = e(g_5, Y)^{\alpha} e(g_5, g)^{\zeta} e(g_1, g)^{t} e(g_2, g)^{m_j} e(g_3, g)^{m_{j+1}} e(g_4, g)^{y_j} / e(C, g)^{z_j}$$

where $\zeta = \alpha z_j$. Therefore, we prove the knowledge of DLs α, ζ, m_j, m_{j+1}, y_j, and z_j to prove the knowledge of $\mathsf{Sign}_{\mathrm{BBS+}}(t, m_j, m_{j+1})$. Note that proving of the knowledge of t is not necessary, since t is just used as a serial number in our RPKE scheme. This relation is appeared in the REnc algorithm of our RPKE scheme for the relation of C_1.

Construction. In this section, we give a concrete instantiation of RPKE. From the viewpoint of efficiency, we apply BB [2] and BBS+ [1,3,13] signatures to implement Teranishi/Nakanishi proof system. In addition, we apply an ElGamal type double encryption $\mathsf{DoubleEnc}$ to implement the building PKE scheme.

In the following scheme, $(g, g_1, g_2, g_3, g_4, Y_1)$ is a verification key of BBS+ signatures $\{(B_j, y_j, z_j)\}_{j=1}^{r-1}$, $(\tilde{g}, Y_2)$ is a verification key of BB signatures $\{F_{1,k}\}_{k=1}^{\lfloor\sqrt{N}\rfloor}$, $(\dot{g}, Y_3)$ is a verification key of BB signatures $\{F_{2,k}\}_{k=0}^{\lfloor 2\sqrt{N}\rfloor}$, and $pk_d = (\hat{f}, \hat{g}_1, \hat{g}_2, \hat{h})$ is a public key of the double encryption scheme $\mathsf{DoubleEnc}$. For a plaintext $M' \in \mathbb{G}'$ and a random number $u \in \mathbb{Z}_p$, $\mathsf{DoubleEnc}_{pk_d}(M'; u) = (\hat{g}_1^u, \hat{g}_2^u, M' \cdot \hat{h}^u)$. Other parameters are for computing NIZK proofs. These NIZK proofs work for exponent in $\mathbb{Z}_p$ so we need to encrypt M by $\hat{f}^M$ for some generator $\hat{f}$. Therefore, to apply this proving system to PKE, we require that a plaintext of the building PKE scheme $\mathsf{Enc}(\cdot)$ is $\hat{f}^M$, and the knowledge of M need to be proved from a ciphertext $\mathsf{Enc}(\hat{f}^M)$ by using NIZK system. When receiver obtains $\hat{f}^M$ by using own sk_d, receiver needs to solve the DL problem to obtain M from $\hat{f}^M$. Therefore, as in Boneh et al. [5] and Okamoto et al. [16], we assume N is small with the condition that the DL problem $(\hat{f}, \hat{f}^M)$ can be solved efficiently (e.g., by using baby-step-giant-step algorithm or Pollard's lambda method with expected time $O(\sqrt{N})$).

The concrete construction we propose is as follows:

$\mathsf{MRASetup}(1^\kappa)$**:** Let $(\mathbb{G}, \mathbb{G}_T)$ be a bilinear group with a κ-bit prime order p and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be a bilinear map. In addition, let $\mathbb{G}'$ be a DDH-hard group with the same order p. Let $H : \{0,1\}^* \to \mathbb{Z}_p$ be a hash function for NIZK proofs.

Choose generators $g, \tilde{g}, \dot{g}, g_1, \tilde{g}_1, g_2, g_3, g_4, g_5 \in \mathbb{G}$, $\hat{f} \in \mathbb{G}'$, a signing key of BBS+ signatures $X_1 \in \mathbb{Z}_p$, and signing keys of BB signatures $X_2, X_3 \in \mathbb{Z}_p$, and compute the a verification key of BBS+ signatures $Y_1 = g^{X_1}$, and verification keys of BB signatures $Y_2 = g^{X_2}$ and $Y_3 = g^{X_3}$. For $k \in [1, \lfloor \sqrt{N} \rfloor]$, compute $Sign'_{BB}(k) := F_{1,k} = \tilde{g}^{\frac{1}{X_2+k}}$. For $k \in [0, \lfloor 2\sqrt{N} \rfloor]$, compute $Sign''_{BB}(k) := F_{2,k} = \dot{g}^{\frac{1}{X_3+k}}$. Output $pk_{MRA} = (p, e, \mathbb{G}, \mathbb{G}_T, \mathbb{G}', H, Y_1, Y_2, Y_3, \{F_{1,k}\}_{k=1}, \{F_{2,k}\}_{k=0}^{\lfloor 2\sqrt{N} \rfloor}, \hat{f})$, and $sk_{MRA} = (X_1, X_2, X_3)$.

RKeyGen(pk_{MRA})**:** Choose $\hat{g}_1, \hat{g}_2 \in \mathbb{G}'$ and $z \in \mathbb{Z}_p$, and compute $\hat{h} = \hat{g}_1^z$. Output a public key of an ElGamal type double encryption scheme $pk_d = (\hat{g}_1, \hat{g}_2, \hat{h})$ and the corresponding secret key $sk_d = z$.

MSSetup(pk_{MRA}, sk_{MRA}, MS)**:** Let $(m_1, m_2, \ldots, m_r)$ be consecutive prohibited messages, $m_0 = 0$, and $m_{r+1} = N + 1$. Choose a current serial number $t \in \mathbb{Z}_p$. For $\ell \in [0, r]$, compute BBS+ signatures of three signed messages $(t, m_\ell, m_{\ell+1})$ $Sign_{BBS+}(t, m_\ell, m_{\ell+1}) := (B_\ell, y_\ell, z_\ell)$, where $B_\ell = (g_1^t g_2^{m_\ell} g_3^{m_{\ell+1}} g_4^{y_\ell} g)^{\frac{1}{X_1+z_\ell}}$, and y_ℓ, $z_\ell \in \mathbb{Z}_p$. Output $pk_{MS} = (t, \{(m_\ell, m_{\ell+1}, B_\ell, y_\ell, z_\ell)\}_{\ell=0}^{r})$.

REnc$(pk_{MRA}, pk_d, MS, pk_{MS}, M)$**:** For $M \in MS$, find the position j such that $m_j < M < m_{j+1}$. If there is no such m_j (which means $M \notin MS$), output $\perp$. compute $\delta_{1,1}, \delta_{2,1} \in [1, \lfloor \sqrt{N} \rfloor]$, and $\delta_{1,2}, \delta_{2,2} \in [0, \lfloor 2\sqrt{N} \rfloor]$, where $M - m_j = \delta_{1,1}^2 + \delta_{1,2}$ and $m_{j+1} - M = \delta_{2,1}^2 + \delta_{2,2}$ and find $Sign'_{BB}(\delta_{1,1}) = F_{1,\delta_{1,1}}$, $Sign''_{BB}(\delta_{1,2}) = F_{2,\delta_{1,2}}$, $Sign'_{BB}(\delta_{2,1}) = F_{1,\delta_{2,1}}$, and $Sign''_{BB}(\delta_{2,2}) = F_{2,\delta_{2,2}}$ from pk_{MRA}. Compute $c = \mathsf{DoubleEnc}_{pk_d}(\hat{f}^M; u)$, and π of the following relations:

$$\pi = NIZK \left\{ \begin{array}{l} (M, S'', S_1, S'_1, S_2, S'_2, \delta_{1,1}, \delta_{1,2}, \delta_{2,1}, \delta_{2,2}) \\ : S'' = \mathsf{Sign}_{\mathrm{BBS+}}(t, m_j, m_{j+1}) \\ \wedge S_1 = \mathsf{Sign}'_{\mathrm{BB}}(\delta_{1,1}) \wedge S_2 = \mathsf{Sign}''_{\mathrm{BB}}(\delta_{1,2}) \\ \wedge S'_1 = \mathsf{Sign}'_{\mathrm{BB}}(\delta_{2,1}) \wedge S'_2 = \mathsf{Sign}''_{\mathrm{BB}}(\delta_{2,2}) \\ \wedge M - m_j = \delta_{1,1}^2 + \delta_{1,2} \bmod p \\ \wedge m_{j+1} - M = \delta_{2,1}^2 + \delta_{2,2} \bmod p \wedge c = \mathsf{DoubleEnc}_{pk_d}(\hat{f}^M; u) \end{array} \right\}$$

Concretely, choose $\alpha, \beta_{1,1}, \beta_{1,2}, \beta_{2,1}, \beta_{2,2}, u, \xi_1, \xi'_1, \xi_2, \xi'_2 \in \mathbb{Z}_p$, compute $C_1 = B_j g_5^{\alpha}$, $C_2 = F_{1,\delta_{1,1}} g_5^{\beta_{1,1}}$, $C_3 = F_{2,\delta_{1,2}} g_5^{\beta_{1,2}}$, $C_4 = F_{1,\delta_{2,1}} g_5^{\beta_{2,1}}$, $C_5 = F_{2,\delta_{2,2}} g_5^{\beta_{2,2}}$, $C_6 = \tilde{g}^{\delta_{1,1}} \tilde{g}_1^{\xi_1}$, $C_7 = \tilde{g}^{\delta_{1,1}^2} \tilde{g}_1^{\xi'_1}$, $C_8 = \tilde{g}^{\delta_{2,1}} \tilde{g}_1^{\xi_2}$, $C_9 = \tilde{g}^{\delta_{2,1}^2} \tilde{g}_1^{\xi'_2}$, $\xi''_1 := \xi'_1 - \xi_1 \delta_{1,1}$, $\xi''_2 := \xi'_2 - \xi_2 \delta_{2,1}$, $C_{10} = \hat{g}_1^u$, $C_{11} = \hat{g}_2^u$, $C_{12} = \hat{f}^M \hat{h}^u$, $\zeta = \alpha z_j$, $\theta_{1,1} := \beta_{1,1} \delta_{1,1}$, $\theta_{1,2} := \beta_{1,2} \delta_{1,2}$, $\theta_{2,1} := \beta_{2,1} \delta_{2,1}$, and $\theta_{2,2} := \beta_{2,2} \delta_{2,2}$. In addition, compute

$$\pi = NIZK \left\{ \begin{array}{l} (M, \zeta, \alpha, y_j, z_j, m_j, m_{j+1}, \delta_{1,1}, \delta_{1,2}, \delta_{2,1}, \delta_{2,2}, \theta_{1,1}, \theta_{1,2}, \theta_{2,1}, \theta_{2,2}, \\ \beta_{1,1}, \beta_{1,2}, \beta_{2,1}, \beta_{2,2}, \xi_1, \xi'_1, \xi''_1, \xi_2, \xi'_2, \xi''_2, u) \\ : \frac{e(C_1, Y_1)}{e(g,g)} = \frac{e(g_5, Y_1)^{\alpha} e(g_5, g)^{\zeta} e(g_1^t, g) e(g_2, g)^{m_j} e(g_3, g)^{m_{j+1}} e(g_4, g)^{y_j}}{e(C_1, g)^{z_j}} \\ \wedge \frac{e(C_2, Y_2)}{e(\tilde{g}, g)} = \frac{e(g_5, Y_2)^{\beta_{1,1}} e(g_5, g)^{\theta_{1,1}}}{e(C_2, g)^{\delta_{1,1}}} \wedge \frac{e(C_3, Y_3)}{e(\dot{g}, g)} = \frac{e(g_5, Y_3)^{\beta_{1,2}} e(g_5, g)^{\theta_{1,2}}}{e(C_3, g)^{\delta_{1,2}}} \\ \wedge \frac{e(C_4, Y_2)}{e(\tilde{g}, g)} = \frac{e(g_5, Y_2)^{\beta_{2,1}} e(g_5, g)^{\theta_{2,1}}}{e(C_4, g)^{\delta_{2,1}}} \wedge \frac{e(C_5, Y_3)}{e(\dot{g}, g)} = \frac{e(g_5, Y_3)^{\beta_{2,2}} e(g_5, g)^{\theta_{2,2}}}{e(C_5, g)^{\delta_{2,2}}} \\ \wedge C_6 = \tilde{g}^{\delta_{1,1}} \tilde{g}_1^{\xi_1} \wedge C_7 = C_6^{\delta_{1,1}} \tilde{g}_1^{\xi''_1} \wedge C_7 = \tilde{g}^{-\delta_{1,2} + M - m_j} \tilde{g}_1^{\xi'_1} \\ \wedge C_8 = \tilde{g}^{\delta_{2,1}} \tilde{g}_1^{\xi_2} \wedge C_9 = C_8^{\delta_{2,1}} \tilde{g}_1^{\xi''_2} \wedge C_9 = \tilde{g}^{-\delta_{2,2} + m_{j+1} - M} \tilde{g}_1^{\xi'_2} \\ \wedge C_{10} = \hat{g}_1^u \wedge C_{11} = \hat{g}_2^u \wedge C_{12} = \hat{f}^M \hat{h}^u \end{array} \right\}$$

(Detailed NIZK proofs are described in the following). Output a ciphertext $C = (C_1, \ldots, C_{12}, \pi)$.

$\mathsf{VerifyMS}(pk_{MRA}, pk_d, pk_{MS}, C)$**:** Output 1 if π is a valid proof, and 0, otherwise.

$\mathsf{RDec}((pk_{MRA}, pk_d, sk_d, C))$**:** Compute $\hat{f}^M = C_{12}/C_{10}^z$, solve the DL problem $(\hat{f}, \hat{f}^M)$, and output M.

If MS is changed (let MS' be a new message space), then MRA chooses t' ($t' \neq t$ for all previous t), and opens BBS+ signatures $Sign_{BBS+}(t', m'_\ell, m'_{\ell+1})$ for all $\ell \in [0, r']$ as $pk_{MS'}$, where $(m'_1, m'_2, \ldots, m'_{r'})$ is the new consecutive prohibited messages, $m'_0 = 0$, and $m'_r = N$.

Detailed NIZK Proofs. Here, we show the detailed proof π of our RPKE scheme. Concretely, π is computed as follows. Note that all pairing values are pre-computable.

1. Choose $r_M, r_\zeta, r_\alpha, r_{y_j}, r_{z_j}, r_{m_j}, r_{m_{j+1}}, r_{\delta_{1,1}}, r_{\delta_{1,2}}, r_{\delta_{2,1}}, r_{\delta_{2,2}}, r_{\theta_{1,1}}, r_{\theta_{1,2}}, r_{\theta_{2,1}}, r_{\theta_{2,2}}, r_{\beta_{1,1}}, r_{\beta_{1,2}}, r_{\beta_{2,1}}, r_{\beta_{2,2}}, r_{\xi_1}, r_{\xi'_1}, r_{\xi''_1}, r_{\xi_2}, r_{\xi'_2}, r_{\xi''_2}, r_u \in \mathbb{Z}_p$.
2. Compute $R_1 = \frac{e(g_5,Y_1)^{r_\alpha}\, e(g_5,g)^{r_\zeta - \alpha r_{z_j}}\, e(g_1,g)^{t}\, e(g_2,g)^{r_{m_j}}\, e(g_3,g)^{r_{m_{j+1}}}\, e(g_4,g)^{r_{y_j}}}{e(B_j,g)^{r_{z_j}}}$, $R_2 = \frac{e(g_5,Y_2)^{r_{\beta_{1,1}}}\, e(g_5,g)^{r_{\theta_{1,1}} - \beta_{1,1} r_{\delta_{1,1}}}}{e(F_{1,\delta_{1,1}},g)^{r_{\delta_{1,1}}}}$, $R_3 = \frac{e(g_5,Y_3)^{r_{\beta_{1,2}}}\, e(g_5,g)^{r_{\theta_{1,2}} - \beta_{1,2} r_{\delta_{1,2}}}}{e(F_{2,\delta_{1,2}},g)^{r_{\delta_{1,2}}}}$, $R_4 = \frac{e(g_5,Y_2)^{r_{\beta_{2,1}}}\, e(g_5,g)^{r_{\theta_{2,1}} - \beta_{2,1} r_{\delta_{2,1}}}}{e(F_{1,\delta_{2,1}},g)^{r_{\delta_{2,1}}}}$, $R_5 = \frac{e(g_5,Y_3)^{r_{\beta_{2,2}}}\, e(g_5,g)^{r_{\theta_{2,2}} - \beta_{2,2} r_{\delta_{2,2}}}}{e(F_{2,\delta_{2,2}},g)^{r_{\delta_{2,2}}}}$, $R_6 = \tilde{g}^{r_{\delta_{1,1}}} \tilde{g}_1^{r_{\xi_1}}$, $R_7 = C_6^{r_{\delta_{1,1}}} \tilde{g}_1^{r_{\xi''_1}}$, $R_8 = \tilde{g}^{-r_{\delta_{1,2}} + r_M - r_{m_j}} \tilde{g}_1^{r_{\xi'_1}}$, $R_9 = \tilde{g}^{r_{\delta_{2,1}}} \tilde{g}_1^{r_{\xi_2}}$, $R_{10} = C_8^{r_{\delta_{2,1}}} \tilde{g}_1^{r_{\xi''_2}}$, $R_{11} = \tilde{g}^{-r_{\delta_{2,2}} + r_{m_{j+1}} - r_M} \tilde{g}_1^{r_{\xi'_2}}$, $R_{12} = \hat{g}_1^{r_u}$, $R_{13} = \hat{g}_2^{r_u}$, and $R_{14} = \hat{f}^{r_M} \hat{h}^{r_u}$.
3. Compute $c = H(R_1, \ldots, R_{14}, C_1, \ldots, C_{12}, pk_{MRA}, pk_{MS}, pk_d)$
4. Compute $s_M = r_M + cM$, $s_\zeta = r_\zeta + c\zeta$, $s_\alpha = r_\alpha + c\alpha$, $s_{y_j} = r_{y_j} + cy_j$, $s_{z_j} = r_{z_j} + cz_j$, $s_{m_j} = r_{m_j} + cm_j$, $s_{m_{j+1}} = r_{m_{j+1}} + cm_{j+1}$, $s_{\delta_{1,1}} = r_{\delta_{1,1}} + c\delta_{1,1}$, $s_{\delta_{1,2}} = r_{\delta_{1,2}} + c\delta_{1,2}$, $s_{\delta_{2,1}} = r_{\delta_{2,1}} + c\delta_{2,1}$, $s_{\delta_{2,2}} = r_{\delta_{2,2}} + c\delta_{2,2}$, $s_{\theta_{1,1}} = r_{\theta_{1,1}} + c\theta_{1,1}$, $s_{\theta_{1,2}} = r_{\theta_{1,2}} + c\theta_{1,2}$, $s_{\theta_{2,1}} = r_{\theta_{2,1}} + c\theta_{2,1}$, $s_{\theta_{2,2}} = r_{\theta_{2,2}} + c\theta_{2,2}$, $s_{\beta_{1,1}} = r_{\beta_{1,1}} + c\beta_{1,1}$, $s_{\beta_{1,2}} = r_{\beta_{1,2}} + c\beta_{1,2}$, $s_{\beta_{2,1}} = r_{\beta_{2,1}} + c\beta_{2,1}$, $s_{\beta_{2,2}} = r_{\beta_{2,2}} + c\beta_{2,2}$, $s_{\xi_1} = r_{\xi_1} + c\xi_1$, $s_{\xi'_1} = r_{\xi'_1} + c\xi'_1$, $s_{\xi''_1} = r_{\xi''_1} + c\xi''_1$, $s_{\xi_2} = r_{\xi_2} + c\xi_2$, $s_{\xi'_2} = r_{\xi'_2} + c\xi'_2$, $s_{\xi''_2} = r_{\xi''_2} + c\xi''_2$, and $s_u = r_u + cu$.
5. Output $C = (C_1, \ldots, C_{12}, \pi)$, where $\pi = (c, s_M, s_\zeta, s_\alpha, s_{y_j}, s_{z_j}, s_{m_j}, s_{m_{j+1}}, s_{\delta_{1,1}}, s_{\delta_{1,2}}, s_{\delta_{2,1}}, s_{\delta_{2,2}}, s_{\theta_{1,1}}, s_{\theta_{1,2}}, s_{\theta_{2,1}}, s_{\theta_{2,2}}, s_{\beta_{1,1}}, s_{\beta_{1,2}}, s_{\beta_{2,1}}, s_{\beta_{2,2}}, s_{\xi_1}, s_{\xi'_1}, s_{\xi''_1}, s_{\xi_2}, s_{\xi'_2}, s_{\xi''_2}, s_u)$.

Next, we show the verification of the above π. Note that all pairing values are pre-computable, except the followings $e(C_1, g^{s_{z_j}} Y_1^c)$, $e(C_2, g^{s_{\delta_{1,1}}} Y_2^c)$, $e(C_3, g^{s_{\delta_{1,2}}} Y_3^c)$, $e(C_4, g^{s_{\delta_{2,1}}} Y_2^c)$, and $e(C_5, g^{s_{\delta_{2,2}}} Y_3^c)$.

1. Compute $R'_1 = \frac{e(g_5,Y_1)^{s_\alpha}\, e(g_5,g)^{s_\zeta}\, e(g_1,g)^{t}\, e(g_2,g)^{s_{m_j}}\, e(g_3,g)^{s_{m_{j+1}}}\, e(g_4,g)^{s_{y_j}}\, e(g,g)^{c}}{e(C_1, g^{s_{z_j}} Y_1^c)}$, $R'_2 = \frac{e(g_5,Y_2)^{s_{\beta_{1,1}}}\, e(g_5,g)^{s_{\theta_{1,1}}}\, e(\tilde{g},g)^{c}}{e(C_2, g^{s_{\delta_{1,1}}} Y_2^c)}$, $R'_3 = \frac{e(g_5,Y_3)^{s_{\beta_{1,2}}}\, e(g_5,g)^{s_{\theta_{1,2}}}\, e(\hat{g},g)^{c}}{e(C_3, g^{s_{\delta_{1,2}}} Y_3^c)}$, $R'_4 = \frac{e(g_5,Y_2)^{s_{\beta_{2,1}}}\, e(g_5,g)^{s_{\theta_{2,1}}}\, e(\tilde{g},g)^{c}}{e(C_4, g^{s_{\delta_{2,1}}} Y_2^c)}$, $R'_5 = \frac{e(g_5,Y_3)^{s_{\beta_{2,2}}}\, e(g_5,g)^{s_{\theta_{2,2}}}\, e(\hat{g},g)^{c}}{e(C_5, g^{s_{\delta_{2,2}}} Y_3^c)}$, $R'_6 = \tilde{g}^{s_{\delta_{1,1}}} \tilde{g}_1^{s_{\xi_1}} C_6^{-c}$, $R'_7 = C_6^{s_{\delta_{1,1}}} \tilde{g}_1^{s_{\xi''_1}} C_7^{-c}$, $R'_8 = \tilde{g}^{-s_{\delta_{1,2}} + s_M - s_{m_j}} \tilde{g}_1^{s_{\xi'_1}} C_7^{-c}$, $R'_9 = \tilde{g}^{s_{\delta_{2,1}}} \tilde{g}_1^{s_{\xi_2}} C_8^{-c}$, $R'_{10} = C_8^{s_{\delta_{2,1}}} \tilde{g}_1^{s_{\xi''_2}} C_9^{-c}$, $R'_{11} = \tilde{g}^{-s_{\delta_{2,2}} + s_{m_{j+1}} - s_M} \tilde{g}_1^{s_{\xi'_2}} C_9^{-c}$, $R'_{12} = \hat{g}_1^{s_u} C_{10}^{-c}$, $R'_{13} = \hat{g}_2^{s_u} C_{11}^{-c}$, and $R'_{14} = \hat{f}^{s_M} \hat{h}^{s_u} C_{12}^{-c}$.
2. Check $c \stackrel{?}{=} H(R'_1, \ldots, R'_{14}, C_1, \ldots, C_{12}, pk_{MRA}, pk_{MS}, pk_d)$.

4.5 Efficiency Comparison

We give an efficiency comparison between the concrete construction given in Sect. 4.4 and a simple OR-proof based construction (whose detailed description is given in Appendix A) below. In the OR-proof based construction, computational time of encryption and verification, the ciphertext length, and the public verification key length are all proportional to the number of allowed messages $N - r$. In contrast, the proposed concrete construction achieves constant computational time of encryption and verification and constant size ciphertexts. The public verification key length now only depends on the number of prohibited messages, no longer depends on the size of the whole message space N. A small drawback of the proposed construction is the size of the public value pk_{MRA}, which is just one group element in the OR-proof based construction, whereas $O(\sqrt{N})$ group element in the proposed concrete construction. A more detailed comparison of the efficiency is summarized in Table 1.

Table 1. Efficiency Comparison. $ME(\mathbb{G})$, $ME(\mathbb{G}')$, and $ME(\mathbb{G}_T)$ denote the computational cost of multi-exponentiation in $\mathbb{G}$, $\mathbb{G}'$, and $\mathbb{G}_T$, respectively. BM denotes that of one bilinear map computation. $|\mathbb{G}|$ and $\left|\mathbb{Z}_p\right|$ denotes the bit-length of the representation of a element of $\mathbb{G}$ and $\mathbb{Z}_p$, respectively.

	Computational Cost		Ciphertext Length	Public Key Length	
	REnc	VerifyMS		pk_{MRA}	pk_{MS}
OR-proof based	$(N-r)ME(\mathbb{G})$ $+(3+3(N-r))ME(\mathbb{G}')$	$(N-r)ME(\mathbb{G})$ $+(3+3(N-r))ME(\mathbb{G}')$	$3\|\mathbb{G}\|+$ $2(N-r)\left\|\mathbb{Z}_p\right\|$	$\|\mathbb{G}\|$	$\|\mathbb{G}\|(N-r)$
Ours	$15ME(\mathbb{G})+6ME(\mathbb{G}')$ $+5ME(\mathbb{G}_T)$	$6ME(\mathbb{G})+3ME(\mathbb{G}')$ $+5ME(\mathbb{G}_T)+5BM$	$12\|\mathbb{G}\|+27\left\|\mathbb{Z}_p\right\|$	$\leq (3\sqrt{N}+4)\|\mathbb{G}\|$	$(r+1)(4\left\|\mathbb{Z}_p\right\|+\|\mathbb{G}\|)$ $+\left\|\mathbb{Z}_p\right\|$

References

1. Au, M.H., Susilo, W., Mu, Y.: Constant-size dynamic k-TAA. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 111–125. Springer, Heidelberg (2006)
2. Boneh, D., Boyen, X.: Short signatures without random oracles and the SDH assumption in bilinear groups. J. Cryptology 21(2), 149–177 (2008)
3. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004)
4. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
5. Boneh, D., Goh, E.-J., Nissim, K.: Evaluating 2-DNF formulas on ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer, Heidelberg (2005)
6. 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)
7. Camenisch, J., Damgård, I.: Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 331–345. Springer, Heidelberg (2000)
8. 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)

9. Cramer, R.: Modular Design of Secure yet Practical Cryptographic Protocols. PhD thesis, CWI and Uni. of Amsterdam (November 1996)
10. Damgård, I.: On Σ-protocol. Cryptologic Protocol Theory, CPT 2010, v.2 (2010), http://www.daimi.au.dk/~ivan/Sigma.pdf
11. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
12. Fuchsbauer, G., Pointcheval, D.: Proofs on encrypted values in bilinear groups and an application to anonymity of signatures. In: Shacham, H., Waters, B. (eds.) Pairing 2009. LNCS, vol. 5671, pp. 132–149. Springer, Heidelberg (2009)
13. Furukawa, J., Imai, H.: An efficient group signature scheme from bilinear maps. IEICE Transactions 89-A(5), 1328–1338 (2006)
14. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. 28(2), 270–299 (1984)
15. Nakanishi, T., Fujii, H., Hira, Y., Funabiki, N.: Revocable group signature schemes with constant costs for signing and verifying. IEICE Transactions 93-A(1), 50–62 (2010)
16. Okamoto, T., Takashima, K.: Homomorphic encryption and signatures from vector decomposition. In: Galbraith, S.D., Paterson, K.G. (eds.) Pairing 2008. LNCS, vol. 5209, pp. 57–74. Springer, Heidelberg (2008)
17. Tate, S.R., Vishwanathan, R.: Improving cut-and-choose in verifiable encryption and fair exchange protocols using trusted computing technology. In: DBSec, pp. 252–267 (2009)
18. Teranishi, I., Furukawa, J., Sako, K.: k-times anonymous authentication. IEICE Transactions 92-A(1), 147–165 (2009)

A A Naive RPKE Scheme Based on OR Proofs

Here, we briefly show a naive (inefficient) RPKE scheme based on simple OR-proof. For a fair comparison, we apply a double encryption DoubleEnc that is implemented over $\mathbb{G}'$ as in our RPKE scheme.

Protocol 1. *The Naive RPKE Scheme*

MRASetup(1^κ)**:** *Let $\mathbb{G}$ be a group with a κ-bit prime order p and $\mathbb{G}'$ be a DDH-hard group with the same order p. Let $H : \{0,1\}^* \to \mathbb{Z}_p$ be a hash function for NIZK proofs. Choose $g \in \mathbb{G}$ and $\hat{f} \in \mathbb{G}'$, and output $pk_{MRA} = (p, \mathbb{G}, \mathbb{G}', H, \hat{f})$, and $sk_{MRA} = \emptyset$.*

RKeyGen(pk_{MRA})**:** *Choose $\hat{g}_1, \hat{g}_2 \in \mathbb{G}'$ and $z \in \mathbb{Z}_p$, and compute $\hat{h} = \hat{g}_1^z$. Output a public key of a double encryption scheme $pk_d = (\hat{g}_1, \hat{g}_2, \hat{h})$ and the corresponding secret key $sk_d = z$.*

MSSetup(pk_{MRA}, sk_{MRA}, MS)**:** *Let $MS = (M_1, M_2, \ldots, M_{N-r})$. Output $pk_{MS} := \{Z_i = g^{M_i}\}_{i=1}^{N-r}$.*

REnc($pk_{MRA}, pk_d, MS, pk_{MS}, M$)**:** *A ciphertext of the naive RPKE scheme is represented as $\pi = NIZK\{M : C = \mathsf{DoubleEnc}_{pk_d}(\hat{f}^M) \wedge ((M = M_1) \vee \cdots \vee (M = M_{N-r}))\}$. Concretely, choose $u \in \mathbb{Z}_p$, and compute $C_1 = \hat{g}_1^u$, $C_2 = \hat{g}_1^u$, and $C_3 = \hat{f}^M \hat{h}^u$. Let a plaintext $M := M_k$ ($k \in [1, N-r]$). Choose $r_{u_k}, r_{M_k} \in \mathbb{Z}_p$, and compute $R_{1,k} = \hat{g}_1^{r_{u_k}}$, $R_{2,k} = \hat{g}_2^{r_{u_k}}$, $R_{3,k} = \hat{f}^{r_{M_k}} \hat{h}^{r_{u_k}}$, and $R_{4,k} = g^{r_{M_k}}$. For all $\ell \in [1, N-r] \setminus \{k\}$, choose $s_{u_\ell}, s_{M_\ell}, c_\ell \in \mathbb{Z}_p$, and compute $R_{1,\ell} = \hat{g}_1^{s_{u_\ell}} C_1^{-c_\ell}$, $R_{2,\ell} = \hat{g}_2^{s_{u_\ell}} C_2^{-c_\ell}$, $R_{3,\ell} = \hat{f}^{s_{M_\ell}} \hat{h}^{s_{u_\ell}} C_3^{-c_\ell}$,*

and $R_{4,\ell} = g^{s_{M_\ell}} Z_\ell^{-c_\ell}$. *Compute* $c = H(pk_{MS}, C_1, C_2, C_3, R_{1,1}, \ldots, R_{4,1}, \ldots, R_{1,N-r}, \ldots, R_{4,N-r})$. *Set* $c_k := c - \sum_{\ell=1,\ell\neq k}^{N-r} c_\ell \bmod p$. *Compute* $s_{M_k} = r_{M_k} + c_k M_k$ *and* $s_{u_k} = r_{u_k} + c_k u$. *Output* $\pi = (\{s_{M_\ell}, c_\ell\}_{\ell=1}^{N-r})$ *and* (C_1, C_2, C_3).

VerifyMS$(pk_{MRA}, pk_d, pk_{MS}, C)$**:** *Output 1 if* $\sum_{\ell=1}^{N-r} c_\ell \bmod p = H(pk_{MS}, C_1, C_2, C_3, \hat{g}_1^{s_{u_1}} C_1^{-c_1}, \hat{g}_2^{s_{u_1}} C_2^{-c_1}, \hat{f}^{s_{M_1}} \hat{h}^{s_{u_1}} C_3^{-c_1}, g^{s_{M_1}} Z_1^{-c_1}, \ldots, \hat{g}_1^{s_{u_{N-r}}} C_1^{-c_{N-r}}, \hat{g}_2^{s_{u_{N-r}}} C_2^{-c_{N-r}}, \hat{f}^{s_{M_{N-r}}} \hat{h}^{s_{u_{N-r}}} C_3^{-c_{N-r}}, g^{s_{M_{N-r}}} Z_{N-r}^{-c_{N-r}})$, *and 0, otherwise.*

RDec$(pk_{MRA}, pk_d, sk_d, C)$**:** *Compute* $\hat{f}^M = C_3 / C_1^z$, *solve the DL problem* $(\hat{f}, \hat{f}^M)$, *and output* M.

B Proof of Theorem 1

Proof. The NIZK proof has an extractor of the proved secret knowledge: given two accepting protocol views, where commitments are the same but challenges are different. By $\sigma^* = \mathsf{Sign}(\cdot,\cdot,\cdot)$ extracted from the output of $\mathcal{A}$, we consider two cases (1) $\sigma^* \notin pk_{MS^*}$, and (2) $\sigma^* \in pk_{MS^*}$. Let $M^* := \mathsf{RDec}(pk_{MRA}, pk_d, sk_d, C^*)$. From the definition of verification soundness, $M^* \in \{m_1, m_2, \ldots, m_r\}$.

Case 1: We construct an algorithm $\mathcal{B}$ that forges one of the underlying signature scheme (Sig.KeyGen, Sign, Verify). Let C be the challenger of unforgeability game of this signature. C sends a public value for verification K_v to $\mathcal{B}$. $\mathcal{B}$ computes other public values, and sends pk_{MRA} to $\mathcal{A}$. By using the signing oracle of the unforgeability game, $\mathcal{B}$ can answer message space queries sent from $\mathcal{A}$. $\mathcal{A}$ outputs C^*. Since $\mathsf{VerifyMS}(pk_{MRA}, pk_d, pk_{MS^*}, C^*) = 1$, using the extractor of the NIZK, $\mathcal{B}$ obtains σ^* and the corresponding signed messages. Since $\sigma^* \notin pk_{MS^*}$, σ^* is not an answer of the signing oracle. Therefore, $\mathcal{B}$ outputs a forged signature σ^* and wins.

Case 2: We construct an algorithm $\mathcal{B}'$ that forges one of the underlying signature scheme (Sig.KeyGen′, Sign′, Verify′). Let C' be the challenger of unforgeability game of this signature under the weakly chosen message attack [2]. First, $\mathcal{B}'$ sends messages $1, \ldots, \lfloor \sqrt{N} \rfloor$ to C'. Although, we describe the attack of signatures $\sigma_{1,*}$, the attack of signatures $\sigma_{2,*}$ is similarly described, and therefore we omit this part (in this case $\mathcal{B}'$ sends messages $0, \ldots, \lfloor 2\sqrt{N} \rfloor$ to C'). C' sends a public value for verification K'_v to $\mathcal{B}'$. $\mathcal{B}'$ obtains $\{\sigma_{1,k}\}_{k=1}^{\lfloor \sqrt{N} \rfloor}$ from $\mathcal{A}$, computes other public values, and sends pk_{MRA} to $\mathcal{A}$. Since $\mathcal{B}'$ has a signing key K_s of the underlying signature scheme (Sig.KeyGen, Sign, Verify), $\mathcal{B}'$ can answer message space queries. $\mathcal{A}$ outputs C^*. Since $\mathsf{VerifyMS}(pk_{MRA}, pk_d, pk_{MS^*}, C^*) = 1$, using the extractor of the NIZK, $\mathcal{B}'$ obtains σ^* and the corresponding signed messages. Since $\sigma^* \in pk_{MS^*}$, let $\sigma^* := \mathsf{Sign}(t, m_{j^*}, m_{j^*+1})$. Using the extractor of the NIZK, $\mathcal{B}'$ obtains $\delta_{1,1}, \delta_{1,2}, \sigma_{1,\delta_{1,1}}, \sigma_{2,\delta_{1,2}}, \delta_{2,1}, \delta_{2,2}, \sigma_{1,\delta_{2,1}}$, and $\sigma_{2,\delta_{2,2}}$, with the conditions $M^* - m_{j^*} \bmod p = \delta_{1,1}^2 + \delta_{1,2}$ and $m_{j^*+1} - M^* \bmod p = \delta_{2,1}^2 + \delta_{2,2}$. Next, we show that $m_{j^*+1} \leq M^*$ or $m_{j^*} \geq M^*$ holds as follows: Since $M^* \notin MS^*$ from the definition of verification soundness, there exists $m_{j'}$ such that $j' \in [1, r]$ and $M^* = m_{j'}$. For all $m_\ell < m_{j'}$, $m_\ell < M^*$ and $m_{\ell+1} \leq M^*$ hold. In addition, for all $m_\ell \geq m_{j'}$, $m_\ell \geq M^*$ and $m_{\ell+1} > M^*$ hold. Therefore, for the consecutive m_{j^*} and m_{j^*+1}, $m_{j^*+1} \leq M^*$ or $m_{j^*} \geq M^*$ holds[1]. This means $m_{j^*+1} - M^* \leq 0$ or

[1] Note that both cases $M^* < m_1$ and $m_r < M^*$ are included in these two cases.

$0 \geq M^* - m_{j^*}$ in $\mathbb{Z}$. If $0 \geq M^* - m_{j^*}$, then set $\delta := M^* - m_{j^*} \bmod p$, $\delta(1) := \delta_{1,1}$, $\delta(2) := \delta_{1,2}$, $\sigma_{1,\delta} := \sigma_{1,\delta_{1,1}}$, and $\sigma_{2,\delta} := \sigma_{2,\delta_{1,2}}$. Otherwise, If $m_{j^*+1} - M^* \leq 0$, then set $\delta := m_{j^*+1} - M^* \bmod p$, $\delta(1) := \delta_{2,1}$, $\delta(2) := \delta_{2,2}$, $\sigma_{1,\delta} := \sigma_{1,\delta_{2,1}}$, and $\sigma_{2,\delta} := \sigma_{1,\delta_{2,2}}$. Under the assumption $\lfloor \sqrt{N} \rfloor^2 + \lfloor 2\sqrt{N} \rfloor < p/2$, $\delta \in [p/2, p-1]$ holds, and therefore $\delta(1) \notin [1, \lfloor \sqrt{N} \rfloor]$ or $\delta(2) \notin [0, \lfloor 2\sqrt{N} \rfloor]$ hold. If $\delta(2) \notin [0, \lfloor 2\sqrt{N} \rfloor]$, then $\mathcal{B}'$ aborts. Note that the case $\delta(2) \notin [0, \lfloor 2\sqrt{N} \rfloor]$ can be captured in the attack of the signature scheme $(\mathsf{Sig.KeyGen}'', \mathsf{Sign}'', \mathsf{Verify}'')$. Now we assume that $\delta(1) \notin [1, \lfloor \sqrt{N} \rfloor]$. $\mathcal{B}'$ outputs a forged signature and message pair $(\delta(1), \sigma_{1,\delta(1)})$, and wins, since $\delta(1)$ is not an input of the signing oracle. □

C Proof of Theorem 2

Proof. Due to the zero-knowledge-ness of NIZK proof, any information is not revealed from π. Therefore, we can reduce the IND-MSR-CCA game to the IND-CCA game of the underlying PKE scheme. Let $\mathcal{A}$ be an adversary who breaks the IND-MSR-CCA security of our RPKE scheme, and C the challenger of the IND-CCA game of the corresponding PKE scheme. Then, we can construct an algorithm $\mathcal{B}$ that breaks the IND-CCA security of the underlying PKE scheme. First, C gives a public key of the PKE scheme pk to $\mathcal{B}$. $\mathcal{B}$ sets pk to pk_d, and sends pk_d to $\mathcal{A}$. When $\mathcal{A}$ issues a decryption query $C = (c, \pi)$, $\mathcal{B}$ checks whether C is a valid ciphertext or not. If C is valid, then $\mathcal{B}$ simply forwards the corresponding part of this query c to C as a decryption query of the IND-CCA game. When $\mathcal{A}$ sends the challenge messages M_0^* and M_1^*, $\mathcal{B}$ forwards M_0 and M_1 to C as the challenge messages. C returns the challenge ciphertext c^*. $\mathcal{B}$ computes the challenge ciphertext of RPKE by applying the simulated NIZK proofs, say π^*. $\mathcal{B}$ sends the challenge ciphertext of RPKE (c^*, π^*) to $\mathcal{A}$. If $\mathcal{A}$ issues a valid (which means the $\mathsf{VerifyMS}$ algorithm returns 1) decryption query $C = (c^*, \pi)$, then we can construct an algorithm $\mathcal{B}'$ who extracts signed messages m_j, $\delta_{1,1}$, and $\delta_{1,2}$ from C, computes $M_b = \delta_{1,1}^2 + \delta_{1,2} + m_j$, outputs b, and wins. For other decryption queries, $\mathcal{B}$ can apply the decryption oracle of the underlying PKE scheme. Finally, $\mathcal{A}$ outputs the guessing bit, and $\mathcal{B}$ also outputs the same bit as the guessing bit of the IND-CCA game. □

Unforgeability of Re-Encryption Keys against Collusion Attack in Proxy Re-Encryption

Ryotaro Hayashi[1], Tatsuyuki Matsushita[1],
Takuya Yoshida[2], Yoshihiro Fujii[2], and Koji Okada[2]

[1] Toshiba Corporation
1, Komukai Toshiba-Cho, Saiwai-Ku, Kawasaki-Shi, Kanagawa 212-8582, Japan
{ryotaro.hayashi,tatsuyuki.matsushita}@toshiba.co.jp
[2] Toshiba Solutions Corporation
3-22, Katamachi, Fuchu-Shi, Tokyo 183-8512, Japan
{Yoshida.Takuya,Fujii.Yoshihiro,Okada.Koji}@toshiba-sol.co.jp

Abstract. Proxy re-encryption (PRE) allows a proxy to convert a ciphertext encrypted for Alice (delegator) into a ciphertext for Bob (delegatee) by using a re-encryption key generated by Alice. In PRE, non-transferability is a desirable property that colluding proxies and delegatees cannot re-delegate decryption rights to a malicious user. However, it seems to be very difficult to directly construct a non-transferable PRE scheme albeit such attempts as in [9,15,8]. In this paper, we discuss the non-transferability and introduce a relaxed notion of the non-transferability, the unforgeability of re-encryption keys against collusion attack (UFReKey-CA), as one approach toward the non-transferability. We then propose two concrete constructions of PRE without random oracles that meet replayable-CCA security and UFReKey-CA assuming the q-wDBDHI and a variant of DHI problems are hard. Although the proposed schemes are partial solutions to non-transferable PRE, we believe that the results are significant steps toward the non-transferability.

Keywords: Proxy re-encryption, non-transferability, unforgeability of re-encryption keys.

1 Introduction

Proxy re-encryption(PRE) schemes introduced by Blaze, Bleumer, and Strauss [4], are cryptosystems with the following special property. Alice, the original recipient of some ciphertext, can delegate the decryption rights to Bob by creating a *re-encryption key* then giving it to a semi-trusted entity called *proxy*. Consequently, Alice lets the proxy to convert ciphertexts for Alice into ciphertexts for Bob without revealing any information about the underlying plaintexts to the proxy. In PRE, Alice and Bob are called a *delegator* and a *delegatee*, respectively.

PRE schemes are often categorized by the following properties: *bidirectional* or *unidirectional*, and *multi-hop* or *single-hop*. In bidirectional PRE, a re-encryption key to convert a ciphertext for Alice into Bob's can be used to convert a ciphertext in the opposite direction (from Bob to Alice). On the other hand, in unidirectional PRE, a re-encryption key from Alice to Bob never helps re-encryption

T. Iwata and M. Nishigaki (Eds.): IWSEC 2011, LNCS 7038, pp. 210–229, 2011.

in the opposite direction. A PRE scheme is multi-hop if a proxy can re-encrypt ciphertexts that are already re-encrypted, while re-encryption is allowed only once in a single-hop PRE scheme. Any of the previously proposed schemes is either a unidirectional single-hop scheme, or a bidirectional multi-hop scheme. We study unidirectional and single-hop schemes in this paper.

After Blaze et al. introduced the concept of PRE, many concrete PRE schemes (e.g. those in [10,12,6,14]) with high confidentiality, i.e. (replayable) chosen ciphertext attack (CCA) security, were proposed. In the definition of the (replayable) CCA security, it is assumed that proxies who have the re-encryption keys to convert ciphertexts of the (target) honest delegator to corrupted users' are not corrupted.

In addition to the above basic security property, it is also important to consider the security where such proxies are corrupted. Recently, as cloud computing emerges, PRE gains much more attention as one of the key security components to provide secure cloud services. The security against corrupted proxies is especially important in such applications since the proxies may be out of control of honest users and the proxies are more likely to be attacked than those in on-premise systems. In [1], Ateniese, Fu, Green, and Hohenberger mention the security notion, *non-transferability*, with respect to the security against malicious proxies, which is described as *"The (malicious) proxy and a set of colluding delegatees cannot re-delegate decryption rights."* They also note that *"achieving a proxy scheme that is* non-transferable*, in the sense that the only way for Bob to transfer offline decryption capabilities to Carol is to expose his own secret key, seems to be the main open problem left for proxy re-encryption."*

Until now, some attempts for non-transferable PRE have been taken. For example, in the scheme proposed by Libert and Vergnaud [9], a delegator can identify the malicious proxies by analyzing a re-encryption key to convert ciphertexts of the delegator into some malicious user's generated (forged) by the colluding proxies and delegatees. Although it is one possible approach to the non-transferable PRE, it still cannot prevent colluding proxies and delegatees from re-delegating the decryption rights. Further, the scheme is less efficient in the sense that the ciphertext size depends on the number of delegations and it is only proved to be secure against chosen plaintext attack(CPA). In the scheme by Wang et al. [15] which is an ID-based PRE scheme, a trusted third party (private-key generator, PKG) takes part in generating re-encryption keys. This approach, however, is not a complete solution to non-transferable PRE because, as pointed out by He, Chim, Hui, and Yiu [8], it is just a transformation of "delegatee-proxy-collusion transferable problem" to "PKG alone transferable problem." In the ID-based scheme by He, Chim, Hui, and Yiu [8], the delegator and the delegatee communicate and send some information to each other to generate the re-encryption key (The delegator also communicates with PKG). Therefore, colluding proxies and delegatees cannot generate a new re-encryption key without delegator's help. This approach itself is worth mentioning, however, their scheme is less practical because it requires additional communication between the

delegator and the delegatee (and PKG, in the ID-based scheme such as [8]). Further, the non-transferability of the scheme in [8] is not proved formally.

In this paper, we introduce *the unforgeability of re-encryption keys against collusion attack (UFReKey-CA)*, which is a relaxed notion (necessary condition) of the non-transferability, as one approach toward the non-transferability. Roughly speaking, UFReKey-CA means that even colluding proxies and delegatees cannot generate a re-encryption key for some user. To the best of our knowledge, there exists no PRE scheme that satisfies UFReKey-CA and does not require either a trusted third party or communication between a delegator and a delegatee during re-encryption key generation. We also propose a security notion, the *strong* unforgeability of re-encryption key against collusion attack (sUFReKey-CA). Since sUFReKey-CA implies UFReKey-CA and sUFReKey-CA is simpler (i.e. easier to treat) definition than UFReKey-CA, sUFReKey-CA is useful to prove UFReKey-CA. We then propose a concrete PRE scheme that meets both the replayable CCA security and sUFReKey-CA assuming the hardness of variants of Diffie–Hellman inversion problems in the standard model. In our scheme, neither trusted third party nor communication between a delegator and a delegatee is required during re-encryption key generation, and the size of the keys and the computational cost does not depend on the number of delegations. We also propose a PRE scheme supporting temporary delegation, which can limit the lifetime of re-encryption keys within a certain time interval. Although the proposed schemes are partial solutions to non-transferable PRE, we believe that the results are significant steps toward non-transferability.

This paper is organized as follows. In Section 2, we review the definitions related to our proposal. We discuss the formal definition of the non-transferability and introduce the (strong) unforgeability of re-encryption keys against collusion attack in Section 3. We propose a concrete PRE scheme in Section 4, and that supporting temporary delegation in Section 5. We conclude in Section 6.

2 Preliminaries

2.1 Bilinear Maps and Complexity Assumptions

Groups $(\mathbb{G}, \mathbb{G}_T)$ of prime order p are called bilinear map groups if there exists a mapping $e : \mathbb{G}\times\mathbb{G} \to \mathbb{G}_T$ with the following properties: (1)bilinearity: $e(g^a, h^b) = e(g,h)^{ab}$ for any $(g,h) \in \mathbb{G} \times \mathbb{G}$ and $a, b \in \mathbb{Z}$, (2) $e(\cdot,\cdot)$ is efficiently computable for any input pair, and (3) non-degeneracy: $e(g,h) \neq 1_{\mathbb{G}_T}$ whenever $g, h \neq 1_{\mathbb{G}}$.

We describe q-weak the decision bilinear Diffie–Hellman Inversion (q-wDBDHI) problem. In [10], it is mentioned that the q-wDBDHI problem is hard in generic groups (See also [7].).

Definition 1 (q-wDBDHI problem). *The q-weak decision bilinear Diffie–Hellman Inversion problem is to distinguish the two distributions* $(g, g^a, g^{a^2}, \ldots, g^{a^q}, g^b, e(g,g)^{b/a})$ *and* $(g, g^a, g^{a^2}, \ldots, g^{a^q}, g^b, e(g,g)^z)$ *for* $a, b, z \xleftarrow{R} \mathbb{Z}_p^*$.

We next propose new problems related to Diffie–Hellman inversion problem.

Definition 2 (2-DHIwRA problem). *The 2-Diffie–Hellman inversion with randomized answers problem is computing* $g^{1/(a+c)}$ *given the following:*

- *input 1:* $g, g^a, g^{a^2}, c \quad (a, c \xleftarrow{R} \mathbb{Z}_p^*)$,
- *input 2:* $(x_i, y_i, D_i, E_i, F_i) = (x_i, y_i, g^{\frac{x_i+\gamma_i}{a(a+c)}}, g^{\frac{ay_i+\gamma_i}{a+c}}, g^{\frac{x_i+\gamma_i}{a}})$ *where* $x_i, y_i,$ $\gamma_i \xleftarrow{R} \mathbb{Z}_p^*$ *for* $i \in \{1, \ldots, L\}$ *and* L *is polynomially bounded.*

Definition 3 (m-2-DHIwRA problem). *The modified 2-Diffie–Hellman inversion with randomized answers problem is computing* $g^{1/(a+c)}$ *given the inputs 1 and 2 of 2-DHIwRA problem and*

- *input 3:* $(y', \mu, D', E', F', G', H') = (y', \mu, g^{\frac{cy'+\gamma'}{a(a+c)}+\delta'_x}, g^{\frac{a\mu y'+\gamma'}{a+c}+\delta'_y}, g^{\frac{cy'+\gamma'}{a}}, g^{a(a+c)\delta'_x}, g^{a(a+c)\delta'_y})$ *where* $y', \mu, \gamma', \delta'_x, \delta'_y \xleftarrow{R} \mathbb{Z}_p^*$.

The 2-DHIwRA and m-2-DHIwRA problems can be seen as the 2-DHI (Diffie–Hellman Inversion) problem [11] with the additional inputs. Therefore, these problems are not harder than the 2-DHI problem. In this paper, we assume the hardness of the above problems to prove the security of our schemes.

2.2 Unidirectional Proxy Re-Encryption

In this section, we describe the syntactic definition of unidirectional proxy re-encryption [1,2] and its security notion [10].

First, we describe the syntactic definition of unidirectional proxy re-encryption.

Definition 4. *A (single-hop) unidirectional proxy re-encryption (PRE) scheme consists of the following algorithms:*

- $\mathsf{Global\text{-}setup}(\lambda)$ *is a probabilistic algorithm which takes a security parameter* λ *and returns a set* par *of public parameters with a plaintext space* $\mathcal{M}$.
- $\mathsf{Keygen}(\lambda, \mathsf{par})$ *is a probabilistic algorithm which takes parameters* λ *and* par *and returns a public/secret key pair* (pk, sk).
- $\mathsf{Enc}_1(m, pk_j, \mathsf{par})$ *is a probabilistic algorithm which takes a plaintext* $m \in \mathcal{M}$, *a user* j*'s public key* pk_j, *and* par, *and returns a first level ciphertext* C_j *for* j, *which cannot be re-encrypted for another user.*
- $\mathsf{Enc}_2(m, pk_i, \mathsf{par})$ *is a probabilistic algorithm which takes a plaintext* $m \in \mathcal{M}$, *a user* i*'s public key* pk_i, *and* par, *and returns a second level ciphertext* C_i *for* i, *which can be re-encrypted with re-encryption keys for another user.*
- $\mathsf{ReKeygen}(sk_i, pk_j, \mathsf{par})$ *is a probabilistic algorithm which takes a user* i*'s secret key* sk_i, *a user* j*'s public key* pk_j, *and* par, *and returns a re-encryption key* R_{ij} *to re-encrypt second level ciphertexts for* i *into first level ciphertexts for* j.
- $\mathsf{ReEnc}(R_{ij}, C_i, \mathsf{par})$ *is a probabilistic algorithm which takes a re-encryption key* R_{ij}, *a second level ciphertext* C_i *encrypted under* pk_i, *and a set* par *of common public parameters, and returns a first level ciphertext* C_j *re-encrypted for* j *or a distinguished message '*`invalid`*.'*
- $\mathsf{Dec}_1(C_j, sk_j, \mathsf{par})$ *is a deterministic algorithm which takes a first level ciphertext* C_j *for* j, *a user* j*'s secret key* sk_j, *and* par, *and returns a plaintext* m *or a distinguished message '*`invalid`*'.*

$\mathsf{Dec}_2(C_i, sk_i, \mathsf{par})$ *is a deterministic algorithm which takes a second level ciphertext* C_i *for* i*, a user* i*'s secret key* sk_i*, and* par*, and returns a plaintext* m *or a distinguished message* 'invalid'*.*

To lighten notations, we will sometimes omit to explicitly write the set par of public parameters, taken as input by all but one of the above algorithms.

Next, we describe the definition of the replayable chosen ciphertext security of unidirectional PRE schemes by Libert and Vergnaud [10]. In unidirectional single-hop PRE schemes, there exist two types of ciphertexts, first level and second level ciphertexts. Therefore, it is necessary to prove the confidentiality of each types of ciphertexts. First, we describe the security definition of second level ciphertexts.

Definition 5 (Second level RCCA security). *A unidirectional single-hop proxy re-encryption scheme is second level secure against replayable chosen-ciphertext attack (RCCA) (or second level RCCA secure for short) if*

$$\begin{aligned}|\Pr[(pk_*, sk_*) \leftarrow \mathsf{Keygen}(\lambda); \{(pk_h, sk_h) \leftarrow \mathsf{Keygen}(\lambda)\}; \{(pk_c, sk_c) \leftarrow \mathsf{Keygen}(\lambda)\};\\ \{R_{*h} \leftarrow \mathsf{ReKeygen}(sk_*, pk_h)\};\ \{R_{h*} \leftarrow \mathsf{ReKeygen}(sk_h, pk_*)\};\\ \{R_{hc} \leftarrow \mathsf{ReKeygen}(sk_h, pk_c)\};\ \{R_{hh'} \leftarrow \mathsf{ReKeygen}(sk_h, pk_{h'})\};\\ (m_0, m_1, St) \leftarrow \mathcal{A}^{\mathcal{O}_{\text{reenc}}, \mathcal{O}_{\text{1-dec}}}(pk_*, \{pk_h\}, \{(pk_c, sk_c)\},\\ \{R_{*h}\}, \{R_{h*}\}, \{R_{hc}\}, \{R_{hh'}\});\\ d^* \xleftarrow{R} \{0,1\};\ C^* \leftarrow \mathsf{Enc}_2(m_{d^*}, pk_*);\ d' \xleftarrow{R} \mathcal{A}^{\mathcal{O}_{\text{reenc}}, \mathcal{O}_{\text{1-dec}}}(C^*, St) : d' = d^*] - 1/2|\end{aligned}$$

is negligible for any polynomial time algorithm $\mathcal{A}$*. Above,* St *is the state information maintained by* $\mathcal{A}$*, and keys subscripted by "*$*$*", h (or* h'*), and* c *are those for the target honest user, a honest user other than the target honest user, and a corrupted user, respectively. Oracles* $\mathcal{O}_{\text{reenc}}$ *and* $\mathcal{O}_{\text{1-dec}}$ *proceed as follows:*

- *Re-encryption oracle* $\mathcal{O}_{\text{reenc}}$*: on input* (pk_i, pk_j, C) *where* C *is a second level ciphertext and* pk_i*,* pk_j *were produced by* Keygen*, this oracle responds with* invalid *if* C *is not properly shaped (ill-formed) with respect to* pk_i*. It returns a special symbol* $\perp$ *if* j *is a corrupted user and* $(pk_i, C) = (pk_*, C^*)$*. Otherwise, the re-encrypted first level ciphertext* $C_j = \mathsf{ReEnc}(\mathsf{ReKeygen}(sk_i, pk_j), C)$ *is returned to* $\mathcal{A}$*.*
- *First level decryption oracle* $\mathcal{O}_{\text{1-dec}}$*: given a pair* (pk, C)*, where* C *is a first level ciphertext and* pk *was produced by* Keygen*, this oracle returns* invalid *if* C *is ill-formed with respect to* pk*. If the query occurs in the guess stage, it outputs* $\perp$ *if* (pk, C) *is a* Derivative *of* (pk_*, C^*)*. Otherwise, the plaintext* $m = \mathsf{Dec}_1(sk, C)$ *is returned to* $\mathcal{A}$*.* Derivatives *of* (pk_*, C^*) *are defined as follows: If* C *is a first level ciphertext and* pk *is an honest user's key,* (pk, C) *is a* Derivative *of* (pk_*, C^*) *if* $\mathsf{Dec}_1(sk, C) \in \{m_0, m_1\}$*.*

Next, we describe the definition of RCCA security of first level ciphertexts.

Definition 6 (First level RCCA security). *A unidirectional single-hop proxy re-encryption scheme is first level secure against replayable chosen-ciphertext attack (RCCA) (or first level RCCA secure for short) if*

$$|\Pr[(pk_*, sk_*) \leftarrow \mathsf{Keygen}(\lambda); \{(pk_h, sk_h) \leftarrow \mathsf{Keygen}(\lambda)\}; \{(pk_c, sk_c) \leftarrow \mathsf{Keygen}(\lambda)\};$$
$$\{R_{*c} \leftarrow \mathsf{ReKeygen}(sk_*, pk_c, \mathsf{par})\};$$
$$\{R_{*h} \leftarrow \mathsf{ReKeygen}(sk_*, pk_h, \mathsf{par})\}; \{R_{h*} \leftarrow \mathsf{ReKeygen}(sk_h, pk_*, \mathsf{par})\};$$
$$\{R_{hc} \leftarrow \mathsf{ReKeygen}(sk_h, pk_c, \mathsf{par})\}; \{R_{hh'} \leftarrow \mathsf{ReKeygen}(sk_h, pk_{h'}, \mathsf{par})\};$$
$$(m_0, m_1, St) \leftarrow \mathcal{A}^{\mathcal{O}_{1\text{-dec}}}(pk_*, \{pk_h\}, \{(pk_c, sk_c)\},$$
$$\{R_{*c}\}, \{R_{*h}\}, \{R_{h*}\}, \{R_{hc}\}, \{R_{hh'}\});$$
$$d^* \xleftarrow{R} \{0,1\}; C^* \leftarrow \mathsf{Enc}_1(m_{d^*}, pk_*); d' \xleftarrow{R} \mathcal{A}^{\mathcal{O}_{1\text{-dec}}}(C^*, St) : d' = d^*] - 1/2|$$

is negligible for any polynomial time algorithm $\mathcal{A}$. Above, St is the state information maintained by $\mathcal{A}$, and keys subscripted by "$*$", h (or h'), and c are those for the target honest user, a honest user other than the target honest user, and a corrupted user, respectively. Oracle $\mathcal{O}_{1\text{-dec}}$ is the same as that in Definition 5 except that the definition of Derivative*s: If C is a first level ciphertext and $pk = pk_*$, (pk, C) is a* Derivative *of (pk_*, C^*) if $\mathsf{Dec}_1(sk, C) \in \{m_0, m_1\}$.*

Above, all re-encryption keys are available to the adversary. Therefore, the re-encryption oracle becomes useless and is not given to the adversary.

2.3 Unidirectional Proxy Re-Encryption with Temporary Delegation

In this section, we describe the syntactic definition of unidirectional proxy re-encryption with temporary delegation and its security notion [10]. In the PRE scheme with temporary delegation, it only allows the proxy to re-encrypt messages from A to B during a limited time period.

The model of unidirectional PRE scheme supporting temporary delegation is almost the same as that in Definition 4 except that re-encryption key generation, encryption, and re-encryption algorithms take a period $\ell \in \{1, ..., L\}$ as input. Intuitively, the re-encryption key generated by ReKeygen with a period ℓ, can be used to re-encrypt the ciphertext generated by Enc_2 with the same period ℓ. Note that the public and secret keys are common to all time periods.

Next, we describe the security notion of a PRE scheme with temporary delegation. In Definition 5, the challenger generates public keys for all parties and allows the adversary to obtain private keys for some of them (*known key model*).

On the other hand, in [10], a stronger security notion is also proposed, called RCCA security in the *chosen key model* (RCCA-CK security). In this model, the adversary can *arbitrarily* choose public keys without demonstrating knowledge of the private keys. This provides the adversary with much more flexibility and power in attacking other honest parties in the system.

The definition of RCCA-CK security can be extended to that for PRE schemes with temporary delegation. We describe the second level RCCA security in the chosen key model for PRE schemes with temporary delegation.

Definition 7 (Second level RCCA-CK security for PRE schemes with temporary delegation). *A unidirectional single-hop proxy re-encryption scheme with temporary delegation is second level secure against replayable chosen-ciphertext attack in the chosen key model (or RCCA-CK secure for short) if*

$$|\Pr[\{(pk_i, sk_i) \leftarrow \mathsf{Keygen}(\lambda)\}_{i \in HU};\ \{R_{ii'} \leftarrow \mathsf{ReKeygen}(sk_i, pk_{i'})\}_{i,i' \in HU};$$
$$(m_0, m_1, i^*, \ell^*, St) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{deleg}}, \mathcal{O}_{\mathrm{reenc}}, \mathcal{O}_{\text{1-dec}}}(\{pk_i\}, \{R_{ii'}\});\ d^* \xleftarrow{R} \{0,1\};$$
$$C^* \xleftarrow{R} \mathsf{Enc}_2(\ell^*, m_{d^*}, pk_{i^*});\ d' \xleftarrow{R} \mathcal{A}^{\mathcal{O}_{\mathrm{deleg}}, \mathcal{O}_{\mathrm{reenc}}, \mathcal{O}_{\text{1-dec}}}(C^*, St) : d' = d^*] - 1/2|$$

is negligible for any polynomial time algorithm $\mathcal{A}$. Above, St is the state information maintained by $\mathcal{A}$, HU is the set of honest users, $i^* \in HU$ is the target user, and ℓ^* is the target time period.

Oracles $\mathcal{O}_{\mathrm{deleg}}$, $\mathcal{O}_{\mathrm{reenc}}$, and $\mathcal{O}_{\text{1-dec}}$ proceed as follows:

- *Delegation oracle $\mathcal{O}_{\mathrm{deleg}}$: on input (ℓ, pk_i, pk_j) where ℓ is a time period, pk_i is a public key of honest user $i \in HU$ (and either $\ell \neq \ell^*$ or $i \neq i^*$ in any stage), and pk_j is a public key which $\mathcal{A}$ chooses arbitrary, this oracle responds with $\mathsf{ReKeygen}(\ell, sk_i, pk_j)$.*
- *Re-encryption oracle $\mathcal{O}_{\mathrm{reenc}}$: on input (ℓ, pk_i, pk_j, C) where ℓ is a time period, C is a second level ciphertext, pk_i is a public key of honest user $i \in HU$, and pk_j is a public key which $\mathcal{A}$ chooses arbitrary, this oracle responds with* `invalid` *if C is ill-formed with respect to pk_i. It returns a special symbol $\perp$ if $j \notin HU$ and $(\ell, pk_i, C) = (\ell^*, pk_*, C^*)$. Otherwise, the re-encrypted first level ciphertext $C_j = \mathsf{ReEnc}(\mathsf{ReKeygen}(\ell, sk_i, pk_j), C)$ is returned to $\mathcal{A}$.*
- *First level decryption oracle $\mathcal{O}_{\text{1-dec}}$: given a pair (pk_i, C), where C is a first level ciphertext and $i \in HU$, this oracle returns* `invalid` *if C is ill-formed with respect to pk_i. If the query occurs in the guess stage, it outputs a special symbol $\perp$ if (pk_i, C) is a* Derivative *of the challenge pair (pk_{i^*}, C^*). Otherwise, the plaintext $m = \mathsf{Dec}_1(sk_i, C)$ is returned to $\mathcal{A}$.* Derivatives *of (pk_{i^*}, C^*) are defined as follows: If C is a first level ciphertext and $i \in HU$, (pk_i, C) is a* Derivative *of (pk_{i^*}, C^*) if C and C^* are encrypted for the same time period ℓ^* and $\mathsf{Dec}_1(sk_i, C) \in \{m_0, m_1\}$.*

Next, we describe the definition of RCCA-CK security of first level ciphertexts for PRE schemes with temporary delegation.

Definition 8 (First level RCCA-CK security for PRE schemes with temporary delegation). *A unidirectional single-hop proxy re-encryption scheme with temporary delegation is first level secure against replayable chosen-ciphertext attack in the chosen key model (RCCA-CK) (or RCCA-CK secure for short) if*

$$|\Pr[\{(pk_i, sk_i) \leftarrow \mathsf{Keygen}(\lambda)\}_{i \in HU};\ \{R_{ii'} \leftarrow \mathsf{ReKeygen}(sk_i, pk_{i'})\}_{i,i' \in HU};$$
$$(m_0, m_1, i^*, \ell^*, St) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{deleg}}, \mathcal{O}_{\text{1-dec}}}(\{pk_i\}, \{R_{ii'}\});\ d^* \xleftarrow{R} \{0,1\};$$
$$C^* \xleftarrow{R} \mathsf{Enc}_1(\ell^*, m_{d^*}, pk_{i^*});\ d' \xleftarrow{R} \mathcal{A}^{\mathcal{O}_{\mathrm{deleg}}, \mathcal{O}_{\text{1-dec}}}(C^*, St) : d' = d^*] - 1/2|$$

is negligible for any polynomial time algorithm $\mathcal{A}$. Above, St is the state information maintained by $\mathcal{A}$, HU is the set of honest users, $i^* \in HU$ is the target user, and ℓ^* is the target time period. An oracle $\mathcal{O}_{\mathrm{deleg}}$ responds with $\mathsf{ReKeygen}(\ell, sk_i, pk_j)$ for any *query (ℓ, pk_i, pk_j) where $i \in HU$. That is, (ℓ^*, pk_{i^*}, pk_j) can be queried to $\mathcal{O}_{\mathrm{deleg}}$. An oracle $\mathcal{O}_{\text{1-dec}}$ is the same as that in Definition 7 except that the definition of* Derivatives*: If C is a first level ciphertext and $pk = pk_*$, (pk, C) is a* Derivative *of (pk_*, C^*) if $\mathsf{Dec}_1(sk, C) \in \{m_0, m_1\}$.*

In the above definition, all re-encryption keys are available to the adversary. Therefore, the re-encryption oracle becomes useless and is not given to the adversary.

3 Unforgeability of Re-Encryption Keys against Collusion Attack

In this section, we discuss about the non-transferability and propose a security definition called the unforgeability of re-encryption keys against collusion attack.

As described before, the non-transferability is described in [1] as "*A proxy scheme is non-transferable when the only way for Bob (corrupted delegatee) to transfer offline decryption capabilities to Carol (malicious user) is to expose his own secret key.*" To our best knowledge, there is no formal definition of the non-transferability. One considerable definition of the non-transferability naturally derived from the above informal statement is as follows. Here, plaintext extractors as in the definition of plaintext awareness [3] are employed.

In the following definitions, keys subscripted by "$*$", h, j, and c_i are those for a target honest delegator, a honest user, a malicious user, and a corrupted delegatee, respectively, and $i \in \{1, \ldots, L\}$ where L is polynomially bounded.

Definition 9 (Non-Transferability, NT). *A unidirectional single-hop proxy re-encryption scheme meets* the non-transferability *if there exists a polynomial time algorithm $\mathcal{P}$ such that*

$$\begin{aligned}\Pr[(pk_*, sk_*) \leftarrow \mathsf{Keygen}(\lambda);\ (pk_h, sk_h) \leftarrow \mathsf{Keygen}(\lambda);\ \{(pk_{c_i}, sk_{c_i}) \leftarrow \mathsf{Keygen}(\lambda)\};\\ (pk_j, sk_j) \leftarrow \mathsf{Keygen}(\lambda);\ \{R_{*c_i} \leftarrow \mathsf{ReKeygen}(sk_*, pk_{c_i})\};\\ \{R_{hc_i} \leftarrow \mathsf{ReKeygen}(sk_h, pk_{c_i})\}; m \xleftarrow{R} \mathcal{M};\ C^* \leftarrow \mathsf{Enc}_2(m, pk_*);\\ \{m_i \xleftarrow{R} \mathcal{M}\};\ \{C_i \leftarrow \mathsf{Enc}_2(m_i, pk_{c_i})\};\ \{m'_i \xleftarrow{R} \mathcal{M}\};\ \{C'_i \leftarrow \mathsf{Enc}_1(m'_i, pk_{c_i})\};\\ \{m''_i \xleftarrow{R} \mathcal{M}\};\ \{C''_i \leftarrow \mathsf{ReEnc}(R_{hc_i}, \mathsf{Enc}_2(m''_i, pk_h))\};\\ X \leftarrow \mathcal{C}(pk_*, \{(pk_{c_i}, sk_{c_i})\}, \{R_{*c_i}\});\ m_{\mathcal{J}} \leftarrow \mathcal{J}(X, (pk_j, sk_j), C^*);\\ m_{\mathcal{P}} \leftarrow \mathcal{P}(X, (pk_j, sk_j), \{C_i\}, \{C'_i\}, \{C''_i\})\\ : m \neq m_{\mathcal{J}} \vee m_{\mathcal{P}} \in \{m_i\} \cup \{m'_i\} \cup \{m''_i\}]\end{aligned}$$

is overwhelming for any polynomial time algorithm $\mathcal{C}$, $\mathcal{J}$, and polynomial L.

Intuitively, this definition states that it is impossible for $\mathcal{C}$ (colluding proxies and delegatees) to re-delegate the decryption rights of the target honest delegator * to $\mathcal{J}$ (a malicious user) without delegating any right related to secret keys of any member in $\mathcal{C}$ to $\mathcal{P}$ (the malicious user).

In a PRE scheme, a user who has a secret key sk_{c_i} only has rights to decrypt second level ciphertexts $C_i = \mathsf{Enc}_2(m_i, pk_{c_i})$, decrypt first level ciphertexts $C'_i = \mathsf{Enc}_1(m'_i, pk_{c_i})$ and $C''_i = \mathsf{ReEnc}(R_{hc_i}, \mathsf{Enc}_2(m''_i, pk_h))$, and generate re-encryption keys to re-encrypt a ciphertext for the user with sk_i into a ciphertext for another user. The definition states that in a non-transferable PRE scheme, whenever the decryption right of the target honest delegator * is re-delegated to the malicious user from colluding proxies and delegatees, at least one of the above rights of any member in the colluding proxies and

delegatees is necessarily transferred to the malicious user at the same time. In the above definition, such transfer is modeled with $\mathcal{P}$ and detected by the event "$m_{\mathcal{P}} \in \{m_i\} \cup \{m'_i\} \cup \{m''_i\}$", that is, the plaintext extractor $\mathcal{P}$ decrypts any ciphertext $C \in \{C_i\} \cup \{C'_i\} \cup \{C''_i\}$ of first or second level ciphertexts for any member in $\mathcal{C}$ by using X. Note that we can omit to consider the rights to generate a re-encryption key, since if the malicious user can generate a re-encryption key $R_{c_i j}$, then the malicious user can also decrypt a second level ciphertext C_i.

Unfortunately, we have not succeeded in constructing the scheme which satisfies the above definition, and we consider that it is very difficult to construct such schemes since the required security level is quite high.

We next propose a security notion called the unforgeability of re-encryption keys against collusion attack, which is a relaxed notion of the non-transferability.

Definition 10 (Unforgeability of Re-Encryption Keys against Collusion Attack, UFReKey-CA). *A unidirectional single-hop proxy re-encryption scheme meets* the unforgeability of re-encryption keys against collusion attack *if there exists a polynomial time algorithm $\mathcal{P}$ such that*

$$\begin{aligned}\Pr[(pk_*, sk_*) \leftarrow \mathsf{Keygen}(\lambda);\ (pk_h, sk_h) \leftarrow \mathsf{Keygen}(\lambda);\ \{(pk_{c_i}, sk_{c_i}) \leftarrow \mathsf{Keygen}(\lambda)\};\\ (pk_j, sk_j) \leftarrow \mathsf{Keygen}(\lambda);\ \{R_{*c_i} \leftarrow \mathsf{ReKeygen}(sk_*, pk_{c_i})\};\\ \{R_{hc_i} \leftarrow \mathsf{ReKeygen}(sk_h, pk_{c_i})\};\ m \xleftarrow{R} \mathcal{M};\ C^* \leftarrow \mathsf{Enc}_2(m, pk_*);\\ \{m_i \xleftarrow{R} \mathcal{M}\};\ \{C_i \leftarrow \mathsf{Enc}_2(m_i, pk_{c_i})\};\ \{m'_i \xleftarrow{R} \mathcal{M}\};\ \{C'_i \leftarrow \mathsf{Enc}_1(m'_i, pk_{c_i})\};\\ \{m''_i \xleftarrow{R} \mathcal{M}\};\ \{C''_i \leftarrow \mathsf{ReEnc}(R_{hc_i}, \mathsf{Enc}_2(m''_i, pk_h))\};\\ X \leftarrow \mathcal{C}(pk_*, \{(pk_{c_i}, sk_{c_i})\}, \{R_{*c_i}\});\ R^{\dagger}_{*j} \leftarrow \mathcal{J}(X, (pk_j, sk_j));\\ m_{\mathcal{P}} \leftarrow \mathcal{P}(X, (pk_j, sk_j), \{C_i\}, \{C'_i\}, \{C''_i\})\\ : m \neq \mathsf{Dec}_1(\mathsf{ReEnc}(R^{\dagger}_{*j}, C^*), sk_j) \vee m_{\mathcal{P}} \in \{m_i\} \cup \{m'_i\} \cup \{m''_i\}]\end{aligned}$$

is overwhelming for any polynomial time algorithm $\mathcal{C}$, $\mathcal{J}$, and polynomial L.

Intuitively, this definition states that it is impossible for $\mathcal{C}$ (the colluding proxies and delegatees) to re-delegate the decryption rights of the target honest delegator * to $\mathcal{J}$ (a malicious user) *by giving the information to forge the re-encryption key for* $\mathcal{J}$ without delegating any right related to secret keys of any member in $\mathcal{C}$ to $\mathcal{P}$ (the malicious user). As compared with the non-transferability, the way to re-delegate the decryption rights is limited to the forgery of the re-encryption key in the definition of UFReKey-CA. Therefore, it is easy to see that UFReKey-CA is a relaxed notion of the non-transferability.

In the above definition, the adversary tries to return a forged re-encryption key $R^{\dagger}_{*j}$ such that

$$m = \mathsf{Dec}_1(\mathsf{ReEnc}(R^{\dagger}_{*j}, \mathsf{Enc}_2(m, pk_*)), sk_j) \tag{1}$$

where $m \xleftarrow{R} \mathcal{M}$. The adversary always wins if she returns the *well-formed* re-encryption key, which is one of the outputs of $\mathsf{ReKeygen}(sk_*, pk_j)$. On the other hand, the adversary does not have to output the well-formed re-encryption key to win the game. That is, the adversary also wins if she forges a re-encryption key which satisfies the equation (1) and it is an *ill-formed* re-encryption key, which is never returned from $\mathsf{ReKeygen}(sk_*, pk_j)$.

To the best of our knowledge, there exists no PRE scheme which satisfies UFReKey-CA and does not require either a trusted third party or communication between a delegator and a delegatee during re-encryption key generation. For example, in case of the scheme in [10], the colluding adversaries can generate the (well-formed) re-encryption key $R_{*j}^{\dagger}$ from $R_{*c} = g^{x_c/x_*}$, $sk_c = x_c$ and $sk_j = x_j$ by simply computing $R_{*j}^{\dagger} = (g^{x_j/x_*})^{x_c/x_j} = g^{x_c/x_*}$. Similarly, for other previously proposed schemes such as [5,12,6,13,16], the colluding adversaries, with secret keys sk_c and sk_j, can easily generate a forged re-encryption key R_{*j} by removing the ingredient(s) related to c from the re-encryption key R_{*c} and adding the ingredient(s) related to j to it.

In the definition of UFReKey-CA, there exists the adversary $\mathcal{P}$ which extracts the plaintext from the information X. Generally speaking, it is difficult (complicated) to prove the (non-)existence of the plaintext extractor $\mathcal{P}$. For convenience, we propose a simple and useful security notion to prove UFReKey-CA.

Definition 11 (Strong Unforgeability of Re-Encryption Keys against Collusion Attack, sUFReKey-CA). *A unidirectional single-hop proxy re-encryption scheme meets* the strong unforgeability of re-encryption keys against collusion attack *if*

$$\begin{aligned}\Pr[(pk_*, sk_*) \leftarrow \mathsf{Keygen}(\lambda);\ \{(pk_{c_i}, sk_{c_i}) \leftarrow \mathsf{Keygen}(\lambda)\};\ (pk_j, sk_j) \leftarrow \mathsf{Keygen}(\lambda);\\ \{R_{*c_i} \leftarrow \mathsf{ReKeygen}(sk_*, pk_{c_i})\};\ m \xleftarrow{R} \mathcal{M};\ C^* \leftarrow \mathsf{Enc}_2(m, pk_*);\\ R_{*j}^{\dagger} \leftarrow \mathcal{A}(pk_*, \{(pk_{c_i}, sk_{c_i})\}, (pk_j, sk_j), \{R_{*c_i}\}) :\ m = \mathsf{Dec}_1(\mathsf{ReEnc}(R_{*j}^{\dagger}, C^*), sk_j)]\end{aligned}$$

is negligible for any polynomial time algorithm $\mathcal{A}$, and polynomial L.

Intuitively, this definition states that it is impossible for $\mathcal{C}$ (colluding proxies and delegatees) to re-delegate the decryption rights of the target honest delegator * to a malicious user $\mathcal{A}$ by giving the forged re-encryption keys for the malicious user, where the secret key(s) of the colluding proxies and delegatees may be revealed to the malicious user, and the secret key of the malicious user may be revealed to to the colluding proxies and delegatees. It is easy to see that the scheme which satisfies sUFReKey-CA also meets UFReKey-CA. Since there exists no plaintext extractor in the definition of sUFReKey-CA, the proof of sUFReKey-CA is simpler than that of UFReKey-CA.

Note that we can consider several variations of UFReKey-CA by changing goals of $\mathcal{C}$. For example, it may be defined that $\mathcal{C}$ tries to generate a secret key sk_{c_i} itself, or generate a forged secret key $sk^{\dagger}$ such that $\mathsf{Dec}_1(sk^{\dagger}, C) = \mathsf{Dec}_1(sk_{c_i}, C)$. We also note that even when such kinds of $\mathcal{C}$ are defined in UFReKey-CA, sUFReKey-CA still implies UFReKey-CA and sUFReKey-CA is still useful to prove UFReKey-CA.

4 The Scheme

In this section, we propose a unidirectional PRE scheme which meets the RCCA security and sUFReKey-CA. Our scheme is based on the scheme in [10].

Before describing it, we review the strong one-time signature which we employ to construct our scheme. One-time signature $\mathsf{Sig} = (\mathcal{G}, \mathcal{S}, \mathcal{V})$ consists of a triple of algorithms. The algorithm $\mathcal{G}$ takes a security parameter λ and returns a pair of signing/verification keys (ssk, svk). Then, for any message M, $\mathcal{V}(\sigma, svk, M)$ returns 1 whenever $\sigma = \mathcal{S}(ssk, M)$ and 0 otherwise. We say that Sig is a strong one-time signature if no polynomial time adversary can create a new signature for a previously signed message (See [10] for the formal security definition.).

The differences between our scheme and the scheme in [10] are as follows. In our PRE scheme, two additional generators g_1, $g_2 = g^\beta$ are introduced and a plaintext is masked with $e(g_1 g_2, g)^r$ where r is a random number. The re-encryption key $R_{ij3} = g^{\frac{x_j + \beta y_j + \gamma}{z_i}}$ of our scheme contains *two secret keys* x_j, y_j of delegatee, a *secret system parameter* β, and a *random number* γ to avoid such forgery attacks as described in Section 3. In the first level decryption, y_j is used to extract $e(g_2, g)^r$ from the ciphertext generated by R_{ij3} and an additional re-encryption key $R_{ij1} = g^{\frac{x_j + \gamma}{x_i}}$. Similarly, x_j is used to extract $e(g_1, g)^r$ from the ciphertext generated by R_{ij3} and an additional re-encryption key $R_{ij2} = g^{\frac{\beta y_j + \gamma}{y_i}}$.

4.1 Description

$\mathsf{Global\text{-}setup}(\lambda)$: given a security parameter λ, choose bilinear map groups $(\mathbb{G}, \mathbb{G}_T)$ of prime order $p > 2^\lambda$, generators $g, g_1 (= g^\alpha), g_2 (= g^\beta), u, v \xleftarrow{R} \mathbb{G}$, and a one-time signature scheme $\mathsf{Sig} = (\mathcal{G}, \mathcal{S}, \mathcal{V})$. The global parameters are $\mathsf{par} := \{p, \mathbb{G}, \mathbb{G}_T, g, g_1, g_2, u, v, \mathsf{Sig}\}$. The message space $\mathcal{M}$ is equal to $\mathbb{G}_T$.

$\mathsf{Keygen}(\lambda, \mathsf{par})$: user i chooses $x_i, y_i, z_i \xleftarrow{R} \mathbb{Z}_p^*$. The secret key is $sk_i = (x_i, y_i, z_i)$. The public key is $pk_i = (X_i, Y_{1i}, Y_{2i}, Z_i, Z_{1i})$ where $X_i \leftarrow g^{x_i}$, $Y_{1i} \leftarrow {g_1}^{y_i}$, $Y_{2i} \leftarrow {g_2}^{y_i}$, $Z_i \leftarrow g^{z_i}$, $Z_{1i} \leftarrow {g_1}^{z_i}$.

$\mathsf{Enc}_1(m, pk_j, \mathsf{par})$: to encrypt a message $m \in \mathbb{G}_T$ under the public key pk_j at the first level, the sender proceeds as follows:

1. Select a one-time signature key pair $(svk, ssk) \leftarrow \mathcal{G}(\lambda)$ and set $C_1 = svk$.
2. Pick $r, s, t, k, \gamma \xleftarrow{R} \mathbb{Z}_p^*$ and compute,
   $C'_{2X} = {Y_{2j}}^s$, $C''_{2X} = {Y_{2j}}^{rs}$, $C'_{2Y} = {X_j}^t$, $C''_{2Y} = {X_j}^{rt}$, $C'_{2Z} = {Y_{2j}}^k$, $C''_{2Z} = {Y_{2j}}^{rk}$, $C'_{2Z1} = {X_j}^k$, $C''_{2Z1} = {X_j}^{rk}$, $C_3 = e(g_1 g_2, g)^r \cdot m$, $C_4 = (u^{svk} \cdot v)^r$, $C_{5X} = (g_1 \cdot g^\gamma)^{\frac{1}{s}}$, $C_{5Y} = g^{\frac{\gamma+1}{t}}$, $C_{5Z} = (g_1 \cdot g^{\gamma+1})^{\frac{1}{k}}$.
3. Generate a one-time signature $\sigma \leftarrow \mathcal{S}(ssk, (C_3, C_4))$ on (C_3, C_4).

The (first level) ciphertext is $C_j = (C_1, C'_{2X}, C''_{2X}, C'_{2Y}, C''_{2Y}, C'_{2Z}, C''_{2Z}, C'_{2Z1}, C''_{2Z1}, C_3, C_4, C_{5X}, C_{5Y}, C_{5Z}, \sigma)$.

$\mathsf{Enc}_2(m, pk_i, \mathsf{par})$: to encrypt a message $m \in \mathbb{G}_T$ under the public key pk_i at the second level, the sender proceeds as follows:

1. Select a one-time signature key pair $(svk, ssk) \leftarrow \mathcal{G}(\lambda)$ and set $C_1 = svk$.
2. Pick $r \xleftarrow{R} \mathbb{Z}_p^*$ and compute,
   $C_{2X} = {X_i}^r$, $C_{2Y} = {Y_{1i}}^r$, $C_{2Z} = {Z_i}^r$, $C_{2Z1} = {Z_{1i}}^r$, $C_3 = e(g_1 g_2, g)^r \cdot m$, $C_4 = (u^{svk} \cdot v)^r$.
3. Generate a one-time signature $\sigma \leftarrow \mathcal{S}(ssk, (C_3, C_4))$ on (C_3, C_4).

The (second level) ciphertext is $C_i = (C_1, C_{2X}, C_{2Y}, C_{2Z}, C_{2Z1}, C_3, C_4, \sigma)$.

$\mathsf{ReKeygen}(sk_i, pk_j, \mathsf{par})$: given user i's secret key sk_i and user j's public key pk_j, generate the re-encryption key $R_{ij} = (R_{ij1}, R_{ij2}, R_{ij3})$ where $\gamma \stackrel{R}{\leftarrow} \mathbb{Z}_p^*$ and
$R_{ij1} = (X_j \cdot g^{\gamma})^{1/x_i} = g^{\frac{x_j+\gamma}{x_i}}$, $R_{ij2} = (Y_{2j} \cdot g^{\gamma})^{1/y_i} = g^{\frac{\beta y_j+\gamma}{y_i}}$,
$R_{ij3} = (X_j \cdot Y_{2j} \cdot g^{\gamma})^{1/z_i} = g^{\frac{x_j+\beta y_j+\gamma}{z_i}}$.

$\mathsf{ReEnc}(R_{ij}, C_i, \mathsf{par})$: on input of the re-encryption key R_{ij} and a second level ciphertext C_i, check the validity of the ciphertext by testing:

$$\begin{aligned} &e(C_{2X}, u^{C_1} \cdot v) = e(X_i, C_4),\ e(C_{2Y}, u^{C_1} \cdot v) = e(Y_{1i}, C_4), \\ &e(C_{2Z}, u^{C_1} \cdot v) = e(Z_i, C_4),\ e(C_{2Z1}, u^{C_1} \cdot v) = e(Z_{1i}, C_4), \\ &\mathcal{V}(C_1, \sigma, (C_3, C_4)) = 1. \end{aligned} \tag{2}$$

If the relations (2) hold (well-formed), C_i is re-encrypted by choosing $s, t, k \stackrel{R}{\leftarrow} \mathbb{Z}_p^*$ and computing
$C'_{2X} = {X_i}^s$, $C''_{2X} = {C_{2X}}^s = {X_i}^{rs}$, $C'_{2Y} = {Y_{1i}}^t$, $C''_{2Y} = {C_{2Y}}^t = {Y_{1i}}^{rt}$,
$C'_{2Z} = {Z_i}^k$, $C''_{2Z} = {C_{2Z}}^k = {Z_i}^{rk}$, $C'_{2Z1} = {Z_{1i}}^k$, $C''_{2Z1} = {C_{2Z1}}^k = {Z_{1i}}^{rk}$,
$C_{5X} = {R_{ij1}}^{\frac{1}{s}}$, $C_{5Y} = {R_{ij2}}^{\frac{1}{t}}$, $C_{5Z} = {R_{ij3}}^{\frac{1}{k}}$,
and a re-encrypted ciphertext $C_j = (C_1, C'_{2X}, C''_{2X}, C'_{2Y}, C''_{2Y}, C'_{2Z}, C''_{2Z}, C'_{2Z1}, C''_{2Z1}, C_3, C_4, C_{5X}, C_{5Y}, C_{5Z}, \sigma)$ is returned. Otherwise, '`invalid`' is returned.

$\mathsf{Dec}_1(C_j, sk_j)$: the validity of the first level ciphertext C_j is checked by testing:

$$\begin{aligned} &e(C''_{2X}, u^{C_1} \cdot v) = e(C'_{2X}, C_4),\ e(C''_{2Y}, u^{C_1} \cdot v) = e(C'_{2Y}, C_4), \\ &e(C''_{2Z}, u^{C_1} \cdot v) = e(C'_{2Z}, C_4),\ e(C''_{2Z1}, u^{C_1} \cdot v) = e(C'_{2Z1}, C_4), \\ &e(C_{5Z}, C'_{2Z}) = e(C_{5X}, C'_{2X}) \cdot e(Y_{2j}, g), \\ &e(C_{5Z}, C'_{2Z1}) = e(C_{5Y}, C'_{2Y}) \cdot e(X_j, g_1), \\ &\mathcal{V}(C_1, \sigma, (C_3, C_4)) = 1. \end{aligned} \tag{3}$$

If the relations (3) hold (well-formed), the plaintext
$m = C_3 \Big/ \left\{ \left(\frac{e(C_{5Z}, C''_{2Z})}{e(C_{5X}, C'_{2X})} \right)^{\frac{1}{y_j}} \cdot \left(\frac{e(C_{5Z}, C''_{2Z1})}{e(C_{5Y}, C''_{2Y})} \right)^{\frac{1}{x_j}} \right\}$
is returned. Otherwise (ill-formed), the algorithm outputs '`invalid`.'

$\mathsf{Dec}_2(C_i, sk_i)$: if the second level ciphertext C_i satisfies the relations (2), the plaintext $m = C_3 \,/\, e(g_1 g_2, C_{2X})^{\frac{1}{x_i}}$ is returned. Otherwise, '`invalid`' is returned.

4.2 Security

Theorem 1. *Our proposed scheme with the strong one-time signature satisfies second level RCCA security if the 3-wDBDHI problem is hard.*

Proof (Sketch). Our scheme is based on the scheme by Libert and Vergnaud [10], and the proof of the above theorem is almost the same as that for the Libert–Vergnaud scheme. We use the following lemma in [10].

Lemma 1. *The 3-wDBDHI problem is equivalent to decide whether T equals $e(g,g)^{b/a^2}$ or a random value given $(g, g^{1/a}, g^a, g^{a^2}, g^b, T)$ as input.*

We prove that our proposed scheme is second level RCCA secure under the above (modified 3-wDBDHI) problem. We build an algorithm $\mathcal{B}$ which is, given $(g, A_{-1} = g^{1/a}, A_1 = g^a, A_2 = g^{a^2}, B = g^b, T)$, solving the modified 3-wDBDHI problem using second level RCCA adversary $\mathcal{A}$.

The algorithm $\mathcal{B}$ simulates $\mathcal{A}$'s input and oracles as follows. The generator are chosen $g_1 \leftarrow g^\alpha$ and $g_2 \leftarrow g^\beta$ where $\alpha, \beta \xleftarrow{R} \mathbb{Z}_p^*$. The key pair $(svk^*, ssk^*) \xleftarrow{R} \mathcal{G}(1^\lambda)$ of the one-time signature scheme is chosen, and the generator u and v are set as $A_1^{\alpha_1}$ and $A_1^{-\alpha_1 svk^*} A_2^{\alpha_2}$, respectively $(\alpha_1, \alpha_2 \xleftarrow{R} \mathbb{Z}_p^*)$.

The public key of the target user is set as $({A_2}^{x_*}, {A_2}^{\alpha y_*}, {A_2}^{\beta y_*}, {A_2}^{z_*}, {A_2}^{\alpha z_*}) = ({g^{a^2}}^{x_*}, {g_1}^{a^2 y_*}, {g_2}^{a^2 y_*}, {g^{a^2}}^{z_*}, {g_1}^{a^2 z_*})$, that of the honest user h is set as $({A_1}^{x_h}, {A_1}^{\alpha y_h}, {A_1}^{\beta y_h}, {A_1}^{z_h}, {A_1}^{\alpha z_h}) = ({g^a}^{x_h}, {g_1}^{a y_h}, {g_2}^{a y_h}, {g^a}^{z_h}, {g_1}^{a z_h})$, and that of the corrupted user c is set as $({g^{x_c}}, {g_1}^{y_c}, {g_2}^{y_c}, {g^{z_c}}, {g_1}^{z_c})$, where $x_*, y_*, z_*, x_h, y_h, z_h, x_c, y_c, z_c \xleftarrow{R} \mathbb{Z}_p^*$ ($\mathcal{B}$ does not have to compute the secret keys of the honest users.).

The re-encryption keys is computed as $R_{h*} = ({A_1}^{\frac{x_*+\gamma'}{x_h}}, {A_1}^{\frac{\beta y_*+\gamma'}{y_h}}, {A_1}^{\frac{x_*+\beta y_*+\gamma'}{z_h}})$, $R_{*h} = ({A_{-1}}^{\frac{x_h+\gamma''}{x_*}}, {A_{-1}}^{\frac{\beta y_h+\gamma''}{y_*}}, {A_{-1}}^{\frac{x_h+\beta y_h+\gamma''}{z_*}})$, $R_{hc} = ({A_{-1}}^{\frac{x_c+\gamma'''}{x_h}}, {A_{-1}}^{\frac{\beta y_c+\gamma'''}{y_h}}, {A_{-1}}^{\frac{x_c+\beta y_c+\gamma'''}{z_h}})$, and $R_{hh'} = ({g}^{\frac{x_{h'}+\gamma}{x_h}}, {g}^{\frac{\beta y_{h'}+\gamma}{y_h}}, {g}^{\frac{x_{h'}+\beta y_{h'}+\gamma}{z_h}})$, where $\gamma', \gamma'', \gamma''', \gamma \xleftarrow{R} \mathbb{Z}_p^*$.

For the re-encryption query (pk_i, pk_j, C_i), $\mathcal{B}$ checks the validity of C_i by using the equations (2). If C_i is ill-formed, $\mathcal{B}$ outputs `invalid`. Otherwise, if i is not the target user or j is not the corrupted user, $\mathcal{B}$ uses the re-encryption key and responds the query. If i is the target user and j is the corrupted user, $C_1 \neq svk^*$ holds with overwhelming probability (because of the strong unforgeability of the one-time signature). Then, the re-encrypted ciphertext C_j can be computed as $(C_1, {A_1}^s, ({A_1}^r)^s, {A_1}^{\alpha t}, ({A_1}^r)^{\alpha t}, {A_1}^k, ({A_1}^r)^k, {A_1}^{\alpha k}, ({A_1}^r)^{\alpha k}, {A_{-1}}^{\frac{x_j+\gamma}{t}}, {A_{-1}}^{\frac{\beta y_j+\gamma}{s}}, {A_{-1}}^{\frac{x_j+\beta y_j+\gamma}{k}}, \sigma)$ and ${A_1}^r = (C_4 / C_{2X}^{\alpha_2/x_*})^{\frac{1}{\alpha_1(C_1 - svk^*)}}$ where $s', t', k', \gamma \xleftarrow{R} \mathbb{Z}_p^*$. This is a valid ciphertext with the randomness $s = s'/ax_*$, $t = t'/ay_*$, $k = k'/kz_*$.

For the first level decryption query (pk_j, C_j), $\mathcal{B}$ checks the validity of C_j by using the equations (3). If C_j is ill-formed, $\mathcal{B}$ outputs `invalid`. When C_j is well-formed, if j is the corrupted user, $\mathcal{B}$ uses the secret key and responds the query. When j is an honest user, if $C_1 = svk^*$ and $(C_3, C_4, \sigma) = (C_3^*, C_4^*, \sigma^*)$, $\mathcal{B}$ returns $\perp$ since C_j is a Derivative of the challenge ciphertext. In the other case, $C_1 \neq svk^*$ holds with overwhelming probability (because of the strong unforgeability of the one-time signature). Then, we consider the following two cases. If j is not the target user, $\mathcal{B}$ computes $X = \{e(C_4, A_{-1}) / (\frac{e(C_{5Z}, C''_{2Z})}{e(C_{5X}, C''_{2X})})^{\frac{\alpha_2}{y_j \beta}}\}^{\frac{\alpha+\beta}{\alpha_1(C_1 - svk^*)}} (= e(g_1 g_2, g)^r)$ and $m = C_3 / X$. If j is the target user, $\mathcal{B}$ computes $Y = \{e(C_4, g) / (\frac{e(C_{5Z}, C''_{2Z})}{e(C_{5X}, C''_{2X})})^{\frac{\alpha_2}{y_j \beta}}\}^{\frac{1}{\alpha_1(C_1 - svk^*)}} (= e(g,g)^{ar})$, $Z = (e(C_4, A_{-1}) / Y^{\alpha_2})^{\frac{\alpha+\beta}{\alpha_1(C_1 - svk^*)}} (= e(g_1 g_2, g)^r)$ and $m = C_3 / Z$. Note that if $m \in \{m_0, m_1\}$, $\mathcal{B}$ returns $\perp$.

The challenge ciphertext is computed as $(svk^*, B^{x_*}, B^{\alpha y_*}, B^{z_*}, B^{\alpha z_*}, m_{d^*} \cdot T^{\alpha+\beta}, B^{\alpha_2}, \mathcal{S}(ssk^*, (C_3^*, C_4^*))$ where $d^* \stackrel{R}{\leftarrow} \{0,1\}$. If $T = e(g,g)^{b/a^2}$, C^* is a valid ciphertext with the random exponent $r = b/a^2$. In contrast, if T is random, $\mathcal{A}$ cannot guess d^* with probability better than 1/2. Therefore, $\mathcal{B}$ decides that $T = e(g,g)^{b/a^2}$ if d^* equals to the adversary's output and that T is random otherwise. □

Theorem 2. *Our proposed scheme with the strong one-time signature satisfies first level RCCA security if the 3-wDBDHI problem is hard.*

Proof (Sketch). The proof is almost the same as that of Theorem 1. We can build an algorithm $\mathcal{B}$ which is, given $(g, A_{-1} = g^{1/a}, A_1 = g^a, A_2 = g^{a^2}, B = g^b, T)$, solving the modified 3-wDBDHI problem using RCCA adversary $\mathcal{A}$ at level 1.

The algorithm $\mathcal{B}$ sets public parameters as in the proof of Theorem 1. The algorithm $\mathcal{B}$ generates a public key for target user as $({A_1}^{x_*}, {A_1}^{\alpha y_*}, {A_1}^{\beta y_*}, {A_1}^{z_*}, {A_1}^{\alpha z_*}) = (g^{ax_*}, {g_1}^{ay_*}, {g_2}^{ay_*}, g^{az_*}, {g_1}^{az_*})$ where $x_*, y_*, z_* \stackrel{R}{\leftarrow} \mathbb{Z}_p^*$, and for the other users as $(g^x, {g_1}^y, {g_2}^y, g^z, {g_1}^z)$ where $x, y, z \stackrel{R}{\leftarrow} \mathbb{Z}_p^*$. Then the re-encryption key can be computed as $R_{ij} = (g^{\frac{x_j+\gamma}{x_i}}, g^{\frac{\beta y_j+\gamma}{y_i}}, g^{\frac{x_j+\beta y_j+\gamma}{z_i}})$, $R_{*j} = (A_{-1}^{\frac{x_j+\gamma}{x_*}}, A_{-1}^{\frac{\beta y_j+\gamma}{y_*}}, A_{-1}^{\frac{x_j+\beta y_j+\gamma}{z_*}})$, $R_{i*} = (A_1^{\frac{x_*+\gamma}{x_i}}, A_1^{\frac{\beta y_*+\gamma}{y_i}}, A_1^{\frac{x_*+\beta y_*+\gamma}{z_i}})$ where i, j is not the target user. For the first level decryption query, $\mathcal{B}$ responds in the same way as that for the honest user's case in the proof of Theorem 1.

The challenge ciphertext is computed as $(svk^*, {A_2}^{\beta y_* s'}, B^{\beta y_* s'}, {A_2}^{x_* t'}, B^{x_* t'}, {A_2}^{\beta y_* k'}, B^{\beta y_* k'}, {A_2}^{x_* k'}, B^{x_* k'}, m_{d^*} \cdot T^{\alpha+\beta}, B^{\alpha_2}, A_{-1}^{\frac{\alpha+\gamma}{s'}}, A_{-1}^{\frac{1+\gamma}{t'}}, A_{-1}^{\frac{\alpha+1+\gamma}{k'}}, \sigma^*)$ where $\sigma^* \leftarrow \mathcal{S}(ssk^*, (C_3^*, C_4^*))$, $s', t', k', \gamma \stackrel{R}{\leftarrow} \mathbb{Z}_p^*$, and $d^* \stackrel{R}{\leftarrow} \{0,1\}$. If $T = e(g,g)^{b/a^2}$, C^* is a valid ciphertext with the random exponents $r = b/a^2$, $s = as'$, $t = at'$, $k = ak'$. In contrast, if T is random, $\mathcal{A}$ cannot guess d^* with probability better than 1/2. Therefore, $\mathcal{B}$ decides that $T = e(g,g)^{b/a^2}$ if d^* equals to the adversary's output and that T is random otherwise. □

Theorem 3. *Our proposed scheme meets sUFReKey-CA if the 2-DHIwRA problem is hard.*

Proof. We specify the polynomial time algorithm $\mathcal{B}$ which solves 2-DHIwRA problem by using the polynomial time algorithm $\mathcal{A}$ which breaks the strong unforgeability of re-encryption keys against collusion attack of the proposed scheme.

1. Given $g, A_1 = g^a, A_2 = g^{a^2}, c$ (input 1 of 2-DHIwRA problem), and $\{(x_i,\ y_i,\ D_i = g^{\frac{x_i+\gamma_i}{a(a+c)}},\ E_i = g^{\frac{ay_i+\gamma_i}{a+c}},\ F_i = g^{\frac{x_i+\gamma_i}{a}})\}$ (input 2 of 2-DHIwRA problem), $\mathcal{B}$ generates the input of $\mathcal{A}$ as follows:
   - par: Choose $\mu \stackrel{R}{\leftarrow} \mathbb{Z}_p^*$ and set $g_1(= g^\alpha) \leftarrow A_1$ and $g_2(= g^\beta) \leftarrow {A_1}^\mu$ (i.e. $\alpha = a$, $\beta = a\mu$). The generators u and v are randomly chosen from $\mathbb{G}$.
   - $pk_* = (X_*, Y_{1*}, Y_{2*}, Z_*, Z_{1*})$: Choose $x, y, z \stackrel{R}{\leftarrow} \mathbb{Z}_p^*$ and compute $X_* = ({A_1}^c \cdot A_2)^x$, $Y_{1*} = ({A_1}^c \cdot A_2)^y$, $Y_{2*} = ({A_1}^c \cdot A_2)^{y\mu}$, $Z_* = {A_1}^z$, $Z_{1*} =$

${A_2}^z$. Here, the corresponding secret key of the target honest user is $sk_* = (x_*, y_*, z_*)$ where $x_* = a(a+c)x, y_* = (a+c)y, z_* = az$. Note that $\mathcal{B}$ does not have to compute x_*, y_*, and z_*.

- (pk_j, sk_j): The secret key $sk_j = (x_j, y_j, z_j)$ of the malicious user is set as $x_j, z_j \xleftarrow{R} \mathbb{Z}_p^*$ and $y_j \leftarrow x_j/c$. The public key pk_j is computed by using the secret key sk_j and the public parameters g, g_1, g_2.
- $(pk_{c_i}, sk_{c_i}), R_{*c_i}$: Let $(x_i, y_i, D_i, E_i, F_i)$ be the i-th quintuple of input 2 of 2-DHIwRA problem. The secret key of i-th corrupted delegatee $sk_{c_i} = (x_{c_i}, y_{c_i}, z_{c_i})$ is set as $x_{c_i} \leftarrow x_i$, $y_{c_i} \leftarrow y_i/\mu$, and $z_{c_i} \xleftarrow{R} \mathbb{Z}_p^*$. The public key pk_{c_i} is computed by using the secret key sk_{c_i} and the public parameters g, g_1, g_2. The re-encryption key $R_{*c_i} = (R_{*c_i1}, R_{*c_i2}, R_{*c_i3})$ is computed as follows. Note that $\beta y_{c_i} = (a\mu)(y_i/\mu) = ay_i$.

$$R_{*c_i1} = {D_i}^{1/x} = g^{\frac{x_i+\gamma_i}{a(a+c)x}} = g^{\frac{x_{c_i}+\gamma_i}{x_*}},\ R_{*c_i2} = {E_i}^{1/y} = g^{\frac{ay_i+\gamma_i}{(a+c)y}} = g^{\frac{\beta y_{c_i}+\gamma_i}{y_*}},$$
$$R_{*c_i3} = (F_i \cdot g^{y_i})^{1/z} = g^{\frac{x_i+ay_i+\gamma_i}{az}} = g^{\frac{x_{c_i}+\beta y_{c_i}+\gamma_i}{z_*}}.$$

2. The algorithm $\mathcal{B}$ runs $\mathcal{A}$ with the input computed in step 1. Then, $\mathcal{A}$'s output $R_{*j}^{\dagger} = (R_1, R_2, R_3)$ is returned to $\mathcal{B}$.
3. $\mathcal{B}$ outputs $W = \left(\frac{{R_3}^z}{{R_1}^{cx} \cdot {R_2}^y}\right)^{\frac{1}{(1+\mu)x_j}}$ as the answer of 2-DHIwRA problem.

We show that the algorithm $\mathcal{B}$ outputs $g^{\frac{1}{a+c}}$ with non-negligible probability. We can easily see that the distributions of the public parameters and the public/secret/re-encryption keys generated in step 1 are identical to those of our proposed scheme. Therefore, the algorithm $\mathcal{A}$ outputs a (forged) re-encryption key $R_{*j}^{\dagger} = (R_1, R_2, R_3)$ which satisfies the equation (1) with non-negligible probability. From the equation (1) and the encryption/decryption/re-encryption algorithms of our proposed scheme, we have

$$m = m \cdot e(g_1 g_2, g)^r \Big/ \left\{ \left(\frac{e({R_3}^{1/k}, g^{z_* rk})}{e({R_1}^{1/s}, g^{x_* rs})}\right)^{\frac{1}{y_j}} \cdot \left(\frac{e({R_3}^{1/k}, g^{\alpha z_* rk})}{e({R_2}^{1/t}, g^{\alpha y_* rt})}\right)^{\frac{1}{x_j}} \right\}$$
$$\Leftrightarrow m \cdot e\left(\frac{{R_3}^{z_*(x_j+\alpha y_j)}}{{R_1}^{x_* x_j} \cdot {R_2}^{\alpha y_* y_j}}, g\right)^r = m \cdot e((g_1 g_2)^{x_j y_j}, g)^r.$$

Therefore,

$$\frac{{R_3}^{z_*(x_j+\alpha y_j)}}{{R_1}^{x_* x_j} \cdot {R_2}^{\alpha y_* y_j}} = (g_1 g_2)^{x_j y_j}. \tag{4}$$

In step 1, we set $x_* = a(a+c)x$, $y_* = (a+c)y$, $z_* = az$, and $x_j = cy_j$. We also set $g_1 = g^a$ and $g_2 = g^{a\mu}$. Together with the equation (4), we have

$$\frac{{R_3}^{az(cy_j+ay_j)}}{{R_1}^{a(a+c)xcy_j} \cdot {R_2}^{a(a+c)yy_j}} = (g^a g^{a\mu})^{x_j y_j} \Leftrightarrow (W =) \left(\frac{{R_3}^z}{{R_1}^{cx} \cdot {R_2}^y}\right)^{\frac{1}{(1+\mu)x_j}} = g^{\frac{1}{a+c}}.$$

Hence, the algorithm $\mathcal{B}$ outputs $g^{\frac{1}{a+c}}$ with non-negligible probability.□

5 The Scheme with Temporary Delegation

In this section, we apply similar modification of the re-encryption keys to the PRE scheme with temporary delegation in [10] and propose the PRE scheme supporting temporary delegation which meets sUFReKey-CA.

5.1 Description

$\mathsf{Global\text{-}setup}(\lambda)$ is the same as that in Section 4, and the others are as follows:

$\mathsf{Keygen}(\lambda, \mathsf{par})$: user i chooses $x_i, y_i, z_i, w_i \xleftarrow{R} \mathbb{Z}_p^*$. The secret key is $sk_i = (x_i, y_i, z_i, w_i)$. The public key is $pk_i = (X_i, Y_{1i}, Y_{2i}, Z_i, Z_{1i}, W_i)$ where $X_i \leftarrow g^{x_i}, Y_{1i} \leftarrow {g_1}^{y_i}, Y_{2i} \leftarrow {g_2}^{y_i}, Z_i \leftarrow g^{z_i}, Z_{1i} \leftarrow {g_1}^{z_i}, W_i \leftarrow g^{w_i}$. A function $F_i : \{1, \ldots, L\} \to \mathbb{G}$ is implicitly defined as $F_i(\ell) = g^{\ell} \cdot W_i = g^{\ell + w_i}$.

$\mathsf{Enc}_1(\ell, m, pk_j, \mathsf{par})$: to encrypt a message $m \in \mathbb{G}_T$ under the public key pk_j at the first level during period ℓ, the sender proceeds as follows:

1. Select a one-time signature key pair $(svk, ssk) \leftarrow \mathcal{G}(\lambda)$ and set $C_1 = svk$.
2. Pick $r, s, t, k, h, \gamma, \delta_x, \delta_y \xleftarrow{R} \mathbb{Z}_p^*$ and compute,
   $C'_{2X} = {Y_{2j}}^{s}$, $C''_{2X} = {Y_{2j}}^{rs}$, $C'_{2Y} = {X_j}^{t}$, $C''_{2Y} = {X_j}^{rt}$,
   $C'_{2Z} = {Y_{2j}}^{k}$, $C''_{2Z} = {Y_{2j}}^{rk}$, $C'_{2Z1} = {X_j}^{k}$, $C''_{2Z1} = {X_j}^{rk}$,
   $C'_{2F} = {Y_{2j}}^{k}$, $C''_{2F} = {Y_{2j}}^{rk}$, $C_3 = e(g_1 g_2, g)^r \cdot m$, $C_4 = (u^{svk} \cdot v)^r$,
   $C_{5X} = (g_1 \cdot g^{\gamma} \cdot F_j(\ell)^{\delta_y})^{\frac{1}{s}} = g^{\frac{\alpha + \gamma + (\ell + w_j)\delta_y}{s}}$,
   $C_{5Y} = (g^{1+\gamma} \cdot F_j(\ell)^{\delta_x})^{\frac{1}{t}} = g^{\frac{1+\gamma+(\ell+w_j)\delta_x}{t}}$, $C_{5Z} = (g_1 \cdot g^{1+\gamma})^{\frac{1}{k}} = g^{\frac{\alpha+1+\gamma}{k}}$,
   $C_{5FX} = (Y_{2j})^{\frac{\delta_y}{h}}$, $C_{5FY} = (X_j)^{\frac{\delta_x}{h}}$.
3. Generate a one-time signature $\sigma \leftarrow \mathcal{S}(ssk, (\ell, C_3, C_4))$ on (ℓ, C_3, C_4).

The (first level) ciphertext is $C_j = (C_1, C'_{2X}, C''_{2X}, C'_{2Y}, C''_{2Y}, C'_{2Z}, C''_{2Z}, C'_{2Z1}, C''_{2Z1}, C_3, C_4, C_{5X}, C_{5Y}, C_{5Z}, C_{5FX}, C_{5FY}, \sigma)$.

$\mathsf{Enc}_2(\ell, m, pk_i, \mathsf{par})$: to encrypt a message $m \in \mathbb{G}_T$ under the public key pk_i at the second level during period ℓ, the sender proceeds as follows:

1. Select a one-time signature key pair $(svk, ssk) \leftarrow \mathcal{G}(\lambda)$ and set $C_1 = svk$.
2. Pick $r \xleftarrow{R} \mathbb{Z}_p^*$ and compute,
   $C_{2X} = {X_i}^{r}$, $C_{2Y} = {Y_{1i}}^{r}$, $C_{2Z} = {Z_i}^{r}$, $C_{2Z1} = {Z_{1i}}^{r}$, $C_{2F} = F_i(\ell)^r$,
   $C_3 = e(g_1 g_2, g)^r \cdot m$, $C_4 = (u^{svk} \cdot v)^r$.
3. Generate a one-time signature $\sigma \leftarrow \mathcal{S}(ssk, (\ell, C_3, C_4))$ on (ℓ, C_3, C_4).

The (second level) ciphertext is $C_i = (\ell, C_1, C_{2X}, C_{2Y}, C_{2Z}, C_{2Z1}, C_{2F}, C_3, C_4, \sigma)$.

$\mathsf{ReKeygen}(\ell, sk_i, pk_j, \mathsf{par})$: given a period number ℓ, user i's secret key sk_i and user j's public key pk_j, generate the re-encryption key $R_{ij\ell} = (R_{ij\ell 1}, R_{ij\ell 2}, R_{ij\ell 3}, R_{ij\ell 4}, R_{ij\ell 5})$ where $\gamma, \delta_x, \delta_y \xleftarrow{R} \mathbb{Z}_p^*$ and

$$R_{ij\ell 1} = (X_j \cdot g^{\gamma})^{1/x_i} \cdot F_i(\ell)^{\delta_x} = g^{\frac{x_j + \gamma}{x_i} + (\ell + w_i)\delta_x},$$
$$R_{ij\ell 2} = (Y_{2j} \cdot g^{\gamma})^{1/y_i} \cdot F_i(\ell)^{\delta_y} = g^{\frac{\beta y_j + \gamma}{y_i} + (\ell + w_i)\delta_y},$$
$$R_{ij\ell 3} = (X_j \cdot Y_{2j} \cdot g^{\gamma})^{1/z_i} = g^{\frac{x_j + \beta y_j + \gamma}{z_i}}, \quad R_{ij\ell 4} = {X_i}^{\delta_x}, \quad R_{ij\ell 5} = {Y_{1i}}^{\delta_y}.$$

$\mathsf{ReEnc}(\ell, R_{ij\ell}, C_i, \mathsf{par})$: on input of the re-encryption key $R_{ij\ell}$ for period ℓ and a second level ciphertext C_i, check the validity of the ciphertext by testing:

$$\begin{aligned} &e(C_{2X}, u^{C_1} \cdot v) = e(X_i, C_4),\ e(C_{2Y}, u^{C_1} \cdot v) = e(Y_{1i}, C_4), \\ &e(C_{2Z}, u^{C_1} \cdot v) = e(Z_i, C_4),\ e(C_{2Z1}, u^{C_1} \cdot v) = e(Z_{1i}, C_4), \\ &e(C_{2F}, u^{C_1} \cdot v) = e(F_i(\ell), C_4),\ \mathcal{V}(C_1, \sigma, (\ell, C_3, C_4)) = 1. \end{aligned} \tag{5}$$

If the relations (5) hold (well-formed), C_i is re-encrypted by computing $C'_{2X} = {X_i}^s$, $C''_{2X} = {C_{2X}}^s = {X_i}^{rs}$, $C'_{2Y} = {Y_{1i}}^t$, $C''_{2Y} = {C_{2Y}}^t = {Y_{1i}}^{rt}$, $C'_{2Z} = {Z_i}^k, C''_{2Z} = {C_{2Z}}^k = {Z_i}^{rk}$, $C'_{2Z1} = {Z_{1i}}^k$, $C''_{2Z1} = {C_{2Z1}}^k = {Z_{1i}}^{rk}$, $C'_{2F} = F_i(\ell)^h, C''_{2F} = {C_{2F}}^h = F_i(\ell)^{rh}$, $C_{5X} = {R_{ij1}}^{\frac{1}{s}}$, $C_{5Y} = {R_{ij2}}^{\frac{1}{t}}$, $C_{5Z} = {R_{ij3}}^{\frac{1}{k}}$, $C_{5FX} = {R_{ij4}}^{\frac{1}{h}}, C_{5FY} = {R_{ij5}}^{\frac{1}{h}}$ where $s, t, k, h \xleftarrow{R} \mathbb{Z}_p^*$, and re-encrypted ciphertext $C_j = (\ell, C_1, C'_{2X}, C''_{2X}, C'_{2Y}, C''_{2Y}, C'_{2Z}, C''_{2Z}, C'_{2Z1}, C''_{2Z1}, C'_{2F}, C''_{2F}, C_3, C_4, C_{5X}, C_{5Y}, C_{5Z}, C_{5FX}, C_{5FY}, \sigma)$ is returned. Otherwise (ill-formed), the algorithm outputs '`invalid`.'

$\mathsf{Dec}_1(C_j, sk_j)$: the validity of the first level ciphertext C_j is checked by testing:

$$\begin{array}{l} e(C''_{2X}, u^{C_1} \cdot v) = e(C'_{2X}, C_4),\ e(C''_{2Y}, u^{C_1} \cdot v) = e(C'_{2Y}, C_4), \\ e(C''_{2Z}, u^{C_1} \cdot v) = e(C'_{2Z}, C_4),\ e(C''_{2Z1}, u^{C_1} \cdot v) = e(C'_{2Z1}, C_4), \\ e(C''_{2F}, u^{C_1} \cdot v) = e(C'_{2F}, C_4),\ \mathcal{V}(C_1, \sigma, (\ell, C_3, C_4)) = 1, \\ e(C_{5Z}, C'_{2Z}) \cdot e(C_{5FX}, C'_{2F}),\ = e(C_{5X}, C'_{2X}) \cdot e(Y_{2j}, g), \\ e(C_{5Z}, C'_{2Z1}) \cdot e(C_{5FY}, C'_{2F}) = e(C_{5Y}, C'_{2Y}) \cdot e(X_j, g_1). \end{array} \tag{6}$$

If the relations (6) hold (well-formed), the plaintext $m = C_3 \Big/ \left\{ \left(\frac{e(C_{5Z}, C''_{2Z}) \cdot e(C_{5FX}, C''_{2F})}{e(C_{5X}, C''_{2X})} \right)^{\frac{1}{y_j}} \cdot \left(\frac{e(C_{5Z}, C''_{2Z1}) \cdot e(C_{5FY}, C''_{2F})}{e(C_{5Y}, C''_{2Y})} \right)^{\frac{1}{x_j}} \right\}$ is returned. Otherwise (ill-formed), the algorithm outputs '`invalid`.'

$\mathsf{Dec}_2(C_i, sk_i)$: if the second level ciphertext C_i satisfies the relations (5) (well-formed), the plaintext $m = C_3 \,/\, e(g_1 g_2, C_{2X})^{\frac{1}{x_i}}$ is returned. Otherwise (ill-formed), the algorithm outputs '`invalid`.'

5.2 Security

We can prove the following theorem with respect to the confidentiality (RCCA-CK security) of our scheme.

Theorem 4. *Assuming the strong unforgeability of one-time signature, our proposed scheme with temporary delegation satisfies first level RCCA-CK security and second level RCCA-CK security if the 1-wDBDHI problem is hard.*

Our scheme supporting temporary delegation is based on the scheme in Section 4 and the scheme supporting temporary delegation by Libert and Vergnaud [10]. The proof of the above theorem is similar to that of our proposed scheme and the Libert–Vergnaud scheme with temporary delegation. Due to lack of space, we omit the the proof of the above theorem.

We show the strong unforgeability of re-encryption keys against collusion attack (sUFReKey-CA) for our proposed scheme with temporary delegation. The definition of sUFReKey-CA for the scheme with temporary delegation is defined as follows: The adversary is given the same public/secret keys as those in Definition 11, the target time period $\ell^* \xleftarrow{R} \{1, ..., L\}$ where L is polynomially bounded, re-encryption keys $R_{*c\ell}$ for any corrupted delegatee $c(\neq j)$ at any

period $1 \leq \ell \leq L$, and re-encryption keys $R_{*j\ell}$ for the malicious user j at period $\ell \neq \ell^*$. Then the adversary tries to compute $R^{\dagger}_{*j\ell^*}$ such that

$$m = \mathsf{Dec}_1(\mathsf{ReEnc}(\ell^*, R^{\dagger}_{*j\ell^*}, \mathsf{Enc}_2(\ell^*, m, pk_*)), sk_j). \tag{7}$$

Theorem 5. *Our proposed scheme with temporary delegation meets sUFReKey-CA if the m-2-DHIwRA problem is hard.*

Proof (Sketch). The algorithm $\mathcal{B}$ for solving the m-2-DHIwRA problem simulates an input of the algorithm $\mathcal{A}$ attacking sUFReKey-CA of our scheme as follows.

For par and pk_*, $\mathcal{B}$ sets $g_2 \xleftarrow{R} {A_1}^{\mu}$ where μ is an element of input 3 of m-2-DHIwRA problem. $\mathcal{B}$ also sets $W_* \leftarrow g^{-\ell^*} \cdot ({A_1}^c \cdot A_2)^w = g^{-\ell^* + a(a+c)w}$ where ℓ^* is the target time period and $w \xleftarrow{R} \mathbb{Z}_p^*$. The other components are computed as in the proof of Theorem 3. For the corrupted delegatee's key, by using the i-th input 2 of the m-2-DHIwRA problem, the secret key sk_{c_i} is set as $x_{c_i} \leftarrow x_i$, $y_{c_i} \leftarrow y_i/\mu$, and $z_{c_i}, w_{c_i} \xleftarrow{R} \mathbb{Z}_p^*$. Then, $R_{*c_i\ell}$ can be computed as $((D_i \cdot g^{\gamma''})^{1/x} \cdot F_*(\ell)^{\delta_x}, (E_i \cdot {A_1}^{\gamma''})^{1/y} \cdot F_*(\ell)^{\delta_y}, F_i \cdot g^{y_i} \cdot (A_1 \cdot g^c)^{\gamma''})^{1/z}, {X_*}^{\delta_x}, {Y_1}_*{}^{\delta_y})$ where $\delta_x, \delta_y, \gamma'' \xleftarrow{R} \mathbb{Z}_p$. $R_{*c_i\ell}$ has a proper shape with the randomness $\gamma = \gamma_i + a(a+c)\gamma''$. For the malicious user's key, by using the input 3 of the m-2-DHIwRA problem, the secret key sk_j is set as $y_j \leftarrow y', x_j \leftarrow cy', z_j, w_j \xleftarrow{R} \mathbb{Z}_p^*$. Then, $R_{*j\ell}$ for $\ell \neq \ell^*$ can be computed as $((D' \cdot g^{\gamma''})^{1/x} \cdot g^{(\ell-\ell^*)\delta''_x} \cdot {G'}^{\frac{w}{(\ell-\ell^*)x}} \cdot (A_1^c \cdot A_2)^{w\delta''_x}, (E' \cdot {A_1}^{\gamma''})^{1/y} \cdot g^{(\ell-\ell^*)\delta''_y} \cdot {H'}^{\frac{w}{(\ell-\ell^*)y}} \cdot (A_1^c \cdot A_2)^{w\delta''_y}, (F' \cdot g^{\mu y'} \cdot (A_1 \cdot g^c)^{\gamma''})^{1/z}, {G'}^{\frac{1}{(\ell-\ell^*)}} \cdot (A_1^c \cdot A_2)^{x\delta''_x}, {H'}^{\frac{1}{(\ell-\ell^*)}} \cdot (A_1^c \cdot A_2)^{y\delta''_y})$ where $\delta''_x, \delta''_y, \gamma'' \xleftarrow{R} \mathbb{Z}_p$. $R_{*c_i\ell}$ has a proper shape with the randomness $\delta_x = \frac{\delta'_x}{(\ell-\ell^*)x} + \delta''_x$, $\delta_y = \frac{\delta'_y}{(\ell-\ell^*)x} + \delta''_y$, and $\gamma = \gamma' + a(a+c)\gamma''$.

Finally, $\mathcal{B}$ outputs $W = \left(\frac{{R_3}^z {R_4}^{cw} {R_5}^w}{{R_1}^{cx} \cdot {R_2}^y}\right)^{\frac{1}{(1+\mu)x_j}}$ where $R^{\dagger}_{*j\ell^*} = (R_1, R_2, R_3, R_4, R_5)$ is an output of $\mathcal{A}$. We can easily see that W is equal to $g^{1/a+c}$ with non-negligible probability since $R^{\dagger}_{*j\ell^*}$ satisfies the equation (7). □

6 Concluding Remarks

We have introduced the notion of (strong) *unforgeability of re-encryption keys* against collusion attack, which is a relaxed notion (necessary condition) of the non-transferability, and proposed two concrete constructions that satisfy the RCCA security and the strong unforgeability of re-encryption keys.

Unfortunately, we have not succeeded in constructing the scheme that meets the non-transferability. With respect to the non-transferability of our proposed scheme, we are aware of the following attack. The colluding proxies and delegatees, given a re-encryption key $(R_{*c1}, R_{*c2}, R_{*c3})$ and a secret key (x_c, y_c, z_c), computes $X = (R_1, R_2, R_{3X}, R_{3Y}) = (R_{*c1}^{1/y_c}, R_{*c2}^{1/x_c}, R_{*c3}^{1/y_c}, R_{*c3}^{1/x_c})$, and passes it to a malicious user. Then, the malicious user computes $C''_{2X}, C''_{2Y}, C''_{2Z}, C''_{2Z1}$ from the (challenge) second level ciphertext C^* according to the re-encryption algorithm with the randomnesses s, t, k. The malicious user may not be able to

compute C_{5X}, C_{5Y}, C_{5Z} since she does not have R_{*j}. However, the malicious user can compute $R'_1 = {R_1}^{y_j/s}$, $R'_2 = {R_2}^{x_j/t}$, $R'_{3X} = {R_{3X}}^{y_j/k}$, $R'_{3Y} = {R_{3Y}}^{x_j/k}$. Then the malicious user can extract the plaintext m of C^* as

$$m = C_3 \Big/ \left\{ \left(\frac{e(R'_{3X}, C''_{2Z})}{e(R'_1, C''_{2X})} \right)^{\frac{1}{y_j}} \cdot \left(\frac{e(R'_{3Y}, C''_{2Z1})}{e(R'_2, C''_{2Y})} \right)^{\frac{1}{x_j}} \right\}$$

and it seems to be hard to extract x_c, y_c from X.

Our future work is to construct a non-transferable PRE scheme. It is also our future work to show the hardness of the (modified) 2-DHIwRA problem.

Acknowledgment. The authors would like to thank the anonymous referees for their valuable and helpful comments and suggestions.

References

1. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage. In: NDSS 2005 (2005)
2. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage. ACM Transactions on Information and System Security (TISSEC) 9(1), 1–30 (2006)
3. 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)
4. 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)
5. Canetti, R., Hohenberger, S.: Chosen-Ciphertext Secure Proxy Re-Encryption. In: ACM CCS 2007, pp. 185–194 (2007)
6. Chow, S.S.M., Weng, J., Yang, Y., Deng, R.H.: Efficient Unidirectional Proxy Re-Encryption. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 316–332. Springer, Heidelberg (2010)
7. Dodis, Y., Yampolskiy, A.: A Verifiable Random Function With Short Proofs and Keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Heidelberg (2005)
8. He, Y.J., Chim, T.W., Hui, L.C.K., Yiu, S.M.: Non-Transferable Proxy Re-Encryption, Cryptology ePrint archive (2010), http://eprint.iacr.org/2010/192
9. Libert, B., Vergnaud, D.: Tracing Malicious Proxies in Proxy Re-encryption. In: Galbraith, S.D., Paterson, K.G. (eds.) Pairing 2008. LNCS, vol. 5209, pp. 332–353. Springer, Heidelberg (2008)
10. Libert, B., Vergnaud, D.: Unidirectional Chosen-Ciphertext Secure Proxy Re-Encryption. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 360–379. Springer, Heidelberg (2008)
11. Mitsunari, S., Sakai, R., Kasahara, M.: A New Traitor Tracing. IEICE Transactions E85-A(2), 481–484 (2002)

12. Shao, J., Cao, Z.: CCA-Secure Proxy Re-Encryption without Pairings. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 357–376. Springer, Heidelberg (2009)
13. Shao, J., Cao, Z., Liu, P.: CCA-Secure PRE Scheme without Random Oracles, Cryptology ePrint archive (2010), http://eprint.iacr.org/2010/112
14. Sur, C., Jung, C.D., Park, Y., Rhee, K.H.: Chosen-Ciphertext Secure Certificateless Proxy Re-Encryption. In: De Decker, B., Schaumüller-Bichl, I. (eds.) CMS 2010. LNCS, vol. 6109, pp. 214–232. Springer, Heidelberg (2010)
15. Wang, L., Wang, L., Mambo, M., Okamoto, E.: New Identity-Based Proxy Re-encryption Schemes to Prevent Collusion Attacks. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 327–346. Springer, Heidelberg (2010)
16. Wang, X.A., Wei, P., Zhang, M.: New CCA-secure Proxy Re-encryption Scheme without Random Oracles. In: CIS 2010, pp. 377–381 (2010)

Author Index