Lecture Notes in Computer Science 5312

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

Kanta Matsuura Eiichiro Fujisaki (Eds.)

Advances in Information and Computer Security

Third International Workshop on Security, IWSEC 2008
Kagawa, Japan, November 25-27, 2008
Proceedings

Springer

Volume Editors

Kanta Matsuura
Institute of Industrial Science
The University of Tokyo
Tokyo, Japan
E-mail: kanta@iis.u-tokyo.ac.jp

Eiichiro Fujisaki
NTT Laboratories
Yokosuka-shi, Japan
E-mail: fujisaki.eiichiro@lab.ntt.co.jp

Library of Congress Control Number: 2008939387

CR Subject Classification (1998): E.3, G.2.1, D.4.6, K.6.5, K.4.1, F.2.1, C.2

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-540-89597-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-89597-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12557493 06/3180 5 4 3 2 1 0

Preface

The Third International Workshop on Security (IWSEC 2008) was held at Kagawa International Conference Hall, Kagawa, Japan, November 25–27, 2008. The workshop was co-sponsored jointly by CSEC, a special interest group on computer security of IPSJ (Information Processing Society of Japan) and ISEC, a technical group on information security of the IEICE (The Institute of Electronics, Information and Communication Engineers). The excellent Local Organizing Committee was led by the IWSEC 2008 General Co-chairs, Masato Terada and Kazuo Ohta.

This year, there were 94 paper submissions from all over the world. We would like to thank all the authors who submitted papers to IWSEC 2008. Each paper was reviewed at least three reviewers. In addition to the members of the Program Committee, many external reviewers joined the review process of papers in their particular areas of expertise. We were fortunate to have this energetic team of experts, and are grateful to all of them for their hard work. The hard work includes very active discussion; the discussion phase was almost as long as the initial individual reviewing. The review and discussion were supported by a very nice Web-based system, iChair. We appreciate its developers.

After all the review phases, 18 papers were accepted for publication in this volume of *Advances in Information and Computer Security*. In the workshop, the contributed papers were supplemented by one invited talk from eminent researcher Alfred Menezes (the Centre for Applied Cryptographic Research, The University of Waterloo).

There are many people who contributed to the success of IWSEC 2008. We wish to express our deep appreciation for their contribution to information and computer security.

November 2008

Kanta Matsuura
Eiichiro Fujisaki

IWSEC 2008
Third International Workshop on Security

Co-sponsored by

CSEC (Special Interest Group on Computer Security of Information Processing Society of Japan)
and
ISEC (Technical Group on Information Security, Engineering Sciences Society, of the Institute of Electronics, Information and Communication Engineers, Japan)

General Co-chairs

Masato Terada	Hitachi Ltd., Japan
Kazuo Ohta	University of Electro-Communications, Japan

Advisory Committee

Norihisa Doi	Chuo University, Japan
Akira Hayashi	Kanazawa Institute of Technology, Japan
Hideki Imai	Chuo University, Japan
Günter Müller	University of Freiburg, Germany
Yuko Murayama	Iwate Prefectural University, Japan
Eiji Okamoto	University of Tsukuba, Japan
Ryoichi Sasaki	Tokyo Denki University, Japan
Shigeo Tsujii	Institute of Information Security, Japan
Doug Tygar	University of California, Berkeley, USA

Program Committee Co-chairs

Kanta Matsuura	The University of Tokyo, Japan
Eiichiro Fujisaki	NTT Labs, Japan

Local Organizing Committee

Co-chairs

Yuji Suga	Internet Initiative Japan Inc., Japan
Takao Okubo	Fujitsu Laboratories Ltd., Japan
Minoru Kuribayashi	Kobe University, Japan

Award Co-chairs

Mira Kim	Institute of Information Security, Japan
Hiroshi Doi	Institute of Information Security, Japan

Finance, Registration, and Liaison Co-chairs

Keisuke Takemori	KDDI R&D Laboratories Inc., Japan
Ryuya Uda	Tokyo University of Technology, Japan
Kazuhiro Ono	Mitsubishi Electric, Japan

Publicity Co-chairs

Koji Chida	NTT Labs, Japan
Kunihiko Miyazaki	Hitachi Ltd., Japan

System Co-chairs

Naoto Sone	Naruto University of Education, Japan
Toshihiro Tabata	Okayama University, Japan

Publication co-Chairs

Tsuyoshi Takagi	Future University Hakodate, Japan
Isao Echizen	National Institute of Infomatics, Japan

Program Committee

Michel Abdalla	ENS & CNRS, France
Koichiro Akiyama	Toshiba Corporation, Japan
Jesus Almansa	NTT Labs, Japan
Tomoyuki Asano	Sony Corporation, Japan
Feng Bao	Institute for Infocomm Research, Singapore
Kevin Butler	Pennsylvania State University, USA
Ee-Chien Chang	National University of Singapore, Singapore
Ed Dawson	Queensland University of Technology, Australia
Bart De Decker	K. U. Leuven, Belgium
Hiroshi Doi	Institute of Information Security, Japan
Steven Furnell	University of Plymouth, UK
Soichi Furuya	Hitachi, Ltd., Japan
David Galindo	University of Malaga, Spain
Philippe Golle	Palo Alto Research Center, USA
Shoichi Hirose	University of Fukui, Japan
Keiichi Iwamura	Tokyo University of Science, Japan
Tetsu Iwata	Nagoya University, Japan

Angelos D. Keromytis	Columbia University, USA
Aggelos Kiayias	University of Connecticut, USA
Hiroaki Kikuchi	Tokai University, Japan
Mira Kim	Institute of Information Security, Japan
Michiharu Kudo	IBM Japan, Japan
Noboru Kunihiro	The University of Tokyo, Japan
Kwok-Yan Lam	Tsinghua University, China
Dong Hoon Lee	Korea University, Korea
Javier Lopez	University of Malaga, Spain
Mark Manulis	UCL Crypto Group, Belgium
Wenbo Mao	EMC Research China, China
Keith Martin	Royal Holloway, University of London, UK
Mitsuru Matsui	Mitsubishi Electric, Japan
Atsuko Miyaji	Japan Advanced Institute of Science and Techinology, Japan
Toru Nakanishi	Okayama University, Japan
Phong Nguyen	ENS, France
Masakatsu Nishigaki	Shizuoka University, Japan
Wakaha Ogata	Tokyo Institute of Technology, Japan
Takeshi Okamoto	Tsukuba University of Technology, Japan
Tatsuaki Okamoto	NTT Labs, Japan
Kazumasa Omote	University of Tsukuba, Japan
Kenneth G. Paterson	Royal Holloway, University of London, UK
Raphael Phan	Loughborough University, UK
Kai Rannenberg	Frankfurt University, Germany
Kyung-Hyune Rhee	Pukyong National University, Korea
Rei Safavi-Naini	University of Calgary, Canada
Kouichi Sakurai	Kyushu University, Japan
Willy Susilo	University of Wollongong, Australia
Tsuyoshi Takagi	Future University Hakodate, Japan
Keiji Takeda	Carnegie Mellon CyLab Japan, Japan
Toshiaki Tanaka	KDDI R&D Laboratories Inc., Japan
Ryuya Uda	Tokyo University of Technology, Japan
Serge Vaudenay	EPFL, Switzerland
Teemupekka Virtanen	Helsinki University of Technology, Finland
Guilin Wang	University of Birmingham, UK
Cliff Wang	Army Research Office, USA
Hajime Watanabe	National Institute of Advanced Science and Technology, Japan
Sung-Ming Yen	National Central University, Taiwan
Hiroshi Yoshiura	University of Electro-Communications, Japan
Yuliang Zheng	University of North Carolina at Charlotte, USA
Jianying Zhou	Institute for Infocomm Research, Singapore

External Reviewers

Andreas Albers
Yusuke Atomori
Nuttapong Attrapadung
Joonsang Baek
Quan Bai
Shane Balfe
Jean-Luc Beuchat
Charles Bouillaguet
Kyu Young Choi
Yvonne Cliff
Jason Crampton
Yang Cui
André Deuker
Pierre-Alain Fouque
Georg Fuchsbauer
Meng Ge
Keisuke Hakuta
Goichiro Hanaoka
Yoshikazu Hanatani
Ryotaro Hayashi
Julio C. Hernandez-Castro
Harunaga Hiwatari
Naofumi Homma
Xinyi Huang
Tibor Jager
Lars Janssen
Christian Kahl
Yasuharu Katsuno
Bum Han Kim
Masafumi Kusakawa
Jeong Ok Kwon
Gyesik Lee
Kwang Su Lee
Adrian Leung
Gaëtan Leurent
Jun Li
Benoit Libert
Tatsuyuki Matsushita
Luke McAven
Hirofumi Muratani
Akito Niwa
Satoshi Obana
Kazuto Ogawa
Koji Okada
Raphael Overbeck
Sylvain Pasini
Maura Paterson
Kun Peng
Jsaon Reid
Hyun Sook Rhee
Denis Royer
Sandra Rueda
Minoru Saeki
Mike Scott
Jinyang Shi
Joshua Schiffman
Kouichi Shimizu
SeongHan Shin
Zhexuan Song
Chunhua Su
Hung-Min Sun
Daisuke Suzuki
Katsuyuki Takashima
Shigenori Uchiyama
Martin Vuagnoux
Yuji Watanabe
Ou Yang
Tomoko Yonemura
Maki Yoshida
Katsunari Yoshioka
Rui Zhang
Jan Zibuschka

Table of Contents

Cryptography

Signature and Signcryption

Software Security

Privacy Protection and Contents Protection

Invited Talk

Authentication and Access Control

Implementation

On Generating Elements of Orders Dividing $p^{2k} \pm p^k + 1$*

Maciej Grześkowiak

Adam Mickiewicz University,
Faculty of Mathematics and Computer Science,
Umultowska 87, 61-614 Poznań, Poland
maciejg@amu.edu.pl

Abstract. In 1999 Gong and Harn proposed a new cryptosystem based on third-order characteristic sequences with period $p^{2k} + p^k + 1$ for a large prime p and fixed k. In order to find key parameters and therefore to construct a polynomial whose characteristic sequence is equal to $p^{2k} + p^k + 1$ one should generate a prime p such that the prime factorization of the number $p^{2k} + p^k + 1$ is known. In this paper we propose new, efficient methods for finding the prime p and the factorization of the aforementioned number. Our algorithms work faster in practice than those proposed before. Moreover, when used for generating of XTR key parameters, they are a significant improvement over the Lenstra-Verheul Algorithm. Our methods have been implemented in C++ using LiDIA and numerical test are presented.

Keywords: The Gong-Harn Public Key System, The XTR Public Key System, primes of a special form, solving polynomial equations modulo p.

1 Introduction and Background

1.1 The Problem

Many new cryptosystems have been introduced in recent years, which require generating primes of special forms as key parameters [9], [13], [17]. For instance, in 1998 and 1999 Gong and Harn [6], [7] proposed public-key cryptosystems based on third-order characteristic sequences over $\mathbf{F}_{p^{3k}}$ of period $\Phi_3(p^k)$, where $\Phi_m(x)$ is the m-th cyclotomic polynomial. In order to generate key parameters to the GH cryptosystem one should find a large prime p and an element $\alpha \in \mathbf{F}_{p^{3k}}$ of order $\Phi_3(p^k)$ for a fixed k [8]. One can determine whether or not the element α has the desired order if one knows the prime factorization of the number $\Phi_3(p^k)$. We extend the definition from [8] and we define a prime p to be *good* if the factorization of $\Phi_m(p^k)$ is known, where $m = 3, 6$. From the security point of view it is essential to find a *good* prime p such that $\Phi_m(p^k)$ has a large prime

* Partially supported by Ministry of Science and Higher Education, grant N N206 2701 33, 2007–2010.

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 1–19, 2008.

factor q having at least 160 bits to make DLP Problem in subgroup of order q of $\mathbf{F}^*_{p^{mk}}$ intractable. Moreover one should finds a prime p such that $mk \log p \approx 1024$ to obtain security equivalent to factoring a positive integer having 1024 bits. There exist two main approaches for generating *good* primes. One of them was proposed by Gong and Harn [8]. Now we give an illustration of this algorithm in the case $m = 3$ and $k = 2$. The main idea it is based on is the following simple observation related to the factorization of polynomials

$$\Phi_3(x^2) = \Phi_3(x)\Phi_6(x) = (x^2 + x + 1)(x^2 - x + 1)$$

The algorithm works as follows. It selects a bound B and a random prime p. Next it computes $\Phi_3(p)$ and $\Phi_6(p)$ and attempts trial division of $\Phi_3(p)$ and $\Phi_6(p)$ for all primes less than B. In the end the algorithm performs a primality test on the remaining large factors of $\Phi_3(p)$ and $\Phi_6(p)$ respectively. If both are prime (say equal to q and $\widetilde{q}$ respectively) then the algorithm returns the *good* prime p. Otherwise, it randomly selects a prime p and the abovementioned steps are repeated. It is worth pointing out that for the appropriate level of security p and q should be have no less than 170 and 160 bits respectively. For such sizes of p and q, the algorithm works well in practice. However, it is essential for the algorithm to work efficiently in spite of the parameters' increase by orders of magnitude.

However for a large p and $k > 2$ it seems to be time-consuming to generate a *good* prime p in the aforementioned way and consequently to check whether or not a sequence has the correct period $\Phi_3(p^k)$ in the GH cryptosystem. For example, note that in the case $k = 4$ we have the following factorization

$$\Phi_3(p^4) = \Phi_{12}(p)\Phi_6(p)\Phi_3(p) = (p^4 - p^2 + 1)(p^2 - p + 1)(p^2 + p + 1)$$

We can see that not only does the order of magnitude of the generated prime factors of $\Phi_3(p^4)$ increase, but also their number, which in turn increases the algorithm's complexity.

On the other hand, theoretical estimation of the algorithm's running time becomes a problem. Namely, it is not known whether there exists an infinite number of primes p such that $\Phi_{12}(p)/s_1$, $\Phi_6(p)/s_2$ and $\Phi_3(p)/s_3$ are simultaneously prime, where s_i, $i = 1, 2, 3$ factor into small primes $< B$.

The second approach for generating *good* primes was proposed by Lenstra and Verheul [14]. However, we have to point out that the goal of that algorithm was to generate such key parameters for the XTR cryptosytem which are resistant to the so called *subgroup attacks*. Unfortunately, we can only use it for our purposes if the polynomial $\Phi_m(x^k)$ is irreducible. We briefly recall the idea of this algorithm. The algorithm randomly selects a prime $q_1 \equiv 7 \pmod{12}$ and computes r_i for $i = 1, 2$ roots of $\Phi_6(x) = x^2 - x + 1 \bmod q_1$. Alternatively the algorithm finds a positive integer r_3 such that $\Phi_6(r_3) = q_1$ is a prime. Next the algorithm selects a prime p such that $p \equiv r_i \pmod{q_1}$ for one of r_i $i = 1, 2, 3$ (from this q_1 divides $\Phi_6(p)$). In the end it checks if the number $\Phi_6(p)/q_1$ is of the form $s \cdot q_2$ where s factors into small primes $< B$ and q_2 is a large prime. It is worth pointing out that the above algorithm works perfectly well in practise.

However in the general case, we can encounter further problems. For example let $k = 6$. Consider the naive way of computing the root of the polynomial

$$\Phi_6(x^6) = \Phi_{36}(x) = (x^{12} - x^6 + 1) \pmod{q}, \tag{1}$$

irreducible in $\mathbf{Z}[x]$, where q is a prime. Substituting $y = x^6$ we reduce the degree of (1) to 2. Next we compute y_1, y_2 roots of $y^2 - y + 1 \pmod{q}$, which requires computing $\sqrt{3} \pmod{q}$. In the end we compute the 6-th root $\pmod{q}$ of y_i for $i = 1$ or $i = 2$. However there are two difficulties, which we can encounter in practice while computing the roots of (1). The first is that we have to use algorithms to both compute the square root and to compute the 6-th root modulo q. Computing of the square root $\pmod{q}$ is basically simple, except for the case where $q \equiv 1 \pmod{8}$, which takes at most $O(\log^4 q)$ [4]. However, computing of the 6-th root can be more difficult. In general, solving of the congruence $x^r \equiv a \pmod{q}$, if a solution exists, is difficult for large r [4]. However, algorithms exist to solve it efficiently if r is a small prime [21]. The simplest case is $r = 3$ [15]. The second problem lies in the handling of square and 6-th roots when these are not in $\mathbf{F}_q$ (they are then in $\mathbf{F}_{q^2}$ and $\mathbf{F}_{q^6}$ respectively). The alternative method in the abovementioned algorithm, involving finding a positive integer r_3 such that $\Phi_6(r_3)$ is a prime, causes both theoretical and practical problems. We do not know if there exist infinitely many primes of the form $\Phi_6(r)$. This is an extremely hard, still unproven mathematical problem. However, there are some conjectures related to this problem [19], [3]. Moreover, numerical tests conducted by us prove that for large r, finding primes which are values of high-degree irreducible polynomials in $\mathbf{Z}[x]$, is time-consuming. The second part of the algorithm also seems problematic. We denote by $P(a, q)$ the least prime $p \equiv a \pmod{q}$ for any $a, q \in \mathbf{N}, 1 \leq a \leq q, (a, q) = 1$. A well known theorem by Heath-Brown [11] gives $P(a, q) \ll q^{5.5}$. From this we cannot exclude the possibility that the number of steps we need to find prime p is exponential. However it can be found quickly in practice [20].

1.2 The Main Result of This Paper

In this paper, we propose an new method of finding *good* primes, which is faster than those previously considered. We also present the developments and improvements of ideas proposed by Lenstra and Verheul. In particular, we improve the method of finding roots of irreducible polynomials

$$\Phi_m(x^k) \pmod{q} \tag{2}$$

in $Z[x]$ where q is a prime, by reducing the number and degree of computed roots. Our method can also be used to construct torus based system using optimal extension fields [18]. Torus based cryptosystem requires one to find a suitable subgroup of prime order in $\mathbf{F}_{q^m}$ with $q = p^k$ for general m nad k. If m, k becomes large, then this causes significant problem because one needs to find a big (and suitable) prime factor of $\Phi_m(p^k)$.

For example, for $m = 3, 6$ and $k = 1$ our method does not require computing any roots in $\mathbf{F}_q$ in order to find the root of (2), which is a big improvement over the Lenstra-Verheul Algorithm [13]. Such an algorithm is presented in Section 2 and it can be used for any arbitrary positive integer k. However the complexity of the algorithm will rise with order of magnitude of k, which is a consequence of the necessity of computing the k-th root in $\mathbf{F}_q$.

Furthermore, for $k = 6$, finding of the root of (2) in $\mathbf{F}_q$ is reduced to computing only one cubic root in $\mathbf{F}_q$, for which there exist an efficient algorithm [15]. This in turn vastly speeds up generation of key parameters of the GH cryptosystem. Algorithm which fulfills this aim is presented in Section 3. This algorithm works for even arbitrary positive integer k. For each such even arbitrary positive integer k computing of k-th root is reduced to computing $k/2$ root in $\mathbf{F}_q$. However, the analysis of this algorithm is more complex and subtle then analysis of algorithm from Section 2.

Moreover, our algorithms guarantee that roots of (2) always exist in $\mathbf{F}_q$. Achieving the described goals is made possible by generating a prime q, which is a value of a primitive quadratic polynomial of two variables with integer coefficients. We prove that the algorithm for finding such a prime is random and executes in polynomial time.

Computational advantages of our method are significant. Numerical tests show that our algorithm works on average three times as fast in practice than those proposed in [8] under comparable of level of security. Our approach splits naturally into two parts. Let $m = 3, 6$. For clarity we will consider only $k = 1, 2, 3, 4, 6$. Observe that we have

$$\Phi_m(p^k) = \Phi_{mk}(p), \tag{3}$$

if $m = 3$ and $k = 1, 3$ or $m = 6$ and $k = 1, 2, 3, 4, 6$. On the other hand it easy to see that we have the following factorization

$$\Phi_3(p^k) = \prod_{d|k} \Phi_{3d}(p), \tag{4}$$

where $k = 2, 4, 6$ and we assume $d > 2$ if $k = 6$. It can be easily checked using any of the available computer algebra systems. First we describe an algorithm for finding a large prime q which divides an irreducible factor of $\Phi_m(p^k)$. Next, we describe an algorithm for finding a *good* prime which satisfies (3), and we extend our method and apply it to the case (4).

2 Generating a Prime Factor of $\Phi_m(p^k)$ for $k = 1, 2, 3$

We start with an algorithm which finds large primes q and p such that q divides $\Phi_m(p^k)$, where $m = 3, 6$ and $k = 1, 2, 3$. Let $\omega = (-1+\sqrt{-3})/2$ be the cubic root of unity. Define the *Eisenstein integers* as the set $\mathbf{Z}[\omega] = \{x + y\omega : x, y \in \mathbf{Z}\}$. Let $\mathbf{Q}(\omega)$ be the corresponding quadratic number field with the ring of integers $\mathbf{Z}[\omega]$. Let $\alpha \in \mathbf{Z}[\omega]$. We denote by $N(\alpha) = \alpha\overline{\alpha} = x^2 - xy + y^2$ the norm of

α relative to $\mathbf{Q}$, where $x,\ y \in \mathbf{Z}$. We will now introduce the main ideas of our algorithm, omitting some details for clarity. Our strategy works as follows. Let us fix a quadratic polynomial $Q(x,y) = x^2 - xy + y^2$ irreducible in $Q[x,y]$. Let us fix k. Of course $Q(x,y) = N(x + y\omega)$ for positive integers x and y. Suppose that one randomly finds positive integers a and b, $(a,b) = 1$ such that $Q(a,b) = q$ is a prime. Then there exists the element $\alpha = a + b\omega \in \mathbf{Z}[\omega]$ such that $N(\alpha) = Q(a,b) = q$. Let $\beta = (p^k - 1) + p^k\omega \in \mathbf{Z}[\omega]$, where p is a prime. Then $N(\beta) = \Phi_6(p^k)$, which can be easily computed. Now assume that α divides β. Then there exists $\gamma = c + d\omega \in \mathbf{Z}[\omega]$ such that

$$\alpha\gamma = (a + b\omega)(c + d\omega) = (p^k - 1) + p^k\omega = \beta. \tag{5}$$

Since $\omega^2 = -\omega - 1$ then the above equalities yield

$$\begin{cases} ac - bd + 1 = p^k, \\ bc + (a - b)d = p^k. \end{cases} \tag{6}$$

Subtracting the first equation from the second, we simplify the above system of equations to a linear equation of variables c and d

$$ad - (a - b)c = 1 \tag{7}$$

Since $(a, a - b) = 1$, there exists infinitely many solutions of this equation. In particular, we find certain solutions of the system of equations (6). We now look for such solutions of (7) that the right-hand side of one of the two equations in the system (6) is a k-th power of prime p. Let c_0 and d_0 be such solutions of (7). Then, from (5) and multiplicative property of norms we get $qN(\gamma) = \Phi_6(p^k)$, where $\gamma = c_0 + d_0\omega$.

Now, we briefly explain the meaning of some functions used in our algorithms. The function $Random(n)$: returns a random integer n. Let $n_1, n_2, \ldots, n_r$ be positive integers. The function $IsPrime(n_1, n_2, \ldots, n_r)$ returns `true` if every $n_1, n_2, \ldots, n_r$ satisfies probabilistic test for primality [16] or some primality test which is deterministic [1]. Moreover both algorithm run in polynomial time. Otherwise it returns `false`. We will denote by $\mathcal{PT}$ number of bit operations necessary to carry out the deterministic primality test. For simplicity, assume that $\mathcal{PT} \gg \log^3 n$.

Theorem 1. *Let $m = 3$ and $k = 1, 3$ or $m = 6$ and $k = 1, 2, 3$ be fixed positive integers. Let l_1, l_2 be positive integers, such that $l_1^2 - l_1 l_2 + l_2^2 \equiv 1 \pmod{mk}$ and $l_1 \not\equiv l_2 \pmod{mk}$. Then Algorithm 1 generates primes q and p such that q divides $\Phi_m(p^k)$. Moreover $q = Q(a,b) = N(\alpha)$ and $\Phi_m(p^k) = N(\beta)$, where $\alpha,\ \beta \in \mathbf{Z}[\omega]$, $\alpha \mid \beta$ and $\alpha = (6ka + l_1) + (6kb + l_2)\omega$ and $\beta = (p^k - 1) + p^k\omega$.*

Proof. Let us fix k and positive integers l_1, l_2. Assume that the procedure FindPrimeQ finds positive integers a, b such that $Q(a,b) = q$ is prime. Then there exists $\alpha = (6ka + l_1) + (6kb + l_2)\omega \in \mathbf{Z}[\omega]$ such that

$$N(\alpha) = (6ka + l_1)^2 - (6ka + l_1)(6kb + l_2) + (6kb + l_2)^2 = Q(a,b).$$

Algorithm 1. Generating primes p and q, such that $q|\Phi_m(p^k)$

input: $m = 3$ and $k = 1, 3$ or $m = 6$ and $k = 1, 2, 3$, $n \in \mathbf{N}$, l_1, $l_2 \in \mathbf{N}$ such that $l_1^2 - l_1 l_2 + l_2^2 \equiv 1 \pmod{mk}$, $l_1 \not\equiv l_2 \pmod{mk}$.
output: primes p, q, such that $q \mid \Phi_m(p^k)$.

$A \leftarrow 36k^2$
$B \leftarrow -36k^2$
$C \leftarrow 36k^2$
$E \leftarrow 6k(2l_1 - l_2)$
$F \leftarrow 6k(2l_2 - l_1)$
$G \leftarrow l_1^2 - l_1 l_2 + l_2^2$
$Q(x, y) \leftarrow Ax^2 + Bxy + Cy^2 + Ex + Fy + G$

procedure FindPrimeQ(n, $Q(x, y)$)
    $q \leftarrow 1$
    while not $IsPrime(q)$ **do**
        $a \leftarrow Random(n)$ ▷ Randomly select $a \in \left[\frac{n-l_1}{6k}, \frac{\sqrt{c}n-l_1}{6k}\right]$
        $b \leftarrow Random(n)$ ▷ Randomly select $b \in \left[\frac{n-l_2}{6k}, \frac{\sqrt{c}n-l_2}{6k}\right]$
        $q \leftarrow Q(a, b)$
    end while
    return (a, b, q)
end procedure

procedure FindRestModuloQ(a, b, q, k, m)
    $a \leftarrow 6ka + l_1$
    $b \leftarrow 6kb + l_2$
    Find $x_0, y_0 \in \mathbf{Z} : ay_0 - (a - b)x_0 = 1$ ▷ Use Extended Euclid's Algorithm
    $s \leftarrow bx_0 + (a - b)y_0 \pmod{q}$
    if $m = 3$ **then**
        $s \leftarrow -s$
    end if
    $r \leftarrow s^{\frac{1}{k}} \pmod{q}$
    return (r)
end procedure

procedure FindPrimeModuloQ(r, q, t)
    $p \leftarrow qk + r$
    while not $IsPrime(p)$ **do**
        $p \leftarrow p + q$
        $t \leftarrow t + 1$
    end while
    return (p, t)
end procedure

return (p, q)

Moreover assume that α divides an element $\beta \in \mathbf{Z}[\omega]$ of the form $\beta = (p^k - 1) + p^k\omega$, where p is a prime. Then there exists $\gamma \in \mathbf{Z}[\omega]$, $\gamma = x + y\omega$, $x, y \in \mathbf{Z}$ such that

$$\alpha\gamma = \beta \tag{8}$$

and

$$N(\alpha)N(\gamma) = N(\beta) = \Phi_6(p^k). \tag{9}$$

We show how one can find elements $\gamma, \beta \in \mathbf{Z}[\omega]$, and the prime p satisfying (9). Since $\omega^2 = -\omega - 1$, equalities (8) yield

$$\begin{cases} (6ka + l_1)x - (6kb + l_2)y + 1 = p^k, \\ (6kb + l_2)x + (6ka + l_1 - 6kb - l_2)y = p^k, \end{cases} \tag{10}$$

where $6ka + l_1$, $6kb + l_2$ are given. Subtracting we get the linear equation

$$(6ka + l_1)y - (6ka + l_1 - 6kb - l_2)x = 1. \tag{11}$$

Since

$$q = Q(a, b) = (6ka + l_1)^2 - (6ka + l_1)(6kb + l_2) + (6kb + l_2)^2$$

is a prime then $(6ka+l_1, 6ka+l_2-6kb-l_2) = 1$ and hence there exists an integer solution x_0, y_0 of (11). It can be found by applying the extended Euclidean algorithm (see step 20 of procedure FindRestModuloQ). Furthermore, the equation (11) has infinitely many solutions x, y of the form

$$\begin{cases} x = x_0 + (6ka + l_1)t, \\ y = y_0 + (6ka + l_1 - 6kb - l_2)t, \end{cases} \tag{12}$$

where $t \in \mathbf{Z}$. Substituting (12) into the second equation of (10), we obtain

$$(6kb + l_2)x_0 + (6ka + l_1 - 6kb - l_2)y_0 + qt = p^k, \quad t \in \mathbf{Z}.$$

Consequently

$$p^k \equiv (6kb + l_2)x_0 + (6ka + l_1 - 6kb - l_2)y_0 \pmod{q}.$$

Now, taking

$$s = (6kb + l_2)x_0 + (6ka + l_1 - 6kb - l_2)y_0 \pmod{q}$$

(see step 21 of procedure FindRestModuloQ) from (9) we obtain

$$\Phi_6(s) \equiv 0 \pmod{q}.$$

Therefore the order of s is equal to 6 modulo q. Since $q \equiv 1 \pmod{6k}$ then

$$s^{\frac{q-1}{k}} \equiv 1 \pmod{q},$$

and there exists r_1 such that

$$r_1^k \equiv s \pmod{q}$$

(see step 25 of procedure FindRestModuloQ). Consequently

$$\Phi_6(r_1^k) \equiv \Phi_6(s) \equiv 0 \pmod{q}.$$

Since $\Phi_6(s) = \Phi_3(-s)$ then the order of $-s$ is equal to 3 modulo q. Hence

$$(-s)^{\frac{q-1}{k}} \equiv 1 \pmod{q}$$

and there exists r_2 such that

$$r_2^k \equiv -s \pmod{q}.$$

Consequently

$$\Phi_3(r_2^k) \equiv \Phi_3(-s) \equiv 0 \pmod{q}.$$

Now, let p be the a prime in an arithmetic progression $p \equiv r_j \pmod{q}$, where $j = 1$ or $j = 2$. Then $\Phi_3(p^k) \equiv \Phi_3(r_2^k) \pmod{q}$ and $\Phi_6(p^k) \equiv \Phi_6(r_1^k) \pmod{q}$ (see procedure FindPrimeModuloQ). This finishes the proof.

Remark 1. In 2000, A. Lenstra and E. Verheul introduced a new public key system called XTR [13]. They proposed a straightforward algorithm for finding primes p and q, such that $q|\Phi_6(p)$, $p \equiv 2 \pmod 3$. The algorithm randomly selects a prime $q \equiv 7 \pmod{12}$ and computes r_i for $i = 2, 3$ roots of $\Phi_6(X)$ mod q. Next, the algorithm selects a prime p such that $p \equiv r_i \pmod q$ and $p \equiv 2 \pmod 3$ for $i = 2$ or $i = 3$. Let us note that computing of the roots of this polynomial requires finding $\sqrt{3} \pmod q$ and an inverse $\pmod q$. The problem of finding the values of a if $a^2 = b$ for a given square $b \in \mathbf{F}_q$ is simple when $q \equiv 3 \pmod 4$. One immediately obtains the two solutions $a_{1,2} = \pm b^{(q+1)/4} \pmod q$ which can be determined using the square and multiply algorithm. However, when $q \equiv 1 \pmod 4$ then the problem is much more difficult [4]. For $m = 6$ and $k = 1$ Algorithm 1 generates key parameters for the XTR. However, our algorithm does not require the computation of roots in $\mathbf{F}_q$. Therefore, the computational benefits are considerable. Furthermore, since 3 is a quadratic residue modulo q if and only if $q \equiv 1 \pmod 6$, while our algorithm generates primes q of this form, it works for a broader class of primes.

3 Generating a Large Prime Factor of $\Phi_6(p^k)$ for $k = 2, 4, 6$

Our next goal is to present an algorithm for generating a prime p and a large prime factor q of $\Phi_6(p^k)$ for $m = 6$ and $k = 2, 4, 6$. Let $\mathbf{Z}[i] = \{x + yi : x, y \in \mathbf{Z}, i = \sqrt{-1}\}$. This ring is called the ring of Gaussian integers. Let $\mathbf{Q}(i)$ be the

quadratic number field with the ring of integers $\mathbf{Z}[i]$. Let $\gamma \in \mathbf{Z}[i]$. We denote by $N(\gamma) = x^2 + y^2$ the norm of α relative to $\mathbf{Q}$, where $x, y \in \mathbf{Z}$. Let us fix a quadratic polynomial $Q(x, y) = x^2 + y^2$ irreducible in $Q[x, y]$. Our strategy works as follows. Let us fix k. Of course $Q(x, y) = N(x+yi)$ for positive integers x and y. Suppose that one randomly finds positive integers a and b, $(a, b) = 1$ such that $Q(a, b) = q$ is a prime. Then there exists the element $\gamma = a + bi \in \mathbf{Z}[i]$ such that $N(\gamma) = Q(a, b) = q$. Let $\xi = (p^k - 1) + \sqrt{p^k}\, i \in \mathbf{Z}[i]$ where p is a prime. We have $N(\xi) = \Phi_6(p^k)$. Now assume that γ divides ξ. Then there exists $\delta = c + di \in \mathbf{Z}[i]$ such that

$$\gamma\delta = (a + bi)(c + di) = (p^k - 1) + \sqrt{p^k}i = \xi. \tag{13}$$

Hence the above equalities yield

$$\begin{cases} ac - bd = p^k - 1 \\ bc + ad = \sqrt{p^k}, \end{cases} \tag{14}$$

where a, b are given. Squaring the second equation and substituting into the first one we get

$$Ac^2 + Bcd + Cd^2 + Dc + Ed = -1, \tag{15}$$

an quadratic polynomial of variables c and d, where $A = -b^2$, $B = -2ab$, $C = -a^2$, $D = a$, $E = -b$. There exist infinitely many solutions of equation (15). In particular, we find certain solutions of (14). Next, we look for solutions of (15), such that the second equation in the system (14) is $k/2$-th power of prime p. In practice, this can be reduced to computing the $k/2$-th degree root in $\mathbf{F}_q$ and to finding the prime p in an arithmetic progression. Let c_0 and d_0 be such solutions. Then by (13) we obtain $qN(\delta) = \Phi_6(p^k)$, where $\delta = c_0 + d_0 i$.

Algorithm 2. Generating primes p and q, such that $q|\Phi_6(p^k)$

input: $k = 2, 4, 6$, $n \in \mathbf{N}$, $F(x, y) = 36k^2x^2 + 36k^2y^2 + 12ky + 1$.
output: primes p and q such that $q \mid \Phi_6(p^k)$.

1: FindPrimeQ$(n, F(x, y))$ ▷ Return prime q and $a, b : q = F(a, b)$

2: **procedure** FindRestModuloQTwo(a, b, q, k)
3: $w \leftarrow 3^{\frac{1}{2}} \pmod q$
4: $s \leftarrow (w(6kb + 1) - 6ka)(-2(6kb + 1))^{-1} \pmod q$
5: $r \leftarrow s^{\frac{2}{k}} \pmod q$
6: **return** r
7: **end procedure**

8: FindPrimeModuloQ(r, q, t) ▷ Return t and prime p
9: **return** (q, p)

Theorem 2. *Let $k = 2, 4, 6$ be fixed positive integer. Then Algorithm 2 generates primes q and p, such that q divides $\Phi_6(p^k)$. Moreover $q = F(a, b) = N(\gamma)$, $\Phi_6(p^k) = N(\xi)$, where γ, $\xi \in \mathbf{Z}[i]$, $\gamma \mid \xi$ and $\gamma = 6ka + (6kb + 1)i$, $\xi = (p^k - 1) + \sqrt{p^k}\, i$.*

Proof. Let us fix k. Assume that the procedure FindPrimeQ finds positive integers a, b such that $F(a, b) = q$ is prime. Then there exists $\gamma = 6ka + (6kb + 1)i \in \mathbf{Z}[i]$ such that

$$N(\gamma) = (6ka)^2 - ((6kb + 1)i)^2 = F(a, b)$$

Moreover assume that γ divides $\xi = (p^k - 1) + \sqrt{p^k}\, i \in \mathbf{Z}[i]$, where p is a prime. Hence there exists $\delta \in \mathbf{Z}[i]$, $\delta = x + yi$, $x, y \in \mathbf{N}$ such that

$$\gamma\delta = \xi, \tag{16}$$

and

$$N(\gamma)N(\delta) = N(\xi) = \Phi_6(p^k). \tag{17}$$

We show how one can find elements δ, $\xi \in \mathbf{Z}[i]$, and a prime p satisfying (16). By (16) it follows that

$$\begin{cases} 6kax - (6kb + 1)y = p^k - 1 \\ (6kb + 1)x + 6kay = \sqrt{p^k}, \end{cases} \tag{18}$$

where $6ka$, $6kb + 1$ are given. Squaring the second equation and substituting to the first one we get

$$Ax^2 + Bxy + Cy^2 + Dx + Ey = -1, \tag{19}$$

where $A = -(6kb + 1)^2$, $B = -2(6ka)(6kb + 1)$, $C = -(6ka)^2$, $D = 6ka$, $E = -(6kb = 1)$. Now we find solutions of (19). We write $\Delta = B^2 - 4AC$. Trivial computation show that $\Delta = 0$. Multiplying (19) by $4A$ we obtain,

$$(2Ax + By)^2 + 4ADx + 4AEy + 4A = 0.$$

Let $2Ax + By = T$ then

$$\begin{aligned} T^2 + 2(2AE - BD)y + 4A + 2DT = 0, \\ (T + D)^2 = 2(BD - 2AE)y + D^2 - 4A. \end{aligned}$$

Consequently equation (19) is equivalent to

$$(2Ax + By + D)^2 + \alpha y = \beta, \tag{20}$$

where

$$\alpha = 2(BD - 2AE) = 4(6kb + 1)((6ka)^2 + (6kb + 1)^2) = 4(6kb + 1)q \tag{21}$$

and

$$\beta = D^2 - 4A = (6ka)^2 + 4(6kb + 1)^2. \tag{22}$$

Let

$$X = 2Ax + By + D, \quad Y = -\alpha y. \tag{23}$$

By (20)

$$X^2 - \beta = Y, \tag{24}$$

we see that a necessary condition for existence of an integers solution of (20) is solubility of the congruence

$$Z^2 \equiv \beta \pmod{\alpha}. \tag{25}$$

Let z_0 be the solution of (25). From (21) and (22) it follows that $z_0 \equiv 0 \pmod{4}$, $z_0 \equiv 6ka \pmod{(6kb+1)}$ and $z_0 \equiv \sqrt{3}(6kb+1) \pmod{q}$. Since $q \equiv 1 \pmod{3}$ then 3 is quadratic residue modulo q and, in consequence, $z_0 \pmod{\alpha}$ exists. It can be easily found by the Chinese Remainder Theorem. By (23), (24) we have

$$y = (z_0^2 - \beta)/(-\alpha), \quad y \in \mathbf{N} \tag{26}$$

Now we prove that in this case x is integer as well. By (23) we have

$$z_0 - D = 2Ax + By \tag{27}$$

Since $q = F(a,b) = (6ka)^2 + (6kb+1)^2$ is a prime then $(2A, B) = 2(2kb+1)$. Hence $z_0 - D \equiv 0 \pmod{2(6kb+1)}$ and so (27) has integer solutions. Consequently, solutions of (20) are integers. This observation works for general solutions of (25)

$$z \equiv z_0 \pmod{\alpha}. \tag{28}$$

Our computation shows that integers solutions x, y of (20) have the form

$$x = \frac{z - By - D}{2A}, \quad y = \frac{z^2 - \beta}{-\alpha}, \quad x, y \in \mathbf{Z}, \tag{29}$$

where $z = \alpha t + z_0$, $t \in \mathbf{Z}$. Substituting (29) to the second equation of (18) we obtain

$$(6kb+1)\left(\frac{\alpha t + z_0 - D}{2A}\right) + \left(6ka - \frac{B(6kb+1)}{2A}\right) y = p^{k/2}$$

Since $(6ka - ((6kb+1)B)/2A)y = 0$ then putting (21) we get

$$-2qt + \frac{z_0 - 6ka}{-2(6kb+1)} = p^{k/2}, \quad t \in \mathbf{Z}.$$

Consequently

$$p^{k/2} \equiv (\sqrt{3}(6kb+1) - 6ka)(-2(6kb+1))^{-1} \pmod{q}, \tag{30}$$

Taking

$$s = (\sqrt{3}(6kb+1) - 6ka)(-2(6kb+1))^{-1} \pmod{q}$$

(see step 4 of procedure FindRestModuloQTwo) from (17) and (30) we get

$$\Phi_6(p^k) \equiv \Phi_6(s^2) \equiv \Phi_{12}(s) \equiv 0 \pmod{q}.$$

Therefore order s is equal to 12 modulo q. Since $q \equiv 1 \pmod{6k}$ then

$$s^{\frac{2(q-1)}{k}} \equiv 1 \pmod{q}.$$

Then there exists r such that

$$r^{k/2} \equiv s \pmod{q}$$

(see step 5 of procedure FindRestModuloQTwo) and so

$$\Phi_6(s^2) \equiv \Phi_6(r^k) \equiv 0 \pmod{q}.$$

Now, let p be a prime in an arithmetic progression $p \equiv r \pmod{q}$, then $\Phi_6(p^k) \equiv \Phi_6(r^k) \equiv 0 \pmod{q}$ (see procedure FindPrimePModuloQ). This finishes the proof.

4 Run-Time Analysis of the Procedure FindprimeQ

We denote by $\pi(x)$ the number of primes p not exceeding x, where $x \geq 1$, that is

$$\pi(x) = \sum_{p \leq x} 1$$

With the notation as above we recall some theorems which are related distributions of primes.

Theorem 3 (Hadamard and de la Vallée Poussin). *For some positive number A*

$$\pi(x) = \text{li } x + O(xe^{-A\sqrt{\log x}}),$$

where

$$\text{li } x = \int_2^x \frac{dt}{\log t}, \quad x > 2.$$

Proof. See [5].

Theorem 4 (Iwaniec). *Let $Q(x,y) = ax^2 + bxy + cy^2 + ex + fy + g \in Z[x,y]$, $\deg Q = 2$, $(a,b,c,e,f,g) = 1$, $Q(x,y)$ be irreducible in $\mathbf{Q}[x,y]$, represent arbitrary large number and depend essentially on two variables. Then*

$$\frac{N}{\log N} \ll \sum_{\substack{q \le N \\ q = Q(x,y)}} 1, \tag{31}$$

if $D = af^2 - bef + ce^2 + (b^2 - 4ac)g = 0$ *or* $\Delta = b^2 - 4ac$ *is a perfect square.*

$$\frac{N}{\log^{\frac{3}{2}} N} \ll \sum_{\substack{q \le N \\ q = Q(x,y)}} 1 \ll \frac{N}{\log^{\frac{3}{2}} N}, \tag{32}$$

if $D \neq 0$ *and* Δ *is different from a perfect square.*

Proof. See [12].

Remark 2. We say that Q depends essentially on two variables if $\partial Q/\partial x$ and $\partial Q/\partial y$ are linearly independent.

Now we give the analysis of computational complexity of the procedure FindPrimeQ.

Theorem 5. *Let* $m = 3$ *and* $k = 1, 3$ *or* $m = 6$ *and* $k = 1, 2, 3$. *Let* l_1, l_2 *be positive integers, such that* $l_1^2 - l_1 l_2 + l_2^2 \equiv 1 \pmod{mk}$ *and* $l_1 \not\equiv l_2 \pmod{mk}$. *Then there exist constants* b_0 *and* n_0 *such that for every integer* $n \ge n_0$ *and an arbitrary real* $\lambda \ge 1$, *the procedure* FindPrimeQ *generates prime* $q = Q(a, b)$ *such that* $q \asymp n^2$, *with probability greater than or equal to* $1 - e^{-\lambda}$ *after repeating* $[b_0 \lambda \log n]$ *steps* $10 - 14$ *of the procedure. Every step of the procedure takes no more than* $\mathcal{PT}$ *bit operations.*

Proof. We start with an estimate for the number of primes in the interval $[n^2, cn^2]$ which are of the form $Q(a, b)$, where $Q(x, y) = Ax^2 + Bxy + Cy^2 + Ex + Fy + G \in Z[x, y]$ and $A = C = 36k^2$, $B = -36k^2$, $E = 6k(2l_1 - l_2)$, $F = 6k(2l_2 - l_1)$ and $G = l_1^2 - l_1 l_2 + l_2^2$. We apply the Theorem 4. Let us fix k and l_1, l_2. Since $G \equiv 1 \pmod{mk}$ then $(A, B, C, E, F, G) = 1$. Since $l_1 \neq l_2$ then a short computation shows that $\alpha_1(Ax + By + E) + \alpha_2(2Cy + Bx + F) = 0$ if $\alpha_1 = \alpha_2 = 0$. Hence $\frac{\partial Q}{\partial x}$ and $\frac{\partial Q}{\partial y}$ are linearly independent and $Q(x, y)$ depends essentially on x and y. Moreover, it is easy to check that $Q(x, y)$ is irreducible in $\mathbf{Q}[x, y]$. Since $D = 0$ polynomial $Q(x, y)$ satisfies assumptions of Theorem 4. We define the set

$$\mathcal{Q} = \{n^2 \le q \le cn^2 : Q(x, y) = q - prime,\ x, y \in \mathbf{N}\},$$

where $c > 0$. Denote by $|\mathcal{Q}|$ the number of the elements of $\mathcal{Q}$. By Theorem 4 there exists $c_0 > 0$ such that for sufficiently large n we have

$$|\mathcal{Q}| \ge \frac{c_0 cn^2}{2 \log n} - \pi(n^2). \tag{33}$$

Since

$$\text{li } n = \frac{n}{\log n} + O\left(\frac{n}{\log^2 n}\right),$$

then by Theorem 3 and (33) it follows that there exist $c_1 = (cc_0 - 1)/2$ such that for sufficiently large n we have

$$|\mathcal{Q}| \geq c_1 \frac{n^2}{\log n} + O\left(\frac{n^2}{\log^2 n}\right), \tag{34}$$

where $c > 1/c_0$. Denote by A_Q the event that a randomly chosen pair of natural numbers a and b satisfying

$$a \in \left[\frac{n - l_1}{6k}, \frac{\sqrt{c}n - l_1}{6k}\right], \qquad b \in \left[\frac{n - l_2}{6k}, \frac{\sqrt{c}n - l_2}{6k}\right]$$

is such that the number $Q(a, b) \in [n^2, cn^2]$ is a prime. Hence by (34) there exist exists $\varepsilon \to 0$ as $n \to \infty$ and $c_2 = (c_1 - \varepsilon(n))/c$ such that for sufficiently large n we have such that the probability that in l trials A_Q does not occur is

$$\left(1 - \frac{c_2}{\log n}\right)^l = \exp\left(l \log\left(1 - \frac{c_2}{\log n}\right)\right) \leq$$
$$\leq \exp\left(\frac{-c_2 l}{\log n}\right) \leq e^{-\lambda}$$

for an arbitrary real $\lambda \geq 1$ and $l = b_0 \lambda \log n$, where $b_0 = c_2^{-1}$. Hence the probability that in l trials A_Q does occur is greater or equal to $1 - e^{-\lambda}$. So after repeating $[b_0 \lambda \log n]$ steps, the procedure finds integers a and b and primes $q = Q(a, b)$ with probability greater than or equal to $1 - e^{-\lambda}$. Now, we estimate the number of bit operations required to carry out the steps of the procedure. It takes a fixed numbers of time to generate a random bit, and $O(\log n)$ bit operations to generate a random integers a and b. Polynomials' Q value computation can be done with $O(\log^2 n)$ bit operations. The most time-consuming step of the algorithm is the deterministic primality test for number q which takes no more than $\mathcal{PT}$ operations. This finished the proof.

Remark 3. A theorem similar to the one proven above, concerning the computational complexity of the procedure FindPrimeQ can be proven for the polynomial $F(x, y)$ defined in Algorithm 2.

5 Generating *Good* Primes for Irreducible Factors

In this section we present an algorithm which finds a *good* prime if $\Phi_m(x^k)$ is irreducible in $Z[x]$. Therefore, we consider the following cases: $m = 3$ and $k = 1, 3$ or $m = 6$ and $k = 1, 2, 3, 4, 6$. For fixed $B \in \mathbf{N}$ we define $\mathcal{B} = \{p \leq B : p \text{ is prime}\}$. We will now describe the main idea of our algorithm for finding primes p and q_1, q_2 such that $\Phi_m(p^k) = sq_1q_2$, where s is small. Let us fix m, k, and choose Algorithm 1 or 2 depending on their values. Let $Q(x, y)$ be a

fixed polynomial, chosen according to parameters m and k. The main idea of the algorithm works as follows. We find a prime $q_1 = Q(a, b)$ for a certain a, b, using procedure FindPrimeQ. Next, we find residue $r \pmod{q_1}$ using procedure FindRestModduloQ. As mentioned before, r is a root of $\Phi_m(x^k) \pmod{q_1}$. The following steps are repeated until the algorithm finds a *good* prime p. Using procedure FindprimeModuloQ we find a prime $p = q_1 t_0 + r$, for some $t_0 \in N$. We check whether $\Phi_m(p^k)/q_1 = sq_2$ where $s = \prod p_i^{e_i}$, for certain $p_i \in \mathcal{B}$ and q_2 is a large prime. If this condition is not satisfied, we find another prime $p = q_1 t_1 + r$ for some $t_1 \in N$, $t_1 > t_0$ and we again check whether $\Phi_m(p^k)/q_1$ is of the desired form. For more details see Algorithm 3 in the appendix.

6 Generating *Good* Primes for Reducible Factors

Let $m = 3$ and $k = 2, 4, 6$. We describe an algorithm which finds a *good* prime p if $\Phi_m(p^k)$ is reducible. To illustrate the idea of this algorithm we will use the following example. Let

$$\Phi_3(x^2) = \Phi_3(x)\Phi_6(x) = (x^2 + x + 1)(x^2 - x + 1)$$

Let $Q(x, y)$ be a fixed polynomial as in Algorithm 1, for $k = 1$. The main idea of the algorithm works as follows. Using procedure FindPrimeQ we find two different large primes $q_1 = Q(a, b)$ and $\widetilde{q_1} = Q(\widetilde{a}, \widetilde{b})$ for some $a, b, \widetilde{a}, \widetilde{b}$. Next, we find residues $r_1 \pmod{q_1}$ and $\widetilde{r_1} \pmod{\widetilde{q_1}}$ using procedures Find RestModuloQ $(a, b, q_1, 1, 3)$ and FindRestModuloQ$(\widetilde{a}, \widetilde{b}, \widetilde{q_1}, 1, 6)$ respectively. Then above procedures return r_1 and $\widetilde{r_1}$ such that $\Phi_3(r_1) \equiv 0 \pmod{q_1}$ and $\Phi_6(\widetilde{r_1}) \equiv 0 \pmod{\widetilde{q_1}}$.

Table 1. Timing results of algorithms generating prime p and the prime divisors of $\Phi_3(p^2)$. The table on the left-hand side contains the timing results of Algorithm 4 for $m = 3$ and $k = 2$. Algorithm 4 generates primes p and q_i such that q_1, q_2 divide $\Phi_3(p)$ and q_3, q_4 divide $\Phi_6(p)$. The table on the right-hand side contains the timing results of Giuliani and Gong Algorithm [8]. The algorithm finds primes p and q_1, q_2 dividing $\Phi_3(p)$ and $\Phi_6(p)$ respectively.

n	♯ searched	time (sec)	p	q_1	q_2	q_3	q_4
38	69	1.451	161	75	231	74	230
42	89	2.310	177	82	259	82	257
46	88	2.907	193	90	283	90	281
50	119	4.546	210	99	307	98	309
54	154	7.215	226	106	333	106	330
58	143	7.864	242	114	356	114	356
62	197	13.208	259	122	382	123	381
66	213	16.361	275	130	405	130	405
70	209	19.616	291	139	428	138	431
74	252	25.905	307	146	455	145	453
78	326	37.215	324	154	476	154	476

♯ searched	time (sec)	p	q_1	q_2
131	4.163	160	305	308
152	6.142	176	334	335
149	6.796	192	373	374
181	10.297	208	399	398
229	14.469	224	436	430
231	18.210	240	458	462
313	26.699	256	491	499
319	32.778	272	534	530
408	46.281	288	559	563
402	55.79	304	593	592
607	89.726	320	625	623

Table 2. Timing results of algorithms generating prime p and the prime divisor of $\Phi_6(p)$ which are key parameters for the XTR cryptosystem. The table on the left-hand side contains the timing results of Algorithm 1 for $m = 6$ and $k = 1$. The table on the right-hand side contains the timing results of Lenstra and Verheul Algorithm [13].

n	time (sec)	p	q
100	0.045	211	205
200	0.361	412	405
300	1.014	611	604
400	2.588	813	805
500	6.780	1020	1012
600	13.862	1213	1205
700	23.629	1409	1403

time (sec)	p	q
0.047	210	203
0.371	411	403
1.469	611	604
4.977	812	802
12.056	1015	1007
21.295	1213	1203
39.558	1412	1404

Table 3. Timing results of Algorithm 4 for $m = 3$ and $k = 4$. The algorithm finds a prime p such that $\Phi_3(p^4) = \Phi_3(p)\Phi_6(p)\Phi_{12}(p)$ and primes q_1, q_2 dividing $\Phi_3(p)$, primes q_3, q_4 dividing $\Phi_6(p)$ and primes q_5, q_6 dividing $\Phi_{12}(p)$.

bits n	♯ searched	time (sec)	p	q_1	q_2	q_3	q_4	q_5	q_6
11	197	10.490	81	20	129	20	127	28	286
14	307	20.101	100	26	159	26	159	34	353
17	956	74.853	120	33	191	34	192	40	428
20	1020	94.845	138	38	222	38	224	46	492
23	1743	189.339	157	44	256	44	256	52	556

Since $(q_1, \widetilde{q_1}) = 1$, we can find, using the Chinese Remainder Theorem, the solution r of the system of congruences

$$\begin{cases} r \equiv r_1 \pmod{q_1} \\ r \equiv \widetilde{r}_1 \pmod{\widetilde{q}_1} \end{cases}$$

The following steps are repeated until we find a *good* prime p. We find a prime $p = q_1\widetilde{q}_1 t_0 + r$ for some $t_0 \in N$. We check if simultaneously $\Phi_3(p)/q_1 = s_1 q_2$ and $\Phi_6(p)/\widetilde{q}_1 = \widetilde{s}_1\widetilde{q}_2$, where $s = \prod p_i^{e_i}$, for certain $p_i \in \mathcal{B}$, $\widetilde{s} = \prod p_j^{e_j}$, for some $p_j \in \mathcal{B}$ and $q_2, \widetilde{q}_2$ are large primes. If the condition is not true, we find the next large prime $p = q_1\widetilde{q}_1 t_1 + r$ for a certain $t_1 \in N$, $t_1 > t_0$ and we check again if $\Phi_3(p^k)/q_1$ and $\Phi_6(p)/\widetilde{q}_1$ are of the desired form. For more details see Algorithm 4 in the appendix.

7 Numerical Test

We have implemented the algorithms presented in this paper. They were implemented in C++ on LINUX server using LiDIA [22] Library For Computational Number Theory. The bound $B = 2^{16}$ was tested. To enable fair comparison, we implemented the algorithm proposed by Giuliani, Gong [8] and Lenstra Verheul [13]. The column labeled "♯ searched" contains the number of primes p searched before the prime factorization of $\Phi_m(p^k)$ was found. Columns labeled "n", "p" and "q_i" contain the number of bits of n, p and q_i respectively.

References

1. Agrawal, M., Kayal, K., Saxena, N.: Primes is P. Ann. of Math. 160, 781–793 (2004)
2. Bach, E., Shallit, J.: Algorithmic Number Theory. Efficient Algorithms, vol. I. MIT Press, Cambridge (1996)
3. Bateman, P.T., Horn, R.A.: A Heuristic Asymptotic Formula Concerning the Distribution of Prime Numbers. Math. Comp. 16, 119–132 (1962)
4. Cohen, H.: A Course in Computational Algebraic Number Theory. Springer, Heidelberg (1995)
5. Davenport, H.: Multiplicative Number Theory. Springer, New York (1980)
6. Gong, G., Harn, L.: Public-Key Cryptosystems Based on Cubic Finite Field Extension. IEEE IT 45(7), 2601–2605 (1999)
7. Gong, G., Harn, L.: A New Approach on Public-key Distribution. ChinaCRYPT, pp. 50–55 (1998)
8. Giuliani, K., Gong, G.: Generating Large Instances of the Gong-Harn Cryptosytem. In: Proceedings of Cryptography and Coding: 8th International Conference Cirencester. LNCS, vol. 2261. Springer, Heidelberg (2002)
9. Giuliani, K., Gong, G.: Analogues to the Gong-Harn and XTR Cryptosystem. Combinatorics and Optimization Research Report CORR 2003-34, University of Waterloo (2003)
10. Heath-Brown, D.R.: Almost-primes in Arithmetic Progression and Short Intervals. Proc. London Proc. Cambridge Phil. Soc. 83, 357–375 (1978)
11. Heath-Brown, D.R.: Zero-free Regions for Dirichlet L-Functions and the Least Prime in an Arithmetic Progressions. Proc. London Math. Soc. 64(3), 265–338 (1992)
12. Iwaniec, H.: Primes Represented by Quadratic Polynomials in Two Variables. Acta Arith. 24, 435–459 (1974)
13. Lenstra, A.K., Verhuel, E.R.: The XTR Public Key System. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 1–19. Springer, Heidelberg (2000)
14. Lenstra, A.K., Verheul, E.R.: Fast Irreducibility and Subgroup Membership Testing in XTR. In: Kim, K.-c. (ed.) PKC 2001. LNCS, vol. 1992, pp. 73–86. Springer, Heidelberg (2001)
15. Müller, S.: On the Computation of Cube Roots Modulo p. In: High Primes and Misdemeanours: lectures in honour of the 60th birthday of Hugh Cowie Williams. Fields Institute Communications Series, vol. 41 (2004)
16. Rabin, M.O.: Probabilistic Algorithm for Testing Primality. J. Number Theory 12, 128–138 (1980)
17. Rubin, K., Silverberg, A.: Using Primitive Subgrups to Do More with Fewer Bits. In: Buell, D.A. (ed.) ANTS 2004. LNCS, vol. 3076, pp. 18–41. Springer, Heidelberg (2004)
18. Rubin, K., Silverberg, A.: Torus-based cryptography. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 349–365. Springer, Heidelberg (2003)
19. Schinzel, A., Sierpiński, W.: Sur Certaines Hypothèses Concernant Les Nombres Premiers. Acta Arith. 4, 185–208 (1956)
20. Wagstaff, S.: Greatest of the Least Primes in Arithmetic Progressions Having a Given Modulus. Math. Comp. 33, 1073–1080 (1979)
21. Williams, K., Hardy, K.: A Refinement of H. C. Williams' qth Root Algorithm. Math. Comp. 61, 475–483 (1993)
22. `http://www.cdc.informatik.tu-darmstadt.de/TI/LiDIA/`

Appendix

Algorithm 3. Finds a prime p and prime factors of $\Phi_m(p^k)$

input: $m = 3$ and $k = 1, 3$ or $m = 6$ and $k = 1, 2, 3, 4, 6$, $n, B \in \mathbf{N}$, $\mathcal{B}$.
output: primes p, q_1, q_2 and $p_i \leq B$ such that $q_1 q_2 \prod p_i^{e_i} = \Phi_m(p^k)$ for some $p_i \in \mathcal{B}$.

1: **if** $m = 6$ and $k = 2, 4, 6$ **then**
2: $F(x, y) = 36k^2x^2 + 36k^2y^2 + 12ky + 1$ ▷ See Algorithm 1
3: **else**
4: $F(x, y) \leftarrow Ax^2 + Bxy + Cy^2 + Ex + Fy + G$ ▷ See Algorithm 1
5: **end if**
6: FindPrimeQ$(n, F(x, y))$ ▷ Return prime q_1 and $a, b : q_1 = F(a, b)$

7: **if** $m = 3, 6$ and $k = 1, 3$ **then**
8: FindRestModuloQ(a, b, k, q_1) ▷ Return $r \pmod{q_1}$
9: **else**
10: FindRestModuloQTwo(a, b, k, q_1) ▷ Return $r \pmod{q_1}$
11: **end if**

12: **procedure** FindPrimeFactors(q_1)
13: $t \leftarrow 0$
14: $b \leftarrow false$
15: **while not** b **do**
16: FindPrimeModuloQ(r, q_1, t) ▷ Return k and prime p
17: $q_2 \leftarrow \Phi_m(p^k)/q_1$
18: **if not** $IsPrime(q_2)$ **then**
19: **for all** $p_i \in \mathcal{B}$ **do**
20: **if** $p_i^{e_i} | q_2$ **then** ▷ Let e_i be the highest power of p_i dividing q_2
21: $q_2 \leftarrow q_2/p_i^{e_i}$
22: store (p_i, e_i) in table $\widetilde{\mathcal{B}}$
23: **end if**
24: **if** $IsPrime(q_2)$ **then**
25: $b \leftarrow true$
26: **goto** line 34:
27: **end if**
28: **end for**
29: **else**
30: $b \leftarrow true$
31: **end if**
32: $t \leftarrow t + 1$
33: **end while**
34: **return** $(p, q_1, q_2, \widetilde{\mathcal{B}})$
35: **end procedure**

Algorithm 4. Finds a prime p and prime factors of $\Phi_3(p^k)$

input: $k = 2, 4, 6$, $n, B \in \mathbf{N}$, $\mathcal{B}$, l be the number of irreducible factors of the $\Phi_m(x^k)$.
output: primes p, q_i, $\widetilde{q}_i$, $i = 1, \ldots, l$ and $p_j \leq B$ such that $s \prod_{i=1}^{l} q_i \widetilde{q}_i = \Phi_m(p^k)$, where $s = \prod p_j^{e_j}$ for some $p_j \in \mathcal{B}$.

1: **if** $k = 2$ **then**
2: $l \leftarrow 2$; $F_1(x, y) = F_2(x, y) \leftarrow Q(x, y)$ ▷ See Algorithm 1
3: **end if**
4: **if** $k = 4$ **then**
5: $l \leftarrow 3$; $F_1(x, y) = F_2(x, y) \leftarrow Q(x, y)$
6: $F_3(x, y) \leftarrow F(x, y)$ ▷ See Algorithms 1 and 2
7: **end if**
8: **if** $k = 6$ **then**
9: $l \leftarrow 2$; $F_1(x, y) = F_2(x, y) \leftarrow F(x, y)$ ▷ See Algorithms 2
10: **end if**

11: **for** $i \leftarrow 1, l$ **do**
12: FindPrimeQ$(n, F_i(x, y))$ ▷ Return prime q_i and $a_i, b_i : q_i = F(a_i, b_i)$
13: **if** $k = 2, 4$ and $i = 1, 2$ **then**
14: FindRestModuloQ(a_i, b_i, k, q_i) ▷ Return $r_i \pmod{q_i}$
15: **else**
16: FindRestModuloQTwo(a_i, b_i, k, q_i) ▷ Return $r_i \pmod{q_i}$
17: **end if**
18: **end for**
19: $q \leftarrow \prod_{i=1}^{l} q_i$
20: *Find* $r \pmod{q} : r \equiv r_i \pmod{q_i}$ ▷ Use the Chinese Remainder Theorem

21: **procedure** FindPrimeFactors(q)
22: FindPrimeModuloQ(r, q, t) ▷ Return k and prime p
23: $t \leftarrow 0$
24: $b \leftarrow false$
25: **while not** b **do**
26: **for** $i \leftarrow 1, l$ **do**
27: $\widetilde{q}_i \leftarrow \Phi_m(p^k)/q_i$
28: **end for**
29: **if not** $IsPrime(\widetilde{q}_1, \ldots, \widetilde{q}_l)$ **then** ▷ q_i, where $i = 1 \ldots l$
30: **for all** i, such that $\widetilde{q}_i$ is not prime **do**
31: **for all** $p_j \in \mathcal{B}$ **do**
32: **if** $p_j^{e_j} | \widetilde{q}_i$ **then** ▷ Let e_j be the highest power of p_j dividing $\widetilde{q}_i$
33: $\widetilde{q}_i \leftarrow \widetilde{q}_i / p_j^{e_j}$
34: store (p_j, e_j) in table $\widetilde{\mathcal{B}}$
35: **end if**
36: **if** $IsPrime(\widetilde{q}_1, \ldots, \widetilde{q}_l)$ **then**
37: $b \leftarrow true$
38: **goto** line 47:
39: **end if**
40: **end for**
41: **end for**
42: **else**
43: $b \leftarrow true$
44: **end if**
45: $t \leftarrow t + 1$
46: **end while**
47: **return** $(p, q_1, \ldots, q_l, \widetilde{q}_1, \ldots, \widetilde{q}_l, \widetilde{\mathcal{B}})$
48: **end procedure**

Chosen Ciphertext Secure Public Key Encryption with a Simple Structure

Goichiro Hanaoka[1], Hideki Imai[1,2],
Kazuto Ogawa[3], and Hajime Watanabe[1]

[1] RCIS, AIST, Japan
[2] Chuo University, Japan
[3] Japan Broadcasting Corporation, Japan

Abstract. In this paper, we present a new public key encryption scheme with an easy-to-understand structure. More specifically, in the proposed scheme, for fixed group elements $g_1, \ldots, g_\ell$ in the public key a sender computes only $g_1^r, \ldots, g_\ell^r$ for encryption where r is a single random number. Due to this simple structure, its security proof becomes very short (and one would easily understand the simulator's behavior for simultaneously dealing with embedding a hard problem and simulating a decryption oracle). Our proposed scheme is provably chosen-ciphertext secure under the gap Diffie-Hellman assumption (without random oracles). A drawback of our scheme is that its ciphertext is much longer than known practical schemes. We also propose a modification of our scheme with improved efficiency.

1 Introduction

1.1 Background

Chosen-ciphertext security (CCA-security, for short) [15, 31] is a standard notion of security for practical public key encryption (PKE) schemes. Furthermore, this security also implies universally composable security [9]. Among existing CCA-secure PKE schemes, ECIES [1] (see Appendix A) has a simple and interesting structure. Namely, (i) its encryption algorithm calculates only g_1^r and g_2^r with a common random r where g_1 and g_2 are *fixed* bases which are contained in the public key, and (ii) its security can be proven with the *gap* Diffie-Hellman (GDH) assumption [28] (in the random oracle model [2]). Due to this simple structure, we can construct a practical PKE scheme, and more importantly, it is easy, especially for non-experts, to understand the basic mechanism for handling CCA adversaries (in the random oracle model).

The main motivation of this paper is to construct a PKE scheme which provides the above two properties in the standard model (i.e. without using random oracles).

1.2 Our Contribution

In this paper, we propose a novel PKE scheme which has similar properties to ECIES. Specifically, in the proposed scheme, for fixed group elements $g_1, \ldots, g_\ell$

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 20–33, 2008.

in the public key a sender computes only $g_1^r, \ldots, g_\ell^r$ for encryption where r is a random, and CCA-security of the proposed scheme can be proven under the GDH assumption.

Due to the simple structure of our proposed scheme (which is similar to ECIES), one can easily understand its essential idea for dealing with chosen-ciphertext attacks. For example, CCA-security of the proposed scheme is *optimally* (i.e. without any security loss) reduced to indistinguishability of a hardcore bit of the (gap) Diffie-Hellman key. In other words, in its security proof, the simulator can perfectly respond to any decryption query without any error probability. Especially, for starters our scheme would give an easy-to-understand insight for designing CCA-secure PKE schemes. Specifically, in the security proof, the simulator can perfectly respond to any decryption query by simply un-mask one of its components with an ordinary exponentiation (not, for example, the Boneh-Boyen-like technique [5]). Namely, since in the simulation one of components of a decryption query must form as K^a where K is the answer of the query and a is a secret for the simulation, the simulator can easily extract the answer K as $K = (K^a)^{\frac{1}{a}}$. It is also easy to see how the given instance of the underlying hard problem is embedded in the challenge ciphertext.

Unfortunately, this scheme is not practical as it has only one-bit plaintext space and its ciphertext consists of $k + 2$ group elements (plus one bit) where k is a security parameter, and hence our scheme has nothing advantageous to existing practical PKE schemes in all practical aspects. In particular, the Kiltz scheme [25] is more efficient than our scheme in terms of both data sizes and computational costs with the same underlying assumption, i.e. the GDH assumption.[1] However, we stress that the main contribution of the proposed scheme is its easy-to-understand structure for protecting chosen-ciphertext attacks.

We also give some extensions of the proposed scheme for enhancing efficiency. By using these ideas, we can significantly reduce data sizes, and especially the ciphertext overhead becomes only two group elements which is the same as [25] (however, key sizes are still much longer than [25]).

1.3 Related Works

The first CCA-secure PKE scheme was proposed by Dolev, Dwork, and Naor [15] by extending the Naor-Yung paradigm [27] which is only non-adaptively CCA-secure. However, this scheme has a complicated structure primarily due to the use of *non-interactive zero knowledge* (NIZK) proof [4].

Cramer and Shoup [13] proposed the first practical CCA-secure scheme under the DDH assumption. This scheme was further improved by Shoup [33] and Kurosawa and Desmedt [26]. Furthermore, Hofheinz and Kiltz [23] showed a variant of the Cramer-Shoup scheme with a weaker assumption, i.e. the *n-linear* DDH assumption.

[1] In [25], security of the Kiltz scheme is discussed mainly based on the gap hashed Diffie-Hellman (GHDH) assumption instead of the standard GDH assumption. However, as (slightly) mentioned in [25], this scheme is also provably CCA-secure under the GDH assumption if the plaintext is only one-bit long.

Canetti, Halevi, and Katz [11] proposed a generic method for converting an (selectively secure) identity-based encryption scheme [6, 32] into a CCA-secure PKE scheme, and Boneh and Katz [7] improved its efficiency. Kiltz [24] discussed a more relaxed condition for achieving CCA-security. Boyen, Mei, and Waters [8] proposed practical CCA-secure schemes by using the basic idea of the Canetti-Halevi-Katz paradigm and specific properties of [35] and [5].

The above schemes, i.e. [11, 13] and their extensions, utilize powerful cryptographic tools such as subset membership problems [14] or identity-based encryption [6, 32] for *efficiently* achieving CCA-security. Therefore, in these schemes minimum functionality for achieving CCA-security seems unclear.

Kiltz [25] also proposed another practical CCA-secure scheme whose security is proven under the *gap hashed Diffie-Hellman* (GHDH) assumption. With a slight modification (by using the hard-core bit), this scheme is also provably secure under the GDH assumption. Our proposed scheme is considered as a redundant version of this scheme with an easier-to-understand structure.

In the random oracle methodology [2], it is possible to construct more efficient CCA-secure schemes, e.g. [1, 3, 17, 18, 29] (though this methodology is known as problematic [10]). Among these schemes, ECIES [1] has a very simple structure on which one can easily understand its essential mechanism for protecting chosen-ciphertext attacks (in the random oracle model). Namely, in ECIES it is impossible to properly encrypt a plaintext without submitting its corresponding Diffie-Hellman key to a random oracle, and therefore, a simulator can respond to any decryption query by simply picking it from random oracle queries (and one can correctly choose the valid Diffie-Hellman key with the help of the DDH oracle). See Appendix A for ECIES. Similarly to this, in our proposed scheme, we can easily understand that for generating a valid ciphertext a CCA adversary has to (implicitly) input the corresponding Diffie-Hellman key into redundant components of the ciphertext, and that the simulator can extract it from such redundant components. See also Sec. 6 for a more detailed description on this observation.

2 Definitions

Here, we give definitions for CCA-security of PKE schemes and some number theoretic assumptions, e.g. the GDH assumption. See also Appendix C for target collision resistant hash functions and data encapsulation mechanisms (DEMs).

2.1 Public Key Encryption

The Model. A public key encryption (PKE) scheme consists of the following three algorithms:

Setup(1^k)**.** Takes as input the security parameter 1^k and outputs a decryption key dk and a public key PK.

Encrypt(PK, M)**.** Takes as input a public key PK and a plaintext $M \in \mathcal{M}$, and outputs a ciphertext ψ.

Decrypt(dk, ψ, PK)**.** Takes as input the decryption key dk, a ciphertext ψ, and the public key PK, and outputs the plaintext $M \in \mathcal{M}$ or a special symbol "$\perp$".

We require that if $(dk, PK) \stackrel{R}{\leftarrow}$ **Setup**(1^k) and $\psi \stackrel{R}{\leftarrow}$ **Encrypt**(PK, M) then **Decrypt**$(dk, \psi, PK) = M$.

Chosen-Ciphertext Security. CCA-security of a PKE scheme is defined using the following game between an attack algorithm A and a challenger. Both the challenger and A are given 1^k as input.

Setup. The challenger runs **Setup**(1^k) to obtain a decryption key dk and a public key PK, and gives PK to A.

Query I. Algorithm A adaptively issues decryption queries $\psi_1, \ldots, \psi_m$. For query ψ_i, the challenger responds with **Decrypt**(dk, ψ_i, PK).

Challenge. At some point, A submits a pair of plaintexts $(M_0, M_1) \in \mathcal{M}^2$. Then, the challenger picks a random $b \in \{0, 1\}$, runs algorithm **Encrypt** to obtain the challenge ciphertext $\psi^\star \stackrel{R}{\leftarrow}$ **Encrypt**(PK, M_b), and gives $\psi^\star$ to A.

Query II. Algorithm A continues to adaptively issue decryption queries ψ_{m+1}, $\ldots, \psi_{q_D}$. For query $\psi_i (\neq \psi^\star)$, the challenger responds as **Query I**.

Guess. Algorithm A outputs its guess $b' \in \{0, 1\}$ for b and wins the game if $b = b'$.

Let $\mathsf{AdvPKE}_\mathsf{A}$ denote the probability that A wins the game.

Definition 1. We say that a PKE scheme is (τ, ϵ, q_D) *CCA-secure* if for all τ-time algorithms A who make a total of q_D decryption queries, we have that $|\mathsf{AdvPKE}_\mathsf{A} - 1/2| < \epsilon$.

2.2 Number Theoretic Assumptions

The Gap (Hashed) Diffie-Hellman Assumption. Let $\mathbb{G}$ be a multiplicative group with prime order p. Then, the *gap Diffie-Hellman* (GDH) problem in $\mathbb{G}$ is stated as follows. Let A be an algorithm, and we say that A has advantage ϵ in solving the GDH problem in $\mathbb{G}$ if

$$\Pr[\mathsf{A}^{\mathcal{O}}(g, g^\alpha, g^\beta) = g^{\alpha\beta}] \geq \epsilon,$$

where the probability is over the random choice of generators g in $\mathbb{G}$, the random choice of α and β in $\mathbb{Z}_p$, and the random bits consumed by A. The oracle $\mathcal{O}$ denotes the DDH oracle which on input $(g_1, g_2, g_3, g_4) \in \mathbb{G}^4$, answers whether $\log_{g_1} g_3 = \log_{g_2} g_4$ or not, and A is allowed to access $\mathcal{O}$ in any time and any order.

Definition 2. *We say that the (τ, ϵ)-GDH assumption holds in $\mathbb{G}$ if no τ-time algorithm has advantage at least ϵ in solving the GDH problem in $\mathbb{G}$.*

Occasionally we drop the τ and ϵ and refer to the GDH in $\mathbb{G}$.

The *gap hashed Diffie-Hellman* (GHDH) problem [25] in $\mathbb{G}$ and function $h : \mathbb{G} \to \mathcal{D}$ is stated as follows. Let A be an algorithm, and we say that A has advantage ϵ in solving the GHDH problem in $\mathbb{G}$ and h if

$$\frac{1}{2} \cdot |\Pr[\mathsf{A}^{\mathcal{O}}(g, g^{\alpha}, g^{\beta}, h(g^{\alpha\beta})) = 0] - \Pr[\mathsf{A}^{\mathcal{O}}(g, g^{\alpha}, g^{\beta}, T) = 0]| \geq \epsilon,$$

where the probability is over the random choice of generators g in $\mathbb{G}$, the random choice of α and β in $\mathbb{Z}_p$, the random choice of $T \in \mathcal{D}$, and the random bits consumed by A. The oracle $\mathcal{O}$ is the DDH oracle.

Definition 3. *We say that the (τ, ϵ)-GHDH assumption holds in $\mathbb{G}$ and h if no τ-time algorithm has advantage at least ϵ in solving the GHDH problem in $\mathbb{G}$ and h.*

Occasionally we drop the τ and ϵ and refer to the GHDH in $\mathbb{G}$ and h.

Hardcore Bits for the (Gap) Diffie-Hellman Key. Roughly speaking, h is called a *hardcore bit* function for the (gap) Diffie-Hellman key in $\mathbb{G}$ if the GHDH assumption in $\mathbb{G}$ and h holds under only the GDH assumption in $\mathbb{G}$.

Let A be a τ-time algorithm which has advantage ϵ in solving the GHDH problem in $\mathbb{G}$ and $h : \mathbb{G} \to \{0, 1\}$.

Definition 4. *We say that function $h : \mathbb{G} \to \{0, 1\}$ is a (p_1, p_2) hardcore bit function in $\mathbb{G}$ if there exists a $p_1(\tau)$-time algorithm B which for any given A, can solve the GDH problem with advantage $p_2(\epsilon)$ for some polynomials p_1 and p_2.*

See [19] for an example of hardcore bit functions for the (gap) Diffie-Hellman key.

3 The Proposed Scheme

In this section, we give the construction of our proposed scheme (its extensions with enhanced efficiency are given in the next section). As a basic scheme, we start with the standard ElGamal PKE scheme [16]. Since semantic security of the ElGamal scheme is based on only the DDH assumption (not the GDH assumption), with the use of hardcore bits we modify it to have semantic security based on the GDH assumption. See Appendix B for this semantically secure PKE scheme. Next, we further modify this scheme to have CCA-security by using the Dolev-Dwork-Naor paradigm [15]. Interestingly, in our proposed scheme, NIZK proofs which is required for [15] is not necessary due to the GDH assumption.[2]

[2] In general, for applying the technique of [15] we need NIZK proofs which make the resulting scheme complicated, however in our scheme it is not needed since the DDH oracle in the GDH assumption provides the necessary functionality for proving equivalence of plaintexts of different ciphertexts without neither any further computational assumption nor any additional ciphertext redundancy.

3.1 The Construction

Let $\mathbb{G}$ be a multiplicative group with prime order p, and $g \in \mathbb{G}$ be a generator. We assume that a group element of $\mathbb{G}$ is k-bit long where k is the security parameter. Then, the construction of our proposed PKE scheme is as follows:

Setup(1^k)**:** Pick $dk = (x_0, \ldots, x_k, y_0, \ldots, y_k, z) \in \mathbb{Z}_p^{2k+3}$, and compute $X_i = g^{x_i}$, $Y_i = g^{y_i}$, and $Z = g^z$ for $i = 0, \ldots, k$. The decryption key is dk, and the public key is $PK = (\mathbb{G}, g, X_0, \ldots, X_k, Y_0, \ldots, Y_k, Z, h)$ where h is a hardcore bit function in $\mathbb{G}$.

Encrypt(PK, M)**:** For a plaintext $M \in \{0,1\}$, pick a random $r \stackrel{R}{\leftarrow} \mathbb{Z}_p$, and compute

$$\psi = (g^r, U_0^r, \ldots, U_k^r, h(Z^r) \oplus M),$$

where $U_i = X_i$ if $v_i = 0$, or $U_i = Y_i$ if $v_i = 1$, v_i is $(i+1)$-th bit of $((g^r)_2 \| (h(Z^r) \oplus M))$, and $(W)_2$ denotes the binary representation of $W \in \mathbb{G}$. The ciphertext is ψ.

Decrypt(dk, ψ, PK)**:** For a ciphertext $\psi = (C_0, C_{1,0}, \ldots, C_{1,k}, C_2)$, check whether for all $i = 0, \ldots, k$, $C_0^{u_i} \stackrel{?}{=} C_{1,i}$ where $u_i = x_i$ if $v_i = 0$, or $u_i = y_i$ if $v_i = 1$ where v_i is $(i+1)$-th bit of $((C_0)_2 \| C_2)$. If not, output $\perp$. Otherwise, output $M = C_2 \oplus h(C_0^z)$.

As earlier mentioned, the required operation for encryption in the proposed scheme is only exponentiations with a common exponent r under fixed bases (which are contained in PK).

3.2 Security

Before going into a formal security proof of the proposed scheme, we consider the (in)security of its simplified scheme which would be helpful for understanding the essential part of the proposed scheme. The (insecure) simplified scheme is as follows: Suppose that the decryption key is reduced to be only $dk = (x_0, z)$ and the public key is $PK = (\mathbb{G}, g, X_0, Z, h)$. Let a ciphertext ψ be $(g^r, X_0^r, h(Z^r) \oplus M)$ for a plaintext $M \in \{0,1\}$. Then, if the component "X_0^r" is valid (this can be checked by testing the consistency with $(g^r)^{x_0}$), the receiver would be convinced that the sender knows r since without knowing r (nor x_0), it seems hard to generate the Diffie-Hellman key $g^{r \cdot x_0}$. However, this is false. Namely, once an adversary sees a valid (challenge) ciphertext (C_0, C_1, C_2), he can generate another valid ciphertext $(C_0^{r'}, C_1^{r'}, C_2)$ without knowing $\log_g C_0^{r'}$, where $r' \in \mathbb{Z}_p$ is a random. In other words, this scheme is *malleable*, and hence, insecure.

Our proposed scheme is considered as an enhanced version of the above scheme with non-malleability by using a similar technique to [15]. More specifically, in the proposed scheme, for each encryption the sender is enforced to choose a distinct subset from $\{X_0, \ldots, X_k, Y_0, \ldots, Y_k\}$ (instead of single X_0), and therefore, the above attack does not work any more. Consequently, the resulting scheme becomes CCA-secure.

The security of the above scheme is formally addressed as follows:

Theorem 1. *Let $\mathbb{G}$ be a multiplicative group with prime order p, and h be a (p_1, p_2) hardcore bit function in $\mathbb{G}$. Then, the above scheme is $(p_1^{-1}(\tau - o(\tau)), p_2^{-1}(\epsilon_{gdh}), q_D)$ CCA-secure assuming the (τ, ϵ_{gdh}) GDH assumption holds in $\mathbb{G}$.*

Proof. Assume we are given an adversary A which breaks CCA-security of the above scheme with running time τ, advantage ϵ, and q_D decryption queries. We use A to construct another adversary B which, by using the DDH oracle, distinguishes hardcore bit h of the (gap) Diffie-Hellman key in $\mathbb{G}$. This suffices for proving CCA-security of our scheme under the GDH assumption since existence of an algorithm which distinguishes a hardcore bit of the gap Diffie-Hellman key immediately implies existence of another algorithm which solves the GDH problem by the definition. Define adversary B as follows:

1. For a given GDH instance (g, g^α, g^β), B picks a random bit γ, and $(a_0, \ldots, a_k, b_0, \ldots, b_k) \in \mathbb{Z}_p^{2k+2}$. Let $v_i^\star$ be $(i+1)$-th bit of $((g^\beta)_2 \| \gamma)$ for $0 \leq i \leq k$.
2. B sets $Z = g^\alpha$, and $(X_i, Y_i) = (g^{a_i}, (g^\alpha)^{b_i})$ if $v_i^\star = 0$, or $(X_i, Y_i) = ((g^\alpha)^{a_i}, g^{b_i})$ if $v_i^\star = 1$, for $i = 0, \ldots, k$.
3. B inputs public key $PK = (\mathbb{G}, g, X_0, \ldots, X_k, Y_0, \ldots, Y_k, Z)$ and challenge ciphertext $\psi^\star = (g^\beta, (g^\beta)^{\mu_0}, \ldots, (g^\beta)^{\mu_k}, \gamma)$ to A where $\mu_i = a_i$ if $v_i^\star = 0$, or $\mu_i = b_i$ if $v_i^\star = 1$ for $i = 0, \ldots, k$.
   Notice that $\psi^\star$ is a valid ciphertext for plaintext $\gamma \oplus h(g^{\alpha\beta}) \in \{0, 1\}$. We also note that since the pair of plaintexts (M_0, M_1) which are challenged is always $(0, 1)$, without loss of generality B may give $\psi^\star$ to A at this stage.
4. When A makes decryption query $\psi = (C_0, C_{1,0}, \ldots, C_{1,k}, C_2) \in \mathbb{G}^{k+2} \times \{0, 1\}$ (if a query is not in this form, then B simply rejects it), B proceeds as follows:
   (a) B determines a binary string $(v_i)_{0 \leq i \leq k} = ((C_0)_2 \| C_2)$, and checks whether for all $i = 0, \ldots, k$,
   $$\log_g U_i \stackrel{?}{=} \log_{C_0} C_{1,i}$$
   by using the DDH oracle (See Sec. 2.2), where $U_i = X_i$ if $v_i = 0$, or $U_i = Y_i$ if $v_i = 1$. If ψ is in an invalid form, then B responds with "$\perp$".
   (b) If ψ is in a valid form, then B picks (one of) i such that $v_i \neq v_i^\star$. (We note that there always exists at least one such i if $\psi \neq \psi^\star$.) B also picks $C_{1,i}$ and calculates $C_{1,i}^{1/a_i} = C_0^\alpha$ if $v_i = 0$, or $C_{1,i}^{1/b_i} = C_0^\alpha$ if $v_i = 1$. B responds with $M' = C_2 \oplus h(C_0^\alpha)$ to A.
5. Finally, A outputs his guess b' on $\gamma \oplus h(g^{\alpha\beta})$, and B outputs $b' \oplus \gamma$ as his guess on $h(g^{\alpha\beta})$.

Obviously, the above simulation is *always* perfect without even *any* negligible error probability. Therefore, B's advantage is completely the same as A's, i.e. ϵ.

4 A Comparison on "Easiness-to-Understand"

In this section, we discuss "easiness-to-understand" of our proposed scheme by comparing it with the Kiltz scheme [25] and the Dolev-Dwork-Naor scheme [15] by focusing on their techniques for responding to decryption queries.

In our proposed scheme, for responding to a query the simulator picks a component from the query, and un-masks it by an ordinary exponentiation. Then, the simulator responds to the query with this result.

In the Kiltz scheme [25], there is only one redundant component in a ciphertext (which is practical and beautiful), and the simulator always picks it for extracting the answer of the query. However, since this construction utilizes the Boneh-Boyen-like technique [5], it requires more complicated and non-intuitive calculation than a single exponentiation. See, for example, Eq. (1) in the full version of [25] for the required calculation.

In the Dolev-Dwork-Naor scheme [15], similarly to our scheme, the simulator picks one of components of a decryption query, and simply decrypts this component ciphertext. This is due to that the NIZK proof guarantees that all decryption results of all component ciphertexts are identitcal. Therefore, this scheme is also considered easy-to-understand if *we do not mind using an NIZK proof as a black box*. However, actually NIZK proofs require the Karp reduction from the specific NP language (i.e. equality of plaintexts) to some NP complete language, and furthermore, the assumption of existence of enhanced trapdoor permutations is also necessary. Therefore, the full description of the scheme with a concrete construction of the NIZK proof becomes more complicated than our scheme.

5 Extensions

In this section, we give some ideas for enhancing efficiency of the proposed scheme. However, the structure of the resulting scheme becomes much less easy-to-understand.

5.1 Compressing Keys

It is possible to compress the size of keys by using target collision resistant hash functions (see Appendix C). Specifically, by replacing the vector $(v_i)_{0\le i\le k} = ((C_0)_2 \| C_2)$ with another vector $(v'_i)_{0\le i\le \ell-1} = \mathsf{TCR}(C_0, C_2)$ where $\mathsf{TCR} : \mathbb{G} \times \{0,1\} \to \{0,1\}^\ell$ is a target collision resistant hash function, sizes for both decryption and public keys are reduced by approximately $\ell/k \simeq 1/2$.[3]

5.2 Compressing Ciphertexts

Interestingly, our strategy of the security proof still works even if we compress the redundant components of a ciphertext by a product as

$$\psi = (g^r, \left(\prod_{v_i=0} X_i \cdot \prod_{v_i=1} Y_i\right)^r, h(Z^r) \oplus M).$$

The ciphertext overhead (i.e. ciphertext size minus plaintext size) of the resulting scheme becomes only two group elements, which is the same as the best known schemes, e.g. [8, 25].

[3] For ℓ-bit security, the size of a group element is required to be at least 2ℓ-bit long, while the size of an output of TCR is required to be at least ℓ-bit long.

5.3 Expanding the Plaintext Space

Similarly to [25], we can expand the size of plaintexts for arbitrary length with the GHDH assumption. Specifically, we replace the hardcore bit function $h : \mathbb{G} \to \{0,1\}$ with another hash function $h' : \mathbb{G} \to \{0,1\}^\nu$ where ν is sufficiently large, and assume the GHDH assumption holds in $\mathbb{G}$ and h'. Then, by encrypting a plaintext with CCA-secure DEM [20, 21, 22, 30] under the data encryption key $h'(Z^r)$ (instead of the simple one-time pad). See Appendix C for CCA-secure DEMs.

5.4 Applying the above Extensions Together

Here, we give a concrete construction of an enhanced version of our proposed scheme with all of the above extensions. Let $\mathbb{G}$ be a multiplicative group with prime order p, and $g \in \mathbb{G}$ be a generator. Let $\mathsf{TCR} : \mathbb{G} \to \{0,1\}^\ell$ be a target collision resistant hash function, and $h : \mathbb{G} \to \{0,1\}^\ell$ be a hash function (such that the GHDH assumption holds in $\mathbb{G}$ and h).[4] Let (E, D) be a CCA-secure DEM. Then, the construction of the enhanced scheme is as follows:

Setup(1^k)**:** Pick $dk = (x_0, \ldots, x_{\ell-1}, y_0, \ldots, y_{\ell-1}, z) \in \mathbb{Z}_p^{2\ell+1}$, and compute $X_i = g^{x_i}$, $Y_i = g^{y_i}$, and $Z = g^z$ for $i = 0, \ldots, \ell-1$. The decryption key is dk, and the public key is

$$PK = (\mathbb{G}, g, X_0, \ldots, X_{\ell-1}, Y_0, \ldots, Y_{\ell-1}, Z, h, \mathsf{TCR}).$$

Encrypt(PK, M)**:** For a plaintext $M \in \mathcal{M}$, pick a random $r \xleftarrow{R} \mathbb{Z}_p$, and compute

$$\psi = \left(g^r, \left(\prod_{v_i=0} X_i \cdot \prod_{v_i=1} Y_i\right)^r, \mathsf{E}(h(Z^r), M)\right),$$

where v_i is $(i+1)$-th bit of $\mathsf{TCR}(g^r)$. The ciphertext is ψ.

Decrypt(dk, ψ, PK)**:** For a ciphertext $\psi = (C_0, C_1, C_2)$, check whether

$$\prod_{v_i=0} (C_0^{x_i}) \cdot \prod_{v_i=1} (C_0^{y_i}) \stackrel{?}{=} C_1,$$

where v_i is $(i+1)$-th bit of $\mathsf{TCR}(C_0)$. If not, output $\perp$. Otherwise, output $M = \mathsf{D}(h(C_0^z), C_2)$.

Theorem 2. *Let $\mathbb{G}$ be a multiplicative group with prime order p, TCR be a (τ, ϵ_{tcr}) target collision resistant hash function, and (E, D) be a (τ, ϵ_{dem}) CCA-secure DEM. Then, the above PKE scheme is $(\tau - o(\tau), \epsilon_{ghdh} + \epsilon_{tcr} + \epsilon_{dem}, q_D)$ CCA-secure assuming the (τ, ϵ_{ghdh}) GHDH assumption holds in $\mathbb{G}$ and h.*

The proof of the theorem is given in the full version of this paper.

The above scheme is much more efficient than the basic scheme in the previous section, and especially its ciphertext overhead is only two group elements which is the same as the best known schemes [8, 25]. However, this scheme is not very easy-to-understand any more, and furthermore, is still less efficient than [25] in all practical aspects.

[4] For ℓ-bit security, length of outputs for both TCR and h are determined as ℓ-bit long.

6 Random Oracle Model vs. Standard Model

Here, we give a brief comparison of our proposed scheme with ECIES by focusing on their methods for simulating decryption oracles (under the GDH assumption). This comparison would clarify an essential difference between the random oracle model and the standard model. In both cases the main issue is that for given g^r and g^x which are contained in a decryption query and a public key, respectively, the simulator has to somehow produce $g^{r \cdot x}$ without knowing x nor r.

In the security proof of ECIES, since the data encryption key is set as $K = H(g^{r \cdot x})$ where H is a random oracle, one cannot properly encrypt a plaintext without submitting $g^{r \cdot x}$ to H. Hence, $g^{r \cdot x}$ can be extracted from H-queries. We notice that $g^{r \cdot x}$ is not embedded in the ciphertext, and consequently this trick does not require ciphertext redundancy which results in a very short ciphertext. See Appendix A for ECIES.

On the other hand, in the security proof of our proposed scheme (and the Kiltz scheme [25]), the simulator cannot observe the CCA adversary's inputs to h, and therefore, we have to enforce the CCA adversary to embed $g^{r \cdot x}$ (with a mask) into some redundant part (i.e. $(C_{1,0}, \ldots, C_{1,k})$ in ψ) of the ciphertext instead. The simulator extracts (masked) $g^{r \cdot x}$ from such a redundant part by using an incomplete decryption key. Hence, we see that it is difficult to remove redundancy of a ciphertext in the standard model.

Construction of redundancy free CCA-secure PKE schemes under reasonable assumptions in the standard model is a major open problem, and even in a relaxed notion of CCA-security, only the DDH-based scheme in [12] achieves it.

Acknowledgement

The authors would like to thank Reynald Affeldt for his invaluable comments and discussions. The authors also would like to thank Takahiro Matsuda, SeongHan Shin and Rui Zhang for their helpful comments and suggestions.

References

1. Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer, Heidelberg (2001)
2. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Proc. of CCS 1993, pp. 62–73 (1993)
3. Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 92–111. Springer, Heidelberg (1995)
4. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In: Proc. of STOC 1988, pp. 103–112 (1988)
5. Boneh, D., Boyen, X.: Efficient selective-ID secure identity-based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004)

6. Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
7. Boneh, D., Katz, J.: Improved efficiency for CCA-secure cryptosystems built using identity-based encryption. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 87–103. Springer, Heidelberg (2005)
8. Boyen, X., Mei, Q., Waters, B.: Direct chosen ciphertext security from identity-based techniques. In: Proc. of CCS 2005, pp. 320–329 (2005)
9. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: Proc. of FOCS 2001, pp. 136–145 (2001)
10. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. In: Proc. of STOC 1998, pp. 209–218 (1998)
11. Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 207–222. Springer, Heidelberg (2004)
12. Cramer, R., Hanaoka, G., Hofheinz, D., Imai, H., Kiltz, E., Pass, R., shelat, A., Vaikuntanathan, V.: Bounded CCA2-secure encryption. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 502–518. Springer, Heidelberg (2007)
13. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998)
14. Cramer, R., Shoup, V.: Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 45–64. Springer, Heidelberg (2002)
15. Dolev, D., Dwork, C., Naor, M.: Non-malleable cryptography. In: Proc. of STOC 1991, pp. 542–552 (1991)
16. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. on Inform. Theory 31(4), 469–472 (1985)
17. Fujisaki, E., Okamoto, T.: How to enhance the security of public-key encryption at minimum cost. In: Imai, H., Zheng, Y. (eds.) PKC 1999. LNCS, vol. 1560, pp. 53–68. Springer, Heidelberg (1999)
18. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (1999)
19. Goldreich, O., Levin, L.A.: A hard-core predicate for all one-way functions. In: Proc. of STOC 1989, pp. 25–32 (1989)
20. Halevi, S.: EME*: extending EME to handle arbitrary-length messages with associated data. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 315–327. Springer, Heidelberg (2004)
21. Halevi, S., Rogaway, P.: A tweakable enciphering mode. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 482–499. Springer, Heidelberg (2003)
22. Halevi, S., Rogaway, P.: A parallelizable enciphering mode. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 292–304. Springer, Heidelberg (2004)
23. Hofheinz, D., Kiltz, E.: Secure hybrid encryption from weakened key encapsulation. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 553–571. Springer, Heidelberg (2007)
24. Kiltz, E.: Chosen-ciphertext security from tag-based encryption. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 581–600. Springer, Heidelberg (2006)

25. Kiltz, E.: Chosen-ciphertext secure key-encapsulation based on gap hashed Diffie-Hellman. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 282–297. Springer, Heidelberg (2007), http://eprint.iacr.org/2007/036
26. Kurosawa, K., Desmedt, Y.: A new paradigm of hybrid encryption scheme. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 426–442. Springer, Heidelberg (2004)
27. Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: Proc. of STOC 1990, pp. 427–437 (1990)
28. 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)
29. Okamoto, T., Pointcheval, D.: REACT: rapid enhanced-security asymmetric cryptosystem transform. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 159–175. Springer, Heidelberg (2001)
30. Phan, D.H., Pointcheval, D.: About the security of ciphers (semantic security and pseudo-random permutations). In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 182–197. Springer, Heidelberg (2004)
31. Rackoff, C., Simon, D.R.: Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 433–444. Springer, Heidelberg (1992)
32. 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)
33. Shoup, V.: Using hash functions as a hedge against chosen ciphertext attack. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 275–288. Springer, Heidelberg (2000)
34. Shoup, V.: A proposal for an ISO standard for public key encryption (version 2.1) (manuscript, 2001)
35. Waters, B.: Efficient identity based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)

A A Brief Review of ECIES

Here, we give a brief review of ECIES [1]. Let $\mathbb{G}$ be a multiplicative group with prime order p, and $g \in \mathbb{G}$ be a generator, and $H : \mathbb{G} \to \{0,1\}^\ell$ be a hash function, which will be viewed as a random oracle in the security analysis. Let (E, D) is a CCA-secure DEM (see Appendix C). Then, the construction of ECIES is as follows:

Setup(1^k)**:** Pick $z \stackrel{R}{\leftarrow} \mathbb{Z}_p$, and compute $Z = g^z$. The decryption key is z, and the public key is $PK = (\mathbb{G}, g, Z, H)$.

Encrypt(PK, M)**:** For a plaintext $M \in \mathcal{M}$, pick a random $r \stackrel{R}{\leftarrow} \mathbb{Z}_p$, and compute $\psi = (g^r, \mathsf{E}(H(Z^r), M))$. The ciphertext is ψ.

Decrypt(dk, ψ, PK)**:** For a ciphertext $\psi = (C_0, C_1)$, output $M = \mathsf{D}(H(C_0^z), C_1)$.

Proposition 1 ([34]). *Let $\mathbb{G}$ be a multiplicative group with prime order p, H be a random oracle, and (E, D) be a (τ, ϵ_{dem}) CCA-secure DEM. Then, the above scheme is $(\tau - o(\tau), \epsilon_{gdh} + \epsilon_{dem}, q_D)$ CCA-secure assuming the (τ, ϵ_{gdh}) GDH assumption holds in $\mathbb{G}$.*

An intuitive explanation of the security is as follows: Due to the standard KEM/DEM composition theorem [33], it is sufficient to prove that the KEM part of the above scheme is CCA-secure (since the DEM part is already CCA-secure). Therefore, for proving security, by using a CCA adversary A against the KEM, we will construct another algorithm B which solves the GDH problem.

For a given GDH instance (g, g^α, g^β), B sets $PK = (\mathbb{G}, g, g^\alpha, H)$, where random oracle H is controlled by B, and $C_0^\star = g^\beta$ as a challenge ciphertext. PK and $(C_0^\star, K^\star)$ is given to A, where $K^\star \stackrel{R}{\leftarrow} \{0,1\}^\ell$ and A's goal is to correctly guess if $K^\star = H(g^{\alpha\beta})$ or not. For a decryption query C_0, by using the DDH oracle $\mathcal{O}$, B searches an H query w which has been submitted by A such that (g, g^α, C_0, w) forms a Diffie-Hellman tuple. B returns $H(w)$ if there is such an H query, or a random $K_{C_0} \in \{0,1\}^\ell$ otherwise. Similarly, for an H query w, by using $\mathcal{O}$, B searches a decryption query C_0 which has been submitted by A such that (g, g^α, C_0, w) forms a Diffie-Hellman tuple. B responds as $H(w) = K_{C_0}$ if there is such a decryption query, or $H(w) \stackrel{R}{\leftarrow} \{0,1\}^\ell$ otherwise. B keeps the above answers in his memory, and returns the same answer for the same query. Since it is information-theoretically impossible to distinguish $H(g^{\alpha\beta})$ from a random ℓ-bit string without submitting $g^{\alpha\beta}$ to H, A submits it to random oracle H at some point. Therefore, B can also output $g^{\alpha\beta}$ with non-negligible probability by picking it from A's H queries (and B can correctly select it with the help of $\mathcal{O}$).

B The ElGamal PKE Scheme with the GDH Assumption

Let $\mathbb{G}$ be a multiplicative group with prime order p, and $g \in \mathbb{G}$ be a generator. Then, the construction of the GDH-based semantically secure PKE scheme is as follows:

Setup(1^k)**:** Pick $z \stackrel{R}{\leftarrow} \mathbb{Z}_p$, and compute $Z = g^z$. The decryption key is $dk = z$, and the public key is $PK = (\mathbb{G}, g, Z, h)$ where h is a hardcore bit function in $\mathbb{G}$.

Encrypt(PK, M)**:** For a plaintext $M \in \{0,1\}$, pick a random $r \stackrel{R}{\leftarrow} \mathbb{Z}_p$, and compute $\psi = (g^r, h(Z^r) \oplus M)$. The ciphertext is ψ.

Decrypt(dk, ψ, PK)**:** For a ciphertext $\psi = (C_0, C_1)$, output $M = C_1 \oplus h(C_0^z)$.

Theorem 3. *Let $\mathbb{G}$ be a multiplicative group with prime order p, and h be a (p_1, p_2) hardcore bit function in $\mathbb{G}$. Then, the above scheme is $(p_1^{-1}(\tau - o(\tau)), p_2^{-1}(\epsilon_{gdh}), 0)$ CCA-secure (i.e. semantically secure) assuming the (τ, ϵ_{gdh}) GDH assumption holds in $\mathbb{G}$.*

Proof. Assume we are given an adversary A which breaks semantic security of the above scheme with running time τ and advantage ϵ. We use A to construct another adversary B which distinguishes hardcore bit h of the Diffie-Hellman key in $\mathbb{G}$. This suffices for proving semantic security of the above scheme under the GDH assumption. Define adversary B as follows: For a given GDH instance (g, g^α, g^β), B sets $PK = (\mathbb{G}, g, g^\alpha, h)$ and $\psi^\star = (g^\beta, \gamma)$ where γ is a random bit. PK and $\psi^\star$ are given to A. Finally, A outputs his guess b' on $\gamma \oplus h(g^{\alpha\beta})$, and B outputs $b' \oplus \gamma$ as his guess on $h(g^{\alpha\beta})$.

Obviously, we can also prove semantic security of the above scheme under the computatinal Diffie-Hellman assumption since in the above proof the DDH oracle is never used. However, for enhancing it to have CCA-security we need the DDH oracle.

C Cryptographic Tools

C.1 Target Collision Resistant Hash Functions

Let $\mathsf{TCR} : \mathcal{X} \to \mathcal{Y}$ be a hash function, A be an algorithm, and A's advantage $\mathsf{AdvTCR}_{\mathsf{A}}$ be

$$\mathsf{AdvTCR}_{\mathsf{A}} = \Pr[\mathsf{TCR}(x') = \mathsf{TCR}(x) \in \mathcal{Y} \wedge x' \neq x |\ x \stackrel{R}{\leftarrow} \mathcal{X};\ x' \stackrel{R}{\leftarrow} \mathsf{A}(x)].$$

Definition 5. We say that TCR is a (τ, ϵ) *target collision resistant hash function* if for all τ-time algorithm A, we have that $\mathsf{AdvTCR}_{\mathsf{A}} < \epsilon$.

It is obvious that any injective mapping can be used as a perfectly secure target collision resistant hash function.

C.2 Data Encapsulation Mechanism

The Model. A data encapsulation mechanism (DEM) scheme consists of the following two algorithms:

$\mathsf{E}(K, M)$ Takes as input a data encryption key $K \in \mathcal{K}$ and a plaintext $M \in \mathcal{M}$, and outputs a ciphertext ψ.

$\mathsf{D}(K, \psi)$ Takes as input a data encryption key $K \in \mathcal{K}$ and a ciphertext ψ, and outputs the plaintext $M \in \mathcal{M}$.

We require that if $\psi \leftarrow \mathsf{E}(K, M)$ then $\mathsf{D}(K, \psi) = M$.

Chosen-Ciphertext Security. CCA-security of a DEM is defined using the following game between an attack algorithm A and a challenger. Both the challenger and A are given 1^k as input.

Setup. The challenger chooses a data encryption key $K \in \{0,1\}^k$.

Query I. Algorithm A adaptively issues decryption queries $\psi_1, \ldots, \psi_m$. For query ψ_i, the challenger responds with $\mathsf{D}(K, \psi_i)$.

Challenge. At some point, A submits a pair of plaintexts $(M_0, M_1) \in \mathcal{M}^2$. Then, the challenger picks a random $b \in \{0,1\}$, runs algorithm E to obtain the challenge ciphertext $\psi^\star \leftarrow \mathsf{E}(K, M_b)$, and give $\psi^\star$ to A.

Query II. Algorithm A continues to adaptively issue decryption queries ψ_{m+1}, $\ldots, \psi_{q_D}$. For query $\psi_i (\neq \psi^\star)$, the challenger responds as **Query I**.

Guess. Algorithm A outputs its guess $b' \in \{0,1\}$ for b and wins the game if $b = b'$.

Let $\mathsf{AdvDEM}_{\mathsf{A}}$ denote the probability that A wins the game.

Definition 6. We say that a DEM is (τ, ϵ, q_D) *CCA-secure* if for all τ-time algorithms A who make a total of q_D decryption queries, we have that $|\mathsf{AdvDEM}_{\mathsf{A}} - 1/2| < \epsilon$.

Remarks on the Attack of Fouque et al. against the ℓIC Scheme

Naoki Ogura and Shigenori Uchiyama

Tokyo Metropolitan University
ogura-naoki@ed.tmu.ac.jp, uchiyama-shigenori@tmu.ac.jp

Abstract. In 2007, ℓ-Invertible Cycles (ℓIC) was proposed by Ding et al. This is one of the most efficient trapdoors for encryption/signature schemes, and of the mixed field type for multivariate quadratic public-key cryptosystems. Such schemes fit on the implementation over low cost smart cards or PDAs. In 2008, Fouque et al. proposed an efficient attack against the ℓIC signature scheme by using Gröbner basis algorithms. However, they only explicitly dealt with the odd case, i.e. ℓ is odd, but the even case; they only implemented their proposed attack in the odd case. In this paper, we propose an another practical attack against the ℓIC encryption/signature scheme. Our proposed attack does not employ Gröbner basis algorithms, and can be applied to the both even and odd cases. We show the efficiency of the attack by using some experimental results. Furthermore, the attack can be also applied to the ℓIC- scheme. To the best of our knowledge, we for the first time show some experimental results of a practical attack against the ℓIC- scheme for the even case.

1 Introduction

In 2007, Ding, Wolf and Yang [4] proposed ℓ-Invertible Cycles (ℓIC), which is a trapdoor for public-key encryption/signature schemes, and also proposed ℓIC-, which is a signature scheme as a variation of it. The security of these schemes is based on the hardness of solving simultaneous multivariate quadratic equations (MQ system) over a finite field whose characteristic is two; we call the problem "MQ problem". The problem of deciding whether an MQ system has a solution or not belongs to NP-complete, and quantum polynomial algorithms for solving the MQ problem are still unknown. It has been expected that public-key cryptosystems based on NP-complete problems like the MQ problem will replace the schemes based on the integer factorization or discrete logarithm problems which will be efficiently solved by quantum computers. Moreover, the schemes based on the MQ problem are very practical under limited computational resources, such as smart cards or PDAs. The ℓIC scheme is more efficient than Quartz[3] under the same security level.

In 2007, Dubois, Fouque, Shamir and Stern [8] proposed a practical attack against SFLASH which is called C^{*-} scheme with specific parameters. The ℓIC-signature scheme is similar to SFLASH. In 2008, Fouque, Gilles, Perret and Stern proposed an efficient attack against the ℓIC signature scheme by using Gröbner

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 34–48, 2008.

basis algorithms. However, they only considered the cryptanalysis under some condition; namely, they only considered the case that a secret-key as a pair of two transformations S, T is a pair of linear maps not affine ones. Also, they only explicitly dealt with the odd case, i.e. ℓ is odd, but the even case; they only implemented their proposed attack in the odd case.

In this paper, we propose an another practical attack against the ℓIC encryption/signature scheme. Our proposed attack does not employ Gröbner basis algorithms, and can be applied to the both even and odd cases. We show the efficiency of the attack by using some experimental results under the most general condition that a secret-key as a pair of two transformations S, T is a pair of affine ones. To the best of our knowledge, we for the first time show some experimental results of a practical attack against the ℓIC- signature scheme for the even case.

This paper is organized as follows. In Section 2, we will briefly introduce ℓ-Invertible Cycles. In Section 3, 4, we will propose the attack against ℓIC. In Section 5, we will show our experimental results. In Section 6, we will conclude this paper.

2 ℓ-Invertible Cycles

In this section, we will describe about public-key cryptosystems based on the multivariate quadratic equations (Multivariate Quadratic public-key Cryptography: MQPKC), and ℓIC (ℓ-Invertible Cycles) and ℓIC- schemes.

2.1 MQPKC

We will briefly review about public-key cryptosystems based on the MQ problem. Matsumoto and Imai[16] proposed the first MQPKC scheme. In the last few decades, there has been enormous amount of work devoted to this area. Most public-key cryptosystems based on the MQ problem use a trapdoor function below. For two vector spaces K^{n_1} and K^{n_2} over a finite field $K = \mathbb{F}_q$ whose characteristic is 2, we construct a public-key as a map $P : K^{n_1} \to K^{n_2}$. Each scheme has the map $F : K^{n_1} \to K^{n_2}$ whose inverse images are efficiently computable. This map, which is constructed by the n_2 quadratic polynomials with n_1 variables, is called "central map." We generate a secret-key as a pair of two bijective affine transformations (S, T). We call these transformations "secret transformations." Of course, the inverse images of secret transformations is easy to compute if these transformations are known. Finally, the map $P = T \circ F \circ S$ is made to public. Since it is believed that solving the MQ problem is intractable, we may assume that it is difficult to compute an inverse image for the public-key if secret transformations are unknown.

For the encryption scheme use, Alice (a sender) sends a ciphertext $C = P(m)$, where m is a plaintext and P is Bob's public-key. Bob (a receiver) obtains $m = P^{-1}(C)$, where $P^{-1} = S^{-1} \circ F^{-1} \circ T^{-1}$. For the signature scheme use, on the other hand, Alice (a signer) generates a signature $\sigma = P^{-1}(m)$, where m is a message and P is Alice's public-key. Bob (a verifier) checks whether $m = P(\sigma)$ or not.

Many central maps are classed as four basic types below.

- MIA (Matsumoto-Imai Scheme A, C^*)[16], [18]
- HFE (Hidden Field Equations)[21]
- UOV (Unbalanced Oil and Vinegar)[20], [17]
- STS (Stepwise Triangular Systems)[15], [25]

Although these schemes are efficient, some of them are broken. So some modifiers have been proposed for enforcing schemes against known attacks. For more information about the modifiers, [24] is very useful for us. Ding et al. [4] recommended three modifiers for ℓIC schemes below.

- Minus [23]
- Internal Perturbed Plus [21], [7], [6], [5]
- Embedding [4], similar to Fixing [2]

We will describe ℓIC-, which is a variation of ℓIC with the Minus method later.

2.2 ℓIC

We will show the central map based on ℓIC scheme [4]. Let $\ell \geq 2$ be a divisor of $n := n_1 = n_2$. Define L by an n/ℓ-dimensional extended field of $K = \mathbb{F}_q$. An L-vector space L^ℓ can be identified to a K-vector space K^n with a fixed basis over K. Fix the values $(\lambda_1, \ldots, \lambda_\ell) \in \{0, \ldots, n/\ell - 1\}^\ell$. Then, the central map of ℓIC scheme is

$$F : L^\ell \to L^\ell;\ (A_1, \ldots, A_\ell) \mapsto (A_1{}^{q^{\lambda_1}} A_2, \ldots, A_\ell{}^{q^{\lambda_\ell}} A_1)\ . \tag{1}$$

In addition, in [4], the concrete values of λ_i were proposed due to efficiency of computation of the inverse image:

$$\lambda_1 = \begin{cases} 0\ (\ell : \text{odd}) \\ 1\ (\ell : \text{even}) \end{cases},\ \lambda_i = 0\ (2 \leq i \leq \ell) \tag{2}$$

In the case when ℓ is even, the condition that $q = 2$ (i.e. $K = \mathbb{F}_2$), are required for bijectivity of the central map. Then, for $A = (A_1, \ldots, A_\ell)$, the explicit central map of these schemes is given as follows.

$$F(A) = \begin{cases} (A_1A_2,\ A_2A_3, \ldots, A_\ell A_1)\ (q : \text{ a power of two}) & \text{for } \ell : \text{odd} \\ (A_1^2A_2,\ A_2A_3, \ldots, A_\ell A_1) \qquad (q = 2) & \text{for } \ell : \text{even} \end{cases} \tag{3}$$

We call these scheme odd-IC scheme and even-IC scheme, respectively.

2.3 ℓIC-

In [4], some schemes with modifiers were also proposed due to enforcing an ℓIC scheme. The ℓIC- scheme is the scheme applied the Minus method to the ℓIC scheme. Select an integer $r < n$, and define a projection map $\Pi : K^n \to K^{n-r}$. This scheme uses not P but $\Pi P = \Pi \circ P$ as a public-key. This modification

prevents attackers from breaking the scheme by using bijectivity of P. Because of filling the requirements for security, r has to be some big integer. So this scheme is used for not an encryption but a signature scheme.

Ding et al. [4] recommended a signature scheme based on a 3IC- scheme. But they did not mention about specific construction of the scheme. We will describe the possible concrete construction of this scheme below. This construction is the same as that of SFLASH [1].

Secret-key

1. Generate the bijective affine transformation $S : K^n \to K^n$. To be more precise, generate randomly an n-dimensional non-singular matrix S_L and an n-dimensional vector S_C over K, and set a function $S(x) = S_L x + S_C$.
2. Similarly, generate the affine transformation $T : K^n \to K^n$ (i.e. a matrix T_L and a vector T_C).
3. Generate a random binary string Δ (enough long to satisfy security requirements).

Let a secret-key be $(S,\ T,\ \Delta)$.

Public-key
Construct a map $P = T \circ F \circ S$, where F is the bilinear map over $(L^*)^\ell$ below.

$$F((A_1,\ A_2,\ A_3)) = (A_1A_2,\ A_2A_3,\ A_3A_1)\ (A_i \in L) \tag{4}$$

Recall that K^n is identified to L^ℓ. By a projection $\Pi : K^n \to K^{n-r}$, let a public-key be $\Pi \circ P$.

Signing
We regard the binary strings with length $\lg q$ as the elements over K.

1. By applying a hash function to a message, generate $V \in K^{n-r}$.
2. By applying a hash function to the concatenation between V and Δ, generate $W \in K^r$.
3. By concatenating between V and W, generate $B \in K^n$. Let a signature be $P^{-1}(B) = S^{-1}(F^{-1}(T^{-1}(B)))$.

Verification

1. By applying the hash function to the message, generate $Y \in K^{n-r}$.
2. Verify that Y corresponds with the element generated by applying $\Pi \circ P$ to the signature.

3 Attack against ℓIC

In this section, we will give an attack against ℓIC schemes.

Table 1. The algorithm of an attack against odd-IC scheme

Input: a public-key P, parameters q, n, ℓ, a message m (resp. a ciphertext c)
Output: the valid signature against m (resp. the plaintext for c)
{(Reduction to linear transformation)}
$\{\nu_j\} \leftarrow$ Linear part of P
$\{\zeta_{ij}\} \leftarrow$ Quadratic (not squared) part of P
$v$space $\leftarrow$ Kernel of $\{\nu_j\}$
$v \leftarrow$ A vector of the $v$space satisfying equation(6)
$P_L(x) := P(x - v) - P(-v)$
{(Forging (resp. Deciphering))}
$\{(\gamma^{(k)}, \gamma'^{(k)})\} \leftarrow \{(\gamma^{(k)}, \gamma'^{(k)})\}$ satisfying forging equations (11)
$V \leftarrow$ The element over K^n generated by applying hash function to m (resp. c)
$y \leftarrow V - P(-v)$
$x$space $\leftarrow$ Space of x satisfying forging equations (11) for y
$x \leftarrow$ A vector of the $x$space satisfying $y = P_L(x)$
return $x - v$

3.1 Summary of Attack against ℓIC

In this subsection, we will show a brief sketch of our attack against ℓIC. The Table 1 shows an algorithm of breaking both a signature scheme (forging) and an encryption scheme (deciphering) based on odd-IC schemes. The attack will be applied to other ℓIC schemes. The attack against ℓIC consists of 2 steps below.

- Reduction to linear transformation
- Forging a signature(resp. Deciphering)

In this attack, we employ a property of the easily invertible central map F, which is taken over a public-key P. The MQ problem is reduced to the linear system problem.

3.2 Reduction to Linear Transformation

Some attacks against MQPKC need that S and T are not only the affine but the linear transformations. The reduction attacks in the cases of SFLASH [14], [13] and HFE [10] were proposed. This process is used for the attack by using a differential against the Minus method. Moreover, this reduction is also useful for our attack against ℓIC schemes. We will see in the next section.

As is well known, a polynomial is called a quadratic form or a homogeneous polynomial if all terms of it have degree 2. As for ℓIC, each coordinate of F is a quadratic form. We will see how to use this property. The aim of this attack is to find $v = (v_1, \ldots, v_n) := S_L^{-1} \cdot S_c$. Since each coordinate of F is a quadratic form, that of $P_L(x) := P(x - v) - P(-v)$ is also the quadratic form, whose coordinate consists of only quadratic (including squared) part of that of P. Here, define the differential[12] by a bilinear function below. This function is useful for cryptanalysis against the schemes based on the MQ problem.

$$DP(a,\ b) := P(a + b) - P(a) - P(b) + P(0) \tag{5}$$

We will describe in detail the way of using the differential for an attack against the Minus method later. By definition of DP, the equation below can be easily shown.

$$-DP(x,\ -v) = P(x) - P_L(x) - P(0) \tag{6}$$

Because the right hand of (6) consists of only linear part of P, it is efficiently computable from the public-key P. Then, we can compute v by solving the linear equations created from (6). Since $(a,\ P(a))$ corresponds to $(b = a + v,\ P_L(b) = P(a) - P(-v))$, the function P_L, whose S and T are linear, give information about the public-key P.

3.3 Forging Equation

We will adapt the attack against C^* scheme[22] to ℓIC scheme. Because of concise representation of index, let μ be the function below for each $i \in \mathbb{Z}$. Note that this symbol is different from [4]'s one.

$$\mu(i) := \begin{cases} \ell & (\ell \mid i) \\ i \mod \ell & (\text{otherwise}) \end{cases} , \tag{7}$$

where $\mod$ express the least non-negative residue. This residue is called the "least positive residue".

The obvious relation below is derived between $(B_1, \ldots, B_\ell) = F((A_1, \ldots, A_\ell))$ $= (A_1 {}^{q^{\lambda_1}} A_2, \ldots, A_\ell {}^{q^{\lambda_\ell}} A_1)$.

$$A_{\mu(i+2)} B_i {}^{q^{\lambda_{\mu(i+1)}}} - A_i {}^{q^{\lambda_i + \lambda_{\mu(i+1)}}} B_{\mu(i+1)} = 0 \tag{8}$$

Let y be $P(x)$. Then, $T^{-1}(y) = F(S(x))$, where S and T are some bijective affine transformation. By applying (8) to this equation, the simple relation below is found.

$$\sum_{1 \le i,j \le n} {\gamma_{ij}}^{(k)} x_i y_j + \sum_{i=1}^{n} ({\alpha_i}^{(k)} x_i + {\beta_i}^{(k)} y_i) + \delta^{(k)} = 0 \ , \tag{9}$$

where ${\gamma_{ij}}^{(k)},\ {\alpha_i}^{(k)},\ {\beta_i}^{(k)},\ \delta^{(k)} \in K$. This relation induces linear equations between x and $y(= P(x))$. We call this relation **forging equation**.

Here, let us examine the number of linear independent equations, i.e. the rank of the coefficient matrix. In the case of even-IC scheme, almost all y makes full rank system. So we can easily forge signatures based on even-IC schemes by using equations (9). In the case of odd-IC scheme, On the other hand, unfortunately an linear dependent equation exists because $\lambda_1 = \lambda_2 = \cdots = \lambda_\ell = 0$. Thus, (9) gives us only $(n - k)$-dimensional equations, so we need exhaustive search from k-dimensional space.

Odd Case. In this subsection, we will give different relation from (8). Put $(B_1, \ldots, B_\ell) = F((A_1, \ldots, A_\ell)) = (A_1 A_2, \ldots, A_\ell A_1)$ for an odd-IC scheme. Using the technique of computation of inverse images shown in [4], we can obtain the relation below.

$$\prod_{j=0}^{(\ell-1)/2} B_{\mu(i+2j)} = {A_i}^2 \prod_{j=0}^{(\ell-1)/2-1} B_{\mu(i+2j+1)} \tag{10}$$

Note that the map $A_i \mapsto {A_i}^2$ is bijective since A_i is in K, whose characteristic is two. Like the previous subsubsection, for $y = P(x)$, we can find the relation $q_1(y_1, \ldots, y_n) = q_2(x_1, \ldots, x_n, y_1, \ldots, y_n)$, where the total degree of the polynomial q_1 and q_2 is $(\ell + 1)/2$.

Now we show an advantage of linearity of S and T. By applying (10) to $y = P(x)$ in this case, more simple relation below is found.

$$\sum_{|\boldsymbol{w_1}|=(\ell+1)/2} {\gamma_{\boldsymbol{w_1}}}^{(k)} y_{\boldsymbol{w_1}} = \sum_{i=1}^{n} \sum_{|\boldsymbol{w_2}|=(\ell-1)/2} {\gamma'_{i\boldsymbol{w_2}}}^{(k)} {x_i}^2 y_{\boldsymbol{w_2}} \ , \tag{11}$$

where we define $y_{\boldsymbol{w}} = y_{c_1} \cdots y_{c_{|\boldsymbol{w}|}}$ for $y = (y_1, \ldots, y_n) \in K^n$ and $\boldsymbol{w} = (c_1, \ldots, c_{|\boldsymbol{w}|})$ such that $1 \leq c_i \leq n$. For example, in the case that $\ell = 3$, the equation below is induced.

$$\sum_{1 \leq i_1 \leq i_2 \leq n} {\gamma_{i_1 i_2}}^{(k)} y_{i_1} y_{i_2} = \sum_{1 \leq i,j \leq n} {\gamma'_{ij}}^{(k)} {x_i}^2 y_j \tag{12}$$

Thus the reduction attack in Section 3.2 has the advantage that we can save computational space.

4 Attack against ℓIC-

We will describe an attack against ℓIC-, which is a Minus variation of ℓIC.

4.1 Summary of Attack Against ℓIC-

In this section, we will show a brief sketch of our attack against ℓIC- schemes. The Table 2 shows an algorithm of breaking signature schemes based on odd-IC$^-$ scheme. Remember that ℓIC- scheme is used for a signature scheme only (not an encryption one). The attack against ℓIC- consists of 3 steps below.

- Reduction to linear transformation
- Recovering the deleted part by using a differential
- Forging a signature

This attack is summarized as follows. First of all, the secret affine transformations S and T are reduced to the linear transformation. Secondly, we recover the part which is deleted by the projection in the time of transforming ℓIC to ℓIC-. Finally, we can apply the attack against ℓIC.

The first and third step were already showed at the previous section. In the next subsection, we will explain details of the second step.

Table 2. The algorithm of an attack against odd-IC$^-$ scheme

Input: a public function ΠP, parameters $q,\ n,\ \ell,\ r$, a message m
Output: the valid signature for m

{**(Reduction to linear transformation)**}
$\{\nu_j\} \leftarrow$ Linear part of ΠP
$\{\zeta_{ij}\} \leftarrow$ Quadratic (not squared) part of ΠP
$v$space $\leftarrow$ Kernel of $\{\nu_j\}$
$v \leftarrow$ A vector of the $v$space satisfying equation(6)
$\Pi P_L(x) := \Pi P(x - v) - \Pi P(-v)$
{**(Recovering the deleted part by using a differential)**}
$\Pi DP_L(a,\ x) := \Pi P_L(x + a) - \Pi P_L(x) - \Pi P_L(a)$
Nspace $\leftarrow$ Space of N which is kernel of function (21)
while true do
{**(Forging)**}
$N_\xi \leftarrow$ Regular(not scalar) matrix of Nspace
$P_f \leftarrow$ Function $K_n \rightarrow K_n$(full rank) by adding $(\Pi P_L) \circ N_\xi$ to ΠP_L
γspace $\leftarrow \{(\gamma^{(k)},\ \gamma'^{(k)})\}$ satisfying forging equations (11)
if rank(γspace) $\leq n$ then
$\{(\gamma^{(k)},\ \gamma'^{(k)})\} \leftarrow \gamma$space
break
end if
end while
$V \leftarrow$ The element over K^{n-r} generated by applying hash function to m
$y \leftarrow$ The element over K^n generated by $V - \Pi P(-v)$ (with random padding)
$x$space $\leftarrow$ Space of x satisfying forging equations (11) for y
$x \leftarrow$ A vector of the $x$space satisfying $y = P_L(x)$
return $x - v$

4.2 Recovering with Differential

In [11], ℓIC- schemes were broken. This is an application of the attack against SFLASH [8], [9]. We will describe in detail this attack. In what follows, we assume that secret transformations are the linear transformation.

For $\xi := (\xi_1, \ldots, \xi_\ell) \in L^\ell$, define

$$\begin{aligned} M_\xi : L^\ell &\rightarrow L^\ell \\ (A_1, \ldots, A_\ell) &\mapsto (\xi_1 A_1, \ldots, \xi_\ell A_\ell)\ . \end{aligned} \tag{13}$$

We can check $F \circ M_\xi = M_{F(\xi)} \circ F$. Here, remember that a differential $DF(A,\ B) = F(A+B) - F(A) - F(B) + F(0)$. For $A = (A_1, \ldots, A_\ell)$ and $B = (B_1, \ldots, B_\ell)$, we have the specific form of DF below.

$$DF(A,\ B) = ({A_1}^{q^{\lambda_1}} B_2 + B_1{}^{q^{\lambda_1}} A_2, \ldots, {A_\ell}^{q^{\lambda_\ell}} B_1 + B_\ell{}^{q^{\lambda_\ell}} A_1) \tag{14}$$

So we can see

$$DF(M_\xi(A),\ B) + DF(A,\ M_\xi(B)) = L_\xi(DF(A,\ B))\ , \tag{15}$$

where $L_\xi := M_{({\xi_1}^{q^{\lambda_1}} + \xi_2, \ldots, {\xi_\ell}^{q^{\lambda_\ell}} + \xi_1)}$.

A differential has various properties. If S is linear, the relation between DF and DP is easily shown as follows.

$$DP(a,\ b) = (T \circ DF)(S(a),\ S(b)) \tag{16}$$

Thus, DP, which is generated by a public-key P, takes over properties of DF. Let N_ξ be $S^{-1} \circ M_\xi \circ S$, let N'_ξ be $T \circ L_\xi \circ T^{-1}$. The property (16) gives us the useful equation below.

$$\begin{aligned}(\Pi \circ DP)(N_\xi(a),\ b) + (\Pi \circ DP)(a,\ N_\xi(b)) \\ = \Pi(N'_\xi(DP(a,\ b)))\end{aligned} \tag{17}$$

Let BS_{N_ξ} be the function which represents left side of (17). Then, we obtain the simple relation below.

$$BS_{N_\xi}(a,\ b) = \Pi(N'_\xi(DP(a,\ b))) \tag{18}$$

We will see how to compute N_ξ by using this relation in the next subsubsection.

If once we find N_ξ, we can recover the projected part. By the definition of N_ξ, we can arrive at the useful property as the following.

$$\begin{aligned}(\Pi \circ P)(N_\xi(x)) &= \Pi(T(F(M_\xi \circ S)(x))) \\ &= \Pi((T \circ M_{F(\xi)})(F \circ S)(x))\end{aligned} \tag{19}$$

Then recovering the deleted part by the projection Π is done as stated below.

$$P_f := (P^{(1)}, \ldots,\ P^{(n-r)},\ P^{(1)} \circ N_\xi, \ldots,\ P^{(r)} \circ N_\xi) \tag{20}$$

Note that P_f does not have to correspond with original P because a verifier checks $\Pi(P_f)$ only.

Compute N_ξ on Odd Case. We consider a way of computation of N_ξ. To find a solution of N_ξ, we apply the technique of [8] or [9]. The difference of the attack between [8] and [9] is caused by whether kernel of L_ξ is trivial or not. On the context of odd-IC scheme, by the definition that $L_\xi := M_{(\xi_1+\xi_2,\ldots,\xi_\ell+\xi_1)}$, we can find a non-trivial kernel, which is $\xi_1 = \xi_2 = \cdots = \xi_\ell$. In other words, a solution that the right side of (18) equals 0 exists. This leads us a computation method of N_ξ, i.e. solving kernel of the function below.

$$(\Pi \circ DP)(N_\xi(a),\ b) + (\Pi \circ DP)(a,\ N_\xi(b)) \tag{21}$$

By the cryptanalysis similar to that of [9], we expect that N_ξ is computable under the condition that $r \leq n - 3$.

Compute N_ξ on Even Case. On the other hand, in the case of even-IC scheme, we can find only a trivial kernel. This is because $L_\xi = M_{({\xi_1}^2+\xi_2,\ldots,\xi_\ell+\xi_1)} = 0$ is equivalent to $\xi_1 = \xi_2 = \cdots = \xi_\ell \in \mathbb{F}_2$. So we apply the technique of [8].

Let c_{ij} be the $(i,\ j)$-element of N'_ξ. Then the equation (18) is equivalent to the relation below for $i = 1, \ldots, n-r$.

$$BS^{(i)}_{N_\xi} = \sum_{j=1}^{n} c_{ij} DP^{(j)} \tag{22}$$

We cannot solve directly the equation (22) since $DP^{(n-r+1)}, \ldots, DP^{(n)}$ are included in it. However, we expect that, for some i, $BS^{(i)}_{N_\xi}$ is in the space generated by only $DP^{(j)}$ $(j = 1, \ldots, \underline{n-r})$. We can solve this relations for $i = 1, 2, 3$ and obtain non-trivial N_ξ. Applying the [8]'s analysis to this scheme, N_ξ can be probably induced under the condition that r is up to $(n-2)/3$.[1] Our experimental results implies that this assertion is almost correct.

5 Experimental Results

In this section, we will show some computational results of attacks against ℓIC and ℓIC-. The computational complexity of our attack will be discussed.

5.1 ℓIC Case

The following is a summary of our attack against ℓIC schemes.

1 Reduction to linear transformation
2 Forging a signature (resp. Deciphering)
  2–1 Finding forging equations (9), (11)
  2–2 Forging a signature (resp. Deciphering) by using the forging equations

Note that we start this algorithm directly from step 2 in the case that $K = \mathbb{F}_2$.

Main computation of this algorithm against odd-IC scheme is solving $O(n^{(\ell+1)/2})$-dimensional linear equations over $\mathbb{F}_q$ in step2–1. So the asymptotic computational complexity of this algorithm is

$$O(n^{3(\ell+1)/2} \lg^2 q) \ . \tag{23}$$

Similarly, the complexity against even-IC scheme is $O(n^6 \lg^2 q)$.

We used the computer whose CPU is 2GHz AMD Opteron 246, memory is 4GB, and hard disk is 160GB. Magma[26] was used as a software for writing the program. The Tables 3, 4 shows experimental results of our attack.

Comparing our attack with [11], our attack would not be so efficient. This is because our attack is simple and completely analogous of Patarin's attack [22]. Fouque et al.[11] stated that Patarin's attack is not efficient and used a technique of Gröbner basis. However, the results show that Patarin's attack is still practical and polynomial time under the condition that ℓ is small.

[1] In [8], the idea of breaking SFLASH under the case that $r < n/2$ was also discussed.

Table 3. Experimental Results against oddIC (over $q = 2^8$)

ℓ	3	3	5
n	69	138	35
k	23	46	7
time[s]	1353.1	73814.7	82309.1

Table 4. Experimental Results against evenIC (over $q = 2$)

ℓ	2	2	4	6
n	120	240	240	240
k	60	120	60	40
time[s]	327.1	5630.3	5618.9	5668.3

Table 5. Experimental Results against 3IC- (over $q = 2^8$)

n	30	36	48
k	10	12	16
r	10	12	16
time[s]	34.1	79.3	321.6

Table 6. Experimental Results against 2IC- with linear (over $q = 2$)

n	40	60	80
k	20	30	40
r	10	15	20
time[s]	317.3	2021.3	7725.6

5.2 ℓIC- Case

We will explain about our attack experiments against ℓIC- schemes. In this section, we classify our algorithm into 5 steps below.

1 Reduction to linear transformation
2 Recovering the deleted part by using a differential
  2–1 Finding linear equations using a differential
  2–2 Solving the equations and recovering the part deleted by a projection
3 Forging a signature
  3–1 Finding forging equations (9), (11)
  3–2 Forging a signature by using the forging equations

This method does not ensure whether we obtain non-trivial $N_\xi = S^{-1} \circ M_\xi \circ S$ in step 2–2. So our algorithm checks whether we can find non-trivial forging equations (9) in step 3–1. Computational results show that almost all parameters pass this check.

Once forging equation(9) are found, we can forge a signature by only executing step 3–2, which is an easy task. The computational complexity of this algorithm is the same as that of the ℓIC case above. The main operations of our algorithm are step 2 and step 3–1.

Results of attack experiments against proposed parameter at [4] is shown at the Table 5. We also show some experimental results against 2IC- schemes under the condition that secret transformations are linear at the Table 6.

6 Conclusion

We proposed a practical attack against the ℓIC schemes. This attack can be applied to the ℓIC- signature scheme under the most general condition that secret transformations are affine. Also, oddIC$^-$ schemes (with affine secret transformations) for small ℓ are efficiently broken by our proposed attack. Furthermore, we carried out computational experiments of the attack against 3IC- signature scheme with the recommended parameters in [4] and showed that forging a signature is efficiently computable. Here we note that these attacks wold not be applied directly to solving the all MQ problems. Main emphasis of our attack is that, by using a property of the central map F, the MQ problem is reduced to solving the linear equations. We used the property to obtain forging equations, which is linear relations between the domain and the image of a public function P. Also, the differential DF enables us to fill the deleted part by a projection. Fouque et al.[11] stated that Patarin's attack would not be efficient, so they used Gröbner basis algorithms. On the other hand, our experimental results explicitly showed that Patarin's attack is still practical and polynomial time under the condition that ℓ is small.

References

1. Courtois, N., Goubin, L., Patarin, J.: SFLASH^{v3}, a fast asymmetric signature scheme. Cryptology ePrint (2003), `http://eprint.iacr.org/2003/211`
2. Courtois, N.T.: The security of Hidden Field Equations (HFE). In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 266–281. Springer, Heidelberg (2001)
3. Courtois, N., Goubin, L., Patarin, J.: Quartz, 128-bit long digital signatures. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 298–307. Springer, Heidelberg (2001)
4. Ding, J., Wolf, C., Yang, B.: ℓ-Invertible Cycles for Multivariate Quadratic(MQ) Public Key Cryptography. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 266–281. Springer, Heidelberg (2007)
5. Ding, J., Gower, J.: Inoculating multivariate schemes against differential attacks. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 290–301. Springer, Heidelberg (2006)
6. Ding, J., Schmidt, D.: Cryptanalysis of HFEv and internal perturbation of HFE. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 288–301. Springer, Heidelberg (2005)
7. Ding, J.: A new variant of the Matsumoto-Imai cryptosystem through perturbation. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 305–318. Springer, Heidelberg (2004)

8. Dubois, V., Fouque, P.A., Shamir, A., Stern, J.: Practical Cryptanalysis of SFLASH. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 1–12. Springer, Heidelberg (2007)
9. Dubois, V., Fouque, P.A., Stern, J.: Cryptanalysis of SFLASH with Slightly Modified Parameters. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 264–275. Springer, Heidelberg (2007)
10. Felke, P.: On the Affine Transformations of HFE-Cryptosystems and Systems with Branches. In: Ytrehus, Ø. (ed.) WCC 2005. LNCS, vol. 3969, pp. 229–241. Springer, Heidelberg (2006)
11. Fouque, P.A., Macario-Rat, G., Perret, L., Stern, J.: Total Break of the ℓ-IC Signature Scheme. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 1–17. Springer, Heidelberg (2008)
12. Fouque, P.A., Granboulan, L., Stern, J.: Differential Cryptanalysis for Multivariate Schemes. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 341–353. Springer, Heidelberg (2005)
13. Geiselmann, W., Steinwandt, R.: A short comment on the affine parts of SFLASH^{v3}. Cryptology ePrint (2003), http://eprint.iacr.org/2003/220
14. Geiselmann, W., Steinwandt, R., Beth, Th.: Attacking the Affine Parts of SFLASH. In: Cryptography and Coding, pp. 355–359. Springer, Heidelberg (2001)
15. Goubin, L., Courtois, N.T.: Cryptanalysis of the TTM cryptosystem. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 44–57. Springer, Heidelberg (2000)
16. Imai, H., Matsumoto, T.: Algebraic methods for constructing asymmetric cryptosystems. In: Algebraic Algorithms and Error-Correcting Codes. LNCS, vol. 299, pp. 108–119. Springer, Heidelberg (1985)
17. Kipnis, A., Patarin, J., Goubin, L.: Unbalanced Oil and Vinegar signature schemes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 206–222. Springer, Heidelberg (1999)
18. 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)
19. Patarin, J., Courtois, N., Goubin, L.: FLASH, a Fast Multivariate Signature Algorithm. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 298–307. Springer, Heidelberg (2001)
20. Patarin, J.: The oil and vinegar signature scheme. In: Dagstuhl Workshop on Cryptography, transparencies (1997)
21. Patarin, J.: Hidden Field Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of asymmetric algorithms. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 33–48. Springer, Heidelberg (1996)
22. Patarin, J.: Cryptanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt 1988. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 248–261. Springer, Heidelberg (1995)
23. Shamir, A.: Efficient signature schemes based on birational permutations. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 1–12. Springer, Heidelberg (1994)
24. Wolf, C.: Multivariate Quadratic Polynomials in Public Key Cryptography. Ph.D. Thesis (2005), http://hdl.handle.net/1979/148
25. Wolf, C., Braeken, A., Preneel, B.: Efficient cryptanalysis of RSE(2)PKC and RSSE(2)PKC. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 294–309. Springer, Heidelberg (2005)
26. Magma, http://magma.maths.usyd.edu.au/magma/

Appendix A Implementation of Our Attack

In this appendix, we concentrate on a specific implementation of our attack in detail.

We only assume that we can compute $P(x)$(or $\Pi P(x)$) for all given x. So, at first we examine a construction of the public key P(or ΠP). Because $P = T \circ F \circ S$, where S, T are affine transformations over K and F can be expressed as a quadratic map over K (by using a K-linear map from L^ℓ to K^n), $P = (P^{(1)}, \ldots, P^{(n)})$(or $\Pi P = (P^{(1)}, \ldots, P^{(n-r)})$) has a polynomial expression over K below.

$$P^{(k)}(x) = \sum_{1 \le i < j \le n} {\zeta_{ij}}^{(k)} x_i x_j + \sum_{1 \le i \le n} {\zeta_{ii}}^{(k)} {x_i}^2 + \sum_{1 \le j \le n} {\nu_j}^{(k)} x_j + \epsilon^{(k)}$$
$$({\zeta_{ij}}^{(k)},\ {\nu_j}^{(k)},\ \epsilon^{(k)} \in K), \tag{24}$$

where $x = (x_1, \ldots, x_n) \in K^n$. We call each summation of (24), the bilinear part, the squared part, the linear part and the constant part, respectively.

The polynomial expression of P is obtained by the following procedure. Let $e_1, \ldots, e_n$ be the canonical basis over K^n. (i.e. $e_1 = (1, 0, \cdots, 0)$, $e_2 = (0, 1, \cdots, 0)$ and so on.) We can check the formula for the bilinear and constant part below.

$${\zeta_{ij}}^{(k)} = P^{(k)}(e_i + e_j) - P^{(k)}(e_i) - P^{(k)}(e_j) + P^{(k)}(0) \text{ (for } i \neq j) \tag{25}$$
$$\epsilon^{(k)} = P^{(k)}(0) \tag{26}$$

Moreover, when $K \neq \mathbb{F}_2$, we can see the equations for the squared and linear part below.

$${\zeta_{ii}}^{(k)} = \frac{1}{z - z^2}(zP^{(k)}(e_i) - P^{(k)}(z \cdot e_i) - (z-1)\epsilon^{(k)})$$
$${\nu_i}^{(k)} = \frac{1}{z^2 - z}(z^2 P^{(k)}(e_i) - P^{(k)}(z \cdot e_i) - (z^2-1)\epsilon^{(k)}),$$

where z is some element of $K \setminus \mathbb{F}_2$.

In the case when $K = \mathbb{F}_2$, we use more complicated techniques. [2] We introduce an (another) polynomial expression over L; P is transformed into $\mathbb{P} = (\mathbb{P}^{(1)}, \ldots, \mathbb{P}^{(\ell)})$ below.

$$\mathbb{P}^{(k)}(X) = \sum_{1 \le i < j \le \ell} \sum_{0 \le \alpha, \beta < n/\ell} {Z_{\alpha\beta ij}}^{(k)} {X_i}^{q^\alpha} {X_j}^{q^\beta} +$$
$$\sum_{1 \le i \le \ell} \sum_{0 \le \alpha \le \beta < n/\ell} {Z_{\alpha\beta ii}}^{(k)} {X_i}^{q^\alpha + q^\beta} +$$
$$\sum_{1 \le j \le \ell} \sum_{0 \le \alpha < n/\ell} {N_{\alpha j}}^{(k)} {X_j}^{q^\alpha} + E^{(k)}$$
$$({Z_{\alpha\beta ij}}^{(k)},\ {N_{\alpha j}}^{(k)},\ E^{(k)} \in L), \tag{27}$$

[2] Of course, if we consider that P has no squared part, computing the linear part is easy. However, since we use a feature that $F^{(k)}$ is a quadratic form, P may have both of the squared and linear part.

where $X = (X_1, \ldots, X_\ell) \in L^\ell$. Note that this expression is the most general representation of multivariate quadratic systems. (i.e. $\ell = 1$ induces the "multivariate representations" and $\ell = n$ the "univariate representations". See [24].) The coefficients ${Z_{\alpha\beta ij}}^{(k)}$, ${N_{\alpha j}}^{(k)}$, $E^{(k)}$ can be computed by using the linear algebra. Precisely speaking, compute $\mathbb{P}(X)$ for distinct $\frac{1}{2}n(n+1)(n+2)$ X, represents a system of linear equations and solve the system. By transforming $\mathbb{P}$ into P, we can obtain the polynomial expression over K.

Here, implementation of (6) is discussed. As we stated above, each coordinate of $P_L(x) = P(x-v) - P(-v)$ is a quadratic form. (The fact that each coordinate of $F(x)$ is a quadratic form is essential.) Then linear part of $P(x - v)$ equals 0:

$$\sum_j {\nu_j}^{(k)} x_j - \sum_{1 \le i < j \le n} {\zeta_{ij}}^{(k)} (v_i x_j + v_j x_i) = 0 \tag{28}$$

Especially, we find the equation $\sum_j {\nu_j}^{(k)} v_j = 0$ by substituting $x = v$ into the equation above. So we compute a kernel of ν_j(which we called $v$space in our algorithms) and find a vector of the $v$space satisfying (28).

Here, implementation of (6) is discussed. By definition of P_L, DP_L is expressed as follows.

$$DP_L{}^{(k)}(a,\ x) = \sum_{1 \le i < j \le n} {\zeta_{ij}}^{(k)} (a_i x_j + x_i a_j), \tag{29}$$

where $a = (a_1, \ldots, a_n),\ x = (x_1, \ldots, x_n)$. So we obtain the precise expression of BS_{N_ξ} for $N_\xi = (n_{ij})_{1 \le i,j \le n}$.

$$BS_{N_\xi}{}^{(k)}(a,\ x) = \sum_{1 \le i,j \le n} \left\{ \sum_{l \ne i} {\zeta_{il}}^{(k)} (a_j x_l + x_j a_l) \right\} n_{ij}, \tag{30}$$

where ${\zeta_{ij}}^{(k)} = {\zeta_{ji}}^{(k)}$ (for $i > j$). By using the discussion on Section 4.2, the linear algebra techniques enables us to compute N_ξ.

Finally, we see how to forge a signature. Note that this step can be used for deciphering. At first, we compute coefficients of forging equations (9), (11). This task is accomplished by solving the linear system for distinct x. Once we obtain the forging equations (9), (11), these equations enable us to forge a signature by using the linear algebra. As we saw above, our attack is not key recovery attack but forgery attack. However, we can compute efficiently many fake signatures by using the same forging equations.

Efficient Batch Verification of Short Signatures for a Single-Signer Setting without Random Oracles

Fuchun Guo[1], Yi Mu[2], and Zhide Chen[1]

[1] Key Lab of Network Security and Cryptology
School of Mathematics and Computer Science
Fujian Normal University, Fuzhou, China
fuchunguo1982@gmail.com, zhidechen@fjnu.edu.cn
[2] Centre for Computer and Information Security Research
School of Computer Science and Software Engineering
University of Wollongong, Wollongong NSW 2522, Australia
ymu@uow.edu.au

Abstract. In Eurocrypt 2007, Camenisch, Hohenberger and Pedersen introduced the notion of multi-signer batch verification and proposed several efficient and practical batch verification schemes for signatures, including a very efficient batch verification scheme for a multi-signer setting without random oracles. This scheme is the most efficient in comparison with other existing schemes, but it can be applied *only* to the multi-signer setting. We observe that amongst all existing batch verification schemes, the fastest scheme for a single-signer setting is based on the BLS short signature whose proof need resort to random oracles. It is clear that batch verification for a single-signer setting is as important as for multi-signer scenarios in some applications, especially where the system has only a single signer, such as a secure time-stamping server or a certificate generation server. However, to our knowledge, the efficient batch verification of short signatures in a single-signer setting without random oracles is a challenging open problem. In this paper, we propose a new signature scheme from Gentry IBE that is as efficient as the BLS short signature scheme in batch verification. We are able to prove its security without random oracles. Our signature is approximately 320 bits in length, and a verification requires only two pairings for verifying n signatures from a single signer.

Keywords: Batch verification, short signature.

1 Introduction

The notion of *Batch Cryptography* was first introduced by Fiat [5] in 1989. Loosely speaking, batch verification is a kind of signature scheme where the cost on verifying n signatures of different messages is less than conducting them one by one. If signatures for a batch verification are generated by multiple signers, we call it *multi-signer* batch verification; otherwise, we call it *single-signer* batch verification.

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 49–63, 2008.

Since the seminal work due to Fiat [5], many schemes about batch verification were published (e.g., [3, 8, 9, 11, 12, 17]). Unfortunately, most of them were broken. Recently, Camenisch, Hohenberger and Pedersen [4] extended the general batch verification definition of Bellare, Garay and Rabin [3] to the case of multi-signer. They proposed two batch verification schemes from existing signature schemes of Waters IBS and BLS [1, 14, 16] and proposed an efficient batch verification scheme of short signatures in a multi-signer setting without random oracles (for simplicity, we denote it by CL*). However, the CL* scheme requires a time limitation where all signatures to be batch-verified should be signed in the same period. We review the previous schemes in terms of two aspects: random oracle model and standard model.

Random Oracles Model. Although the batch verification scheme can be derived from RSA and DSA schemes, the much more efficient one is based on the BLS short signature scheme [1], which possesses two main properties: (1) The signature length is very short, about 160 bits; (2) The computation only requires two pairing operations for verifying n signatures from a single-signer. The BLS scheme can also achieve multi-signer batch verification, but it's not as efficient as single-signer batch verification due to multiple pairings [7].

Standard Model. There are only two schemes presented in [4]. The first scheme, denoted by Waters IBS, has the batch verifiability for both multi-signer and single-signer settings. However, they are not efficient due to multiple pairings. The second one, denoted by CL*, has a short signature length and is very fast in multi-signer batch verification, where a verification only requires three pairings for verifying n signatures from multiple signers. However, the CL* scheme requires that the signatures to be verified must be signed in the same period and any signer can only issue one signature per period; therefore it cannot be applied to single-signer batch verification.

In summary, the most efficient batch verification for a single-signer setting is from the BLS scheme and for a multi-signer setting is from the CL* scheme. Therefore, the remaining open problems are how to find a fast batch verification for a single-signer setting *without random oracles* and how to find a fast batch verification for both multi-signer and single-signer with or without random oracles.

1.1 Our Contribution

In this paper, we propose an efficient batch verification of short signatures for a single-signer setting in the standard model from Gentry IBE [6] in order to fill the gap outlined earlier. Our signature length is short (about 320 bits) and the batch verification requires only two pairings for n signatures, which is almost as efficient as the BLS scheme for single-signer that requires the random oracle model. Similarly to the BB04 short signature [2], our short signature scheme is provably secure under the q-Strong Diffie-Hellman (q-SDH) assumption.

Apart from the feature of batch verification, our signature scheme also captures other nice properties, i,e, ability of online/offline signing and feature of

Table 1. Comparison on the cost for batch verification

	CL*	**BLS**	**Waters IBS**	**Ours**
Security	Standard Model	Random Oracles	Standard Model	Standard Model
Length	1	1	2	2
Multi-signers	Fast	Normal	Normal	Normal
Single-signer	×	Fast	Normal	Fast

Table 2. Comparison within the feature of online/offline signing

	CL*	**BLS**	**Waters IBS**	**Ours**
Security	Standard Model	Random Oracles	Standard Model	Standard Model
Length	2	2	3	2
Multi-signers	Normal	Normal	Normal	Normal
Single-signer	×	Normal	Normal	Fast

short signature length. Considering all these features, we show that our scheme is much more efficient than all previous schemes.

To demonstrate that our single-signer batch verification scheme fills the gap, we give a comparison in the following tables (Tables 1 and 2). We denote by "1", "2" , "3" the number of elements of a signature, respectively. We denote by "Normal" the verification that requires multiple pairings related with the number of signatures and by "Fast" the verification that requires only a few pairings.

The rest of this paper are organized as follows. In Section 2, we provide the preliminaries associated with our scheme, including the security model, batch verification of signature, bilinear pairing and assumption. We present our signature scheme and its security proof in Section 3. In Section 4, we show the batch verifiability of our signature scheme. We give a discussion in Section 5 and conclude our paper in last Section.

2 Definition

In this section, we give the definition of signature security, batch verification of signatures, bilinear pairing, assumptions and the notations of cost.

2.1 Signature Scheme

A signature scheme consists of three algorithms $(\mathcal{G}, \mathcal{S}, \mathcal{V})$, for key generation, signing, and signature verification, respectively. The security of a signature scheme is modeled with unforgeability under a chosen message attack (UF-CMA), where a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ can be described as follows :

Setup: The challenger $\mathcal{C}$ runs algorithm $\mathcal{G}$ to obtain a pair of signing key and verification key (sk, vk). The adversary $\mathcal{A}$ is given vk.

Query: $\mathcal{A}$ makes a signature query on message m_i of his choice. $\mathcal{C}$ responds to each query with a signature $\sigma_{m_i} = \mathcal{S}_{sk}(m_i)$. $\mathcal{A}$ can require q_s message queries at most.

Forge: $\mathcal{A}$ outputs a signature pair (m^*, σ^*) and wins the game if

1. $\mathcal{A}$ did not make a signature query on m^*;
2. σ^* is a valid signature on m^* for vk, i.e., $\mathcal{V}_{vk}(m^*, \sigma^*) = True$.

We define $Adv_{\mathcal{A}}$ as the probability that $\mathcal{A}$ wins in the above game.

Definition 1. *A signature scheme is (t, q_s, ϵ)-secure against UF-CMA attack if no forger (t, q_s, ϵ)-breaks it, where the forger $\mathcal{A}$ runs in time at most t, makes at most q_s signature queries, and $Adv_{\mathcal{A}}$ is at least ϵ.*

2.2 Batch Verification of Signatures

Definition 2. (Batch Verification of Signatures)[4] *Let k be the security parameter. Suppose $(\mathcal{G}, \mathcal{S}, \mathcal{V})$ is a signature scheme, $k \in poly(k)$, and (pk_1, sk_1), $\cdots, (pk_n, sk_n)$ are generated independently according to $\mathcal{G}(1^k)$. Then we call probabilistic Batch a batch verification algorithm when the following conditions hold:*

- *If $\mathcal{V}(pk_i, m_i, \sigma_{m_i}) = 1$ for all $i \in [1, n]$, then*

$$Batch((pk_1, m_1, \sigma_{m_1}), \cdots, (pk_n, m_n, \sigma_{m_n})) = 1;$$

- *If $\mathcal{V}(pk_i, m_i, \sigma_{m_i}) = 0$ for any $i \in [1, n]$, then*

$$Batch((pk_1, m_1, \sigma_{m_1}), \cdots, (pk_n, m_n, \sigma_{m_n})) = 0.$$

2.3 Bilinear Pairing

The notation of bilinear pairing and bilinear pairing groups used in our short signature scheme are as follows:

1. $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are three multiplicative cyclic groups of prime order p ;
2. g_1 is a generator of $\mathbb{G}_1$ and g_2 is a generator of $\mathbb{G}_2$;
3. ψ is an isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$, with $\psi(g_2) = g_1$;
4. $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is the bilinear pairing.

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be three groups as the above, such that $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T| = p$. A bilinear pairing is $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

- Bilinearity: for all $u \in \mathbb{G}_1, v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$;
- Non-degeneracy: $e(g_1, g_2) \neq 1$. In other words, if g_1 be a generator of $\mathbb{G}_1$ and g_2 be a generator of $\mathbb{G}_2$, then $e(g_1, g_2)$ generates $\mathbb{G}_T$;
- Computability: It is efficient to compute $e(u, v)$ for all $u \in \mathbb{G}_1, v \in \mathbb{G}_2$.

2.4 The q-Strong Diffie-Hellman Assumption

Let $\mathbb{G}_1, \mathbb{G}_2$ be two cyclic groups of prime order p. Let g_1 be a generator of $\mathbb{G}_1$ and g_2 be a generator of $\mathbb{G}_2$ such that $g_1 = \psi(g_2)$. The q-SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$ is defined as follows: given $g_1, g_2, g_2^a, g_2^{(a^2)}, \cdots, g_2^{(a^q)}$ as input, it outputs a pair $(c, g_1^{1/(a+c)})$ for any $c \in \mathbb{Z}_p$. An algorithm $\mathcal{A}$ has advantage ϵ in solving the q-SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$ if

$$Pr\Big[\mathcal{A}(g_1, g_2, g_2^a, g_2^{(a^2)}, \cdots, g_2^{(a^q)}) = (c, g_1^{\frac{1}{a+c}})\Big] \geq \epsilon,$$

where the probability is over the random choice of generator $g_2 \in \mathbb{G}_2$ with $g_1 = \psi(g_2)$, the random choice of a in $\mathbb{Z}_p$, and the random bits consumed by $\mathcal{A}$.

Definition 3. *We say that the (q, t, ϵ)-SDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$ if no t-time algorithm has advantage at least ϵ in solving the q-SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$.*

2.5 Notations

We use the same definition of computation cost in [4]. Note that Lim [10] provides a number of efficient methods for doing n-term exponentiations and Granger and Smart [7] give improvements over the naive method for computing a product of pairings. We define them explicitly.

n-MP^s :	s n-term pairings $\prod_{i=1}^{n} e(g_i, h_i)$ where $g_i, h_i \in \mathbb{G}$.
n-ME$^s_{\mathbb{G}}(k)$:	s n-term exponentiations $\prod_{i=1}^{n} g_i^{a_i}$ where $g_i \in \mathbb{G}, \lvert a_i\rvert = k$.
P^s :	s pairings $e(g_i, h_i)$ for $i = 1, 2, \cdots, s$ where $g_i, h_i \in \mathbb{G}$.
E$^s_{\mathbb{G}}(k)$:	s exponentiations g^{a_i} for $i = 1, 2, \cdots, s$ where $g \in \mathbb{G}, \lvert a_i\rvert = k$.
GT^s :	Testing whether or not s elements are in the group $\mathbb{G}$.
H^s :	Hashing s values into the group $\mathbb{G}$.
M^s :	s multiplications in one or more groups.
m^s :	s modular multiplications in $\mathbb{Z}_p$.

3 Our Signature Scheme

We now present our short signature construction and then provide its security proof. The construction is a modification from the private key generation of Gentry IBE construction in [6].

3.1 Construction

Let $(\mathbb{G}_1, \mathbb{G}_2)$ be bilinear groups where $|\mathbb{G}_1| = |\mathbb{G}_2| = p$ for some prime p. The short signature scheme can be described as follows:

Key generation: Pick a random generator $g_2 \in \mathbb{G}_2$ and set $g_1 = \psi(g_2)$. Randomly choose $\alpha, \beta \in \mathbb{Z}_p$, and compute $u = e(g_1, g_2) \in \mathbb{G}_T, v = e(g_1, g_2)^\alpha \in \mathbb{G}_T, z = g_2^\beta \in \mathbb{G}_2$, the verification key and signing key are, respectively,

$$vk = (e, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, u, v, z), \qquad sk = (\alpha, \beta).$$

Sign: Given a message $m \in \mathbb{Z}_p$ and the signing key $sk = (\alpha, \beta)$, the signer picks a random $r \in \mathbb{Z}_p$ and outputs the signature σ_m:

$$\sigma_m = (\sigma_1, \sigma_2) = \left(\alpha - r(\beta - m), g_1^r\right) \in \mathbb{Z}_p \times \mathbb{G}_1.$$

Verification: Given a verification key $vk = (e, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, u, v, z)$, a message $m \in \mathbb{Z}_p$ and a signature $\sigma_m = (\sigma_1, \sigma_2)$, verify that

$$u^{\sigma_1} \cdot e(\sigma_2, z/g_2^m) = v.$$

If the equality holds the signature σ_m on m of vk is valid; otherwise invalid.

Correctness:

$$\begin{aligned} u^{\sigma_1} \cdot e(\sigma_2, z/g_2^m) &= e(g_1, g_2)^{\sigma_1} \cdot e(g_1^r, g_2^\beta/g_2^m) \\ &= e(g_1, g_2)^{\alpha - r(\beta - m)} \cdot e(g_1, g_2)^{r(\beta - m)} \\ &= e(g_1, g_2)^{\alpha - r(\beta - m) + r(\beta - m)} \\ &= e(g_1, g_2)^\alpha \\ &= v. \end{aligned}$$

A signature contains two elements $(\alpha - r(\beta - m), g^r) \in \mathbb{Z}_p \times \mathbb{G}_1$. The length of a signature is approximately $\log_2(p)$ bits; therefore the total signature length is approximately $2\log_2(p)$. When using the same pairing as BLS [1] and BB04 [2] scheme, the signature length is approximately 320 bits.

3.2 Security

We now prove that our short signature is secure against chosen message attack under the q-SDH assumption.

Theorem 1. *Let $q = q_s + 1$. Then, our short signature scheme is (q_s, t, ϵ) UF-CMA secure assuming the (q, t', ϵ')-SDH assumption holds for $(\mathbb{G}_1, \mathbb{G}_2)$:*

$$t' = t + O(t_{exp} \cdot q^2), \epsilon' \geq (1 - \frac{q_s}{p})\epsilon,$$

where t_{exp} is the maximum time for an exponentiation in $\mathbb{G}_1$ and $\mathbb{G}_2$.

Proof. Assume $\mathcal{A}$ is the forger (adversary) that (q_s, t, ϵ)-breaks the signature scheme. We construct an algorithm $\mathcal{B}$ that, by interacting with $\mathcal{A}$, solves the q-SDH problem in time t' with advantage ϵ'. Algorithm $\mathcal{B}$ is given a random

$g_1, g_2, g_2^a, g_2^{(a^2)}, \cdots, g_2^{(a^q)}$ of the q-SDH problem for some unknown $a \in \mathbb{Z}_p$. Algorithm $\mathcal{B}$'s goal is to produce a pair $(c, g_1^{1/(a+c)})$ for some $c \in \mathbb{Z}_p$. Algorithm $\mathcal{B}$ does so by interacting with the forger $\mathcal{A}$ as follows:

Setup: Algorithm $\mathcal{B}$ randomly chooses a polynomial $f(x) \in \mathbb{Z}_p[x]$ of degree q and sets the signing key to be $sk = (\alpha, \beta) = (f(a), a)$. The value $g_2^{f(a)}$ can be computed from $g_2, g_2^a, g_2^{(a^2)}, \cdots, g_2^{(a^q)}$ and $f(x)$. So, the verification key

$$u = e(g_1, g_2), \quad v = e(g_1, g_2)^\alpha = e(g_1, g_2)^{f(a)} = e(g_1, g_2^{f(a)}), \quad z = g_2^\beta = g_2^a$$

can be computed from $g_1, g_2, g_2^a, g_2^{f(a)}$. $\mathcal{B}$ gives the verification key $vk = (e, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, u, v, z)$ to the forger $\mathcal{A}$.

Query: The forger makes a signature query on message m_i, algorithm $\mathcal{B}$ returns the following signature:

$$\sigma_{m_i} = (\sigma_1, \sigma_2) = \left(f(m_i), \psi(g_2^{\frac{f(a)-f(m_i)}{a-m_i}})\right). \tag{1}$$

Because $(x - m_i) | f(x) - f(m_i)$, we have $\frac{f(a)-f(m_i)}{a-m_i} = f_i'(a)$ where $f_i'(x) \in \mathbb{Z}_p[x]$ is a polynomial of degree $q-1$ and $\sigma_2' = g_2^{(f(a)-f(m_i))/(a-m_i)} = g_2^{f'(a)}$ can be computed from $f_i'(x)$ and $g_2, g_2^a, g_2^{a^2}, \cdots, g_2^{a^q}$ and then $\sigma_2 = \psi(\sigma_2')$.

Let $r = \frac{f(a)-f(m_i)}{a-m_i}$, we have

$$f(m_i) = f(a) - r(a - m_i) = \alpha - r(\beta - m_i),$$

$$\left(f(m_i), \psi(g_2^{\frac{f(a)-f(m_i)}{a-m_i}})\right) = \left(\alpha - r(\beta - m_i), \psi(g_2^r)\right) = \left(\alpha - r(\beta - m_i), g_1^r\right).$$

So, signature (1) is a valid signature on m_i. $\mathcal{B}$ gives it to $\mathcal{A}$.

Forge: Let (m^*, σ^*) be a valid signature forged by the adversary $\mathcal{A}$, where the tuple of signature σ^* is

$$\sigma^* = (\sigma_1^*, \sigma_2^*) = \left(\alpha - r^*(\beta - m^*), g_1^{r^*}\right).$$

If $(x - m^*) \mid f(x) - \sigma_1^*$, $\mathcal{B}$ aborts; Otherwise, let $s^* = \alpha - r^*(\beta - m^*)$, the forged signature can be re-written as

$$\sigma^* = (s^*, g_1^{\frac{\alpha - s^*}{\beta - m^*}}) = (s^*, g_1^{\frac{f(a)-s^*}{a-m^*}}).$$

Then, there exists a polynomial $f''(x) \in \mathbb{Z}_p[x]$ of degree $q-1$ and d, which can be computed by $\mathcal{B}$, such that:

$$g_1^{\frac{f(a)-s^*}{a-m^*}} = g_1^{f''(a) + \frac{d}{a-m^*}}.$$

$\mathcal{B}$ produces the pair $\left(-m^*, g_1^{\frac{1}{a-m^*}}\right)$ using $g_2, g_2^a, g_2^{a^2}, \cdots, g_2^{a^q}, \psi$ and $f''(x)$ by:

$$g_1^{\frac{1}{a-m^*}} = \left(\frac{\psi(g_2^{\frac{f(a)-s^*}{a-m^*}})}{\psi(g_2^{f''(a)})}\right)^{\frac{1}{d}} = \left(\frac{g_1^{\frac{f(a)-s^*}{a-m^*}}}{g_1^{f''(a)}}\right)^{\frac{1}{d}} = \left(g_1^{\frac{d}{a-m^*}}\right)^{\frac{1}{d}} = g_1^{\frac{1}{a-m^*}}.$$

This completes the description of the simulation. It remains to analyze the correctness of simulation and the probability of $\mathcal{B}$ for not aborting.

For $q = q_s + 1$, from $\mathcal{A}$'s view, the random value $f(m_1), f(m_2), \cdots, f(m_{q_s})$ in the signature queries are random and independent. But this follows from a fact that $f(x) \in \mathbb{Z}_p[x]$ is a uniformly random polynomial of degree q. So, it's a perfect simulation in signature query.

The adversary $\mathcal{A}$ outputs the forged signature (m^*, σ^*) and $\mathcal{B}$ will abort if and only if $(x - m^*) \mid f(x) - \sigma_1^*$, i.e., $\sigma_1^* = f(m^*)$. When the adversary $\mathcal{A}$ forges the signature $\sigma^* = (\sigma_1^*, \sigma_2^*)$ on m^*, it deserts the signature and re-forges a signature on m^* if σ_2^* is equal to the value $g_1^{r_j}$ in the queries of m_j. If $\mathcal{A}$ makes q_s queries in total and we have $\sigma_2^* = g_1^{(f(a)-\sigma_1^*)/(a-m^*)}$, then σ_1^* is constrained within $p - q_s$ values in $\mathbb{Z}_p$. So the probability that $\mathcal{B}$ did not abort is $1 - q_s/p$. We therefore have

$$\Pr\left[\mathcal{B}(g_1, g_2, g_2^a, g_2^{(a^2)}, \cdots, g_2^{(a^q)}) = (-m^*, g_1^{\frac{1}{a-m^*}})\right] \geq \epsilon - \frac{q_s}{p}\epsilon.$$

The time complexity of the algorithm $\mathcal{B}$ is dominated by each signature query, producing the pair $(-m^*, g_1^{1/(a-m^*)})$, which costs q exponentiations at most in each signature query for producing the pair. There are $q_s = q - 1$ signature queries, so it takes $O(t_{exp} \cdot q^2)$ at most, where t_{exp} is the time required to exponentiate in $(\mathbb{G}_1, \mathbb{G}_2)$. □

3.3 Signing Arbitrary Messages

We can also extend our short signature scheme to sign arbitrary messages in $\{0,1\}^*$, as opposed to merely messages in the space of $\mathbb{Z}_p$, by using a collision-resistant hash function $H : \{0,1\}^* \to \mathbb{Z}_p$. Obviously, the challenger can simulate the signatures in the query phase for any q_s ($q_s < q$) messages if the property of hash function holds. In the forge phase, the challenger will output the pair of $(-H(m^*), g_1^{1/(a-H(m^*))})$ as the solution to the q-SDH problem.

4 Batch Verification

We use the way of "Small Exponents Test" described by Bellare, Garay and Rabin in 1998 [3] to achieve secure batch verification.

4.1 Single-Signer Batch Verification

Given the verification key $vk = (e, \mathbb{G}_1, \mathbb{G}_2, g_1, g_2, u, v, z)$, suppose there are n signatures $\sigma_{m_1}, \sigma_{m_2}, \cdots, \sigma_{m_n}$ on $m_1, m_2, \cdots, m_n$ using the signing key $sk = (\alpha, \beta)$, where $r_1, r_2, \cdots, r_n$ are random values in $\mathbb{Z}_p$, respectively.

$$\sigma_{m_1} = \left(\alpha - r_1(\beta - m_1), g_1^{r_1}\right) \in \mathbb{Z}_p \times \mathbb{G}_1,$$
$$\sigma_{m_2} = \left(\alpha - r_2(\beta - m_2), g_1^{r_2}\right) \in \mathbb{Z}_p \times \mathbb{G}_1,$$
$$\cdots\cdots\cdots$$
$$\sigma_{m_n} = \left(\alpha - r_n(\beta - m_n), g_1^{r_n}\right) \in \mathbb{Z}_p \times \mathbb{G}_1.$$

The batch verification works as follows:

- Check whether $g_1^{r_1}, g_1^{r_2}, \cdots, g_1^{r_n} \in \mathbb{G}_1^n$ or not;
- Randomly choose a vector $\delta = (\delta_1, \delta_2, \cdots, \delta_n)$ with each element being a l-bits random numbers and compute

$$A = (g_1^{r_1})^{\delta_1} \cdot (g_1^{r_2})^{\delta_2} \cdots (g_1^{r_n})^{\delta_n} = g_1^{\sum_{i=1}^n \delta_i r_i},$$
$$B = (g_1^{r_1})^{m_1\delta_1} \cdot (g_1^{r_2})^{m_2\delta_2} \cdots (g_1^{r_n})^{m_n\delta_n} = g_1^{\sum_{i=1}^n \delta_i r_i m_i},$$
$$C = \delta_1(\alpha - r_1(\beta - m_1)) + \delta_2(\alpha - r_2(\beta - m_2)) + \cdots + \delta_n(\alpha - r_n(\beta - m_n))$$
$$= \alpha \sum_{i=1}^n \delta_i - \beta \sum_{i=1}^n \delta_i r_i + \sum_{i=1}^n \delta_i r_i m_i.$$

- Accept all signatures if the equation holds

$$\frac{u^C \cdot e(A, z)}{e(B, g_2)} = v^{\sum_{i=1}^n \delta_i}.$$

Correctness:

$$\frac{u^C \cdot e(A, z)}{e(B, g_2)} = \frac{e(g_1, g_2)^{\alpha \sum_{i=1}^n \delta_i - \beta \sum_{i=1}^n \delta_i r_i + \sum_{i=1}^n \delta_i r_i m_i} \cdot e(g_1, g_2)^{\beta \sum_{i=1}^n \delta_i r_i}}{e(g_1, g_2)^{\sum_{i=1}^n \delta_i r_i m_i}}$$
$$= e(g_1, g_2)^{\alpha \sum_{i=1}^n \delta_i}$$
$$= v^{\sum_{i=1}^n \delta_i}.$$

We use the group defined in [1, 2] such that the order is about 160 bits in length, and same as the below. The cost on above single-signer batch verification is

$$n\text{-}ME_{\mathbb{G}}(160) + n\text{-}ME_{\mathbb{G}}(l) + P^2 + E_{\mathbb{G}_T}(160) + E_{\mathbb{G}_T}(nl) + GT^n + M + m^{n+1},$$

where the cost on native verification is $P^n + E_{\mathbb{G}}^n(160) + E_{\mathbb{G}_T}^n(160) + GT^n + M^{2n}$.

Security:

Theorem 2. *The probability that any adversary breaks the above batch verification is no more than 2^{-l}, where l is the size of small exponent number.*

Proof. Suppose there are n signatures $\sigma'_{m_1}, \sigma'_{m_2}, \cdots, \sigma'_{m_n}$ on $m_1, m_2, \cdots, m_n$ using the signing key $sk = (\alpha, \beta)$, which are accepted as n valid signatures by batch verification. Now, we prove that the probability that the adversary breaks the batch verification is decided by the size of small exponent number.

No matter the signatures are valid or not, the signatures can be re-written as the follows with random $r_i, k_i \in \mathbb{Z}_p$ in each signature:

$$\begin{aligned}
\sigma'_{m_1} &= \left(\alpha - r_1(\beta - m_1) + k_1, g_1^{r_1}\right) \in \mathbb{Z}_p \times \mathbb{G}_1 \\
\sigma'_{m_2} &= \left(\alpha - r_2(\beta - m_2) + k_2, g_1^{r_2}\right) \in \mathbb{Z}_p \times \mathbb{G}_1 \\
&\cdots\cdots\cdots \\
\sigma'_{m_n} &= \left(\alpha - r_n(\beta - m_n) + k_n, g_1^{r_n}\right) \in \mathbb{Z}_p \times \mathbb{G}_1
\end{aligned}$$

Such that

$$\sigma'_{m_i} = \left(\alpha - r_i(\beta - m_i) + k_i, g_1^{r_i}\right) = \begin{cases} \text{valid}, & \text{if } k_i = 0; \\ \text{invalid}, & \text{otherwise}; \end{cases}$$

Then, we have

$$\begin{aligned}
A &= (g_1^{r_1})^{\delta_1} \cdot (g_1^{r_2})^{\delta_2} \cdots (g_1^{r_n})^{\delta_n} = g_1^{\sum_{i=1}^{n} \delta_i r_i} = g_1^R, \\
B &= (g_1^{r_1})^{m_1\delta_1} \cdot (g_1^{r_2})^{m_2\delta_2} \cdots (g_1^{r_n})^{m_n\delta_n} = g_1^{\sum_{i=1}^{n} r_i m_i \delta_i} = g_1^T, \\
C' &= \delta_1(\alpha - r_1(\beta - m_1) + k_1) + \cdots + \delta_n(\alpha - r_n(\beta - m_n) + k_n) \\
&= \alpha \sum_{i=1}^{n} \delta_i - R\beta + T + (k_1\delta_1 + \cdots + k_n\delta_n)
\end{aligned}$$

Let $\delta_0 = \delta_1 + \cdots + \delta_n$, for the batch verification equation still holds, we have

$$\begin{aligned}
\frac{u^{C'} \cdot e(A, z)}{e(B, g_2)} &= \frac{e(g_1, g_2)^{\delta_0\alpha - R\beta + T + (k_1\delta_1 + \cdots + k_n\delta_n)} \cdot e(g_1^R, g_2^\beta)}{e(g_1^T, g_2)} \\
&= e(g_1, g_2)^{\delta_0\alpha - R\beta + T + (k_1\delta_1 + \cdots + k_n\delta_n)} \cdot e(g_1, g_2)^{R\beta} \cdot e(g_1, g_2)^{-T} \\
&= e(g_1, g_2)^{\delta_0\alpha - R\beta + T + (k_1\delta_1 + \cdots + k_n\delta_n) + R\beta - T} \\
&= e(g_1, g_2)^{\delta_0\alpha + (k_1\delta_1 + \cdots + k_n\delta_n)} \\
&= v^{\delta_0} e(g_1, g_2)^{k_1\delta_1 + \cdots + k_n\delta_n} \\
&= v^{\delta_0} \\
&\Rightarrow k_1\delta_1 + \cdots + k_n\delta_n \equiv 0 (\text{mod p})
\end{aligned}$$

Assume that the batch verification holds but for at least one signature is invalid. Assume that the signature σ_{m_j} is invalid, which means $k_j \neq 0$. Since p is a prime value then δ_j has an inverse k_j^{-1} such that

$$\delta_j \equiv -k_j^{-1} \sum_{i=1, i\neq j}^{n} \delta_i k_i (\text{mod p}).$$

If $\delta_i, i = 1, 2, \cdots, n$ are random l-bits chosen by a verifier, the above equation holds with the probability of

$$\Pr\Big[\delta_j \equiv -k_j^{-1} \sum_{i=1, i\neq j}^{n} \delta_i k_i (\text{mod p}) \Big| k_1\delta_1 + \cdots + k_n\delta_n \equiv 0 (\text{mod p})\Big] = \frac{1}{2^l}.$$

Therefore, the probability that any adversary (signer) can break the batch verification is not more than 2^{-l}. □

As noted in [3, 4], the size of l depends on the application and how critical it is not to accept even a single invalid signature. For just a rough check that all signatures are correct 20 bits seems reasonable. In a higher security setting we should probably be using around 64 bits.

4.2 Multi-signer Batch Verification

From our signature construction, we know that different signers can use the common parameters of $(e, \mathbb{G}_1, \mathbb{G}, g_1, g_2, u)$ as part of their public keys. So, we set (α_i, β_i) to be the private key and $(e, \mathbb{G}_1, \mathbb{G}, g_1, g_2, u, v_i, z_i)$ as the corresponding public keys, where $v_i = e(g_1, g_2)^{\alpha_i}$ and $z_i = g_2^{\beta_i}$.

Suppose there are n signatures $\sigma_{m_1}, \sigma_{m_2}, \cdots, \sigma_{m_n}$ and σ_{m_i} on m_i is singed by the private key of (α_i, β_i), where $r_1, r_2, \cdots, r_n$ are random values in $\mathbb{Z}_p$, respectively.

$$\begin{aligned}
\sigma_{m_1} &= \Big(\alpha_1 - r_1(\beta_1 - m_1), g_1^{r_1}\Big) \in \mathbb{Z}_p \times \mathbb{G}_1,\\
\sigma_{m_2} &= \Big(\alpha_2 - r_2(\beta_2 - m_2), g_1^{r_2}\Big) \in \mathbb{Z}_p \times \mathbb{G}_1,\\
&\cdots\cdots\cdots\\
\sigma_{m_n} &= \Big(\alpha_n - r_n(\beta_n - m_n), g_1^{r_n}\Big) \in \mathbb{Z}_p \times \mathbb{G}_1.
\end{aligned}$$

The batch verification works as follows:

- Check whether $g_1^{r_1}, g_1^{r_2}, \cdots, g_1^{r_n} \in \mathbb{G}_1^n$ or not;
- Randomly choose a vector $\delta = (\delta_1, \delta_2, \cdots, \delta_n)$ with each element being a l-bits random numbers and compute

$$\begin{aligned}
A &= (g_1^{r_1})^{m_1\delta_1} \cdot (g_1^{r_2})^{m_2\delta_2} \cdots (g_1^{r_n})^{m_n\delta_n} = g_1^{\sum_{i=1}^{n} r_i m_i \delta_i},\\
B &= \Big(e(g_1, g_2)^{\alpha_1}\Big)^{\delta_1} \Big(e(g_1, g_2)^{\alpha_2}\Big)^{\delta_2} \cdots \Big(e(g_1, g_2)^{\alpha_n}\Big)^{\delta_n},\\
C &= \delta_1(\alpha_1 - r_1(\beta_1 - m_1)) + \cdots + \delta_n(\alpha_n - r_n(\beta_n - m_n))\\
&= \sum_{i=1}^{n} \delta_i\alpha_i - \sum_{i=1}^{n} \delta_i r_i \beta_i + \sum_{i=1}^{n} \delta_i r_i m_i;
\end{aligned}$$

- Accept all the signatures if the equation holds

$$\frac{u^C \cdot \prod_{i=1}^{n} e\Big(g_1^{r_i}, z_i\Big)^{\delta_i}}{e(A, g_2)} = B.$$

Correctness:

$$\begin{aligned}
&\frac{u^C \cdot \prod_{i=1}^{n} e\left(g_1^{r_i}, z_i\right)^{\delta_i}}{e(A, g_2)} \\
&= \frac{e(g_1, g_2)^{\sum_{i=1}^{n} \delta_i \alpha_i - \sum_{i=1}^{n} \delta_i r_i \beta_i + \sum_{i=1}^{n} \delta_i r_i m_i} \cdot e(g_1, g_2)^{\sum_{i=1}^{n} \delta_i r_i \beta_i}}{e(g_1, g_2)^{\sum_{i=1}^{n} \delta_i r_i m_i}} \\
&= e(g_1, g_2)^{\sum_{i=1}^{n} \delta_i \alpha_i} \\
&= B.
\end{aligned}$$

The cost on the above multi-signer batch verification is

$$n\text{-}MP + n\text{-}ME_{\mathbb{G}}(160) + n\text{-}ME^2_{\mathbb{G}_T}(l) + P + E_{\mathbb{G}_T}(160) + GT^n + M + m^{n+1},$$

where the cost on the native verification is $P^n + E^n_{\mathbb{G}}(160) + E^n_{\mathbb{G}_T}(160) + GT^n + M^{2n}$.

Theorem 3. *The algorithm above is a batch verification for our signatures.*

Proof. The proof is similar to the proof in Theorem 2 and omitted.

4.3 Comparison

The BLS [4] scheme is much more efficient and practical than the other schemes in single-signer batch verification; therefore we only give our comparison to the the BLS scheme.

BLS: The cost on multi-signer batch verification is

$$n\text{-}MP + n\text{-}ME_{\mathbb{G}}(l) + P + E^n_{\mathbb{G}_T}(l) + GT^n + H^n;$$

The cost on single-signer batch verification is

$$n\text{-}ME^2_{\mathbb{G}}(l) + P^2 + GT^n + H^n.$$

Ours: The cost on the multi-signer batch verification is

$$n\text{-}MP + n\text{-}ME_{\mathbb{G}}(160) + n\text{-}ME^2_{\mathbb{G}_T}(l) + P + E_{\mathbb{G}_T}(160) + GT^n + M + m^{n+1};$$

The cost on the single-signer batch verification is

$$n\text{-}ME_{\mathbb{G}}(160) + n\text{-}ME_{\mathbb{G}}(l) + P^2 + E_{\mathbb{G}_T}(160) + E_{\mathbb{G}_T}(nl) + GT^n + M + m^{n+1}.$$

Our scheme is as efficient as the BLS based scheme in terms of in pairing computation in both for multi-signer and single-signer cases, where both need one multi-pairing and two pairing operations. The cost for all is almost the same when the inefficient H^n with $n\text{-}ME_{\mathbb{G}_T}(l)$ are equal to $n\text{-}ME_{\mathbb{G}_T}(160)$. However, our scheme is provably secure without random oracles.

5 Discussion

5.1 Online/Offline Signature

We briefly describe how to modify our scheme into an efficient online/offline signature scheme in a natural way. There is no need to modify the key generation and the verification phases. We here briefly describe how to separate signing into two phases:

- Offline phase: Choose at random $r \in \mathbb{Z}_p^*$ and compute $g_1^r \in \mathbb{G}_1$. Keep (r, g_1^r) in the memory.
- Online phase: For a given message $m \in \mathbb{Z}_p^*$ and the signing key $sk = (\alpha, \beta)$, retrieve from the memory the pair (r, g_1^r) and compute $\sigma_1 = \alpha - r(\beta - m) \bmod p$. The signature is

$$\sigma_m = (\sigma_1, \sigma_2) = \Big(\alpha - r(\beta - m), g_1^r\Big).$$

The offline cost is one exponentiation and the online cost is only one modular multiplication.

5.2 Comparison

We now provide an interesting comparison for online/offline signing applications. We have described that our scheme can be naturally separated into online and offline phases. However, the schemes based on the BLS scheme must use the "hash-sign-switch" paradigm [15] in order to enable online/offline signing. We denote by BLS* and CL** two online/offline enabled signature schemes, where they are generated from BLS and CL* schemes based on the "hash-sign-switch" paradigm. Clearly, they require one additional trapdoor hash computation (i.e., one 2-ME$_{\mathbb{G}}(160)$ operation) for verification and one element (i.e., about 160 bits) in signature length. Thus, by allowing online/offline signing, our scheme is much more efficient than the BLS* scheme, since the cost increases 2-$ME_{\mathbb{G}}^n(160)$ for n signatures prior to the batch verification for both cases. Because of this, we only make a comparison to the CL** scheme.

CL**: The cost on multi-signer batch verification is

$$2\text{-}ME_{\mathbb{G}}^n + n\text{-}ME_{\mathbb{G}}^2(l) + n\text{-}ME_{\mathbb{G}}(160) + P^3 + GT^n + H^n + m^n.$$

Ours: The cost on multi-signer batch verification is

$$n\text{-}MP + n\text{-}ME_{\mathbb{G}}(160) + n\text{-}ME_{\mathbb{G}_T}^2(l) + P + E_{\mathbb{G}_T}(160) + GT^n + M + m^{n+1}.$$

The cost on single-signer batch verification is

$$n\text{-}ME_{\mathbb{G}}(160) + n\text{-}ME_{\mathbb{G}}(l) + P^2 + E_{\mathbb{G}_T}(160) + E_{\mathbb{G}_T}(nl) + GT^n + M + m^{n+1}.$$

From [7, 10], we know that the cost on n-MP and 2-$ME_{\mathbb{G}}^n(160)$ is nearly equal. So, our signature scheme is as efficient as the CL** scheme with the following improvement. (1) Our multi-signer batch verification does not have the signing time limitation. (2) Our scheme can be applied to single-signer batch verification and is very efficient, whereas the CL** scheme can be applied to a multi-signer setting only.

6 Conclusion

Amongst all the previous schemes, the most efficient and practical batch verification scheme for single-signer is based on the BLS scheme, where it costs only two pairing operations for verifying n signatures. However, the security proof must employ random oracles. In this paper, we proposed a batch verification of short signatures for a single-signer setting as efficient as BLS scheme without using the random oracle model, where the signature is about 320 bits in length and the security is based on the q-Strong Diffie-Hellman (q-SDH) assumption in the standard model. Our scheme can be further speeded up by converting it into online/offline signature.

References

1. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001)
2. Boneh, D., Boyen, X.: Short Signatures Without Random Oracles. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004)
3. Bellare, M., Garay, J., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 236–250. Springer, Heidelberg (1998)
4. Camenisch, J., Hohenberger, S., Pedersen, M.: Batch Verification of Short Signatures. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 246–263. Springer, Heidelberg (2007)
5. Fiat, A.: Batch RSA. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 175–185. Springer, Heidelberg (1990)
6. Gentry, C.: Practical Identity-Based Encryption Without Random Oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 445–464. Springer, Heidelberg (2006)
7. Granger, R., Smart, N.: On computing products of pairings. Cryptology ePrint Archive: Report 2006/172 (2006)
8. Harn, L.: Batch verifying multiple DSA digital signatures. Electronics Letters 34(9), 870–871 (1998)
9. Harn, L.: Batch verifying multiple RSA digital signatures. Electronics Letters 34(12), 1219–1220 (1998)
10. Lim, C.H.: Efficient multi-exponentation and application to batch verification of digital signatures (2000), `http://dasan.sejong.ac.kr/~chlim/english$_$pub.html`
11. Yen, S.M., Laih, C.S.: Improved digital signature suitable for batch verification. IEEE Trans. Comput. 44(7), 957–959 (1995)
12. Naccache, D., MRaÏhi, D., Vaudenay, S., Raphaeli, D.: Can D.S.A. be improved? complexity trade-offs with the digital signature standard. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 77–85. Springer, Heidelberg (1995)
13. Naccache, D., Stern, J.: Signing on a Postcard. In: Frankel, Y. (ed.) FC 2000. LNCS, vol. 1962, pp. 121–135. Springer, Heidelberg (2001)

14. Paterson, K., Schuldt, J.: Efficient identity-based signatures secure in the standard model. In: Batten, L., Safavi-Naini, R. (eds.) ACISP 2006. LNCS, vol. 4058, pp. 207–222. Springer, Heidelberg (2006)
15. Shamir, A., Tauman, Y.: Improved online/offline signature schemes. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 355–367. Springer, Heidelberg (2001)
16. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)
17. Yoon, H., Cheon, J., Kim, Y.: Batch verifications with ID-based signatures. In: Park, C., Chee, S. (eds.) ICISC 2004. LNCS, vol. 3506, pp. 233–248. Springer, Heidelberg (2005)

Signcryption Scheme in Multi-user Setting without Random Oracles

Chik How Tan

NISlab, Department of Computer Science and Media Technology
Gjøvik University College, Norway
chik.tan@hig.no

Abstract. Since the notion of signcryption scheme was introduced by Zheng in 1997, many signcryption schemes were proposed. In 2002, An et al. introduced an insider and outsider security of signcryption scheme. Their insider security model was based on the so-called g-CCA2 security which was a relaxation of CCA2 security (adaptive chosen ciphertext attacks security). The g-CCA2 security means that an attacker is not allowed to query the "equivalent" challenged ciphertext (which is produced by the same challenged plaintext). Recently, many insider-secure signcryption schemes were proposed and provably insider-secure in the random oracles model based on the usual CCA2 security (not the g-CCA2 security), for example, Libert-Quisquater's signcryption schemes at PKC'2004 and SCN'2004 respectively, Yang et al.'s signcryption scheme at ISC'2005 and Ma's signcryption scheme at Inscrypt'2006. But, Tan showed that these signcryption schemes were not insider-secure against either adaptive chosen ciphertext attacks or forgery in two-user setting. In this paper, we proposed an insider-secure signcryption scheme based on encrypt-then-sign structure and it is provably insider-secure in a multi-user setting without random oracles based on the usual CCA2 security. The other advantage of the proposed signcryption scheme is that the ciphertext is publicly verifiable and corresponds to the unique plaintext.

Keywords: Signcryption, decisional bilinear Diffie-Hellman assumption.

1 Introduction

In 1997, Zheng [25] introduced the notion of signcryption scheme. The purpose of signcryption is to perform encryption and signature simultaneously more efficient than the usual encryption and signing approach. Since then, many signcryption schemes were proposed [7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 22, 24] and [25], etc. The formal security models of signcryption scheme in a two-user setting was introduced by Baek et al. [2], [3] and An et al. [1] independently in 2002. While the security model of signcryption scheme in a multi-user setting was proposed by An et al. [1]. The two-user setting means that the security is only considered a sender and a receiver, while the multi-user setting means that the security takes into the consideration of many users and does not only restrict to a sender and a receiver.

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 64–82, 2008.

In addition, An et al. [1] also defined two types of security on a signcryption scheme, that is, an outsider security and an insider security. The outsider security only protects an external attacker, that is, an attacker only knows the public keys of a sender and a receiver. While the insider security protects against attacks from his/her partner's private key being exposed. Consider two types of key exposed: (a) If the sender's private key is exposed, then an attacker is not able to recover the message from the ciphertext. (b) If the receiver's private key is exposed, then an attacker is not able to forge any signcryption message. An et al. [1]'s insider security model was based on the so-called g-CCA2 security which did not allow an attacker querying the "equivalent" challenged ciphertext (which was produced by the same challenged plaintext) in adaptive chosen ciphertext attacks. Recently, few attempts to construct insider-secure signcryption schemes based on the usual CCA2 security (adaptive chosen ciphertext attacks security), for example, Libert-Quisquater's signcryption schemes [13], [14] at PKC'2004 and SCN'2004 respectively. Both signcryption schemes [13], [14] were claimed to be insider-secure against adaptive chosen ciphertext attacks in the random oracles model (ROM), but Tan [19], [20] showed that both signcryption schemes were not insider-secure against adaptive chosen ciphertext attacks in the two-user setting in 2005 and 2006 respectively. Later, Yang et al. [24] further improved Libert-Quisquater's signcryption scheme [13] in the insider security and claimed that their improved signcryption scheme was insider-secure against adaptive chosen ciphertext attacks in the random oracles model. In 2006, Tan [21] further showed that Yang et al.'s signcryption scheme [24] was also not insider-secure against adaptive chosen ciphertext attacks even in the two-user setting. In 2006, Ma [15] also attempted to construct an insider-secure signcryption scheme (called short signcryption scheme) which was claimed to be insider-secure in the random oracles model, but it is showed in [23] that Ma's scheme is not insider-secure against weakly existential forgeability as the receiver can forge any signcrypt message of the sender. Due to many unsuccessful attempts of constructed signcryption schemes, a construction of signcryption scheme in multi-user setting still remains an interesting problem, especially an insider-secure signcryption scheme without random oracles.

Our Contributions

In this paper, we propose a signcryption scheme in the multi-user setting without random oracles. The proposed signcryption scheme is based on encrypt-then-sign structure and is different from most of signcryption schemes [13, 14, 15, 24] as they were based on sign-then-encrypt structure. As the proposed signcryption scheme is of encrypt-then-sign structure, the validity of ciphertext is publicly verifiable. If there is a dispute between the sender and the receiver, the receiver can send the plaintext and its signature to the trusted third party in the sign-then-encrypt structure. Then the trusted third party can check the correctness of the plaintext through its signature verification. But there is a challenge in the encrypt-then-sign structure to convince the trusted third party on the correctness of the plaintext/ciphertext pair. In this paper, we show that the proposed signcryption scheme has a unique plaintext-ciphertext pair property. This property

prevents the receiver sending incorrect plaintext/ciphertext pair to the trusted third party if there is a dispute between the sender and receiver. Furthermore, we also show that the proposed signcryption scheme is insider-secure against adaptive chosen ciphertext attacks without random oracles in the multi-user setting under the decisional bilinear Diffie-Hellman assumption, the collision resistant hash function assumption and the target collision hash function assumption. In addition, the proposed signcryption scheme is also insider-secure against strongly existential forgeability without random oracles in the multi-user setting.

Outline of the paper
This paper is organised as follows. Section 2 gives a definition of the insider security of signcryption scheme, the decisional bilinear Diffie-Hellman assumption, the collision resistant hash functions and the target collision resistant hash functions. In Section 3, a signcryption scheme is proposed. The proposed scheme is secure against insider attacker in the multi-user setting without random oracles. In Section 4, the security proofs of the proposed scheme are presented. The proposed signcryption scheme is provably insider-secure against adaptive chosen ciphertext attacks under the decisional bilinear Diffie-Hellman assumption and insider-secure against strongly existential forgeability without random oracles in the multi-user setting.

2 Preliminaries

2.1 Bilinear Maps and Assumptions

Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be cyclic groups of prime order p and g be a generator of $\mathcal{G}_1$. Let e be an admissible bilinear map from $\mathcal{G}_1 \times \mathcal{G}_1$ to $\mathcal{G}_2$ satisfying the following:

(a) Bilinear: for all $u, v \in \mathcal{G}_1$ and integers x, y, then $e(u^x, v^y) = e(u, v)^{xy}$.

(b) Non-degenerate: $e(g, g) \neq 1$.

(c) Computability: $\forall\ u, v \in \mathcal{G}_1$, $e(u, v)$ is efficiently computable.

The computational Diffie-Hellman (CDH) problem states that given a triple $(g,\ g^x,\ g^y) \in \mathcal{G}_1^3$ where $x,\ y \in \mathcal{Z}_p^*$, output g^{xy}.

Definition 1 (CDH Assumption). *A* CDH *problem in* $\mathcal{G}_1$ *is* $(\epsilon_{\mathtt{cdh}},\ t)$*-hard if for any probabilistic* t*-polynomial time algorithm* $\mathcal{A}$ *has a probability less than* $\epsilon_{\mathtt{cdh}}$ *in solving* CDH *problem in* $\mathcal{G}_1$*, that is,* $\Pr[\mathcal{A}(g,\ g^x,\ g^y) = g^{xy} \mid x,\ y \in \mathcal{Z}_p^*] < \epsilon_{\mathtt{cdh}}$.

Definition 2 (Bilinear Diffie-Hellman Problem (BDH)). *Given a quadruple* $(g, g^x, g^y, g^z) \in \mathcal{G}_1^4$ *as input where* $x, y, z \in \mathcal{Z}_p^*$*, output* $e(g,\ g)^{xyz}$.

Definition 3 (Decisional Bilinear Diffie-Hellman Problem (DBDH)). *Given a* 5*-tuple* $(g, g^x, g^y, g^z, T) \in \mathcal{G}_1^4 \times \mathcal{G}_2$ *as input where* $x, y, z \in \mathcal{Z}_p^*$*, output* 1 *if* $T = e(g,\ g)^{xyz}$ *and* 0 *otherwise. The advantage of an algorithm* $\mathcal{A}$ *in deciding* DBDH *problem is defined as follows:*

$$\mathrm{Adv}^{\mathtt{DBDH}}_{\mathcal{G}_1,\mathcal{G}_2}(\mathcal{A}) = \left|\Pr\left[\begin{matrix}\mathcal{A}(g, g^x, g^y, g^z, T) = 1 : \\ x, y, z \in \mathcal{Z}_p^* \text{ and } T = e(g, g)^{xyz}\end{matrix}\right] - \Pr\left[\begin{matrix}\mathcal{A}(g, g^x, g^y, g^z, T) = 1 : \\ x, y, z \in \mathcal{Z}_p^* \text{ and } T \in \mathcal{G}_2\end{matrix}\right]\right|.$$

Definition 4 (DBDH Assumption). *A* DBDH *problem in* $\mathcal{G}_1 \times \mathcal{G}_2$ *is* $(\epsilon_{\mathtt{dbdh}}, t)$*-hard if for any probabilistic* t*-polynomial time algorithm* $\mathcal{A}$ *has an advantage less than* $\epsilon_{\mathtt{dbdh}}$ *in deciding the* DBDH *problem* $(\mathrm{Adv}^{\mathtt{DBDH}}_{\mathcal{G}_1,\mathcal{G}_2}(\mathcal{A}) < \epsilon_{\mathtt{dbdh}})$.

Now, we give a definition of a collision resistant hash function and a target collision resistant hash function as follows.

Definition 5 (Collision Resistance). *Let* l *and* n *be two positive integers and* $\mathcal{H} = \{H_\lambda : \{0,1\}^l \to \{0,1\}^n\}_{\lambda \in \Lambda}$ *be a family of hash function, where* Λ *is an index set.* $\mathcal{H}$ *is said to be* $(\epsilon_{\mathtt{cr}}, t)$-CR *collision resistant hash function if for any probabilistic* t*-polynomial time algorithm* $\mathcal{A}$*, which outputs* $x,\ y \in \{0,1\}^l$ *such that* $H_\lambda(x) = H_\lambda(y)$ *and* $x \neq y$ *for a given* $\lambda \in \Lambda$*, is of probability less than* $\epsilon_{\mathtt{cr}}$*. Formally, a collision resistant hash function is defined as follows*

$$\Pr\left[\ y \leftarrow \mathcal{A}(H_\lambda) \text{ and } y \neq x \ \middle|\ \text{given } H_\lambda \in \mathcal{H};\ H_\lambda(y) = H_\lambda(x)\ \right] < \epsilon_{\mathtt{cr}}$$

for any probabilistic t*-polynomial time algorithm* $\mathcal{A}$.

Definition 6 (Target Collision Resistance). *Let* l *and* n *be two positive integers and* $\mathcal{H} = \{H_\lambda : \{0,1\}^l \to \{0,1\}^n\}_{\lambda \in \Lambda}$ *be a family of hash function, where* Λ *is an index set.* $\mathcal{H}$ *is said to be* $(\epsilon_{\mathtt{tcr}}, t)$-TCR *target collision resistant hash function if for any probabilistic* t*-polynomial time algorithm* $\mathcal{A}$*, which outputs* y *such that* $H_\lambda(y) = H_\lambda(x)$ *and* $y \neq x$ *for a given* $x \in \{0,1\}^l$ *and chosen* $\lambda \in \Lambda$*, is of probability less than* $\epsilon_{\mathtt{tcr}}$*. Formally, a target collision resistant hash function is defined as follows*

$$\Pr\left[\ H_\lambda(y) = H_\lambda(x) \text{ and } y \neq x \ \middle|\ \begin{array}{l}\text{given } x \in \{0,1\}^l, \text{ choose } H_\lambda \in \mathcal{H};\\ y \leftarrow \mathcal{A}(H_\lambda, x)\end{array}\right] < \epsilon_{\mathtt{tcr}}$$

for any probabilistic t*-polynomial time algorithm* $\mathcal{A}$.

2.2 Boneh-Boyen Short Signature Scheme (BB)

In this subsection, we briefly describe Boneh-Boyen short signature scheme [4] which will be used in the construction of the signcryption scheme later. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be groups of prime order p, let g be a generator of $\mathcal{G}_1$ and e be an admissible bilinear map from $\mathcal{G}_1 \times \mathcal{G}_1$ into $\mathcal{G}_2$. Let $H : \{0,1\}^l \to \{0,1\}^n$ be a collision resistant hash function such that $n \leq \lfloor \log_2 p \rfloor$.

Key Generation: Choose two randoms $\mathsf{x}, \mathsf{y} \in \mathcal{Z}_p^*$ and compute $\mathsf{g_1} = g^{\mathsf{x}}$ and $\mathsf{g_2} = g^{\mathsf{y}}$. Then, the public key is $(\mathsf{g_1}, \mathsf{g_2})$ and the private key is (x, y).

Signature Generation: To sign a message m, choose a random $r \in \mathcal{Z}_p^*$ and compute $\sigma = g^{\frac{1}{H(m)+\mathsf{x}+\mathsf{y}r} \bmod p}$. If $H(m) + \mathsf{x} + \mathsf{y}r \neq 0 \bmod p$, then the signature of m is (σ, r), otherwise try a different r.

Signature Verification: Given a public key $(\mathsf{g_1}, \mathsf{g_2})$, to verify a signature (σ, r) on a message m, the verifier checks $e(\sigma,\ g^{H(m)}.\mathsf{g_1}.\mathsf{g_2}^r) \stackrel{?}{=} e(g,\ g)$. If the equation holds, then the signature is valid, otherwise invalid.

In [4], the authors showed that Boneh-Boyen short signature scheme is secure against adaptive chosen message attacks without random oracles based on the q-strong Diffie-Hellman problem [4] which is stated as follows:

Definition 7 (*q*-Strong Diffie-Hellman Problem (q-SDH)). *A q-SDH problem in $\mathcal{G}_1$ is defined as follows: given an input of $(q+2)$-tuple $(g_1, g_2, g_2^x, g_2^{x^2}, \cdots, g_2^{x^q})$, output a pair $(c, g_1^{1/(x+c)})$ where $c \in Z_p^*$.*

2.3 Signcryption Scheme (SC)

A signcryption scheme $\mathtt{SC} = (\mathcal{P}_{\mathtt{sc}}, \mathcal{K}_{\mathtt{sc}}, \mathcal{S}_{\mathtt{sc}}, \mathcal{D}_{\mathtt{sc}})$ consists of four algorithms, that is, system parameters generation algorithm $\mathcal{P}_{\mathtt{sc}}$, key generation algorithm $\mathcal{K}_{\mathtt{sc}}$, signcrypt algorithm $\mathcal{S}_{\mathtt{sc}}$ and de-signcrypt algorithm $\mathcal{D}_{\mathtt{sc}}$, which are defined as follows.

∘ System Parameters Generation $\mathcal{P}_{\mathtt{sc}}$: Given a security parameter, this algorithm generates a common system parameters for use in key generation algorithm.

∘ Key Generation $\mathcal{K}_{\mathtt{sc}}$: Given system parameters, the algorithm generates a receiver's (r) and sender's (s) public/private key pairs $(pk_{\mathsf{r}}, sk_{\mathsf{r}})$ and $(pk_{\mathsf{s}}, sk_{\mathsf{s}})$ respectively, where pk_{u} and sk_{u} are the public key and the private key respectively for $\mathsf{u} \in \{\mathsf{r}, \mathsf{s}\}$.

∘ Signcrypt Algorithm $\mathcal{S}_{\mathtt{sc}}$: Given system parameters, a sender's private key sk_{s}, a receiver's public key pk_{r} and a plaintext m, the algorithm $\mathcal{S}_{\mathtt{sc}}$ returns a ciphertext as $c = \mathcal{S}_{\mathtt{sc}}(sk_{\mathsf{s}}, pk_{\mathsf{r}}, m)$.

∘ De-signcrypt Algorithm $\mathcal{D}_{\mathtt{sc}}$: Given system parameters, a receiver's private key sk_{r}, a sender's public key pk_{s} and a ciphertext c, the algorithm $\mathcal{D}_{\mathtt{sc}}$ returns either a plaintext $m = \mathcal{D}_{\mathtt{sc}}(sk_{\mathsf{r}}, pk_{\mathsf{s}}, c)$ or reject symbol $\perp$.

The security of a signcryption scheme is confidentiality and authenticity. We first define confidential security of a signcryption scheme as an attack game against adaptive chosen ciphertext attacks in the insider security between a challenger and an adversary as follows:

• Setup. The challenger first generates system parameters and runs the key generation algorithm $\mathcal{K}_{\mathtt{sc}}$ which generates a receiver's public/private key pair $(pk_{\mathsf{r}}^*, sk_{\mathsf{r}}^*)$. The challenger gives system parameters and the receiver's public key pk_{r}^* to an adversary $\mathcal{A}_{\mathtt{sc}}$ and keeps sk_{r}^* secret.

• Phase-1. $\mathcal{A}_{\mathtt{sc}}$ makes a number of de-signcrypt queries. In a de-signcrypt query, $\mathcal{A}_{\mathtt{sc}}$ submits a ciphertext c with a receiver's/sender's public key $(pk_{\mathsf{r}}, pk_{\mathsf{s}})$ to the challenger (pk_{r} can be equal to pk_{r}^*). The challenger runs the de-signcrypt oracle $\mathcal{D}_{\mathtt{sc}}^{\mathcal{O}}$ which returns either the plaintext $m = \mathcal{D}_{\mathtt{sc}}^{\mathcal{O}}(sk_{\mathsf{r}}, pk_{\mathsf{s}}, c)$ or reject symbol $\perp$.

• Challenge. $\mathcal{A}_{\mathtt{sc}}$ chooses two equal-length plaintexts m_0, m_1 and a sender's public/private key pair $(pk_{\mathsf{s}}^*, sk_{\mathsf{s}}^*)$; and sends these to the challenger. The challenger chooses a random $b \in \{0, 1\}$, computes a ciphertext $c^* = \mathcal{S}_{\mathtt{sc}}^{\mathcal{O}}(sk_{\mathsf{s}}^*, pk_{\mathsf{r}}^*, m_b)$ where $\mathcal{S}_{\mathtt{sc}}^{\mathcal{O}}$ is the signcrypt oracle; and gives c^* to $\mathcal{A}_{\mathtt{sc}}$ as a challenged ciphertext.

• Phase-2. $\mathcal{A}_{\mathtt{sc}}$ continually makes a number of de−signcrypt queries. $\mathcal{A}_{\mathtt{sc}}$ is not allowed to query the de−signcrypt oracle $\mathcal{D}_{\mathtt{sc}}^{\mathcal{O}}$ on the challenged ciphertext c^* with the same sender's public/private key $(pk_{\mathtt{s}}^*, sk_{\mathtt{s}}^*)$, but $\mathcal{A}_{\mathtt{sc}}$ is allowed to query the de−signcrypt oracle $\mathcal{D}_{\mathtt{sc}}^{\mathcal{O}}$ on the challenged ciphertext c^* with a different sender's public/private key or different receiver's public key.

• Guess. At the end of the game, $\mathcal{A}_{\mathtt{sc}}$ outputs bits b' and wins if $b' = b$.

The advantage of $\mathcal{A}_{\mathtt{sc}}$ is defined as $\mathrm{Adv}_{\mathtt{sc}}^{\mathtt{is-cca2}}(\mathcal{A}_{\mathtt{sc}}) = |\Pr[b' = b] - 1/2|$.

Definition 8 (Insider-secure CCA2). *A signcryption scheme* SC *is* $(\epsilon_{\mathtt{sc}}, t, q_d)$-IS-CCA2 *secure in the multi-user setting if for any probabilistic t-polynomial time adversary* $\mathcal{A}_{\mathtt{sc}}$ *has an advantage less than* $\epsilon_{\mathtt{sc}}$ *after at most* q_d *de−signcrypt queries.*

For the authenticity of a signcryption scheme, it is normally provided by signature scheme. A digital signature scheme is said to be secure if it is existentially unforgeable against adaptive chosen message attacks. This means that the adversary is not able to produce a valid signature of a new message. This normally refers to a weakly existential unforgeability [11]. While a strongly existential unforgeability means that an adversary is unable to produce a new valid signature of a message m where m may be the previously signed message. Now, we define an attack game of a strongly existential unforgeability of signcryption scheme SC against chosen "message" (message refers to a plaintext if it is sign-then-encrypt structure, and a ciphertext if it is encrypt-then-sign structure) attacks in the insider security between a challenger and a forger as follows:

• Setup. Given system parameters, the challenger first generates the sender's public/private key pair $(pk_{\mathtt{s}}^*, sk_{\mathtt{s}}^*)$ and gives $pk_{\mathtt{s}}^*$ to a forger $\mathcal{F}_{\mathtt{sc}}$.

• Signcrypt Query. When the forger $\mathcal{F}_{\mathtt{sc}}$ makes a signcrypt query, the forger $\mathcal{F}_{\mathtt{sc}}$ first chooses a plaintext m, a receiver's public/private key $(pk_{\mathtt{r}}, sk_{\mathtt{r}})$ and a sender's public key $pk_{\mathtt{s}}$ ($pk_{\mathtt{s}}$ can be chosen to be $pk_{\mathtt{s}}^*$), then sends $(m, pk_{\mathtt{r}}, sk_{\mathtt{r}}, pk_{\mathtt{s}})$ to the challenger. The challenger then runs the signcrypt oracle $\mathcal{S}_{\mathtt{sc}}^{\mathcal{O}}$ which returns the ciphertext $c = \mathcal{S}_{\mathtt{sc}}^{\mathcal{O}}(sk_{\mathtt{s}}, pk_{\mathtt{r}}, m)$; and gives c to $\mathcal{F}_{\mathtt{sc}}$.

• Forgery. After a number of signcrypt query, $\mathcal{F}_{\mathtt{sc}}$ chooses a receiver's public/private key $(pk_{\mathtt{r}}^*, sk_{\mathtt{r}}^*)$ and produces a ciphertext c^* of a plaintext m^* for the sender's public key $pk_{\mathtt{s}}^*$. The forger $\mathcal{F}_{\mathtt{sc}}$ wins the game if c^* is a valid ciphertext, where c^* is not the outputs from signcrypt query before.

Definition 9 (Strongly Existential Unforgeability). *A signcryption scheme* SC *is* $(\epsilon_{\mathtt{sc}}, t, q_{\mathtt{s}})$*-strongly existential unforgeability against adaptive chosen "message"*[1] *attacks in the insider security and the multi-user setting if for any probabilistic t-polynomial time forger* $\mathcal{F}_{\mathtt{sc}}$ *outputs a valid ciphertext with probability less than* $\epsilon_{\mathtt{sc}}$ *after at most* $q_{\mathtt{s}}$ *signcrypt queries.*

[1] In sign-then-encrypt structure, message refers to a plaintext, while in encrypt-then-sign structure, it refers to a ciphertext.

3 Proposed Signcryption Scheme

The construction of a signcryption scheme is based on Boneh-Boyen identity-based encryption scheme [5], [6] and Boneh-Boyen short signature scheme [4]. The proposed signcryption scheme (called SC) is based on encrypt-then-sign structure which is different from most of signcryption schemes as they were mainly based on sign-then-encrypt structure. The proposed signcryption scheme also has a unique plaintext-ciphertext pair property and is described as follows:

System Parameters Setup: Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two groups of prime order p, let g be a generator of $\mathcal{G}_1$ and e be an admissible bilinear map from $\mathcal{G}_1 \times \mathcal{G}_1$ into $\mathcal{G}_2$. Let $\mathcal{H}_i = \{H_{k_i} : \{0,1\}^{l_i} \rightarrow \{0,1\}^{n_i}\}_{\lambda_i \in \Lambda_i}$ be the families of hash functions indexed by $\lambda_i \in \Lambda_i$, l_i and n_i are integers for $i \in \{1,2,3\}$ such that $\mathcal{H}_1$ is a target collision resistant hash function, $\mathcal{H}_2$ and $\mathcal{H}_3$ are collision resistant hash functions; and $n_i \leq \lfloor \log_2 p \rfloor$ for $i = 1,2,3,$. Choose $g_1, g_2 \in \mathcal{G}_1$ and $H_i \in \mathcal{H}_i$ for $i = 1,2,3$, then the system parameters are $\mathsf{SP} = (g, g_1, g_2, H_1, H_2, H_3)$.

Key Generation: We consider the key generation of a receiver and a sender separately as follows:

(i) For a receiver : Choose randoms $x, u \in \mathcal{Z}_p^*$ and compute $\mathsf{h}_1 = g^x$, $\mathsf{h}_2 = \mathsf{h}_1^u$, and $Z = e(\mathsf{h}_1, g_1)$. Then the public key is $\mathsf{PK_r} = (\mathsf{h}_1, \mathsf{h}_2, Z)$ and the private key is $\mathsf{SK_r} = (x, u)$.

(ii) For a sender: Choose randoms $\mathsf{x}, \mathsf{y} \in \mathcal{Z}_p^*$ and compute $\mathsf{g_1} = g^{\mathsf{x}}$, $\mathsf{g_2} = g^{\mathsf{y}}$ and $\mathsf{Z} = e(g, g)$. Then the public key is $\mathsf{PK_s} = (\mathsf{g_1}, \mathsf{g_2}, \mathsf{Z})$ and the private key is $\mathsf{SK_s} = (\mathsf{x}, \mathsf{y})$[2].

Signcrypt: To signcrypt a plaintext m for a user r, a sender s first chooses randoms $w,\ \beta,\ \mu \in \mathcal{Z}_p^*$ and computes the followings:

$$c_1 = g^w,\ \ c_2 = Z^w m,\ \ v = H_2(c_1, c_2, \mathsf{PK_r}, \mathsf{PK_s}),\ \ \sigma = g^{1/(v+\mathsf{x}+\mathsf{y}\beta) \bmod p},$$
$$\eta = H_3(\sigma, \beta, \mathsf{PK_r}, \mathsf{PK_s}),\ \ \alpha = H_1(\mathsf{h}_1^\eta \mathsf{h}_2^\mu),\ \ c_3 = (g_1^\alpha\ g_2)^w.$$

Then, the ciphertext is $C = (c_1, c_2, c_3, \sigma, \beta, \mu)$.

De-signcrypt: Upon receipt of a ciphertext $C = (c_1, c_2, c_3, \sigma, \beta, \mu)$, the receiver r computes

$$v = H_2(c_1, c_2, \mathsf{PK_r}, \mathsf{PK_s}),\ \ \eta = H_3(\sigma, \beta, \mathsf{PK_r}, \mathsf{PK_s}),\ \ \alpha = H_1(\mathsf{h}_1^{\eta+\mu u})$$

and checks $e(c_3,\ g) \stackrel{?}{=} e(g_1^\alpha\ g_2,\ c_1)$ and $e(\sigma,\ g^v\ \mathsf{g_1}\ \mathsf{g_2}^\beta) \stackrel{?}{=} \mathsf{Z}$. If one of the two equations is not equal, output reject symbol $\perp$, otherwise compute the plaintext as $m = \frac{c_2}{e(c_1,\ g_1^x)}$.

Publicly Verifiable Ciphertext: If given system parameters $\mathsf{SP} = (g, g_1, g_2, H_1, H_2, H_3)$, the receiver's public key $\mathsf{PK_r} = (\mathsf{h}_1, \mathsf{h}_2, Z)$, the sender's public key $\mathsf{PK_s} = (\mathsf{g_1}, \mathsf{g_2}, \mathsf{Z})$ and a ciphertext $C = (c_1, c_2, c_3, \sigma, \beta, \mu)$, anyone can compute

[2] It is note that $\mathsf{PK_r}$ and $\mathsf{PK_s}$ are different structure as Z and Z, h_2 and g_2 are different structure respectively.

$$v = H_2(c_1, c_2, \mathsf{PK_r}, \mathsf{PK_s}),\ \ \eta = H_3(\sigma, \beta, \mathsf{PK_r}, \mathsf{PK_s}),\ \ \alpha = H_1(\mathsf{h}_1^{\eta}\mathsf{h}_2^{\mu})$$

and checks $e(c_3,\ g) \stackrel{?}{=} e(g_1^{\alpha}\ g_2,\ c_1)$ and $e(\sigma,\ g^{v}\ \mathsf{g}_1\ \mathsf{g}_2^{\beta}) \stackrel{?}{=} \mathsf{Z}$. If both equations are equal, then C is a correct ciphertext, otherwise incorrect.

Verifiable Plaintext/Ciphertext Pair: If there is a dispute between the sender and the receiver, then the receiver will prove to the trusted third party on the plaintext-ciphertext pair which is in fact produced by the sender. First, for a given ciphertext $C = (c_1, c_2, c_3, \sigma, \beta, \mu)$, the receiver checks the validity of the ciphertext C as

$$e(c_3,\ g) \stackrel{?}{=} e(g_1^{\alpha}\ g_2,\ c_1) \quad \text{and} \quad e(\sigma,\ g^{v}\ \mathsf{g}_1\ \mathsf{g}_2^{\beta}) \stackrel{?}{=} \mathsf{Z},$$

where $v = H_2(c_1, c_2, \mathsf{PK_r}, \mathsf{PK_s})$, $\eta = H_3(\sigma, \beta, \mathsf{PK_r}, \mathsf{PK_s})$ and $\alpha = H_1(\mathsf{h}_1^{\eta}\mathsf{h}_2^{\mu})$. If all the above equations hold, then the receiver de-signcrypts C to obtain the plaintext m and computes $c_4 = c_1^x$. The receiver sends C and (c_4, m) to the trusted third party for verification of ciphertext produced by the sender s. The trusted third party first computes

$$v = H_2(c_1, c_2, \mathsf{PK_r}, \mathsf{PK_s}),\ \ \eta = H_3(\sigma, \beta, \mathsf{PK_r}, \mathsf{PK_s}),\ \ \alpha = H_1(\mathsf{h}_1^{\eta}\mathsf{h}_2^{\mu})$$

and checks the followings

$$e(c_4,\ g) \stackrel{?}{=} e(c_1,\ \mathsf{h}_1), \quad e(c_4,\ g_1) \stackrel{?}{=} \frac{c_2}{m}, \tag{1}$$

$$e(c_3,\ g) \stackrel{?}{=} e(g_1^{\alpha}\ g_2,\ c_1) \quad \text{and} \quad e(\sigma,\ g^{v}\ \mathsf{g}_1\ \mathsf{g}_2^{\beta}) \stackrel{?}{=} \mathsf{Z}. \tag{2}$$

If all the above equations hold, then the trusted third party is convinced that the ciphertext is produced by the sender s with a plaintext m as no forger can forge the sender's signature on (c_1, c_2) (the signature is generated by Boneh-Boyen short signature scheme) and the receiver cannot produce another $(\bar{m}, \bar{c}_4)$ for the same (c_1, c_2). We show that (c_4, m) is unique for a given (c_1, c_2) as follows.

Proposition 1. *Given* $(g, g_1, \mathsf{h}_1 = g^x, c_1, c_2) \in \mathcal{G}_1^4 \times \mathcal{G}_2$, *then* (c_4, m) *is a unique pair satisfying* (1) *where* $c_4 = c_1^x$ *and* $m = \frac{c_2}{e(c_4,\ g_1)}$.

Proof: Assume that there is another pair $(\bar{c}_4, \bar{m}) \neq (c_4, m)$ satisfying (1), then we have

$$e(\bar{c}_4,\ g) = e(c_1,\ \mathsf{h}_1), \quad e(\bar{c}_4,\ g_1) = \frac{c_2}{\bar{m}}. \tag{3}$$

Hence, we obtain $e(c_4,\ g) = e(\bar{c}_4,\ g)$ and $e(\bar{c}_4 c_4^{-1},\ g) = 1$. Assume that $\bar{c}_4 = g^k$ and $c_4 = g^j$ for some $0 < k, j < p$, then, we have $e(g, g)^{k-j} = 1$. By the bilinear property of e, $e(g, g) \neq 1$, hence, $k = j$ and $\bar{c}_4 = c_4$. By (3), we obtain $\bar{m} = m$. This contradicts to the assumption that $(\bar{c}_4, \bar{m}) \neq (c_4, m)$. Hence, we complete the proof. ♢

Performance Analysis: Let L_1 and L_2 be the length of the binary representation of an element in $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively. Let $L_p = \lceil \log_2 p \rceil$. Then, the length

Table 1. Performance Comparisons and Properties

	Proposed Scheme	**Libert-Quisquater's Scheme** [13]	**Yang et al.'s Scheme** [24]
Structure	Encrypt-then-Sign	Sign-then-Encrypt	Sign-then-Encrypt
Security	secure without Random Oracles*	insecure in ROM	insecure in ROM
Publicly Verifiable Ciphertext	Yes	No	No
`Sender's`			
PK size (bits)	$2L_1 + L_2$	L_1	L_1
SK size (bits)	$2L_p$	L_p	L_p
`Receiver's`			
PK size (bits)	$2L_1 + L_2$	L_1	L_1
SK size (bits)	$2L_p$	L_p	L_p
Ciphertext size (bits)	$3L_1 + L_2 + 2L_p$	$3L_1 + L_m$	$3L_1 + L_m$
Signcrypt	3 exp, 2 m-exp	3 exp	3 exp
Designcrypt	2 exp, 1 m-exp, 4 pairings	1 exp, 2 pairings	1 exp, 2 pairings
Trusted Third Party Verification	1 exp, 1 m-exp, 6 pairings	2 pairings	2 pairings

Note:

PK : Public Key
SK : Private Key
L_m : message size in bits
* : The security proof is in Section 4

of the system parameter is $3L_1$ bits. The lengths of the sender's and the receiver's public/private key are the same respectively and is equal to $2L_1 + L_2$ bits and $2L_p$ bits respectively. The ciphertext length of the proposed signcryption scheme is $3L_1 + L_2 + 2L_p$ bits. The computational complexity of the signcrypt algorithm requires 3 exponentiations (exp) and 2 multi-exponentiations (m-exp). While the computational complexity of the de-signcrypt algorithm requires 2 exponentiations, 1 multi-exponentiation and 4 pairings. For the verifiable plaintext/ciphertext pair, a trusted third party requires 1 exponentiation, 1 multi-exponentiation and 6 pairings.

In the following table, we list the performance comparisons and properties among the proposed signcryption scheme, Libert-Quisquater's signcryption scheme [13] and Yang et al.'s signcryption scheme [24].

From Table 1, although Libert-Quisquater's signcryption scheme and Yang et al.'s signcryption scheme are more efficient than the proposed signcryption scheme, they are not even insider-secure in the random oracles model and the ciphertext is not publicly verifiable. While the proposed signcryption scheme is insider-secure without random oracles (the security proof is in Section 4) and the ciphertext is publicly verifiable.

4 Security Analysis

In this section, we show that the proposed signcryption scheme is insider-secure against adaptive chosen ciphertext attacks and insider-secure against strongly existential forgeability under adaptive chosen "message" attacks in the multi-user setting without random oracles. Before we state the main theorem, we first list the "Difference Lemma" defined by Shoup [18] as follows:

Lemma 1 ([18], Difference Lemma)**.** *Let* T_1*,* T_2 *and* F *be events defined on some probability space. Suppose that the event* $\mathsf{T}_2 \wedge \neg\mathsf{F}$ *occurs if and only if* $\mathsf{T}_1 \wedge \neg\mathsf{F}$ *occurs. Then* $\left|\Pr[\mathsf{T}_2] - \Pr[\mathsf{T}_1]\right| \leq \Pr[\mathsf{F}]$.

We first prove that the proposed signcryption scheme is insider-secure in the multi-user setting against adaptive chosen ciphertext attacks as follows:

Theorem 1. *The proposed signcryption scheme* SC *is* $(\epsilon_{\mathtt{sc}}, t, q_d)$-IS-CCA2 *secure in the multi-user setting without random oracles if the* $(\epsilon_{\mathtt{dbdh}}, t)$-DBDH *assumption holds in* $\mathcal{G}_1 \times \mathcal{G}_2$*,* H_1 *is a* $(\epsilon_{\mathtt{tcr}}, t)$-TCR *target collision resistant hash function and* H_2 *and* H_3 *are a* $(\epsilon_{\mathtt{cr}}, t)$-CR *and a* $(\bar{\epsilon}_{\mathtt{cr}}, t)$-CR *collision resistant hash functions respectively, such that*

$$\epsilon_{\mathtt{sc}} \leq \epsilon_{\mathtt{dbdh}} + 2\epsilon_{\mathtt{tcr}} + \epsilon_{\mathtt{cr}} + \bar{\epsilon}_{\mathtt{cr}} + \frac{q_d}{p},$$

where q_d *is the number of de-signcrypt queries and* $q_d < p$*.*

Proof: The theorem proceeds by reductionist proof, that is, suppose an adversary $\mathcal{A}_{\mathtt{sc}}$ can break the proposed signcryption scheme in the sense of insider-secure CCA2 (IS-CCA2) in the multi-user setting, then one can construct an algorithm $\mathcal{B}$ which solves the decisional bilinear Diffie-Hellman assumption (DBDH) in a random instance with advantage $\epsilon_{\mathtt{dbdh}}$. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two cyclic groups of prime order p and g be a generator of $\mathcal{G}_1$. Let e be an admissible bilinear map from $\mathcal{G}_1 \times \mathcal{G}_1$ to $\mathcal{G}_2$. First, the algorithm $\mathcal{B}$ (as a simulator) is given a 5-tuple $(g, g^{x^*}, g^{y^*}, g^{w^*}, T) \in \mathcal{G}_1^4 \times \mathcal{G}_2$. The goal of $\mathcal{B}$ is to output 1 if $T = e(g,g)^{x^*y^*w^*}$ and 0 otherwise. The algorithm $\mathcal{B}$ interacts with an adversary $\mathcal{A}_{\mathtt{sc}}$ in the IS-CCA2 game as follows:

Setup: The algorithm $\mathcal{B}$ sets $\mathsf{h}_1^* = g^{x^*}$, $g_1 = g^{y^*}$ and $c_1^* = g^{w^*}$, then $\mathcal{B}$ chooses randoms k^*, γ, $u^* \in \mathcal{Z}_p^*$ and computes

$$\alpha^* = H_1((\mathsf{h}_1^*)^{k^*}), \quad g_2 = g_1^{-\alpha^*} \cdot g^{\gamma}, \quad \mathsf{h}_2^* = (\mathsf{h}_1^*)^{u^*} \quad \text{and} \quad Z^* = e(\mathsf{h}_1^*, g_1).$$

Let H_1 be a target collision resistant hash function and H_2 and H_3 be collision resistant hash functions chosen from $\mathcal{H}_1$, $\mathcal{H}_2$ and $\mathcal{H}_3$ respectively. Then, the system parameters are $\mathsf{SP} = (g, g_1, g_2, H_1, H_2, H_3)$, the receiver's public key is $\mathsf{PK}_{\mathsf{r}}^* = (\mathsf{h}_1^*, \mathsf{h}_2^*, Z^*)$ and the receiver's private key is $\mathsf{SK}_{\mathsf{r}}^* = (x^*, u^*)$ where x^* is unknown to $\mathcal{B}$. The algorithm $\mathcal{B}$ gives the system parameters SP and the receiver's public key $\mathsf{PK}_{\mathsf{r}}^*$ to $\mathcal{A}_{\mathtt{sc}}$.

Phase-1: In this phase, the adversary $\mathcal{A}_{\mathsf{sc}}$ makes a number of de-signcrypt queries. If the adversary submits a ciphertext $C = (c_1, c_2, c_3, \sigma, \beta, \mu)$ with the receiver's public key $\mathsf{PK_r} = (\mathsf{h}_1, \mathsf{h}_2, Z)$ and the sender's public key $\mathsf{PK_s} = (\mathsf{g}_1, \mathsf{g}_2, \mathsf{Z})$ for de-signcrypt query, $\mathcal{B}$ first checks $c_1 = c_1^*$. If they are equal, the simulation aborts, otherwise $\mathcal{B}$ computes

$$v = H_2(c_1, c_2, \mathsf{PK_r}, \mathsf{PK_s}), \quad \eta = H_3(\sigma, \beta, \mathsf{PK_r}, \mathsf{PK_s}), \quad \alpha = H_1(\mathsf{h}_1^{\eta}\mathsf{h}_2^{\mu})$$

and checks $e(c_3,\ g) \stackrel{?}{=} e(g_1^{\alpha}\ g_2,\ c_1)$ and $e(\sigma,\ g^{v}\ \mathsf{g}_1\ \mathsf{g}_2^{\beta}) \stackrel{?}{=} \mathsf{Z}$. If one of the two equations are not equal or $\alpha = \alpha^*$, then return reject symbol $\perp$, otherwise $\mathcal{B}$ computes g_1^w (where $c_1 = g^w$ and $\mathcal{B}$ does not know the exponents of g_1 and c_1 over g respectively) as follows: Since $g_2 = g_1^{-\alpha^*} g^{\gamma}$, then $g^{\gamma} = g_1^{\alpha^*} g_2$. $\mathcal{B}$ computes c_1^{γ} which is equal to $(g_1^{\alpha^*} g_2)^w$. By $c_1^{\gamma} = (g_1^{\alpha^*} g_2)^w$ and $c_3 = (g_1^{\alpha} g_2)^w$, $\mathcal{B}$ obtains

$$g_1^w = (c_3 c_1^{-\gamma})^{\frac{1}{\alpha - \alpha^*}}$$

and returns the plaintext m as

$$m = \frac{c_2}{e(g_1^w,\ \mathsf{h}_1)}.$$

Challenge: After the number of queries in `Phase 1`, $\mathcal{A}_{\mathsf{sc}}$ chooses two equal length plaintexts m_0 and m_1 and a sender's public/private key $(\mathsf{PK_s^*}, \mathsf{SK_s^*})$, where $\mathsf{PK_s^*} = (\mathsf{g}_1^*,\ \mathsf{g}_2^*,\ \mathsf{Z}^*)$ and $\mathsf{SK_s^*} = (\mathsf{x}^*,\ \mathsf{y}^*)$ such that $\mathsf{g}_1^* = g^{\mathsf{x}^*}$ and $\mathsf{g}_2^* = g^{\mathsf{y}^*}$, on which it wishes to be challenged. Then $\mathcal{A}_{\mathsf{sc}}$ sends (m_0, m_1) and $(\mathsf{PK_s^*}, \mathsf{SK_s^*})$ to $\mathcal{B}$. $\mathcal{B}$ chooses $\beta^* \in \mathcal{Z}_p^*$ and $b \in \{0, 1\}$; and responds the challenge ciphertext $C^* = (c_1^*, c_2^*, c_3^*, \sigma^*, \beta^*, \mu^*)$ as follows

$$\begin{aligned}
&c_1^* = g^{w^*}, \quad c_2^* = T \cdot m_b, \quad c_3^* = (c_1^*)^{\gamma} = (g_1^{\alpha^*} \cdot g_2)^{w^*},\\
&v^* = H_2(c_1^*, c_2^*, \mathsf{PK_r^*}, \mathsf{PK_s^*}), \quad \sigma^* = g^{\frac{1}{v^* + \mathsf{x}^* + \mathsf{y}^* \beta^*} \bmod p},\\
&\eta^* = H_3(\sigma^*, \beta^*, \mathsf{PK_r^*}, \mathsf{PK_s^*}), \quad \mu^* = \frac{k^* - \eta^*}{u^*} \bmod p.
\end{aligned}$$

Phase-2: The adversary $\mathcal{A}_{\mathsf{sc}}$ continually makes de-signcrypt query on ciphertext C which is similar as Phase-1 except that $\mathcal{A}_{\mathsf{sc}}$ is not allowed to make a de-signcrypt query on $C = C^*$ with the same receiver's/sender's public key $(\mathsf{PK_r^*}, \mathsf{PK_s^*})$, but $\mathcal{A}_{\mathsf{sc}}$ is allowed to query de-signcrypt oracle on C^* with different receiver's public key $\mathsf{PK_r}$ or sender's public key $\mathsf{PK_s}$ such that $\mathsf{PK_r} \neq \mathsf{PK_r^*}$ or $\mathsf{PK_s} \neq \mathsf{PK_s^*}$ respectively.

Guess: After the number of de-signcrypt queries, the adversary $\mathcal{A}_{\mathsf{sc}}$ finally outputs a bit $b' \in \{0, 1\}$. If $b = b'$, $\mathcal{B}$ returns 1, otherwise returns 0.

This completes the simulation of the attacks game of the signcryption between the algorithm $\mathcal{B}$ and the adversary $\mathcal{A}_{\mathsf{sc}}$.

Analysis: To ease the analysis on the success probability of $\mathcal{B}$, a sequence of games technique is used, which was introduced by Shoup [18]. The sequence of games

starts from game G_0 to game G_5, where G_0 is the original attack game and the last game G_5 gives no advantage to the adversary $\mathcal{A}_{\mathtt{sc}}$. Let T_i be the event that $b' = b$ in the game G_i for $0 \leq i \leq 5$. Then, we have

$$\mathrm{Adv}_{\mathtt{sc}}^{\mathtt{is-cca2}}(\mathcal{A}_{\mathtt{sc}}) = \left|\Pr[\mathsf{T}_0] - 1/2\right|$$

and the sequence of games are described as follows:

Game G_1 (Eliminate the correct guess of c_1 in Phase-1)**.** First, game G_1 is obtained from game G_0 by modifying the de-signcrypt oracle in Phase-1 to reject a ciphertext $C = (c_1, c_2, c_3, \sigma, \beta, \mu)$ such that $c_1 = c_1^*$, then the de-signcrypt oracle outputs reject and halt. Since an adversary has no information about $c_1 = c_1^*$ from the challenge ciphertext C^*, the probability of this type of ciphertext submitted by the adversary is at most $\frac{q_d}{p}$. So, by Lemma 1, we have $\left|\Pr[\mathsf{T}_1] - \Pr[\mathsf{T}_0]\right| \leq \frac{q_d}{p}$.

Game G_2 (Eliminate hash collision of H_2)**.** In this game, the game G_1 is modified to game G_2 by modifying the de-signcrypt oracle to reject a ciphertext $C = (c_1, c_2, c_3, \sigma, \beta, \mu)$ with the receiver's/sender's public key $(\mathsf{PK}_{\mathsf{r}}^*, \mathsf{PK}_{\mathsf{s}}^*)$ such that $(c_3, \sigma, \beta, \mu) = (c_3^*, \sigma^*, \beta^*, \mu^*)$, $(c_1, c_2) \neq (c_1^*, c_2^*)$ and $v = v^*$ where $v = H_2(c_1, c_2, \mathsf{PK}_{\mathsf{r}}^*, \mathsf{PK}_{\mathsf{s}}^*)$. In this case, the de-signcrypt oracle outputs reject and halt. $\mathcal{B}$ then randomly outputs $d \in \{0, 1\}$. The probability of this rejection is negligible with the probability $\epsilon_{\mathtt{cr}}$ as the chance of finding $v = v^*$ with $(c_1, c_2, \mathsf{PK}_{\mathsf{r}}^*, \mathsf{PK}_{\mathsf{s}}^*) \neq (c_1^*, c_2^*, \mathsf{PK}_{\mathsf{r}}^*, \mathsf{PK}_{\mathsf{s}}^*)$ is $\epsilon_{\mathtt{cr}}$. So, by Lemma 1, we have $\left|\Pr[\mathsf{T}_2] - \Pr[\mathsf{T}_1]\right| \leq \epsilon_{\mathtt{cr}}$.

Game G_3 (Eliminate hash collision of H_3)**.** Game G_3 is similar to game G_2 except that the de-signcrypt oracle is modified to reject a ciphertext $C = (c_1, c_2, c_3, \sigma, \beta, \mu)$ with the receiver's/sender's public key $(\mathsf{PK}_{\mathsf{r}}^*, \mathsf{PK}_{\mathsf{s}})$ (PK_{s} can be chosen to be $\mathsf{PK}_{\mathsf{s}}^*$) such that $(\sigma, \beta) \neq (\sigma^*, \beta^*)$, $(c_1, \mu) = (c_1^*, \mu^*)$ and $\eta = \eta^*$ where $\eta = H_3(\sigma, \beta, \mathsf{PK}_{\mathsf{r}}^*, \mathsf{PK}_{\mathsf{s}})$. In this case, the de-signcrypt oracle outputs reject and halt. $\mathcal{B}$ then randomly outputs $d \in \{0, 1\}$. The probability of this rejection is negligible with the probability $\bar{\epsilon}_{\mathtt{cr}}$ as the chance of finding $\eta = \eta^*$ is $\bar{\epsilon}_{\mathtt{cr}}$. So, by Lemma 1, we have $\left|\Pr[\mathsf{T}_3] - \Pr[\mathsf{T}_2]\right| \leq \bar{\epsilon}_{\mathtt{cr}}$.

Game G_4 (Eliminate hash collision of H_1)**.** In this game, the game G_4 is obtained from game G_3 by modifying the de-signcrypt oracle to reject a ciphertext $C = (c_1, c_2, c_3, \sigma, \beta, \mu)$ with a receiver's/sender's public key $(\mathsf{PK}_{\mathsf{r}}, \mathsf{PK}_{\mathsf{s}})$ which is not rejected in game G_i for $i = 1, 2, 3$ before, we consider two types of this rejection rules on ciphertext as below:

(a) $[(\mathsf{PK}_{\mathsf{r}}, \mathsf{PK}_{\mathsf{s}}) = (\mathsf{PK}_{\mathsf{r}}^*, \mathsf{PK}_{\mathsf{s}}^*)] : (\sigma, \beta) = (\sigma^*, \beta^*)$, $\mu \neq \mu^*$ and $\alpha = \alpha^*$ where $\eta = H_3(\sigma, \beta, \mathsf{PK}_{\mathsf{r}}, \mathsf{PK}_{\mathsf{s}})$ and $\alpha = H_1((\mathsf{h}_1^*)^{\eta}(\mathsf{h}_2^*)^{\mu})$, then the de-signcrypt oracle outputs reject and halt. $\mathcal{B}$ then randomly outputs $d \in \{0, 1\}$. The probability of this rejection rule is negligible with the probability $\epsilon_{\mathtt{tcr}}$ as the chance of finding $\alpha = \alpha^*$ for $\mathsf{h}_1^{\eta}\mathsf{h}_2^{\mu} \neq (\mathsf{h}_1^*)^{\eta^*}(\mathsf{h}_2^*)^{\mu^*}$ is $\epsilon_{\mathtt{tcr}}$.

(b) $[\mathsf{PK}_{\mathsf{r}} = (\mathsf{h}_1, \mathsf{h}_2)^3 \neq \mathsf{PK}_{\mathsf{r}}^*$ and PK_{s} may be equal to $\mathsf{PK}_{\mathsf{s}}^*] : \eta \neq \eta^*$, $\mathsf{h}_1^{\eta}\mathsf{h}_2^{\mu} \neq (\mathsf{h}_1^*)^{\eta^*}(\mathsf{h}_2^*)^{\mu^*}$ and $\alpha = \alpha^*$ where $\eta = H_3(\sigma, \beta, \mathsf{PK}_{\mathsf{r}}, \mathsf{PK}_{\mathsf{s}})$ and $\alpha = H_1(\mathsf{h}_1^{\eta}\mathsf{h}_2^{\mu})$,

[3] It is noted that PK_{r} may be chosen to be $((\mathsf{h}_1^*)^a, (\mathsf{h}_2^*)^a)$ by the adversary for some random $a \in \mathcal{Z}_p^*$.

then the de-signcrypt oracle outputs reject and halt. $\mathcal{B}$ then randomly outputs $d \in \{0, 1\}$. The probability of this rejection rule is negligible with the probability $\epsilon_{\mathtt{tcr}}$ as the chance of finding $\alpha = \alpha^*$ for $\mathsf{h}_1^{\eta}\mathsf{h}_2^{\mu} \neq (\mathsf{h}_1^*)^{\eta^*}(\mathsf{h}_2^*)^{\mu^*}$ is $\epsilon_{\mathtt{tcr}}$.

Combining case (a) & (b) and Lemma 1, we have $\big|\Pr[\mathsf{T}_4] - \Pr[\mathsf{T}_3]\big| \leq 2\epsilon_{\mathtt{tcr}}$.

Game G_5 (Modify the challenge ciphertext)**.** In this game, the signcrypt oracle is modified by replacing c_2^* with a random c_2' in $\mathcal{G}_2$. Due to this change of c_2^* to c_2', c_2' is independent of the challenge bit b and no information about b is given to the adversary. Hence $\Pr[\mathsf{T}_5] = 1/2$. Furthermore, as game G_5 and game G_4 are equal unless adversary $\mathcal{A}$ can distinguish $e(g, g)^{x^*y^*w^*}$ from the random element in $\mathcal{G}_2$, therefore, we have $\big|\Pr[\mathsf{T}_5] - \Pr[\mathsf{T}_4]\big| \leq \epsilon_{\mathtt{dbdh}}$.

Combine the results from the above games, we obtain

$$\epsilon_{\mathtt{sc}} \leq \epsilon_{\mathtt{dbdh}} + 2\epsilon_{\mathtt{tcr}} + \epsilon_{\mathtt{cr}} + \bar{\epsilon}_{\mathtt{cr}} + \frac{q_d}{p}$$

and complete the proof of the Theorem. ♢

In the following theorem, we show that the proposed signcryption scheme is secure against strongly existential forgeability in the insider security and the multi-user setting as follows:

Theorem 2. *Suppose there is a forger that breaks the (ϵ, t, q_s)-strongly existential unforgeability of the proposed signcryption scheme* SC *in the insider security and the multi-user setting, then there exists an adversary who can break one of the following four cases:*
(a) *The $(\frac{2\epsilon}{5}, t', q_s)$-strongly existential unforgeability of Boneh-Boyen short signature scheme,*
(b) *The $(\frac{\epsilon}{5}, t')$-*CR *collision resistant hash function H_2,*
(c) *The $(\frac{\epsilon}{5}, t'')$-hard computational Diffie-Hellman* (CDH) *assumption,*
(d) *The $(\frac{\epsilon}{5}, t')$-*TCR *target collision resistant hash function H_1,*
such that $t' = t + q_s\mathsf{E}$ and $t'' = t + q_s\mathsf{E} + \mathsf{exp}$ where E and exp *are the signcrypt cost of* SC *and the computation cost of exponentiation respectively; and q_s is the number of signcrypt queries.*

Proof: Let $\mathcal{F}_{\mathtt{sc}}$ be a forger that breaks the proposed signcryption scheme SC. The forger $\mathcal{F}_{\mathtt{sc}}$ is first given the system parameters SP and the sender's public key $\mathsf{PK}_{\mathsf{s}}'$ from the challenger. In an i-th signcrypt query, the forger $\mathcal{F}_{\mathtt{sc}}$ first chooses a plaintext m_i, a sender's public key PK_{is} (PK_{is} can be chosen to be $\mathsf{PK}_{\mathsf{s}}'$ or different from $\mathsf{PK}_{\mathsf{s}}'$), a receiver's public/private key $(\mathsf{PK}_{ir}, \mathsf{SK}_{ir})$; and sends $(\mathsf{PK}_{ir}, \mathsf{SK}_{ir}, \mathsf{PK}_{is}, m_i)$ to the challenger for signcrypt query. The challenger runs the signcrypt oracle $\mathcal{S}_{\mathtt{sc}}^{\mathcal{O}}$ which returns a ciphertext $C_i = (c_{i1}, c_{i2}, c_{i3}, \sigma_i, \beta_i, \mu_i)$ and the challenger gives C_i to $\mathcal{F}_{\mathtt{sc}}$. After the q_s number of signcrypt queries, the forger $\mathcal{F}_{\mathtt{sc}}$ chooses a receiver's public/private key $(\mathsf{PK}_{\mathsf{r}}', \mathsf{SK}_{\mathsf{r}}')$ and finally outputs a ciphertext $C' = (c_1', c_2', c_3', \sigma', \beta', \mu')$ under the sender's public key $\mathsf{PK}_{\mathsf{s}}'$ such that $C' \neq C_i$ where $C_i = (c_{i1}, c_{i2}, c_{i3}, \sigma_i, \beta_i, \mu_i)$ is the output from the signcrypt queries for those $i \in \{1, \cdots, q_s\}$ with the sender's public key $\mathsf{PK}_{\mathsf{s}}'$. Let $v' =$

$H_2(c'_1, c'_2, \mathsf{PK}'_\mathsf{r}, \mathsf{PK}'_\mathsf{s})$ and $v_i = H_2(c_{i1}, c_{i2}, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$ for $1 \le i \le q_s$. We consider the following five types of forgeries:

Type 1A Forgery: $(c'_1, c'_2, \mathsf{PK}'_\mathsf{r}) \neq (c_{i1}, c_{i2}, \mathsf{PK}_{ir})$, $\mathsf{PK}_{is} = \mathsf{PK}'_\mathsf{s}$ and $v' \neq v_i$ for all $i \in \{1, \cdots, q_s\}$.
Type 1B Forgery: $(c'_1, c'_2, \mathsf{PK}'_\mathsf{r}) \neq (c_{i1}, c_{i2}, \mathsf{PK}_{ir})$, $\mathsf{PK}_{is} = \mathsf{PK}'_\mathsf{s}$ and $v' = v_i$ for some $i \in \{1, \cdots, q_s\}$.
Type 2A Forgery: $(c'_1, c'_2, \mathsf{PK}'_\mathsf{r}, \mathsf{PK}'_\mathsf{s}) = (c_{i1}, c_{i2}, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$ and $(\sigma', \beta') \neq (\sigma_i, \beta_i)$ for some $i \in \{1, \cdots, q_s\}$.
Type 2B Forgery: $(c'_1, c'_2, \sigma', \beta', \mathsf{PK}'_\mathsf{r}, \mathsf{PK}'_\mathsf{s}) = (c_{i1}, c_{i2}, \sigma_i, \beta_i, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$, $\mu' \neq \mu_i$ and $c'_3 \neq c_{i3}$ for some $i \in \{1, \cdots, q_s\}$.
Type 2C Forgery: $(c'_1, c'_2, c'_3, \sigma', \beta', \mathsf{PK}'_\mathsf{r}, \mathsf{PK}'_\mathsf{s}) = (c_{i1}, c_{i2}, c_{i3}, \sigma_i, \beta_i, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$ and $\mu' \neq \mu_i$ for some $i \in \{1, \cdots, q_s\}$.

By using $\mathcal{F}_{\mathsf{sc}}$ as a subroutine, one can construct five algorithms, that are $\mathcal{F}^1_{\mathsf{bb}}$, $\mathcal{B}_1$, $\mathcal{F}^2_{\mathsf{bb}}$, $\mathcal{B}_2$ and $\mathcal{B}_3$ which breaks Boneh-Boyen short signature scheme, the collision resistant hash function, Boneh-Boyen short signature scheme, the computational Diffie-Hellman assumption and the target collision resistant hash function respectively corresponding to Type 1A, Type 1B, Type 2A, Type 2B and Type 2C forgery. First, a simulator randomly guesses type of forgeries for $\mathcal{F}_{\mathsf{sc}}$. The simulator success in guessing type of forgeries is of probability 1/5. For ease of understanding, we list the five forgeries in the following table.

Forgery	Algorithm	Break
Type 1A Forgery	$\mathcal{F}^1_{\mathsf{bb}}$	Boneh-Boyen short signature scheme
Type 1B Forgery	$\mathcal{B}_1$	Collision resistant hash function
Type 2A Forgery	$\mathcal{F}^2_{\mathsf{bb}}$	Boneh-Boyen short signature scheme
Type 2B Forgery	$\mathcal{B}_2$	Computational Diffie-Hellman assumption
Type 2C Forgery	$\mathcal{B}_3$	Target collision resistant hash function

Type 1A Forgery: Assume a forger $\mathcal{F}_{\mathsf{sc}}$ is of Type 1A Forgery that breaks the proposed signcryption scheme SC, then using $\mathcal{F}_{\mathsf{sc}}$ as a subroutine, one constructs an algorithm $\mathcal{F}^1_{\mathsf{bb}}$ that forges a Boneh-Boyen short signature scheme.

Setup: The algorithm $\mathcal{F}^1_{\mathsf{bb}}$ is first given the system parameters $\mathsf{SP} = (g, g_1, g_2, H_1, H_2, H_3)$ and the under attack signer's public key $\mathsf{PK}'_\mathsf{s} = (\mathsf{g}_1, \mathsf{g}_2, \mathsf{Z})$. The algorithm $\mathcal{F}^1_{\mathsf{bb}}$ gives SP and PK'_s to the forger $\mathcal{F}_{\mathsf{sc}}$.

Signcrypt Query: In an i-th signcrypt query on a plaintext m_i, the forger $\mathcal{F}_{\mathsf{sc}}$ first generates a receiver's public/private key $(\mathsf{PK}_{ir}, \mathsf{SK}_{ir})$ where $\mathsf{PK}_{ir} = (\mathsf{h}_{i1}, \mathsf{h}_{i2}, Z_i)$ and $\mathsf{SK}_{ir} = (x_i, u_i)$, chooses a sender's public key PK_{is} (PK_{is} can be chosen to be PK'_s) and sends $(\mathsf{PK}_{ir}, \mathsf{SK}_{ir}, \mathsf{PK}_{is}, m_i)$ to $\mathcal{F}^1_{\mathsf{bb}}$. $\mathcal{F}^1_{\mathsf{bb}}$ first chooses $w_i \in \mathcal{Z}^*_p$ and computes $c_{i1} = g^{w_i}$ and $c_{i2} = Z_i^{w_i} m_i$. $\mathcal{F}^1_{\mathsf{bb}}$ makes a signature query on $(c_{i1}, c_{i2}, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$ to signing oracle $\mathcal{S}^{\mathcal{O}}_{\mathsf{sig}}$ which returns (σ_i, β_i). Upon receipt of (σ_i, β_i), $\mathcal{F}^1_{\mathsf{bb}}$ chooses $\mu_i \in \mathcal{Z}^*_p$, computes $\eta_i = H_3(\sigma_i, \beta_i, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$, $\alpha_i = H_1(\mathsf{h}_{i1}^{\eta_i} \mathsf{h}_{i2}^{\mu_i})$ and $c_{i3} = (g_1^{\alpha_i} g_2)^{w_i}$; and sends $C_i = (c_{i1}, c_{i2}, c_{i3}, \sigma_i, \beta_i, \mu_i)$ to $\mathcal{F}_{\mathsf{sc}}$.

Output: After the q_s number of signcrypt queries, $\mathcal{F}_{\mathsf{sc}}$ finally outputs a ciphertext $C' = (c'_1, c'_2, c'_3, \sigma', \beta', \mu')$ with receiver's public/private key $(\mathsf{PK}'_\mathsf{r}, \mathsf{SK}'_\mathsf{r})$ and the

sender's public key $\mathsf{PK}'_{\mathsf{s}}$ such that $(c'_1, c'_2, \mathsf{PK}'_{\mathsf{r}}) \neq (c_{i1}, c_{i2}, \mathsf{PK}_{i\mathsf{r}})$ and $v' \neq v_i$ for those $i \in \{1, \cdots, q_s\}$ with $\mathsf{PK}_{i\mathsf{s}} = \mathsf{PK}'_{\mathsf{s}}$ where $v' = H_2(c'_1, c'_2, \mathsf{PK}'_{\mathsf{r}}, \mathsf{PK}'_{\mathsf{s}})$ and $v_i = H_2(c_{i1}, c_{i2}, \mathsf{PK}_{i\mathsf{r}}, \mathsf{PK}_{i\mathsf{s}})$. Then, one can easily see that (σ', β') is a valid signature of $(c'_1, c'_2, \mathsf{PK}'_{\mathsf{r}}, \mathsf{PK}'_{\mathsf{s}})$. Hence, $\mathcal{F}^1_{\mathsf{bb}}$ is able to forge a signature for Boneh-Boyen short signature scheme. Furthermore, it is easy to see that the time to forge Boneh-Boyen short signature scheme is $t' = t + q_s\mathsf{E}$ where E is the signcrypt cost of $\mathtt{SC}$.

Type 1B Forgery: Assume a forger $\mathcal{F}_{\mathtt{sc}}$ is of Type 1B Forgery that breaks the proposed signcryption scheme $\mathtt{SC}$, then using $\mathcal{F}_{\mathtt{sc}}$ as a subroutine, one constructs an algorithm $\mathcal{B}_1$ that breaks the collision resistant hash function H_2.

Setup: Given the system parameter $\mathsf{SP} = (g, g_1, g_2, H_1, H_2, H_3)$, $\mathcal{B}_1$ first generates a sender's public/private key $(\mathsf{PK}'_{\mathsf{s}}, \mathsf{SK}'_{\mathsf{s}})$ and gives SP and $\mathsf{PK}'_{\mathsf{s}}$ to $\mathcal{F}_{\mathtt{sc}}$.

Signcrypt Query: In an i-th signcrypt query on a plaintext m_i, the forger $\mathcal{F}_{\mathtt{sc}}$ first generates a receiver's public/private key $(\mathsf{PK}_{i\mathsf{r}}, \mathsf{SK}_{i\mathsf{r}})$ where $\mathsf{PK}_{i\mathsf{r}} = (\mathsf{h}_{i1}, \mathsf{h}_{i2}, Z_i)$ and $\mathsf{SK}_{i\mathsf{r}} = (x_i, u_i)$; and chooses a sender's public key $\mathsf{PK}_{i\mathsf{s}}$ ($\mathsf{PK}_{i\mathsf{s}}$ can be chosen to be $\mathsf{PK}'_{\mathsf{s}}$). $\mathcal{F}_{\mathtt{sc}}$ sends $(\mathsf{PK}_{i\mathsf{r}}, \mathsf{SK}_{i\mathsf{r}}, \mathsf{PK}_{i\mathsf{s}}, m_i)$ to $\mathcal{B}_1$ for signcrypt query. Consider the following two cases :

(a) If $\mathsf{PK}_{i\mathsf{s}} = \mathsf{PK}'_{\mathsf{s}}$, then $\mathcal{B}_1$ outputs $C_i = \mathcal{S}^{\mathcal{O}}_{\mathtt{sc}}(\mathsf{SK}_{i\mathsf{s}}, \mathsf{PK}_{i\mathsf{r}}, m_i)$ and sends $C_i = (c_{i1}, c_{i2}, c_{i3}, \sigma_i, \beta_i, \mu_i)$ to $\mathcal{F}_{\mathtt{sc}}$.

(b) If $\mathsf{PK}_{i\mathsf{s}} \neq \mathsf{PK}'_{\mathsf{s}}$, then $\mathcal{B}_1$ chooses $w_i \in \mathcal{Z}^*_p$ and computes $c_{i1} = g^{w_i}$ and $c_{i2} = Z_i^{w_i} m_i$. $\mathcal{B}_1$ makes a signature query on $(c_{i1}, c_{i2}, \mathsf{PK}_{i\mathsf{r}}, \mathsf{PK}_{i\mathsf{s}})$ to signing oracle $\mathcal{S}^{\mathcal{O}}_{\mathtt{sig}}$ which returns (σ_i, β_i). $\mathcal{B}_1$ then chooses $\mu_i \in \mathcal{Z}^*_p$, computes $\eta_i = H_3(\sigma_i, \beta_i, \mathsf{PK}_{i\mathsf{r}}, \mathsf{PK}_{i\mathsf{s}})$, $\alpha_i = H_1(\mathsf{h}_{i1}^{\eta_i} \mathsf{h}_{i2}^{\mu_i})$ and $c_{i3} = (g_1^{\alpha_i} g_2)^{w_i}$; and sends $C_i = (c_{i1}, c_{i2}, c_{i3}, \sigma_i, \beta_i, \mu_i)$ to $\mathcal{F}_{\mathtt{sc}}$.

Output: After the q_s number of signcrypt queries, the forger $\mathcal{F}_{\mathtt{sc}}$ finally outputs a receiver's public/private key $(\mathsf{PK}'_{\mathsf{r}}, \mathsf{SK}'_{\mathsf{r}})$ and a ciphertext $C' = (c'_1, c'_2, c'_3, \sigma', \beta', \mu')$ under the sender's public key $\mathsf{PK}'_{\mathsf{s}}$ such that $(c'_1, c'_2, \mathsf{PK}'_{\mathsf{r}}) \neq (c_{i1}, c_{i2}, \mathsf{PK}_{i\mathsf{r}})$ and $v' = v_i$ for some $i \in \{1, \cdots, q_s\}$, where $v' = H_2(c'_1, c'_2, \mathsf{PK}'_{\mathsf{r}}, \mathsf{PK}'_{\mathsf{s}})$ and $v_i = H_2(c_{i1}, c_{i2}, \mathsf{PK}_{i\mathsf{r}}, \mathsf{PK}_{i\mathsf{s}})$. Then, $\mathcal{B}_1$ obtains a collision of hash function H_2. Furthermore, it is easy to see that the time to break the collision resistant hash function H_2 is $t' = t + q_s\mathsf{E}$ where E is the signcrypt cost of $\mathtt{SC}$.

Type 2A Forgery: The proof of this type of forgery is similar to Type 1A Forgery, which leads to the strongly existential forgery of Boneh-Boyen short signature scheme. Therefore, we omit here.

Type 2B Forgery: Assume a forger $\mathcal{F}_{\mathtt{sc}}$ is of Type 2B Forgery that breaks the proposed signcryption scheme $\mathtt{SC}$, then using $\mathcal{F}_{\mathtt{sc}}$ as a subroutine, one constructs an algorithm $\mathcal{B}_2$ that breaks the computational Diffie-Hellman problem ($\mathtt{CDH}$).

Setup: $\mathcal{B}_2$ is first given $(g, g^\tau, g^\omega) \in \mathcal{G}^3_1$, then $\mathcal{B}_2$ chooses H_1 be the target collision resistant hash function and H_2 and H_3 be the collision resistant hash functions. $\mathcal{B}_2$ sets $g_1 = g^\tau$, chooses $k, \gamma \in \mathcal{Z}^*_p$ and computes $g_2 = g_1^{-H_1(g^k)} g^\gamma$. $\mathcal{B}_2$ also generates a sender's public key as $\mathsf{PK}'_{\mathsf{s}} = (\mathsf{g}'_1, \mathsf{g}'_2, \mathsf{Z}')$ and the private key as

$\mathsf{SK_s} = (\mathsf{x}', \mathsf{y}')$. $\mathcal{B}_2$ then gives the system parameters $\mathsf{SP} = (g, g_1, g_2, H_1, H_2, H_3)$ and PK'_s to the forger $\mathcal{F}_{\mathtt{sc}}$.

Signcrypt Query: In an i-th signcrypt query on a plaintext m_i, the forger $\mathcal{F}_{\mathtt{sc}}$ first generates a receiver's public/private key $(\mathsf{PK}_{ir}, \mathsf{SK}_{ir})$ where $\mathsf{PK}_{ir} = (\mathsf{h}_{i1}, \mathsf{h}_{i2}, Z_i)$ and $\mathsf{SK}_{ir} = (x_i, u_i)$; and chooses a sender's public key PK_{is} (PK_{is} can be chosen to be PK'_s). $\mathcal{F}_{\mathtt{sc}}$ sends $(\mathsf{PK}_{ir}, \mathsf{SK}_{ir}, \mathsf{PK}_{is}, m_i)$ to $\mathcal{B}_2$ for signcrypt query. Consider the following two cases :

(a) If $\mathsf{PK}_{is} = \mathsf{PK}'_\mathsf{s}$, then $\mathcal{B}_2$ chooses $a_i, \beta_i \in \mathcal{Z}_p^*$ and computes

$$c_{i1} = (g^\omega)^{a_i}, \;\; c_{i2} = e(c_{i1}, g_1)^{x_i} m_i, \;\; v_i = H_2(c_{i1}, c_{i2}, \mathsf{PK}_{ir}, \mathsf{PK}'_\mathsf{s}),$$
$$\sigma_i = g^{1/(v_i + \mathsf{x}' + \mathsf{y}'\beta_i) \bmod p}, \;\; \eta_i = H_3(\sigma_i, \beta_i, \mathsf{PK}_{ir}, \mathsf{PK}'_\mathsf{s}),$$
$$\mu_i = \frac{k - x_i\eta_i}{x_i u_i} \bmod p, \;\; c_{i3} = c_{i1}^{\gamma a_i} = (g_1^{H_1(g^k)}\, g_2)^{\omega a_i}.$$

Then $\mathcal{B}_2$ sends $C_i = (c_{i1}, c_{i2}, c_{i3}, \sigma_i, \beta_i, \mu_i)$ to $\mathcal{F}_{\mathtt{sc}}$.

(b) If $\mathsf{PK}_{is} \neq \mathsf{PK}'_\mathsf{s}$, then $\mathcal{B}_2$ chooses $w_i \in \mathcal{Z}_p^*$ and computes $c_{i1} = g^{w_i}$ and $c_{i2} = Z_i^{w_i} m_i$. $\mathcal{B}_2$ makes a signature query on $(c_{i1}, c_{i2}, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$ to signing oracle $\mathcal{S}^{\mathcal{O}}_{\mathtt{sig}}$ which returns (σ_i, β_i). $\mathcal{B}_2$ then chooses $\mu_i \in \mathcal{Z}_p^*$, computes $\eta_i = H_3(\sigma_i, \beta_i, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$, $\alpha_i = H_1(\mathsf{h}_{i1}^{\eta_i}\mathsf{h}_{i2}^{\mu_i})$ and $c_{i3} = (g_1^{\alpha_i} g_2)^{w_i}$; and sends $C_i = (c_{i1}, c_{i2}, c_{i3}, \sigma_i, \beta_i, \mu_i)$ to $\mathcal{F}_{\mathtt{sc}}$.

Output: After the q_s number of signcrypt queries, the forger $\mathcal{F}_{\mathtt{sc}}$ finally outputs a receiver's public/private key $(\mathsf{PK}'_\mathsf{r}, \mathsf{SK}'_\mathsf{r})$ and a ciphertext $C' = (c'_1, c'_2, c'_3, \sigma', \beta', \mu')$ under the sender's public key PK'_s such that $(c'_1, c'_2, \sigma', \beta', \mathsf{PK}'_\mathsf{r}, \mathsf{PK}'_\mathsf{s}) = (c_{i1}, c_{i2}, \sigma_i, \beta_i, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$, $\mu' \neq \mu_i$ and $c'_3 \neq c_{i3}$ for some $i \in \{1, \cdots, q_s\}$. As $(\sigma', \beta', \mathsf{PK}'_\mathsf{r}, \mathsf{PK}'_\mathsf{s}) = (\sigma_i, \beta_i, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$, then $\eta' = \eta_i$ where $\eta' = H_3(\sigma', \beta', \mathsf{PK}'_\mathsf{r}, \mathsf{PK}'_\mathsf{s})$ and $\eta_i = H_3(\sigma_i, \beta_i, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$. Since $\mu' \neq \mu_i$, this implies that $\mathsf{h}_{i1}^{\eta'}\mathsf{h}_{i2}^{\mu'} \neq \mathsf{h}_{i1}^{\eta_i}\mathsf{h}_{i2}^{\mu_i}$ (It is noted that $\mathsf{PK}'_\mathsf{r} = \mathsf{PK}_{ir} = (\mathsf{h}_{i1}, \mathsf{h}_{i2})$). Furthermore, $c'_1 = c_{i1} = g^{\omega a_i}$, then $e(c'_1, g_1^{\alpha'} g_2) = e(c'_3, g)$ where $\alpha' = H_1(\mathsf{h}_{i1}^{\eta'}\mathsf{h}_{i2}^{\mu'})$. We obtain $e((g_1^{\alpha'} g_2)^{-\omega a_i} c'_3, g) = 1$. By the non-degenerate of bilinear map e, we have $c'_3 = (g_1^{\alpha'} g_2)^{\omega a_i}$. As $c'_3 \neq c_{i3}$, this implies that $\alpha' \neq \alpha_i$ where $\alpha_i = H_1(\mathsf{h}_{i1}^{\eta_i}\mathsf{h}_{i2}^{\mu_i})$. Then from $c'_3 = (g_1^{\alpha'} g_2)^{\omega a_i}$ and $c_{i3} = (g_1^{\alpha_i} g_2)^{\omega a_i}$, $\mathcal{B}_2$ can compute

$$g^{\tau\omega} = (c'_3 c_{i3}^{-1})^{\frac{1}{a_i(\alpha' - \alpha_i)}}.$$

Hence, $\mathcal{B}_2$ can solve the CDH problem. Furthermore, it is easy to see that the time to break the CDH problem is $t' = t + q_s\mathsf{E} + \mathsf{exp}$ where E and exp are the signcrypt cost of SC and the computation cost of exponentiation respectively.

Type 2C Forgery: Assume a forger $\mathcal{F}_{\mathtt{sc}}$ is of Type 2C Forgery that breaks the proposed signcryption scheme SC, then using $\mathcal{F}_{\mathtt{sc}}$ as a subroutine, one constructs an algorithm $\mathcal{B}_3$ that breaks the target collision resistant hash function H_1.

Setup: Given the system parameter $\mathsf{SP} = (g, g_1, g_2, H_1, H_2, H_3)$, $\mathcal{B}_3$ first generates a sender's public/private key $(\mathsf{PK}'_\mathsf{s}, \mathsf{SK}'_\mathsf{s})$ and gives SP and PK'_s to $\mathcal{F}_{\mathtt{sc}}$.

Signcrypt Query: In an i-th signcrypt query on a plaintext m_i, the forger $\mathcal{F}_{\mathsf{sc}}$ first generates a receiver's public/private key $(\mathsf{PK}_{ir}, \mathsf{SK}_{ir})$ where $\mathsf{PK}_{ir} = (\mathsf{h}_{i1}, \mathsf{h}_{i2}, Z_i)$ and $\mathsf{SK}_{ir} = (x_i, u_i)$; and chooses a sender's public key PK_{is} (PK_{is} can be chosen to be $\mathsf{PK}'_{\mathsf{s}}$). $\mathcal{F}_{\mathsf{sc}}$ sends $(\mathsf{PK}_{ir}, \mathsf{SK}_{ir}, \mathsf{PK}_{is}, m_i)$ to $\mathcal{B}_3$ for signcrypt query. Consider the following two cases :

(a) If $\mathsf{PK}_{is} = \mathsf{PK}'_{\mathsf{s}}$, then $\mathcal{B}_3$ outputs $C_i = \mathcal{S}^{\mathcal{O}}_{\mathsf{sc}}(\mathsf{SK}_{is}, \mathsf{PK}_{ir}, m_i)$ and sends $C_i = (c_{i1}, c_{i2}, c_{i3}, \sigma_i, \beta_i, \mu_i)$ to $\mathcal{F}_{\mathsf{sc}}$.

(b) If $\mathsf{PK}_{is} \neq \mathsf{PK}'_{\mathsf{s}}$, then $\mathcal{B}_3$ chooses $w_i \in \mathcal{Z}^*_p$ and computes $c_{i1} = g^{w_i}$ and $c_{i2} = Z_i^{w_i} m_i$. $\mathcal{B}_3$ makes a signature query on $(c_{i1}, c_{i2}, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$ to signing oracle $\mathcal{S}^{\mathcal{O}}_{\mathsf{sig}}$ which returns (σ_i, β_i). $\mathcal{B}_3$ then chooses $\mu_i \in \mathcal{Z}^*_p$, computes $\eta_i = H_3(\sigma_i, \beta_i, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$, $\alpha_i = H_1(\mathsf{h}_{i1}^{\eta_i} \mathsf{h}_{i2}^{\mu_i})$ and $c_{i3} = (g_1^{\alpha_i}\ g_2)^{w_i}$; and sends $C_i = (c_{i1}, c_{i2}, c_{i3}, \sigma_i, \beta_i, \mu_i)$ to $\mathcal{F}_{\mathsf{sc}}$.

Output: After the q_s number of signcrypt queries, the forger $\mathcal{F}_{\mathsf{sc}}$ finally outputs a receiver's public/private key $(\mathsf{PK}'_{\mathsf{r}}, \mathsf{SK}'_{\mathsf{r}})$ and a ciphertext $C' = (c'_1, c'_2, c'_3, \sigma', \beta', \mu')$ under the sender's public key $\mathsf{PK}'_{\mathsf{s}}$ such that $(c'_1, c'_2, c'_3, \sigma', \beta', \mathsf{PK}'_{\mathsf{r}}, \mathsf{PK}'_{\mathsf{s}}) = (c_{i1}, c_{i2}, c_{i3}, \sigma_i, \beta_i, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$ and $\mu' \neq \mu_i$ for some $i \in \{1, \cdots, q_s\}$. As $(\sigma', \beta', \mathsf{PK}'_{\mathsf{r}}, \mathsf{PK}'_{\mathsf{s}}) = (\sigma_i, \beta_i, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$, then $\eta' = \eta_i$ where $\eta' = H_3(\sigma', \beta', \mathsf{PK}'_{\mathsf{r}}, \mathsf{PK}'_{\mathsf{s}})$ and $\eta_i = H_3(\sigma_i, \beta_i, \mathsf{PK}_{ir}, \mathsf{PK}_{is})$. Furthermore, $c'_1 = c_{i1}$ and $c'_3 = c_{i3}$, this implies that $\alpha' = \alpha_i$ where $\alpha_i = H_1(\mathsf{h}_{i1}^{\eta_i} \mathsf{h}_{i2}^{\mu_i})$ and $\alpha' = H_1(\mathsf{h}_{i1}^{\eta'} \mathsf{h}_{i2}^{\mu'})$ (It is noted that $\mathsf{PK}'_{\mathsf{r}} = \mathsf{PK}_{ir}$). Since $\mu' \neq \mu_i$, this implies that $\mathsf{h}_{i2}^{\mu'} \neq \mathsf{h}_{i2}^{\mu_i}$. Hence, $\mathcal{B}_3$ obtains a target collision of hash function H_1. Furthermore, it is easy to see that the time to break the target collision resistant hash function H_1 is $t' = t + q_s\mathsf{E}$ where E is the signcrypt cost of SC. ♢

5 Conclusion

In this paper, we presented the insider-secure signcryption scheme in the multi-user setting based on the decisional bilinear Diffie-Hellman assumption, secure Boneh-Boyen short signature scheme, the collision resistant hash function and the target collision resistant hash function. The proposed signcryption scheme is provably insider-secure against adaptive chosen ciphertext attacks and secure against strongly existential forgeability without random oracles in the multi-user setting. Furthermore, the ciphertext of the proposed signcryption scheme is publicly verifiable and the proposed signcryption scheme also allows the trusted third party to verify the correct plaintext/ciphertext pair if there is a dispute between the sender and the receiver.

Acknowledgments

The author wishes to thank the anonymous reviewers for their comments and invaluable suggestions for improving this paper.

References

1. An, J.H., Dodis, Y., Rabin, T.: On the security of joint signature and encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 83–107. Springer, Heidelberg (2002)
2. Baek, J., Steinfeld, R., Zheng, Y.: Formal proofs for the security of signcryption. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 80–98. Springer, Heidelberg (2002)
3. Baek, J., Steinfeld, R., Zheng, Y.: Formal proofs for the security of signcryption. J. Cryptology 20(2), 203–235 (2007)
4. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004)
5. Boneh, D., Boyen, X.: Efficient selective-ID secure identity based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004)
6. Boneh, D., Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. SIAM J. Comput. 36(5), 1301–1328 (2006)
7. Dent, A.W.: Hybrid signcryption schemes with insider security. In: Boyd, C., González Nieto, J.M. (eds.) ACISP 2005. LNCS, vol. 3574, pp. 253–266. Springer, Heidelberg (2005)
8. Dent, A.W.: Hybrid signcryption schemes with outsider security. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 203–217. Springer, Heidelberg (2005)
9. Dodis, Y., An, J.-H.: Concealment and its applications to authenticated encryption. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 312–329. Springer, Heidelberg (2003)
10. Dodis, Y., Freedman, M.J., Jarecki, S., Walfish, S.: A versatile padding schemes for joint signature and encryption. In: The 11th ACM Conference on Computer and Communications Security, pp. 344–353. ACM press, New York (2004)
11. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Computing 17(2), 281–308 (1988)
12. Jeong, I.-R., Jeong, H.-J., Rhee, H.-S., Jong, I.-L.: Provably secure encrypt-then-sign composition in hybrid signcryption. In: Lee, P.J., Lim, C.H. (eds.) ICISC 2002. LNCS, vol. 2587, pp. 16–34. Springer, Heidelberg (2003)
13. Libert, B., Quisquater, J.J.: Efficient signcryption with key privacy from gap Diffie-Hellman groups. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 187–200. Springer, Heidelberg (2004)
14. Libert, B., Quisquater, J.J.: Improved signcryption from q-Diffie-Hellman problems. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 220–234. Springer, Heidelberg (2005)
15. Ma, C.: Efficient short signcryption scheme with public verifiability. In: Lipmaa, H., Yung, M., Lin, D. (eds.) Inscrypt 2006. LNCS, vol. 4318, pp. 118–129. Springer, Heidelberg (2006)
16. Malone-Lee, J.: Signcryption with non-interactive non-repudiation. Designs, Codes and Cryptography 37(1), 81–109 (2005)
17. Malone-Lee, J., Mao, W.: Two birds one stone: Signcryption using RSA. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 211–225. Springer, Heidelberg (2003)
18. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs (manuscript, 2004), `http://eprint.iacr.org/2004/332`

19. Tan, C.H.: On the security of signcryption scheme with key privacy. IEICE Trans. on Fundamentals E88-A(4), 1093–1095 (2005)
20. Tan, C.H.: Security analysis of signcryption scheme from q-Diffie-Hellman problems. IEICE Trans. on Fundamentals E89-A(1), 206–208 (2006)
21. Tan, C.H.: Analysis of improved signcryption scheme with key privacy. Information Processing Letters 99(4), 135–138 (2006)
22. Tan, C.H.: Insider-secure hybrid signcryption scheme without random oracles. In: The 2nd International Conference on Availability, Reliability and Security 2007 (ARES 2007), pp. 1148–1154. The IEEE Computer Press, Los Alamitos (2007)
23. Tan, C.H.: Forgery of provable secure short signcryption scheme. IEICE Trans. on Fundamentals E90-A(9), 1879–1880 (2007)
24. Yang, G., Wong, D.S., Deng, X.: Analysis and improvement of a signcryption scheme with key privacy. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 218–232. Springer, Heidelberg (2005)
25. Zheng, Y.: Digital signcryption or how to achieve cost(signature & encryption) $\ll$ cost(signature) + cost(encryption). In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 165–179. Springer, Heidelberg (1997)

Simple and Efficient Group Signature Scheme Assuming Tamperproof Devices

Takuya Yoshida and Koji Okada

Toshiba Solutions Corporation
3–22, Katamachi, Fuchu-Shi, Tokyo 183-8512, Japan
{Takuya.Yoshida,Koji.Okada}@toshiba-sol.co.jp

Abstract. We propose a simple and efficient group signature scheme assuming tamperproof devices, which is based only on the DDH assumption and the Random Oracle. Compared to the state-of-the-art group signature schemes of [9, 13, 15], although the proposed scheme does not have coalition resistance and exculpability thus requires tamperproof devices, it is about twice faster in signature generation in exchange. It is also efficient in signature size and signing key size. Even though the application of the proposed scheme is not extensive, it can be much more suitable than previous schemes for platforms with tamperproofness and limited resources such as smart cards or mobile phones. Our approach is important in designing group signature schemes (with some restrictions) in the sense that it has shown a concrete construction that one can build a better scheme by carefully considering redundant properties for required security level of given platform. The proposed scheme also supports Verifier Local Revocation, Self-Traceability and weaker version of Non-Frameability.

1 Introduction

A group signature is an anonymous digital signature scheme introduced by Chaum and van Heyst [11] in which only legitimate members of a group can anonymously sign messages on behalf of the group. Anyone can verify the signature but cannot get any information about which member made it. On the other hand, only the trusted authority called *Group Manager* can reveal the identity of the signer if necessary. Group signature schemes have constantly attracted wide attention as a useful and powerful tool to realize privacy preserving and personal information protection and a number of schemes have been proposed in the literature.

Previous Works. From view point of efficiency, the first breakthrough was the scheme proposed by Camenisch and Stadler [8] and a number of studies follow. The earlier schemes are inefficient because the signature size or computational cost depends on the size of the group. The scheme in [8] realizes constant signature size and computational cost for the first time by introducing the notion of *membership certificate*. A membership certificate is the signature by Group

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 83–99, 2008.

Manager on the member secret key and is used as the proof that its owner is one of the legitimate member of the group. The group signature of a member consists of a ciphertext and non-interactive zero knowledge proofs or arguments (NIZK). The ciphertext is encryption of his membership certificate under the public key of the Group Manager and the NIZKs are those of a member secret key, the membership certificate on it and the ciphertext is well-formed. No information on the identity of the signer is obtained because of the confidentiality and zero knowledge property of underlying encryption scheme and NIZK. On the other hand, the validity of the signature can be checked by verifying the correctness of the NIZK and the Group Manager can reveal the identity of the signer by decrypt the membership certificate. A number of schemes have been proposed afterwards and all the practical schemes are membership certificate based. Among them remarkable ones are the following. The first practical scheme was proposed by Ateniese et al. [2] and state-of-the-art schemes are those by Camenisch et al. [9], Furukawa et al. [15] and Delerablée et al. [13]. These are all membership certificate based schemes.

Another research interest on group signatures is additional functions.

Revocation is probably the most difficult but important one because unlike ordinary signature schemes the authority needs to revoke users anonymously. There are two ways to realize revocation. First strategy is to force members to renew their own member secret keys or membership certificates from public revocation information using such tools as dynamic accumulators. Although it achieves high security because revoked members can no longer generate valid signatures, it is costly to force key renewal process to all members each time revocation occurs. The other strategy does not require members any computation for revocation. The first two efficient schemes were independently proposed by Boneh et al. [7] and Camenisch et al. [9] as verifier local revocation and full revocation respectively. We use the term *Verifier Local Revocation(VLR)* in this paper. Even though there are some issues, called forward security or backward unlinkability treated in [1, 3, 22, 26], relating the anonymity of the past signature of revoked user, it is computationally practical. Forward security or backward unlinkability is out of the scope of this paper.

Separability of Management Functions is important to realize higher security level of Group Manager and flexible group management. There are some functions for Group Manager such as generating group public/secret key pairs, issuing member secret keys or membership certificates, opening the identity of signatures and revocation. In the earlier schemes, a group secret key enables to execute all these functions and the risk of the key management is considerably high. The problem is considered in [5, 14, 19] and schemes in which the secret information and the authority is separated by functionalities are proposed. We call the separated authority *Issuer*, *Opener* and *Revocation Manager*, respectively.

If Group Manager or Opener is not fully trusted, it is required that anyone can verify that a signer identification procedure is properly executed. *Non-Frameability* is the security requirement introduced in [5, 20] to deal with it. The authority is required to output a proof called *Publicly Verifiable Proof of*

Opening(PVPO) along with the identity of the signer. Anyone can confirm the opened identity by verifying the proof.

Another type of traceability called *Self-Traceability* is introduced in [20]. It enables a user to claim, if he wishes, a certain signature is his own without compromising any privacy of other signatures.

Yet another aspect on research of group signature schemes is intractability assumptions on which the security of group signature schemes are based. Previously known efficient schemes are either RSA based or pairing based. The schemes in [2, 9] are RSA based and their security relies on the Strong RSA assumption. Although NIZK over RSA modulus is inefficient in both computational cost and length, RSA has been extensively investigated for decades and the Strong RSA assumption is widely accepted as a reliable assumption. Pairing based schemes, which use bilinear groups and mappings, are increasing after the first scheme [6]. [7, 13, 15] are efficient and popular among them. Bilinear group is an effective tool to realize short signature length and revocation. On the other hand, pairing related assumptions such as SDH, DLDH, DBDH or LRSW are new, the computational cost of bilinear mappings is still expensive and the range of bilinear mappings is relatively large, which is 1020bit on the contrary to the 171bit domain for example.

Our Results. Our main contribution is the construction of a very simple and efficient group signature scheme assuming tamperproof devices. This is achieved by carefully considering redundant properties for required security level of given platform and by omitting two security requirements which are redundant in certain cases. We point out that there are certain applications in which some of the security requirements discussed in the literature as indispensable are too strict or redundant and efficiency is the most important requirement.

One property the proposed scheme does not have is *coalition resistance*. In other words, the proposed scheme becomes vulnerable when a malicious coalition of members misuse their member secret keys. However, those kinds of illegal act can be eliminated by securely storing a member secret key so that even the owner of the key cannot know the value. Many technologies to achieve it are known and widely used as tamperproof hardware or software. One good application of the proposed scheme is implementation on smart cards or mobile phones. Smart cards and mobile phones have tamperproofness but their resources for computation, storage and communication bandwidth are quite limited. Therefore, a scheme with better efficiency will be desirable even if it does not have coalition resistance.

The other property we neglect is *exculpability* against false abuse by Issuer. However, the proposed scheme is sufficiently practical because it is secure against fraudulent acts by Revocation Manager and Opener, thus the trust level requirement to authorities are not too high.

Due to these relaxed security requirements, the proposed scheme is remarkably simple and efficient. The proposed scheme is about twice faster than the state-of-the-art schemes of [9, 13, 15]. It is also efficient in signature size and member secret key size. These are owing to the assumption on which the proposed

scheme depends. The proposed scheme is the first practical one based only on the DDH assumption and the Random Oracle. Therefore, it requires neither NIZK in hidden order group nor bilinear mappings which oblige costly computations and long data length. It also implies efficient implementation on elliptic curves, which result in shorter signature length. The proposed scheme is also quite simple in the sense that it is both conceptually comprehensible and easy to implement. Long exponent operations (e.g. 2000bit exponent in 1024bit modulus) or bilinear mappings, which are used in previous schemes, may have to be specially implemented. On the other hand, the proposed scheme is composed of standard multi-byte modulus operations or ordinary elliptic curve operations and a hash function.

Our construction shows one concrete example that one can build a better scheme by carefully considering redundant properties for required security level of given platform. Our approach is important in the sense that it has shown an alternative direction in designing new group signature schemes (with some restrictions).

The proposed scheme also supports Verifier Local Revocation(VLR), separability of management functions, Self-Traceability and weaker version of Non-Frameability.

2 Definitions

In this section, we define the model and the functions of our group signature scheme assuming tamperproof devices $\mathcal{GS}$. The security definitions are also given, which are based on the definitions of [4, 5] by Bellare et al.

2.1 Model and Functions

In order to identify the signer, we use what we call *tracing information*, which is uniquely bound to each member of the group, instead of membership certificates.

$\mathcal{GS}$ consists of the following seven polynomial time algorithms and an interactive protocol.

Group Key Set Generation: GKg
GKg is a probabilistic polynomial time algorithm invoked by Group Manager to obtain the group key set that on input a security parameter k generates and outputs a group public key gpk, the corresponding secret issuing key for Issuer ik and the secret opening key for Opener ok.

Member Secret Key Generation: MKg
Let n be the maximum number of group members. Without loss of generality, we assume the set of member identities is $\{1, \ldots, n\}$. MKg is a probabilistic polynomial time algorithm invoked by Issuer (or Group Manager) to obtain a member secret key and related data that on input a public key gpk, the corresponding issuing key ik and member's identity $i \in \{1, \ldots, n\}$ generates and outputs the member secret key gsk_i, the corresponding tracing information T_i and the revocation token grt_i.

Signature Generation: GSig
GSig is a probabilistic polynomial time algorithm invoked by a member of a group to generate a signature that on input a group public key gpk, a member secret key gsk_i, the corresponding tracing information T_i and a message m computes and outputs the signature σ.

Signature Verification: GVf
GVf is a deterministic polynomial time algorithm invoked by a verifier to check the validity of a signature that on input a group public key gpk, a message m, a signature σ and a revocation list $\mathcal{RL}$ outputs either valid or invalid.

Claiming Generation of Signature: Claim
Claim is an interactive two party protocol between a signer and a verifier to realize Self-Traceability by which the signer of a signature can convince the verifier that the signature was certainly generated by him. A group public key gpk, a signature σ and a message m are given to both the signer and the verifier as common inputs. A member secret key gsk_i is a private input only to the signer. At the end of the protocol, the verifier either outputs true or false.

Signer Identification: Open
Open is a probabilistic polynomial time algorithm invoked by Opener (or Group Manager) to identify the signer from given signature that on input a group public key gpk, an opening key ok, a message m and a signature σ outputs either the identity of the signer i and its Publicly Verifiable Proof of Opening(PVPO) τ if the signature is valid, or invalid otherwise.

Checking Identified Signer: Judge
Judge is a deterministic polynomial time algorithm invoked by a verifier to check if the signer identification was properly carried out that on input a group public key gpk, a signature σ, the identity of a member i and a PVPO τ outputs either true or false.

Membership Revocation: Revoke
Revoke is a deterministic polynomial time algorithm invoked by Revocation Manager (or Group Manager) to revoke a set of users that on input the set of all revocation tokens **grt** and a set of revoked users' identities RU outputs a revocation list $\mathcal{RL}$. $\mathcal{RL}$ is a list of revocation tokens of revoked users, that is, $\mathcal{RL} = \{grt_i \mid i \in RU\}$.

2.2 Security

Many security requirements had been discussed in earlier studies and Bellare et al. [4, 5] summarized into fewer requirements. Their requirements captures many security requirements which had been separately discussed, including coalition resistance and exculpability. However, their requirements are too strict in practice and most of the recent efficient schemes use weaker variants. Our definitions are weaker than those conventional ones in the sense that they do not capture coalition resistance and exculpability. Notice that only one oracle query to

MKg is allowed and ik is never given to the adversaries in the games described below, which are different from the conventional ones.

We require the following four properties, Correctness, Anonymity, Traceability and Weak Non-Frameability.

Correctness. $\mathcal{GS}$ is said to have *Correctness* if for σ=GSig(gsk_i, m) GVf(gpk, m, σ) = valid, Open$(ok, m,$ GSig$(gsk_i, m)) = (i, \tau)$ and Judge(gpk, i, σ, τ) = true holds. Namely, a signature correctly generated by a legitimate member is accepted as valid, the signer of the signature is correctly identified from the signature and the identification can be publicly verified.

Anonymity. Roughly speaking, Anonymity is the requirement that no information on the identity of a signature leaks.

Consider the following game played by the adversary $\mathcal{A}^{\text{anon}}$.

Setup: Execute GKg(1^k) then execute MKg for each user $i \in \{1, \ldots, n\}$ to obtain gpk, ik, ok, the set **gsk** of all gsk_i, the set **T** of all T_i and the set **grt** of all grt_i. Then provide $\mathcal{A}^{\text{anon}}$ with gpk and **T**.

Queries: $\mathcal{A}^{\text{anon}}$ can make arbitrary query calls to GSig, Open, Revoke and Claim. $\mathcal{A}^{\text{anon}}$ is also allowed to make *one* query call to MKg to obtain gsk_u for arbitrarily chosen member u.

Challenge: $\mathcal{A}^{\text{anon}}$ outputs a message $\hat{m}$ and two identities $\hat{i}_0$ and $\hat{i}_1$. When $\mathcal{A}^{\text{anon}}$ has made a query to MKg on u, $\hat{i}_0$ and $\hat{i}_1$ must satisfy neither $u \neq \hat{i}_0$ nor $u \neq \hat{i}_1$. Challenger picks $b \leftarrow \{0, 1\}$ at random, computes $\hat{\sigma} \leftarrow$ GSig$(gpk, gsk_{\hat{i}_b}, \hat{m})$ and gives $\hat{\sigma}$ to $\mathcal{A}^{\text{anon}}$.

Restricted Queries: $\mathcal{A}^{\text{anon}}$ can make arbitrary query calls to GSig, Open, Revoke and Claim with the restriction that (1) $\mathcal{A}^{\text{anon}}$ cannot make an oracle query to Revoke on $\hat{i}_0$ nor $\hat{i}_1$ and (2) $\mathcal{A}^{\text{anon}}$ cannot make an oracle query to Open nor Claim on $\hat{\sigma}$. $\mathcal{A}^{\text{anon}}$ is also allowed to make *one* query call to MKg if it has not been made in the preceding Queries phase. In this case, $\mathcal{A}^{\text{anon}}$ must not ask $gsk_{\hat{i}_0}$ or $gsk_{\hat{i}_1}$.

Output: $\mathcal{A}^{\text{anon}}$ outputs b'.

We say that $\mathcal{GS}$ has *Anonymity* if the probability $|\Pr[b = b'] - 1/2|$ is negligible for any polynomial time adversary $\mathcal{A}^{\text{anon}}$.

Traceability. Roughly speaking, Traceability is the requirement that no adversary can forge a signature from which the adversary is not traced.

Consider the following game played by the adversary $\mathcal{A}^{\text{trace}}$.

Setup: Execute GKg(1^k) then execute MKg for each user $i \in \{1, \ldots, n\}$ to obtain gpk, ik, ok, the set **gsk** of all gsk_i, the set **T** of all T_i and the set **grt** of all grt_i. Then provide $\mathcal{A}^{\text{trace}}$ with gpk, ok, **grt** and **T**.

Queries: $\mathcal{A}^{\text{trace}}$ can make arbitrary query calls to GSig and Claim. $\mathcal{A}^{\text{trace}}$ is also allowed to make *one* query call to MKg to obtain gsk_u for arbitrarily chosen member u.

Response: $\mathcal{A}^{\text{trace}}$ outputs a message $\hat{m}$ and a signature $\hat{\sigma}$.

The adversary $\mathcal{A}^{\text{trace}}$ wins if $\mathsf{GVf}(gpk, \hat{m}, \hat{\sigma}) = \mathsf{valid}$, $i \neq u$ where $\mathsf{Open}(ok, \hat{m}, \hat{\sigma}) = (i, \tau)$ and no signing query is invoked on input i and $\hat{m}$. We say that $\mathcal{GS}$ has *Traceability* if there exists no polynomial time adversary $\mathcal{A}^{\text{trace}}$ who can win the above game with non-negligible probability.

Weak Non-Frameability. Roughly speaking, Weak Non-Frameability is the requirement that no adversary can forge a signature from which an honest user is traced. Our definition is weaker than Non-Frameability defined in [5] because it does not capture exculpability against false abuse by Issuer whereas the original definition does. Our definition captures, however, the exculpability against fraudulent Opener thus can make the security requirement to the Opener lower.

Consider the following game played by the adversary $\mathcal{A}^{\text{wnf}}$.

Setup: Execute $\mathsf{GKg}(1^k)$ then execute MKg for each user $i \in \{1, \ldots, n\}$ to obtain gpk, ik, ok, the set **gsk** of all gsk_i, the set **T** of all T_i and the set **grt** of all grt_i. Then provide $\mathcal{A}^{\text{wnf}}$ with gpk, ok, **T** and **grt**.

Queries: $\mathcal{A}^{\text{wnf}}$ can make arbitrary query call to GSig and Claim. $\mathcal{A}^{\text{wnf}}$ is also allowed to make *one* query call to MKg to obtain gsk_u for arbitrarily chosen member u.

Response: $\mathcal{A}^{\text{wnf}}$ outputs a message $\hat{m}$, a signature $\hat{\sigma}$, an identity of a member $\hat{i}$ and a PVPO $\hat{\tau}$.

The adversary $\mathcal{A}^{\text{wnf}}$ wins if $\mathsf{GVf}(gpk, \hat{m}, \hat{\sigma}) = \mathsf{valid}$, $\hat{i} \neq u$, $\mathsf{Judge}(gpk, \hat{i}, \hat{\sigma}, \hat{\tau}) = \mathsf{true}$ and no signing query is invoked on input $\hat{i}$ and $\hat{m}$. We say that a group signature scheme has *Weak Non-Frameability* if no polynomial time adversary $\mathcal{A}^{\text{wnf}}$ can win the above game with non-negligible probability.

3 Preliminaries

In this section, we present the intractability assumption and the building blocks we are using to construct the proposed scheme.

3.1 The DDH Assumption

The Decisional Diffie-Hellman (DDH) assumption is an intractabity assumption that no polynomial time algorithm can distinguish the following two distributions

- the distribution $\mathcal{R}$ of random quadruples $(G_1, G_2, U_1, U_2) \in \mathbb{G}^4$.
- the distribution $\mathcal{D}$ of quadruples $(G_1, G_2, U_1, U_2) \in \mathbb{G}^4$, where $G_1, G_2 \in \mathbb{G}$ are random and $U_1 = {G_1}^r$, $U_2 = {G_2}^r$ for random $r \in \mathbb{Z}_q$.

where $\mathbb{G}$ is a multiplicative group of prime order q.

The security of the proposed scheme is reduced to the DDH assumption.

3.2 Relaxed Discrete Logarithm (RDL)

Let $\mathbb{G}$ be a multiplicative group of prime order q and $H, G_1, G_2, \ldots, G_l$ be elements of $\mathbb{G}$. An RDL of H with respect to the bases $(G_1, G_2, \ldots, G_l)$ is a tuple $(e_1, e_2, \ldots, e_l)$ satisfying the equation $H = {G_1}^{e_1} {G_2}^{e_2} \cdots {G_l}^{e_l}$.

Relaxed Discrete Logarithm (RDL) is an extension of discrete logarithm (DL). Although it is often referred to as *representation*, we call it RDL to indicate the relation to DL. It appears in [10] and has been used as a cryptographic tools. An NIZK of RDL is presented in [8]. We omit the details here.

We use RDL as a member secret key and its NIZK is contained in a signature.

3.3 Cramer-Shoup Cryptosystem

Cramer-Shoup cryptosystem is a practical CCA secure cryptosystem based on the DDH assumption. The description is as follows.

Key Pair Generation
Given a security parameter k, a public/secret key pair is generated as follows.

- Generate a $k$bit prime q, a multiplicative group $\mathbb{G}$ of order q and a generator G_1 of the group $\mathbb{G}$.
- Choose $a \in \mathbb{Z}_q$ at random.
- Compute $G_2 = {G_1}^a$.
- Choose $x_1, x_2, y_1, y_2, z \in \mathbb{Z}_q$ at random.
- Compute $C = {G_1}^{x_1} {G_2}^{x_2}$, $D = {G_1}^{y_1} {G_2}^{y_2}$ and $H = {G_1}^z$.
- Choose $\mathcal{H}$ from a universal one-way hash family.
- The public key is $pk = (G_1, G_2, C, D, H, \mathcal{H})$ and the secret key is $sk = (x_1, x_2, y_1, y_2, z)$.

Encryption
Given a public key $pk = (G_1, G_2, C, D, H, \mathcal{H})$ and a message $M \in \mathbb{G}$, the message is encrypted as follows.

- Choose $r \in \mathbb{Z}_q$ at random.
- Compute $U_1 = {G_1}^r$, $U_2 = {G_2}^r$ and $E = H^r M$.
- Compute $\alpha = \mathcal{H}(U_1, U_2, E)$.
- Compute $V = C^r D^{r\alpha}$.
- The ciphertext is a tuple (U_1, U_2, E, V).

Decryption
Given a ciphertext (U_1, U_2, E, V), a public key $pk = (G_1, G_2, C, D, H, \mathcal{H})$, and the corresponding secret key $sk = (x_1, x_2, y_1, y_2, z)$, the message is retrieved as follows.

- Compute $\alpha = \mathcal{H}(U_1, U_2, E)$.
- Check if ${U_1}^{x_1 + y_1 \alpha} {U_2}^{x_2 + y_2 \alpha} = V$. If it does not hold, output invalid and abort.
- The message is obtained as $M = E / {U_1}^z$.

The proposed scheme uses Cramer-Shoup cryptosystem to encrypt the signer's tracing information.

4 The Proposed Scheme

In this section, we describe the details of the proposed scheme and analyze its security and efficiency.

4.1 Our Approach

The target of this paper is to propose an efficient group signature scheme. Our approach is twofold. One is to make the scheme discrete logarithm based and the other is to revisit the security requirements.

As mentioned in section 1, there still is room to refine speed and size in state-of-the-art efficient schemes, both RSA based schemes [9] and pairing based schemes [13, 15]. On the other hand, discrete logarithm based schemes are expected to achieve both speed and size because of efficient NIZKs with low computational cost and small data size and implementation over elliptic curves. We use an RDL tuple as a member secret key. An NIZK of the tuple is contained in signatures as a proof of membership. It is suitable for group oriented systems because there are many tuples to given bases and a target on the contrary to discrete logarithm, which has only one value to a given base and a target. On the other hand, a convex combination of RDLs is also another RDL so that the proposed scheme does not have coalition resistance. The scheme proposed by Kiayias et al. in [18] uses RDL. However, the scheme is inefficient because it uses RDL itself to identify the signer. In order to identify the signer, the proposed scheme uses not RDL itself but what we call tracing information which is uniquely determined by and bound to the RDL.

The other approach is to revisit the security requirements. We omit coalition resistance and exculpability, which can be redundant in certain cases as discussed in section 1.

As a result, the proposed scheme realizes simple structure and efficiency in exchange for the two security requirements.

4.2 Description

Group Key Set Generation: GKg
Given a security parameter k, GKg runs as follows.

- Generate a $k$bit prime q, a multiplicative group $\mathbb{G}$ of order q and a generator G_1 of the group $\mathbb{G}$.
- Choose $a, b \in \mathbb{Z}_q$ at random.
- Compute $G_2 = {G_1}^a$ and $F = {G_1}^b$.
- Choose $x_1, x_2, y_1, y_2, z \in \mathbb{Z}_q$ at random.
- Compute $C = {G_1}^{x_1}{G_2}^{x_2}$, $D = {G_1}^{y_1}{G_2}^{y_2}$ and $H = {G_1}^z$.
- Choose $\mathcal{H}_1$, $\mathcal{H}_2$ and $\mathcal{H}_3$ from a universal one-way hash family.
- Output the group public key $gpk = (G_1, G_2, F, C, D, H, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3)$, the issuing key $ik = (a, b)$, and the opening key $ok = (x_1, x_2, y_1, y_2, z)$.

Member Secret Key Generation: MKg
Given a group public key $gpk = (G_1, G_2, F, C, D, H, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3)$, a issuing key $ik = (a, b)$ and the identity of a member, MKg runs as follows. The number of members n must be polynomially bound by k.

- Choose $k_{i2} \in \mathbb{Z}_q$ at random.
- If k_{i2} is already assigned to another member, restart from the beginning.

- Compute $k_{i1} = b - ak_{i2} \bmod q$ and let $grt_i = k_{i1}$ then give it to Opener and Revocation Manager. it is used in Open and Revoke.
- Compute the tracing information $T_i = {G_1}^{k_{i1}}$ and publish it on a public directory, which is used in Judge. Note that T_i is not necessarily made public if Opener is trusted and Judge is not required.
- Output the member secret key $gsk_i = (k_{i1}, k_{i2})$, the tracing information T_i and the group revocation token $grt_i = k_{i1}$.

Signature Generation: GSig
Given a group public key $gpk = (G_1, G_2, F, C, D, H, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3)$, a member secret key $gsk_i = (k_{i1}, k_{i2})$, the tracing information T_i and a message $m \in \{0,1\}^*$, GSig runs as follows.

- Choose $r \in \mathbb{Z}_q$ at random.
- Compute $U_1 = {G_1}^r$, $U_2 = {G_2}^r$ and $E = H^r T_i$.
- Compute $\alpha = \mathcal{H}_1(U_1, U_2, E)$.
- Compute $V = C^r D^{r\alpha}$.
- Compute $R = {T_i}^r (= {U_1}^{k_{i1}} = {G_1}^{k_{i1} r})$.
- Choose $r_1, r_2, r_r \in \mathbb{Z}_q$ at random.
- Compute $R_1 = {G_1}^{r_1} {G_2}^{r_2}$, $R_2 = {G_1}^{r_r}$, $R_3 = H^{r_r} {G_1}^{r_1}$ $R_4 = C^{r_r} D^{r_r \alpha}$ and $R_5 = {U_1}^{r_1}$.
- Compute $\beta = \mathcal{H}_2(gpk, U_1, U_2, E, V, R, R_1, R_2, R_3, R_4, R_5, m)$.
- Compute $s_1 = r_1 + \beta k_{i1} \bmod q$, $s_2 = r_2 + \beta k_{i2} \bmod q$ and $s_r = r_r + \beta r \bmod q$.
- Output the signature $\sigma = (U_1, U_2, E, V, R, \beta, s_1, s_2, s_r)$.

Signature Verification: GVf
Given a group public key $gpk = (G_1, G_2, F, C, D, H, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3)$, a message $m \in \{0,1\}^*$, a signature $\sigma = (U_1, U_2, E, V, R, \beta, s_1, s_2, s_r)$ and a revocation list $\mathcal{RL}$, GVf runs as follows. It consists of two phases Signature Check, which checks if the signature is well-formed, and Revocation Check, which checks whether the signer is revoked or not.

Signature Check

- Compute $\alpha = \mathcal{H}_1(U_1, U_2, E)$.
- Compute $R'_1 = F^{-\beta} {G_1}^{s_1} {G_2}^{s_2}$, $R'_2 = {U_1}^{-\beta} {G_1}^{s_r}$, $R'_3 = E^{-\beta} H^{s_r} {G_1}^{s_1}$ $R'_4 = V^{-\beta} C^{s_r} D^{s_r \alpha}$ and $R'_5 = R^{-\beta} {U_1}^{s_1}$.
- Compute $\beta' = \mathcal{H}_2(gpk, U_1, U_2, E, V, R, R'_1, R'_2, R'_3, R'_4, R'_5, m)$.
- Check if $\beta = \beta'$ holds. If it does not hold, output invalid and abort. Otherwise, go to Revocation Check.

Revocation Check

- For each $k_1 \in \mathcal{RL}$, check if $R = {U_1}^{k_1}$. Output invalid if there is such k_1 that the equation holds. Output valid otherwise.

Claiming Generation of Signature: Claim
In Claim, the signer initiates a protocol with the verifier to show the knowledge of $\log_{U_1} R (= k_{i1})$ without revealing k_{i1}. The verifier outputs true if it succeeds and false otherwise. Claim can be an arbitrary zero knowledge protocol to prove the

knowledge of the discrete logarithm provided it is based on weaker assumptions than the DDH assumption such as DH assumption or the discrete logarithm assumption.

Signer Identification: Open
Given a group public key $gpk = (G_1, G_2, F, C, D, H, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3)$, an opening key $ok = (x_1, x_2, y_1, y_2, z)$, a message m and a signature $\sigma = (U_1, U_2, E, V, R, \beta, s_1, s_2, s_r)$, Open runs as follows.

- Check the validity of the signature by the same procedure of Signature Check in GVf. Output invalid and abort if it fails.
- Compute $\alpha = \mathcal{H}_1(U_1, U_2, E)$.
- Check if ${U_1}^{x_1+y_1\alpha}{U_2}^{x_2+y_2\alpha} = V$. If it does not hold, output invalid and abort.
- Compute $T = E/U_1^z$.
- Look for i such that $T_i = T$. If no such i is found, return invalid and abort.
- Choose $r_1 \in \mathbb{Z}_q$ at random.
- Compute ${S_1} = {G_1}^{r_1}$ and ${S_2} = {U_1}^{r_1}$.
- Compute $\gamma = \mathcal{H}_3(gpk, T_i, S_1, S_2, \sigma)$.
- Compute $t_1 = r_1 + \gamma k_{i1} \bmod q$.
- Output (i, τ) where $\tau = (\gamma, t_1)$.

Checking Identified Signer: Judge
Given a group public key $gpk = (G_1, G_2, F, C, D, H, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3)$, a signature $\sigma = (U_1, U_2, E, V, R, \beta, s_1, s_2, s_r)$, the identity of a member i and a PVPO $\tau = (\gamma, t_1)$, Judge runs as follows.

- Compute $S_1' = {T_i}^{-\gamma}{G_1}^{t_1}$ and $S_2' = R^{-\gamma}{U_1}^{t_1}$.
- Compute $\gamma' = \mathcal{H}_3(gpk, T_i, S_1', S_2', \sigma)$.
- Check if $\gamma = \gamma'$ holds then output true if the equation holds and false otherwise.

Membership Revocation: Revoke
Given the set of all revocation tokens **grt** and a set of revoked users' identities RU, let $\mathcal{RL} = \{grt_i \mid i \in RU\}$ then publish it on a public directory.

4.3 Security

Theorem 1. *The proposed scheme has* Correctness.

Theorem 2. *The proposed scheme has* Anonymity *under the DDH assumption in the random oracle model.*

Theorem 3. *The proposed scheme has* Traceability *under the assumption that discrete logarithm problem is hard in the random oracle model.*

Theorem 4. *The proposed scheme has* Weak Non-Frameability *under the assumption that discrete logarithm problem is hard in the random oracle model.*

The proofs of these theorems are given in section 5.

Remark 1. In the security definition in section 2, for simplicity, all member secret keys are generated at once at the setup phase. In the proposed scheme, however, allows Issuer to generate them each time a member is added to the group. This saves the storage.

4.4 Efficiency

In this section, we show the efficiency of the proposed scheme by comparing with previous schemes of [9, 13, 15]. Note that the previous schemes have coalition resistance and exculpability whereas the proposed scheme does not.

Comparison with RSA based Schemes. We compare the proposed scheme with RSA signature and currently the most efficient RSA based schemes proposed in [9]. Three schemes are proposed in [9]. including the basic scheme, which does not support Join and Revoke, and the full-revocation scheme, which supports VLR. We denote them [CG04]Basic and [CG04]VLR respectively and compare them with the proposed scheme.

We focus only on the exponentiation operations because the cost is so large that the cost of other operations is negligible. The computational cost of modular exponentiation is linearly dependent of the bit length of exponents if the bit length of modulus is the same. Thus, we use the total bit length of exponents in order to compare the computational costs. We also consider the Chinese remainder theorem, by which we estimate the computational cost is reduced to 1/4, and simultaneous multiple exponentiation(e.g. [21]).

We compare the schemes on the same security level as the recommended security parameter of [9]. In particular, we use 2048bit RSA modulus, a subgroup of $\mathbb{Z}_p{}^*$ of order q where p is a 2048bit prime and q is a 224bit prime ($\mathbb{Z}_p{}^*2048$) and an 224bit elliptic curve (EC224). Note that a point on EC224 can be represented as a 225bit string. These security level is considered to be equivalent. See [27], for example.

The efficiency is summarized in Table 1.

Table 1. Comparison with RSA based Schemes

	RSA2048	[CG04]		The Proposed Scheme	
		Basic	VLR	$\mathbb{Z}_p{}^*2048$	EC224
Computational Cost of GSig	512	4180	4462	2240	
Computational Cost of GVf	small	2310	2592	1120	
Bit Length of Signature	2048	12426	14976	11136	2021
Bit Length of Signing Key	4096	4438	4720	448	
Bit Length of Verification Key	≈2048	16666	18714	12288	1350
Bit Length of Authority Key(s)	—	282		448+1120	

Comparison with Pairing based Schemes. We compare the proposed scheme with currently the most efficient pairing based schemes proposed in [15] and [13], which we denote [FI05] and [DP06], respectively.

It is not easy to compare the speed because the cost of computing bilinear mappings heavily depends on implementation. However, one implementation result by Hansen et al. in [17] shows that [CG04] is a bit faster than [FI05] as pointed out in the full version of [9]. [DP06] is faster but is not expected

to be twice faster than [FI05] because it requires 7 multi-exponentiations in $\mathbb{G}_1$ and 1 multi-exponentiations in $\mathbb{G}_T$ whereas [FI05] requires 7 and 4 multi-exponentiations, respectively, where $\mathbb{G}_1$ and $\mathbb{G}_T$ is the range and the domain of the bilinear mapping. Also note that according to [25], the computational cost of bilinear mappings with precomputation is 1.5–2times of RSA decryption and that without precomputation is 2.5–4.5times, that is, the cost is still relatively expensive. Moreover, the above estimation of [FI05] and [DP06] is based on precomputing bilinear mappings. However, there is a time-memory tradeoff and it is required to store several elements of $\mathbb{G}_T$. Those elements are 1020bit each, for example, and require larger storage. [FI05] and [DP06] become much slower without precomputation.

From view point of signature length, the proposed scheme is more efficient. The length of signature of the proposed scheme is 1365bit, while those of [FI05] and [DP06] are 1711bit and 1444bit, respectively, on the same security level.

4.5 Basic Scheme without Revocation

A scheme without revocation can be easily obtained by removing R from the signature. Although it does not have Self-Traceability and Weak Non-Frameability, it is more efficient than the scheme with revocation.

5 Proofs of Security

5.1 Proof of Theorem 1 (Correctness)

Proof. By inspection. □

5.2 Proof of Theorem 2 (Anonymity)

Proof. Suppose there exists a polynomial time adversary $\mathcal{A}^{\text{anon}}$ which breaks Anonymity of the proposed scheme. We describe how to construct a polynomial time adversary $\mathcal{A}^{\text{cs}}$ which breaks the indistinguishability of Cramer-Shoup cryptosystem by using $\mathcal{A}^{\text{anon}}$.

Given a public key of Cramer-Shoup cryptosystem $pk = (G_1, G_2, C, D, H, \mathcal{H}_1)$, generate gpk, $\mathbf{T}$, $\mathbf{grt}$ and gsk_u as follows then provide $\mathcal{A}^{\text{anon}}$ with gpk and $\mathbf{T}$.

- For each $i \in \{1, \ldots, n\}$, choose $k_{i1} \in \mathbb{Z}_q$ at random then let $T_i = {G_1}^{k_{i1}}$ and $grt_i = k_{i1}$.
- Choose $u \in \{1, \ldots, n\}$ at random.
- Choose $k_{u2} \in \mathbb{Z}_q$ at random.
- Let $F = {G_1}^{k_{u1}} {G_2}^{k_{u2}}$.
- Let $gpk = (G_1, G_2, F, C, D, H, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3)$ and $gsk_u = (k_{u1}, k_{u2})$.

When $\mathcal{A}^{\text{anon}}$ makes a query to MKg on i, $\mathcal{A}^{\text{cs}}$ returns gsk_u to $\mathcal{A}^{\text{anon}}$ if $i = u$ or aborts otherwise.

We then show how to simulate the oracles GSig and Open. Simulating Revoke and Claim are straightforward by using $\mathbf{grt}$. $\mathcal{H}_2$ and $\mathcal{H}_3$ are simulated as random oracles.

Upon receiving signing oracle query from $\mathcal{A}^{\text{anon}}$ on a member i and a message m, compute and return its signature σ by encrypting T_i, backpatching $\mathcal{H}_2$ and simulating NIZK as follows.

- Computing U_1, U_2, E, V, R is the same as GSig.
- Choose $\alpha, \beta, s_1, s_2, s_r \in \mathbb{Z}_q$ at random.
- Compute $R_1 = F^{-\beta} {G_1}^{s_1} {G_2}^{s_2}$, $R_2 = {U_1}^{-\beta} {G_1}^{s_r}$, $R_3 = E^{-\beta} H^{s_r} {G_1}^{s_1}$ $R_4 = V^{-\beta} C^{s_r} D^{s_r \alpha}$ and $R_5 = R^{-\beta} {U_1}^{s_1}$.
- Backpatch $\mathcal{H}_1$ and $\mathcal{H}_2$ by defining $\alpha = \mathcal{H}_1(U_1, U_2, E)$ and $\beta = \mathcal{H}_2(gpk, U_1, U_2, E, V, R, R_1, R_2, R_3, R_5, m)$.
- Return $\sigma = (U_1, U_2, E, V, R, \beta, s_1, s_2, s_r)$.

Upon receiving opening oracle query from $\mathcal{A}^{\text{anon}}$ on a message m and a signature $\sigma = (U_1, U_2, E, V, R, \beta, s_1, s_2, s_r)$, check the validity of σ, decrypt tracing information by using the decryption oracle of Cramer-Shoup cryptosystem and return the result to $\mathcal{A}^{\text{anon}}$ as follows.

- Compute $R'_1 = F^{-\beta} {G_1}^{s_1} {G_2}^{s_2}$, $R'_2 = {U_1}^{-\beta} {G_1}^{s_r}$, $R'_3 = E^{-\beta} H^{s_r} {G_1}^{s_1}$ $R'_4 = V^{-\beta} C^{s_r} D^{s_r \alpha}$ and $R'_5 = R^{-\beta} {U_1}^{s_1}$.
- Check if $\mathcal{H}_2$ is defined on input $(gpk, U_1, U_2, E, V, R, R'_1, R'_2, R'_3, R'_4, R'_5, m)$.
- If it is not defined or the output is not equal to β, return invalid and abort.
- Make an oracle query to the decryption oracle of Cramer-Shoup cryptosystem on input ciphertext (U_1, U_2, E, V) to obtain the plaintext T.
- Look for i such that $T_i = T$. If no such i is found, return invalid and abort.
- Generate τ in the same way as Open and return (i, τ).

Given the challenge query $\hat{m}$, $\hat{i}_0$ and $\hat{i}_1$ from $\mathcal{A}^{\text{anon}}$, simulate the response signature $\hat{\sigma}$ as follows.

- Make a challenge query on two plaintexts $T_{\hat{i}_0}$ and $T_{\hat{i}_1}$ to the encryption oracle of Cramer-Shoup cryptosystem.
- Let the response be $(\hat{U}_1, \hat{U}_2, \hat{E}, \hat{V})$, which is the encryption of either $T_{\hat{i}_0}$ or $T_{\hat{i}_1}$.
- Backpatch $\mathcal{H}_2$ and simulate NIZK as in the simulation of signing query to get $\hat{\sigma} = (\hat{U}_1, \hat{U}_2, \hat{E}, \hat{V}, \hat{R}, \hat{\beta}, \hat{s}_1, \hat{s}_2, \hat{s}_r)$ then return it to $\mathcal{A}^{\text{anon}}$.

Finally, let the output of $\mathcal{A}^{\text{cs}}$ be the final output b' of $\mathcal{A}^{\text{anon}}$.

Since the above simulations succeed with non-negligible probability, $\mathcal{A}^{\text{cs}}$ breaks the indistinguishability of Cramer-Shoup cryptosystem. This contradict the DDH assumption. □

5.3 Proof of Theorem 3 (Traceability)

Proof. Suppose there exists a polynomial time adversary $\mathcal{A}^{\text{trace}}$ which breaks Traceability of the proposed scheme. We describe how to construct a polynomial time adversary $\mathcal{A}^{\text{dl}}$ which solves discrete logarithm problem by using $\mathcal{A}^{\text{trace}}$. Suppose the input to $\mathcal{A}^{\text{dl}}$ be (G_1, F). The goal is to find $\log_{G_1} F$.

Given $\log_{G_1} F$, generate gpk, ok, $\mathbf{T}$, $\mathbf{grt}$ and gsk_u as follows then provide $\mathcal{A}^{\text{trace}}$ with gpk, ok, $\mathbf{T}$ and $\mathbf{grt}$.

- For each $i \in \{1, \ldots, n\}$, choose $k_{i1} \in \mathbb{Z}_q$ at random then let $T_i = {G_1}^{k_{i1}}$ and $grt_i = k_{i1}$.
- Choose $u \in \{1, \ldots, n\}$ at random.
- Choose $k_{u2} \in \mathbb{Z}_q$ at random.
- Choose $ok = (x_1, x_2, y_1, y_2, z) \in {\mathbb{Z}_q}^5$ at random.
- Let $G_2 = (F{G_1}^{-k_{i1}})^{1/k_{i2}}$.
- Compute $C = {G_1}^{x_1}{G_2}^{x_2}$, $D = {G_1}^{y_1}{G_2}^{y_2}$ and $H = {G_1}^{z}$.
- Let $gpk = (G_1, G_2, F, C, D, H, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3)$ and $gsk_u = (k_{u1}, k_{u2})$.

When $\mathcal{A}^{\text{trace}}$ makes a query to MKg on i, $\mathcal{A}^{\text{dl}}$ returns gsk_u to $\mathcal{A}^{\text{trace}}$ if $i = u$ or aborts otherwise.

Simulation of GSig, Claim, $\mathcal{H}_2$ and $\mathcal{H}_3$ are the same as in the proof of Theorem 2.

Rewind $\mathcal{A}^{\text{trace}}$ to obtain two signatures $\sigma = (U_1, U_2, E, V, R, \beta, s_1, s_2, s_r)$ and $\sigma' = (U_1, U_2, E, V, R, \beta', s'_1, s'_2, s'_r)$. By applying the Forking Lemma [23], this succeeds with non-negligible probability. Let $k'_1 = (s_1 - s'_1)/(\beta - \beta')$ and $k'_2 = (s_2 - s'_2)/(\beta - \beta')$ then $F = {G_1}^{k'_1}{G_2}^{k'_2}$. On one hand, from the definition of Traceability u is not traced from these signatures, therefore $(k'_1, k'_2) \neq (k_{u1}, k_{u2})$ and $G_2 = {G_1}^{-(k_{u1}-k'_1)/(k_{u2}-k'_2)}$. Finally, $\log_{G_1} F$ can be obtained as $k_{u1} + ck_{u2}$ where $c = -(k_{u1} - k'_1)/(k_{u2} - k'_2)$.

Since the above simulations succeed with non-negligible probability, $\mathcal{A}^{\text{dl}}$ solves discrete logarithm problem with non-negligible probability. This is a contradiction. □

5.4 Proof of Theorem 4 (Weak Non-Frameability)

Proof. Weak Non-Frameability of the proposed scheme immediately follows from Traceability and the security of NIZK. That is, an adversary which forges with non-negligible probability an NIZK of discrete logarithm without knowing it can be constructed by using $\mathcal{A}^{\text{wnf}}$ and simulating GSig as in the proof of Theorem 3. □

References

1. Ateniese, G., Tsudik, G.: Some Open Issues and New Directions in Group Signatures. In: Franklin, M.K. (ed.) FC 1999. LNCS, vol. 1648, pp. 196–211. Springer, Heidelberg (1999)
2. Ateniese, G., Camenisch, J., Joye, M., Tsudik, G.: A Practical and Provably Secure Coalition-Resistant Group Signature Scheme. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 255–270. Springer, Heidelberg (2000)
3. Ateniese, G., Song, D., Tsudik, G.: Quasi-Efficient Revocation of Group Signatures. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 183–197. Springer, Heidelberg (2003)
4. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 614–629. Springer, Heidelberg (2003)

5. Bellare, M., Shi, H., Zhang, C.: Foundations of Group Signatures: The Case of Dynamic Groups. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 136–153. Springer, Heidelberg (2005)
6. Boneh, D., Boyen, X., Shacham, H.: Short Group Signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004)
7. Boneh, D., Shacham, H.: Group Signatures with Verifier-Local Revocation. In: Proc. of the 11'th ACM conference on Computer and Communications Security (CCS), pp. 168–177 (2004)
8. Camenisch, J., Stadler, M.: Efficient Group Signature Schemes for Large Groups. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997)
9. Camenisch, J., Groth, J.: Group Signatures: Better Efficiency and New Theoretical Aspects. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 120–133. Springer, Heidelberg (2005), http://www.brics.dk/~jg/
10. Chaum, D., Evertse, J.H., van de Graaf, J.: An Improved Protocol for Demonstrating Possession of Discrete Logarithms and Some Generalizations. In: Price, W.L., Chaum, D. (eds.) EUROCRYPT 1987. LNCS, vol. 304, pp. 127–141. Springer, Heidelberg (1988)
11. Chaum, D., van Heyst, E.: Group Signatures. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 257–265. Springer, Heidelberg (1991)
12. Cramer, R., Shoup, V.: A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998)
13. Delerablée, C., Pointcheval, D.: Dynamic Fully Anonymous Short Group Signatures. In: Nguyên, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 193–210. Springer, Heidelberg (2006)
14. Furukawa, J., Yonezawa, S.: Group Signatures with Separate and Distributed Authorities. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 77–90. Springer, Heidelberg (2005)
15. Furukawa, J., Imai, H.: An Efficient Group Signature Scheme from Bilinear Maps. In: Boyd, C., González Nieto, J.M. (eds.) ACISP 2005. LNCS, vol. 3574, pp. 455–467. Springer, Heidelberg (2005)
16. Furukawa, J., Imai, H.: An Efficient Group Signature Scheme from Bilinear Maps. IEICE Trans. Fundamentals E89-A(5), 1328–1337 (2006)
17. Hansen, H.S., Pagels, K.K.: Implementation og analyse af fem gruppesignatursystemer. Technical report (2006), http://www.daimi.au.dk/~slot/speciale/speciale.pdf
18. Kiayias, A., Yung, M.: Extracting Group Signatures from Traitor Tracing Schemes. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 630–648. Springer, Heidelberg (2003)
19. Kiayias, A., Yung, M.: Group Signatures: Provable Security, Efficient Constructions and Anonymity from Trapdoor-Holders. Cryptology ePrint Archive: Report 2004/076 (2004)
20. Kiayias, A., Tsiounis, Y., Yung, M.: Traceable Signatures. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 571–589. Springer, Heidelberg (2004)
21. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography
22. Nakanishi, T., Funabiki, N.: Verifier-Local Revocation Group Signature Schemes with Backward Unlinkability from Bilinear Maps. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 533–548. Springer, Heidelberg (2005)

23. Pointcheval, D., Stern, J.: Security Arguments for Digital Signatures and Blind Signatures. Journal of Cryptology 13(3), 361–396 (2000)
24. Schnorr, C.P.: Efficient Signature Generation by Smart Cards. Journal of Cryptology 4(3), 161–174 (1991)
25. Scott, M.: Implementing Cryptographic Pairings. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 197–207. Springer, Heidelberg (2007)
26. Song, D.: Practical Forward-Secure Group Signature Schemes. In: Proc. of 2001 ACM Symposium on Computer and Communication Security, pp. 225–234 (2001)
27. Draft Federal Information Processing Standard (FIPS) 186-3 — Digital Signature Standard (DSS), `http://csrc.nist.gov/publications/drafts.html`

The Superdiversifier: Peephole Individualization for Software Protection

Matthias Jacob[1], Mariusz H. Jakubowski[2], Prasad Naldurg[3], Chit Wei (Nick) Saw[2], and Ramarathnam Venkatesan[2,3]

[1] Nokia
[2] Microsoft Research
[3] Microsoft Research India

Abstract. We present a new approach to individualize programs at the machine- and byte-code levels. Our *superdiversification* methodology is based on the compiler technique of superoptimization, which performs a brute-force search over all possible short instruction sequences to find minimum-size implementations of desired functions. Superdiversification also searches for equivalent code sequences, but we guide the search by restricting the allowed instructions and operands to control the types of generated code. Our goal is not necessarily the shortest or most optimal code sequence, but an individualized sequence identified by a secret key or other means, as determined by user-specified criteria. Also, our search is not limited to commodity instruction sets, but can work over arbitrary byte-codes designed for software randomization and protection. Applications include patch obfuscation to complicate reverse engineering and exploit creation, as well as binary diversification to frustrate malicious code tampering. We believe that this approach can serve as a useful element of a comprehensive software-protection system.

1 Introduction

One element of a comprehensive program-security toolbox is *code individualization*, which enables software diversity as a defense against various attacks. When different users receive individualized copies of the same sensitive application, a break on one system may not work on others, ideally forcing crackers to expend the same effort on breaking each copy. Individualization also helps to alleviate the so-called *monoculture problem* [15], which involves quick propagation of malicious programs on networks of equally vulnerable systems. As in biology and fault-tolerant systems, diversity can be an effective generic technique to guard against known and unknown threats.

Software individualization uses a number of techniques to modify code at various levels without changing its semantics [1, 2, 8]. For example, at an algorithmic level, a bubble-sort function can be replaced by quicksort or heapsort. At a syntactic level, a tool can inject inert "chaff" code, reorder basic blocks by flipping branch conditions, and replace instructions with operationally equivalent sequences. Such measures may be implemented via compilers and transformation tools.

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 100–120, 2008.

In this paper, we focus on individualization at the instruction or byte-code level. Our main idea is search for a large number of instruction sequences that are semantically equivalent to an input code fragment. To accomplish this, we leverage and extend the compiler technique of superoptimization [4, 19, 21], which performs brute-force searches for minimum-size code fragments that implement a given simple function. Our work thus proposes a novel application of superoptimization towards software security, including areas such as exploit prevention and tamper-resistance, and leading towards code obfuscation.

1.1 Software Protection

Since the early days of the IBM PC and 8-bit home computers, software protection has been a never-ending battle between protectors and crackers. While no secure generic solution has been demonstrated in practice, various techniques of tamper-resistance and obfuscation have helped to delay reverse engineering and cracking of sensitive code and data [3, 7, 9, 10, 11, 18, 24, 27]. Among such approaches are code checksums or hashes to prevent easy patching, anti-debugging and anti-disassembly measures to hinder attack tools, and various obfuscation measures to complicate control- and data-flow analysis. For increased security, modern protection tools typically combine a large variety of such features.

Recent theoretical work [5, 16, 20, 28] suggests that only limited obfuscation is possible in general. However, such modeling has been generally distant from practice, particularly given the large variety of possible application and attack scenarios. We note that most published work on real-world software protection does not provide a cryptographic model or formally analyzable security. Even in systems whose breaking reduces to solving some difficult problem, it is often possible to "go outside the model" to produce efficient attacks. A system with guaranteed or estimated tamper-resistance measured in days, weeks or months may be formally insecure, yet quite sufficient for various business applications. Thus, presently deployed protection strategies generally rely not on theoretical models, but rather on heuristic analysis, quality of engineering, and penetration testing on software protected by combined techniques. In this context, individualization serves as an element of a comprehensive protection strategy or model [12].

1.2 Superoptimization and Software Diversity

Superoptimization was introduced by Massalin as a means of minimizing size of compiled code [21]. Though software security was not a stated goal of his original work, his description hints at some obfuscation and individualization inherent in superoptimized code. Massalin wrote that "startling programs have been generated, many of them engaging in convoluted bit-fiddling bearing little resemblance to the source programs which defined the functions" [21]. One of his examples is a sign function superoptimized to four 680x0 instructions, where "like a typical superoptimized program, the logic is really convoluted" [21] and warrants the detailed explanation he provides. Such code sequences often contain

clever tricks that appear to be human-generated, yet were found automatically by simply enumerating all possible instructions up to a certain length. In essence, brute-force search has substituted for human cleverness.

Motivated by the above observations, we build upon superoptimization to develop a diversification toolkit. In particular, we output not only the shortest instruction sequence, but any sequences that implement the input function. Additionally, we typically use a secret key and other user-specified parameters to guide the brute-force search and control the nature of generated code (e.g., by changing the set of instructions and operands over which the search is performed). Our search may also use other heuristics, such as instruction frequencies collected from real-life programs. For example, the search may prefer instruction combinations observed often in real code, deferring or omitting instruction sequences not commonly found in binaries. We also use other performance enhancements, such as limiting exhaustive enumeration of constant operands (e.g., restricting constants to small ranges or sets of values harvested from real-life software). We refer to our approach as *superdiversification* or *superindividualization*.

The rest of the paper is organized as follows. We outline our basic methodology of superdiversification in Section 2. Implementation details and performance results are the topic of Section 3. In Section 4, we show several applications of our basic methodology. Finally, Section 5 summarizes our approach with directions for future work.

2 Superdiversification

Like a superoptimizer, our code individualizer accepts a sequence of instructions as input and attempts to generate equivalent code sequences up to a given length, by exhaustive enumeration. A secret key and other user-specified parameters may further determine characteristics of these code fragments and order of generation, as explained in detail later. Each candidate sequence is tested for equivalence to the input sequence. The test proceeds in two phases – a quick, probabilistic equivalence check, which rejects sequences that do not agree on random inputs, and Boolean equivalence checking of formulas when the sequences agree on a threshold number of random inputs:

- During the most common, quick phase of the test, the sequence is first executed or simulated on one or more random sample inputs; if the output is incorrect, the sequence is quickly discarded.
- If the candidate sequence generates correct outputs, a Boolean test is used to verify that the sequence is actually equivalent to the input sequence. For this, we encode instructions as Boolean formulas in conjunctive normal form (CNF) and use a SAT solver to test equivalence, as described later.

2.1 Generation of Code Sequences

In our basic approach, we enumerate a large number of candidate code sequences to test for equivalence with an input sequence. In particular, superdiversification performs one or more of the following actions:

- Restrict the set of instructions and operands over which the search is performed. For example, if we wish to generate a sign function that uses only addition, subtraction and negation, we simply search over just these three instructions. This has an effect similar to switching from brute-force to heuristically-guided search.
- Guide the search based on empirical data about probabilities of instruction occurrences in real-life programs. When choosing the next instruction in a candidate sequence, our enumeration procedure prefers instructions that are likely to appear, as determined by a table of instruction frequencies harvested from actual binaries. This is intended both to speed up the search and produce diversified code that blends in with target applications.
- Use various optimizations and pruning techniques to cut down on the time required for search. For example, a user-specified parameter determines whether or not to search over instruction pairs and longer sequences that have never been observed in actual binaries. In addition, to prevent brute-force enumeration of constant operands (e.g., 64-bit), we collect or harvest constants from real-life programs and search only over these. Alternately, we may use random (key-based) sampling to choose constants, as well as to enumerate instruction sequences in general. With such sampling, the search will not be exhaustive, but may still yield usefully individualized sequences.

Given an input code sequence of length m, we generate all code sequences up to a specified maximum length n, where $n \geq m$. For instruction selection, we use a subset of the processor's instruction set. The generated sequence uses the same inputs and modifies the same outputs as the input sequence. In addition, the generated sequence may also use a limited set of constant operands, restricted to commonly occurring values (e.g., 0, 1, -1), or alternately drawn from a set of constants harvested from real-life programs. We may also choose to reject generated sequences containing obviously redundant instructions that amount to no-ops unlikely to occur in actual binaries (e.g., `mov r1, r1`, or `add r1, 0`). To find such instructions, we can run the superdiversifier on a no-op input sequence, and ask it to find all equivalent sequences of length 1.

In order to improve search time, as well as to produce sequences that blend in more closely with actual programs, we can use a table containing empirical data about instruction frequencies. The table data are pre-computed from a sample set of actual binaries, where f_{jk} is the frequency with which instruction i_k is observed to follow instruction i_j in the sample set. The code generator will pick the next instruction in the generated sequence in order of decreasing frequency, based on the most recent instruction generated. This will naturally favor sequences that appear to blend in with compiled code in real binaries, thus making it more difficult to distinguish between unmodified compiler-generated code and code produced by the superdiversifier in an individualized binary.

We may further guide the search by randomly re-ordering the enumeration based on a secret key, or by eliminating instructions – either randomly or using heuristic information based on knowledge about the input sequence.

2.2 Practical Issues

In both superdiversification and superoptimization, the search space for code generation is exponential in the number of instructions in the sequence. Since superoptimization discards all instruction sequences longer than the current minimum-length equivalent sequence, its brute-force search often may be quite fast. In contrast, superdiversification searches over sequences of all lengths (within a user-specified range), and is thus inherently slower. However, even two decades ago, Massalin [21] was able to find superoptimized sequences of 10 or more instructions. Since computing systems today are several orders of magnitude more powerful, we expect that superdiversification can still be quite effective, particularly with the use of optimizations and pruning mechanisms.

Our approach is useful for generating various types of code, as determined by parameters. Using empirical instruction frequencies, as mentioned earlier, we can produce diversified code that does not stand out from the target application, helping to improve security. Alternately, we may produce unusual code by preferring instructions unlikely to appear (i.e., using the inverse of our instruction-frequency table). In general, parameterized diversification is a powerful enhancement over superoptimization in terms of security.

2.3 Testing for Equivalence of Code Sequences

To determine whether two machine-code fragments compute the same function, we first execute each sequence using one or more sets of randomly generated inputs, and compare their corresponding outputs. If their outputs are equal, we consider the sequences to be potentially equivalent and convert the instructions to Boolean formulas in conjunctive normal form (CNF). For each instruction, we assign distinct Boolean variables representing each bit of its inputs and outputs. The output variables are the result of the Boolean function performed by the instruction over its inputs. For example, consider the generic instruction

```
and r1, r2
```

which performs a bitwise-AND on registers `r1` and `r2`, placing the result in register `r1`. Let the variable x represent `r1` and y represent `r2`. Then we can represent the function for the `and` instruction by

$$F(x, y) = (x \wedge y)$$

Using static single assignment (SSA) notation such that each variable is assigned no more than once, we can rewrite this as the relation

$$x1 \leftrightarrow (x0 \wedge y0)$$

Note that we would have one such relation for each data bit. For example, for a w-bit data width, we would have w relations each representing the effect of the `and` instruction on each bit of the output. Since all variables are unique, we can simply combine them in a single conjunction as follows:

$$\varphi = \bigwedge_{i=1}^{w} (x1_i \leftrightarrow (x0_i \wedge y0_i))$$

To ensure that the resulting formula is in CNF, each term in the conjunction needs to be converted to CNF. We apply the Tseitin transformation [25] to the above expression, resulting in

$$\varphi = \bigwedge_{i=1}^{w} ((\neg x1_i \vee x0_i) \wedge (\neg x1_i \vee y0_i) \wedge (x1_i \vee \neg x0_i \vee \neg y0_i))$$

We repeat the above for each instruction in the sequence, encoding the operation performed by the instruction as a Boolean expression in CNF form. Thus, for a length-N instruction sequence, we would have N sub-formulas, and the entire sequence would be represented as

$$\Phi = \bigwedge_{i=1}^{N} \varphi_j$$

Let Φ' be the generated length-M sequence that we wish to test for equivalence against Φ.

$$\Phi' = \bigwedge_{i=1}^{M} \varphi'_j$$

We first ensure that the input states of both sequences are synchronized such that the same variables are used to represent inputs that are common to both sequences. In the above example, the initial values of `r1` and `r2` are represented by the variables $x0$ and $y0$ in both sequences. All other variables representing the intermediate and output states should be distinct and unique to each sequence. Let A be the set of variables representing the output state of Φ, and A' be the set of variables representing the output state of Φ'. Then we can define equivalence in terms of the output state as follows:

$$\Phi \equiv \Phi' \qquad \text{iff} \qquad \forall x \in A, y \in A' : A \leftrightarrow A'$$

or,

$$\Phi \not\equiv \Phi' \qquad \text{iff} \qquad \exists x \in A, y \in A' : A \nleftrightarrow A'$$

For each bit variable x_i in A corresponding to the same bit in A' (represented by y_i), we generate the relation

$$z_i \leftrightarrow (x_i \nleftrightarrow y_i)$$

This is equivalent to

$$z_i \leftrightarrow (x_i \oplus y_i)$$

Thus, for K output bits, we can state that

$$\Phi \not\equiv \Phi' \qquad \text{iff} \qquad \bigvee_{i=1}^{K} z_i = \top$$

In other words, $\Phi \equiv \Phi'$ iff the following CNF formula is *un*satisfiable:

$$(\Phi \wedge \Phi' \wedge (\bigvee_{i=1}^{K} z_i))$$

Generating the above formula and applying a SAT solver to it, we can thus determine whether two sequences are equivalent. If the SAT solver finds that the formula is satisfiable, then we have established that the sequences are *not* equivalent. Conversely, if the SAT solver determines that the formula is unsatisfiable, then the sequences are equivalent.

3 Implementation and Experimental Results

We implemented an initial prototype of the code individualizer in C++. This consists of a code-generation module (front-end) and a code verification module (back-end). We included only a subset of Intel x86 instructions (`mov`, `not`, `neg`, `xor`, `or`, `and`, `add`, `sub`, `inc`, `dec`, `cmp`, `shl`, `shr`, `sar`, `push`, `pop`), but the design allows future support for other instruction sets to be added. (In particular, we may use custom instruction sets geared towards individualization and obfuscation, not necessarily generality and performance.) For the verification phase, we perform a quick execution test on every sequence generated, followed by a Boolean test using the zChaff [26] SAT solver only if the quick test passes. With this approach, we were able to generate code sequences of up to 5 or 6 instructions in length in reasonable time over the full range of supported instructions. We ran all our tests on a Pentium 3.0 GHz CPU with 2 GB of RAM. Note that emulation would be needed to run the quick test on non-native instruction sets.

The code individualizer works as follows. Given an input machine-code fragment, the generator enumerates the next candidate code sequence. The verifier then compares the generated sequence against the original sequence. If it is determined that the two sequences are equivalent, the generated sequence is added to the list of equivalent sequences. The process is then repeated until the generator has generated all sequences.

While our approach generates code sequences that are exact matches with respect to the output states, in practice it is frequently sufficient to find a match over a subset of the output state. For example, in the Intel x86 instruction set, the `inc` and `add` instructions affect the carry flag differently. However, if we do not care about the value of the carry flag (i.e., it is not live) in the context of the original program, the instruction `inc r1` could be substituted with `add r1, 1` and vice versa. In order to allow for such possibilities, we made provisions in the code individualizer to designate flags and output variables that may be safely ignored during the verification phase. To store intermediate results, we also provided the ability to introduce free temporary registers not already in the input sequence into the generated sequence.

In general, modeling arbitrary memory accesses is a complex problem. In our current implementation, we limited the inputs and outputs to constants, registers, and stack memory variables only.

3.1 Sample Results and Discussion

We ran the code generator and verifier on a sample set of input sequences. In order to produce a greater variety of generated sequences, we chose to ignore the processor flags when comparing output states. Consequently, further program analysis to determine live flags would be necessary prior to considering any of the generated sequences for substitution in the original program.

We present results for some input sequences taken from actual programs. Consider a 3-instruction sequence, *seq1*, that swaps the contents of two registers, `r1` and `r2`, using `r3` as a temporary register[1]:

seq1:

```
mov r3, r1
mov r1, r2
mov r2, r3
```

Assuming `r3` is free and ignoring flag side-effects, the code individualizer was asked to generate all equivalent sequences up to length 5 using only the `mov`, `xor`, `push` and `pop` instructions. A total of 2426 equivalent sequences were found out of 8308224 generated in total. To minimize the correlation between generated equivalents and the input sequence, we filtered out sequences containing any of the original instructions in the input sequence. We also filtered out obvious no-op instructions. Table 1 shows a small sample of some of the more interesting sequences generated for different lengths n. The number in parentheses is the total number of equivalent sequences of length n that were found.

By selecting a different set of instructions for generation (e.g., using a secret key), we generate a completely different set of equivalent sequences for *seq1*. For example, Table 2 shows some of the equivalents we found using only arithmetic and logical operations.

In contrast to superoptimization, the generated sequences were no more optimized in size and execution time than the original sequence, and in most cases resulted in longer, less efficient code. However, consistent with the objectives of superdiversification, each sequence produced by the code individualizer presents a different implementation of the same input function, and in some cases the implementation can be very different (and non-obvious) from the original code.

As another example, consider a sequence consisting of a `mov` followed by an `add` instruction. This sample was taken from a binary generated by the Microsoft Visual C++ compiler:

seq2:

```
mov r1, r2
add r1, 0x10
```

[1] For generality, we use r1, r2, etc., to denote the register operands in these examples rather than their x86 names.

Table 1. Sample Generated Equivalents For Sequence 1 (a)

$n = 3$ (4)	$n = 4$ (113)	$n = 5$ (2309)

`xor r1, r2`
`xor r2, r1`
`xor r1, r2`
	`push r1`
	`push r2`
	`pop r1`
	`pop r2`
		`xor r2, r1`
		`mov r3, r2`
		`mov r2, r1`
		`xor r3, r2`
		`mov r1, r3`

`push r2`
`mov r2, r1`
`pop r1`
	`push r1`
	`xor r1, r1`
	`xor r1, r2`
	`pop r2`
		`xor r2, r1`
		`push r2`
		`mov r2, r1`
		`pop r3`
		`xor r1, r3`

`mov r3, r2`
`mov r2, r1`
`mov r1, r3`
	`mov r3, r2`
	`push r1`
	`mov r1, r3`
	`pop r2`
		`xor r2, r3`
		`xor r3, r1`
		`xor r1, r2`
		`xor r2, r1`
		`xor r1, r3`

Table 2. Sample Generated Equivalents For Sequence 1 (b)

$n = 4$ (4)	$n = 5$ (176)

`add r1, r2`
`sub r2, r1`
`add r1, r2`
`neg r2`
	`and r3, r1`
	`or r3, r1`
	`sub r3, r2`
	`add r2, r3`
	`sub r1, r3`

`sub r1, r2`
`add r2, r1`
`sub r1, r2`
`neg r1`
	`sub r3, r3`
	`add r3, r2`
	`sub r3, r1`
	`add r1, r3`
	`sub r2, r3`

Feeding this to the individualizer and allowing it to utilize a small set of constants, Table 3 illustrates some of the equivalents we were able to generate:

Table 3. Sample Generated Equivalents For Sequence 2

$n = 2$ (1)	$n = 3$ (122)	$n = 4$ (13538)

`mov r1, 0x10`
`add r1, r2`
	`mov r1, 0x4`
	`shl r1, 0x2`
	`add r1, r2`
		`mov r1, 0xf`
		`xor r1, 0xffffffff`
		`sub r1, r2`
		`neg r1`

	`mov r1, 0x8`
	`add r1, r1`
	`add r1, r2`
		`mov r1, 0xf`
		`sub r1, 0x2`
		`add r1, r2`
		`add r1, 0x3`

Finally, we created the following 2-instruction input sequence to illustrate some additional types of sequences the individualizer generates. This sequence simply adds 1 to the value in register r1, then clears its most significant bit:

seq3:

```
add r1, 0x1
and r1, 0x7fffffff
```

Of the equivalent sequences generated by the individualizer, we observe groups of sequences that share certain common characteristics, which we attempt to classify in Table 4. Each sample is an example of a class of transforms bearing distinct features that we frequently observe in the generated results.

These categories of sequences were representative of what we observed when generating equivalents for a variety of input sequences. It appears that those belonging to Categories I and II are most "different" from the original, as might be measured by their edit distance.

We found that without explicitly filtering out sequences containing the original instructions, many of the longer-length generated sequences fell into Category V or Category VI, comprising simply the original sequence with the addition of inert "chaff" instructions. While these sequences do not constitute a transformed version of the original, they may still be useful from the point of view of diversity, since we can generate a large number of such sequences. However, they may simply be filtered out as mentioned above and discarded if we want to exclude these categories of sequences. A related class of transforms are those comprising previously generated sequences of shorter length, with the addition of chaff instructions. Similarly, these may be filtered out by excluding all sequences that contain any sub-sequence that has already been encountered.

3.2 Performance

As with superoptimization, the quick execution test is key to dramatically reducing the search time (by approximately a factor of 50 in our current implementation) by ruling out obviously non-equivalent sequences without having to perform the much slower Boolean test. This does, however, impose the requirement of either running on native hardware or under emulation, the latter of which would likely have performance implications.

We observed that the number of times the quick execution was run on each generated sequence can be a factor in whether or not the quick test yields false-positive matches, leading to additional Boolean test runs on non-equivalent sequences and hence reduced speed. Generally, the fewer output-bits that are altered by the sequence, the more execution test runs are required (over different random inputs) to reduce the number of false positives. Figures 1 and 2 illustrate the false-positive hit-rates and their impact on execution speed for different numbers of quick test runs.A relatively small number of quick-test runs on random input sets (about 4) is sufficient to reduce the rate of false positives

Table 4. Categories of Equivalent Sequence Transforms

Category	Characteristics	Examples

I. Corruption	Transformation or corruption of the original input, operation in the transformed domain, followed by uncorruption to yield the correct output.	`add r1, 0x1`
		`and r1, 0x7fffffff` ⟶
		`shl r1, 0x1`
		`add r1, 0x2`
		`shr r1, 0x1`

II. Substitution	Substitution of one or more operations in the original sequence with a different operation that has the same effect.	`add r1, 0x1`
		`and r1, 0x7fffffff` ⟶
		`add r1, 0x2`
		`sub r1, 0x1`
		`shl r1, 0x1`
		`shr r1, 0x1`

III. Instruction Re-ordering	Altering the order of the instructions where their order does not change the final result.	`add r1, 0x2`
		`sub r1, 0x1`
		`shl r1, 0x1`
		`shr r1, 0x1` ⟶
		`sub r1, 0x1`
		`add r1, 0x2`
		`shl r1, 0x1`
		`shr r1, 0x1`

IV. Operand Re-ordering	Altering the order in which the operands appear in the sequence while preserving the final result.	`mov r1, r2`
		`add r1, 0x10` ⟶
		`mov r1, 0x10`
		`add r1, r2`

V. Chaff Code	Addition of extra inert instructions or sequences of instructions into the original sequence (or a previously derived equivalent), such that they appear to be integral to the overall function but are in fact functionally irrelevant.	`mov r1, r2`
		`add r1, 0x10` ⟶
		`mov r1, r2`
		`or r1, r2`
		`and r1, r2`
		`add r1, 0x10`

VI. Chaff (Obvious)	Addition of extra inert instructions or sequences of instructions into the original sequence (or a previously derived equivalent) that are clearly irrelevant.	`mov r1, r2`
		`add r1, 0x10` ⟶
		`mov r1, r2`
		`mov r1, r1`
		`add r1, 0x10`

to the point where it no longer materially impacts the search. Note that in the case of Sequence 3, significantly increasing the number of test runs beyond this level does little to eliminate a residual number of false positives. In fact, with as

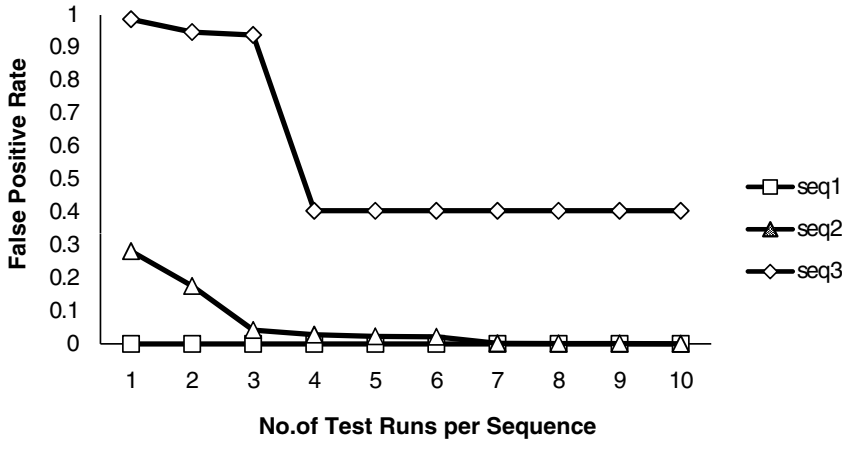


Fig. 1. Quick Execution Test False Positives

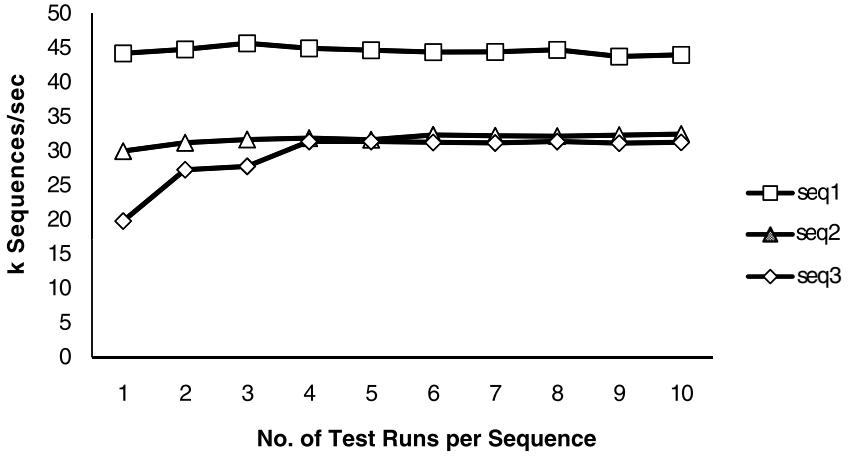


Fig. 2. Sequences Processed per Second

many as 1000 test runs, the same false-positive matches were present. This is due to the fact that for many input values, only a few bits of output are affected. In the case of Sequence 3, for example, consider the following false-positive match, where the instructions are simply interchanged:

```
and r1, 0x7fffffff
add r1, 0x1
```

It can be seen that for most initial values of `r1`, this sequence will yield the same result as the original sequence. It is only when we have input values of `0x7fffffff` or `0xffffffff` that the difference is exposed, and it is hard to

achieve this even with many random test inputs. Compare this to Sequence 1 (swapping two registers), where for most randomly selected inputs, a large number of output bits are changed. In fact, we did not observe any false positives for this sequence, even with only a single run of the quick test.

Due to the size of the search space, the individualizer was practically limited to generating sequences of approximately 5 or 6 instructions long, depending on the number of inputs and outputs. Even so, it was able to find a diverse set of equivalents for a variety of input sequences. We expect to improve the performance in a future version with additional optimizations to the sequence-generation algorithm and better search-pruning strategies.

4 Applications

Our methodology fits well within the larger context of software individualization as a security mechanism. In particular, superdiversification is useful at the machine-instruction and byte-code levels by offering a systematic means of enumerating all possible customizations. Combined with higher-level methods, such as source-based control-flow transformations, our approach helps to complete a full-fledged diversification solution against attacks enabled by software uniformity and "monocultures." We briefly describe some specific scenarios that benefit from such defenses.

4.1 Signature-Based Attacks

Certain malicious software attempts to patch binaries by scanning for particular byte patterns within executables and replacing these sequences with attack code. For example, patching tools modify DRM and game binaries to eliminate security measures like license checking and binding to a CD/DVD. Other attack tools will even scan anti-virus binaries and other security software to alter and disable their protection, allowing infiltration by rootkits, adware, bot drivers and other malware. By eliminating signatures from customized copies of security-sensitive binaries, our techniques can hinder such attacks, especially when superdiversification is used alongside higher-level individualization.

Both malware and legitimate security software have used certain individualization techniques to defeat signature-based detection [22]. Polymorphic viruses, for example, may use variable code encryption and instruction replacement based on a small library of equivalents, while metamorphic code may transform itself to an intermediate representation (and back to native instructions) for broader diversity [29]. However, no technique is likely to be very strong when used alone, and a comprehensive individualization solution should combine various approaches. In this context, our technique fits well as a means of generating nontrivial code equivalents that appear to require human cleverness.

Our methodology could potentially be used to "normalize" code sections, or transform them into a standardized form to enable easier signature-based searches. However, we would normally apply superdiversification over code blocks

chosen via a secret key. In addition, since we typically iterate the process, the next block to be individualized may overlap with other blocks, potentially including ones that have already been diversified. Other individualization tactics could be used to move code blocks and inject "chaff" code, further complicating any attempts at comparing and matching code sections. Thus, "code normalization" is unlikely to be effective, particularly if we employ a combination of individualizing transforms.

4.2 Patch Obfuscation

A compelling application of our technique is patch obfuscation. As mentioned earlier, in order to make reverse engineering of binaries harder, software providers employ a variety of code-transformation techniques. This can make the code difficult to reverse-engineer in its entirety, but may have little effect on the difficulty of comparing two similar binaries. When a security patch is released, tools such as BinDiff [23] and EBDS [13] can quickly discover differences between patched and unpatched versions of software, easily pinpointing vulnerable code fragments addressed by the patch. BinDiff represents a class of graph-based matching tools that diff two binaries via various heuristics on basic blocks and control-flow graphs (CFGs). As an advancement over simple byte-based diffing, such tools are robust to minor or syntactic transformations on control- and data-flow paths. Recent work [6] has even shown that it's possible to automate the process of finding inputs that exercise vulnerabilities in unpatched binaries.

In this context, we claim that our superdiversification method can delay or prevent reverse-engineering by maximizing the differences between patched and unpatched binaries. In conjunction with source-level and other individualization methods, our approach can provide different semantically equivalent patches to vulnerabilities. In addition, a patch update can include replacements for other code fragments that are not related to the vulnerability being fixed. In fact, most of the protection against sophisticated patch diffing can stem from individualizing unrelated parts of the binary, as well as adding inert "chaff" code and temporary corruption of variables. A cracker attempting to reverse-engineer the code using a graph-based matching tool will find many differences between the original and patched code. A determined cracker may be able to narrow this down eventually, but the slowdown in analysis will translate to a slowdown in vulnerability exploitation and improve the utility of the patch. Colluding crackers will have an even harder time at comparing patterns. While it is difficult to make any formal arguments, we believe that this approach can provide a practical solution to the patch-reverse-engineering issue. As a bonus, we may even be able to improve the performance of some patched versions if our equivalent code sequences are smaller in size than the original.

As a simple example, consider the following basic block:

```
51                push        ecx
8B 45 08          mov         eax,dword ptr [ebp+8]
8B 48 0C          mov         ecx,dword ptr [eax+0Ch]
```

```
81 F1 73 AE 28 3F  xor         ecx,3F28AE73h
89 4D FC           mov         dword ptr [ebp-4],ecx
8B 45 FC           mov         eax,dword ptr [ebp-4]
33 D2              xor         edx,edx
B9 07 00 00 00     mov         ecx,7
```

Using a sliding code window, we applied our superdiversification method over short sequences in this block. By using different keys and other parameters to vary the search, we can generate many different equivalent blocks. One example is below:

```
51                 push        ecx
FF 75 08           push        dword ptr [ebp+8]
58                 pop         eax
68 73 AE 28 3F     push        3F28AE73h
59                 pop         ecx
33 48 0C           xor         ecx,dword ptr [eax+0Ch]
51                 push        ecx
58                 pop         eax
50                 push        eax
8F 45 FC           pop         dword ptr [ebp-4]
03 D2              add         edx,edx
2B D2              sub         edx,edx
6A 07              push        7
59                 pop         ecx
```

The following is another individualized version of the original block, based on a different secret key:

```
51                 push        ecx
F7 D0              not         eax
FF 75 08           push        dword ptr [ebp+8]
58                 pop         eax
FF 70 0C           push        dword ptr [eax+0Ch]
59                 pop         ecx
81 F1 51 04 00 15  xor         ecx,15000451h
81 F1 22 AA 28 2A  xor         ecx,2A28AA22h
51                 push        ecx
8F 45 FC           pop         dword ptr [ebp-4]
FF 75 FC           push        dword ptr [ebp-4]
58                 pop         eax
42                 inc         edx
F7 D2              not         edx
42                 inc         edx
4A                 dec         edx
33 D2              xor         edx,edx
51                 push        ecx
```

```
59                  pop         ecx
6A 07               push        7
59                  pop         ecx
```

We then patched the original block by changing a few critical instructions:

```
51                  push        ecx
8B 45 08            mov         eax,dword ptr [ebp+8]
8B 48 10            mov         ecx,dword ptr [eax+10h]
81 F1 53 5D 3E 2C   xor         ecx,2C3E5D53h
89 4D FC            mov         dword ptr [ebp-4],ecx
8B 45 FC            mov         eax,dword ptr [ebp-4]
83 C0 64            add         eax,64h
83 C8 01            or          eax,1
33 D2               xor         edx,edx
B9 07 00 00 00      mov         ecx,7
```

It is not difficult to see which instructions were changed by simply comparing the original and the patched block. By applying superdiversification to the patched block, we can again generate many different equivalent blocks with more differences between them and the original than just the patched instructions, making it harder to isolate the patched code. One example of a generated block is shown below:

```
FF 75 08            push        dword ptr [ebp+8]
58                  pop         eax
51                  push        ecx
6A 01               push        1
59                  pop         ecx
83 E1 02            and         ecx,2
F7 D9               neg         ecx
F7 D9               neg         ecx
03 48 10            add         ecx,dword ptr [eax+10h]
41                  inc         ecx
49                  dec         ecx
81 F1 53 5D 3E 2C   xor         ecx,2C3E5D53h
8B C1               mov         eax,ecx
50                  push        eax
58                  pop         eax
50                  push        eax
8F 45 FC            pop         dword ptr [ebp-4]
83 F0 01            xor         eax,1
0B C0               or          eax,eax
83 F0 01            xor         eax,1
3B C0               cmp         eax,eax
3B C0               cmp         eax,eax
83 C0 64            add         eax,64h
```

```
48                dec         eax
40                inc         eax
83 C8 01          or          eax,1
3B C9             cmp         ecx,ecx
3B C9             cmp         ecx,ecx
3B C9             cmp         ecx,ecx
6A 01             push        1
5A                pop         edx
83 E2 02          and         edx,2
6A 07             push        7
59                pop         ecx
```

Below is another individualized version of the above block:

```
51                push        ecx
FF 75 08          push        dword ptr [ebp+8]
58                pop         eax
6A 01             push        1
59                pop         ecx
FF 70 10          push        dword ptr [eax+10h]
59                pop         ecx
81 F1 51 55 14 04 xor         ecx,4145551h
81 F1 02 08 2A 28 xor         ecx,282A0802h
6A 01             push        1
8F 45 FC          pop         dword ptr [ebp-4]
51                push        ecx
8F 45 FC          pop         dword ptr [ebp-4]
FF 75 FC          push        dword ptr [ebp-4]
58                pop         eax
F7 D8             neg         eax
F7 D8             neg         eax
0B C0             or          eax,eax
83 F0 01          xor         eax,1
83 F0 01          xor         eax,1
83 C0 44          add         eax,44h
83 C0 20          add         eax,20h
83 C8 01          or          eax,1
F7 DA             neg         edx
F7 DA             neg         edx
F7 DA             neg         edx
0B D2             or          edx,edx
42                inc         edx
4A                dec         edx
83 CA 01          or          edx,1
42                inc         edx
81 CA 9E 1E 00 00 or          edx,1E9Eh
```

```
81 F2 9E 1E 00 00  xor         edx,1E9Eh
83 E2 03           and         edx,3
6A 07              push        7
59                 pop         ecx
```

Methods Against Graph-Diffing. As part of a comprehensive strategy against patch diffing, superdiversification addresses only one aspect of protection, namely the task of making basic blocks and small code sections appear different to data-based comparisons. We describe two additional techniques that help against graph-oriented diffing as well. These modify the patched binary's CFG by adding nodes and edges:

Code Outlining: Adding Nodes This technique moves sections of code into newly created functions, replacing the sections with calls to the functions. Such a process is used by compilers for certain optimizations, and has also found applications in software protection [17]. In the context of this paper, the new functions introduced by outlining serve as new nodes in the patched binary's CFG. We can iterate this procedure to outline code from already outlined functions, creating new patterns of edges and nodes in the CFG.

Chaff Control Flow: Adding Edges This method injects new control-flow transfers, such as branches, jumps, and calls, into the patched binary's CFG. To avoid interfering with the program's operation, the new transfers may never be executed, but should be protected via opaque predicates [11]. Such predicates themselves insert additional edges into the CFG.

In principle, these two techniques in combination suffice to convert the patched binary's CFG into a more complex CFG of arbitrary structure. The original CFG remains embedded in the new CFG. In a reasonable attack model, a graph-diffing tool may be required to find the original CFG in the new CFG. In the worst case, this reduces to solving the subgraph-isomorphism problem (NP-Complete), which is expensive for large graphs.

For this model to be useful in practice, certain implementation assumptions must be satisfied; for example, the new chaff edges and outlined functions should not be easily discernible. The design needs to consider the particular subgraph-isomorphism instances created for real-life programs, mainly to ensure that these are not easily solvable on average. In addition, both security and performance penalties will depend on the degree of outlining and chaff-edge insertion. Nonetheless, the approach takes initial steps towards a formal solution and avoids the need for ad hoc methodologies.

When used alone, neither superdiversification nor the above graph anti-diffing methodology suffices in general. For example, even if the CFGs of two binaries are very different, it may be possible to match up basic blocks by inspecting their contents. This is where superdiversification comes in: The numbers and types of instructions in basic blocks can be individualized to prevent easy block matching. Thus, superdiversification and graph anti-diffing techniques complement each other to create a more complete solution against patch diffing.

5 Summary and Future Work

We presented an initial version of our code individualizer, which we believe can be a useful tool for binary diversification. Another potential application is steganography, or data hiding via variably individualized code sequences [14]. Our technique also provides one possible practical implementation of code individualization assumed by certain software-protection models [12]. Given short input sequences, the individualizer is able to generate reasonably large numbers of equivalent sequences with a variety of different characteristics. Due to the exponentially large search space, performance of the tool is limited to generating sequences of only a few instructions in length over a small subset of opcodes. Further research into better search strategies and other optimizations (e.g., adaptive pruning based on input-sequence heuristics) will be undertaken to enable longer sequences and a larger set of instructions to be considered.

An important extension for superdiversification is search over arbitrary instruction sets and byte-codes, not just instruction sets of commodity processors. Modern instruction sets, including x86, x64, and MSIL, are geared towards generality and performance, not individualization and obfuscation. On the other hand, superdiversification is free to use arbitrary custom byte codes designed specifically for randomization and protection. We may generate such byte-codes explicitly, allowing them to contain randomized operations (e.g., a single instruction to multiply by 3, rotate right by 2 places, and add 1). The nature of byte-code instructions we allow is heuristically determined to increase the chances of efficient and successful searches. For example, when superobfuscating a particular input function, we may vary the allowed instruction set based on the operations contained in that function. A custom byte-code compiler may transform our superobfuscated functions into mainstream source or native code.

References

1. Anckaert, B., Jakubowski, M., Venkatesan, R.: Proteus: Virtualization for diversified tamper-resistance. In: DRM 2006: Proceedings of the ACM Workshop on Digital Rights Management, pp. 47–58. ACM Press, New York (2006), doi:10.1145/1179509.1179521
2. Anckaert, B., De Sutter, B., De Bosschere, K.: Software piracy prevention through diversity. In: DRM 2004: Proceedings of the 4th ACM Workshop on Digital Rights Management, pp. 63–71. ACM Press, New York (2004)
3. Aucsmith, D.: Tamper resistant software: An implementation. In: Anderson, R. (ed.) IH 1996. LNCS, vol. 1174, pp. 317–333. Springer, Heidelberg (1996)
4. Bansal, S., Aiken, A.: Automatic generation of peephole superoptimizers. In: ASPLOS-XII: Proceedings of the 12th International Xonference on Architectural Support for Programming Languages and Operating Systems, pp. 394–403. ACM Press, New York (2006), doi:10.1145/1168857.1168906
5. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001)

6. Brumley, D., Poosankam, P., Song, D., Zheng, J.: Automatic patch-based exploit generation is possible: Techniques and implications. In: Proceedings of the 2008 IEEE Security and Privacy Symposium (2008)
7. Chen, Y., Venkatesan, R., Cary, M., Pang, R., Sinha, S., Jakubowski, M.H.: Oblivious hashing: A stealthy software integrity verification primitive. In: Information Hiding (2002)
8. Cohen, F.: Operating system protection through program evolution (1992), http://all.net/books/IP/evolve.html
9. Collberg, C., Thomborson, C., Low, D.: A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Science, The University of Auckland, New Zealand (July 1997)
10. Collberg, C., Thomborson, C., Low, D.: Breaking abstractions and unstructuring data structures. In: International Conference on Computer Languages, pp. 28–38 (1998)
11. Collberg, C., Thomborson, C., Low, D.: Manufacturing cheap, resilient, and stealthy opaque constructs. In: Principles of Programming Languages, POPL 1998, pp. 184–196 (1998)
12. Dedic, N., Jakubowski, M.H., Venkatesan, R.: A graph game model for software tamper protection. In: 2007 Information Hiding Workshop (2007)
13. eEye Digital Security. eEye Binary Diffing Suite (2007), http://research.eeye.com
14. El-khalil, R., Keromytis, A.D.: Hydan: Hiding information in program binaries. In: López, J., Qing, S., Okamoto, E. (eds.) ICICS 2004. LNCS, vol. 3269, pp. 187–199. Springer, Heidelberg (2004)
15. Geer, D., Bace, R., Gutmann, P., Pfleeger, C.P., Quarterman, J.S., Schneier, B.: CyberInsecurity: The cost of monopoly–how the dominance of Microsoft's products poses a risk to security (2003), http://www.ccianet.org/paperscyberinsecurity.pdf
16. Goldwasser, S., Kalai, Y.T.: On the impossibility of obfuscation with auxiliary input. In: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science (FOCS 2005) (2005)
17. Jacob, M., Jakubowski, M.H., Venkatesan, R.: Towards integral binary execution: Implementing oblivious hashing using overlapped instruction encodings. In: 2007 ACM Multimedia and Security Workshop, Dallas, TX (2007)
18. Jakubowski, M.H., Venkatesan, R.: Protecting digital goods using oblivious checking, US Patent No. 7,080,257, filed on August 30, 2000, granted on July 18 (2006)
19. Joshi, R., Nelson, G., Randall, K.: Denali: a goal-directed superoptimizer. In: PLDI 2002: Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation, pp. 304–314. ACM Press, New York (2002)
20. Lynn, B., Prabhakaran, M., Sahai, A.: Positive results and techniques for obfuscation. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 20–39. Springer, Heidelberg (2004)
21. Massalin, H.: Superoptimizer: A look at the smallest program. In: ASPLOS-II: Proceedings of the Second International Conference on Architectual Support for Programming Languages and Operating Systems, pp. 122–126. IEEE Computer Society Press, Los Alamitos (1987)
22. The Metasploit Project. Metasploit, http://www.metasploit.com
23. SABRE Security and Zynamics. Using SABRE BinDiff for malware analysis (2007), http://www.sabresecurity.com/files/BinDiff_Malware.pdf
24. Tan, G., Chen, Y., Jakubowski, M.H.: Delayed and controlled failures in tamper-resistant software. In: Proceedings of the 2006 Information Hiding Workshop (2006)

25. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Studies in Constructive Mathematics and Mathematical Logic, pp. 115–125 (1968)
26. Princeton University. zChaff, http://www.princeton.edu/~chaff/zchaff.html
27. Wang, C., Hill, J., Knight, J., Davidson, J.: Software tamper resistance: Obstructing static analysis of programs. Technical Report CS-2000-12, University of Virginia (December 2000)
28. Wee, H.: On obfuscating point functions. In: STOC 2005: Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing, pp. 523–532. ACM Press, New York (2005)
29. Wikipedia. Metamorphic code, http://en.wikipedia.org

Detecting Java Theft Based on Static API Trace Birthmark*

Heewan Park, Seokwoo Choi, Hyun-il Lim, and Taisook Han

Division of Computer Science
KAIST
Gwahangno 335, Yuseong-gu, Daejeon 305-701, Republic of Korea
{hwpark,swchoi,hilim,han}@pllab.kaist.ac.kr

Abstract. Software birthmark is the inherent program characteristics that can identify a program. In this paper, we propose a static API trace birthmark to detect Java theft. Because the API traces can reflect the behavior of a program, our birthmark is more resilient than the existing static birthmarks. Because the API traces are extracted by static analysis, they can be applied to library programs which earlier dynamic birthmarks cannot handle properly. We evaluate the proposed birthmark in terms of credibility and resilience. Experimental results show that our birthmark can detect common library modules of two packages while other birthmarks fail to detect.

1 Introduction

Recently many programs are under development in the form of open source projects. Open source programs are allowed to modify or distribute program codes under the certain types of software licenses. The most popular software license is the GNU Public License (GPL). Under the GPL, program codes are freely used, while the software using the original codes should also be under the GPL. Open source softwares such as MySQL have different licenses to support software firms. We call the license types as multi-licenses. Under such licenses the GPL is used for open source projects, and commercial licenses are applied to the commercial softwares of which the source codes are not distributed under the GPL if the license fees are paid.

Several reports such as the IBM and SCO battle, where the SCO claims that IBM illegally used open source codes in proprietary programs, confirm that the violations of software licenses are common. These software thefts cause much damage to open source developers and companies. There are plagiarism detectors such as YAP, JPlag, and MOSS which are commonly used to discover code theft [1][2][3]. The plagiarism detectors target to the source codes of programs. Because most commercial programs distribute binary executables only, binary code analysis techniques are necessary. Software birthmarking techniques can be used to detect code theft of binary executables. A software birthmark is a combination of unique characteristics from a program. For

* This work was supported by the Korea Science and Engineering Foundation(KOSEF) grant funded by the Korea government(MEST) (No. R01-2008-000-11856-0).

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 121–135, 2008.

example, the strings extracted from a binary or binary checksums are candidates of a software birthmark.

To detect code theft, two birthmarks are compared to get the similarity between two binary programs. If the similarity is sufficiently high, we can conclude that one program is a copy of the other. Software watermark is similar to software birthmark in that they both can be used to detect code theft. The difference is that for a software birthmark we extract inherent characteristics from the software itself while for a software watermark we extract pre-embedded fingerprints from the software. Software watermarks provide a certain evidence of code theft by extracted copyright information, while software birthmarks provide the possibility of code theft by similarity between programs. However, in some cases, software birthmarks are better than software watermarks due to the highly restricted computing power and memory size.

Software birthmarks can be classified into static and dynamic. Static birthmarks are extracted from a program itself without execution. Dynamic birthmarks are extracted from observable program behaviors during the execution of a program. Static birthmarks can cover the whole program paths, while dynamic birthmarks depend on the run-time trace of a program. Dynamic birthmarks are known to be more resilient to program transformations such as code obfuscation than static birthmarks.

In this research, we propose a static API trace birthmark for Java programs. The static API trace birthmark is the set of all possible run-time API traces of each Java method. Unlike existing birthmarks, the static API trace birthmark does not simply extract adjacent API sequences but analyzes the control flow of methods and generates the possible run-time API traces. Because the API traces can reflect the behavior of a program, our birthmark is more resilient than other static birthmarks.

We evaluate the proposed birthmark in terms of two criteria: credibility and resilience. The credibility of birthmark is the ability to distinguish different programs. The resilience of birthmark is the ability to resist against program transformations. Experimental results show that our birthmark can detect common library modules of two packages while the other birthmarks fail to detect.

2 Related Work

Software birthmarks can be classified into static and dynamic. A static birthmark is extracted from a program itself by static analysis. A dynamic birthmark is extracted from observable run-time behaviors of a program. The advantage of static birthmarks is that they can cover the whole program paths, while dynamic birthmarks can contain only the run-time trace of a program. One serious problem with dynamic birthmarks is that they vary depending on the inputs and the run-time environments. Dynamic birthmarks are known to be more resilient to program transformations such as code obfuscation than static birthmarks.

H. Tamada et al. suggested a static birthmark for Java [4]. Their birthmark is defined with the combination of four features: constant values assigned to fields, the sequence of method calls following the order of instructions in the class files, class inheritance hierarchy, and used class information. Their birthmark is resilient to code obfuscations but it cannot compare algorithms in the methods because it only compares externally observable features.

G. Myles et al. suggested the k-gram birthmark for Java [5]. The k-gram birthmark is the k length sequence of bytecode instructions. It is highly credible, but frail to program transformations.

G. Myles et al. suggested the Whole Program Path(WPP)-birthmark for Java [6]. The WPP birthmark records run-time instruction traces and constructs a dynamic control flow graph. Because the WPP birthmark records executed instructions only, it is resilient to some obfuscation transformations such as insertion of opaque predicates [7] that inserts garbage instructions. However, because the WPP birthmark is dynamic, it can be changed depending on inputs and environments.

D. Schuler et al. suggested a dynamic API birthmark for Java [8]. Their dynamic API birthmark is a set of API call traces per object. The collection of API calls per object makes this birthmark credible than the earlier dynamic API birthmark for Windows [9]. The limitation of Schuler's birthmark is that it can handle only applications that call API frequently.

S. Choi et al. suggested a static API birthmark for Windows [10]. Their static API birthmark is a set of possible API calls which are extracted by statically analyzing disassembled codes. It can be applied only to the Windows API applications.

3 A Static API Trace Birthmark

A software birthmark of a program means the inherent characteristics that can identify the program. A software birthmarking system is the system that provides two functions for the birthmark: extraction and comparison. In this section, we propose a static API trace birthmark and suggest the birthmark extraction method and the comparison method.

3.1 The Definition of the Static API Trace Birthmark

Instead of comparing two control flow graphs directly, we compare two sets of API traces because the traces reflect the run-time behaviors of programs compared to the control flow graph comparison. The static API trace birthmark considers API traces of a method as the essential characteristics of the method. Definition 1 explains an API flow graph. Definition 2 explains a static API trace.

Definition 1. *(API flow graph) Given a control flow graph $G = (V, E, v_s, V_e)$, where v_s is a start node and V_e is a set of exit nodes, $G' = (V', E', v_s, V_e)$ is an API flow graph if G' satisfies following two conditions:*

Condition 1

$$V' = V - \{v \in V | v \text{ does not have API call}\} \cup \{v_s\} \cup V_e,$$

Condition 2

$$E' = \{(v_f, v_t) | \exists \text{a path } v_f v_1 \cdots v_n v_t \wedge v_f \in V' \wedge v_t \in V' \wedge v_1, \cdots, v_n \notin V'\}.$$

Definition 2. *(Static API Trace) Given an API flow graph $G' = (V', E', v_s, V_e)$, a static API trace of a method is a sequence of API calls from v_s to V_e.*

A static API trace may match with a dynamic trace of the method. The static analysis to get static API traces may possibly generate abstract traces which contain all dynamic traces. In previous definition, however, we define the static API trace as a concrete trace rather than as an abstract trace. Although concrete traces cannot cover all possible dynamic traces, a comparison of two sets of concrete traces gives us sufficient accuracy because we are not comparing concrete traces with dynamic traces.

Definition 3. *(Edge Covering Static API Trace Set) An edge covering static API trace set of a method is a set of static API traces where the union of the static API traces covers all the edges of the API flow graph of the method.*

To incorporate all possible execution traces, our birthmark requires edge covering.

Definition 4. *(Static API Trace Birthmark) The static API trace birthmark is the minimum size edge covering static API trace set.*

3.2 Static API Trace Generation

We generate the control flow graphs from Java bytecodes. One peculiarity when extracting birthmarks from Java bytecodes is that Java bytecodes contain subroutine call instructions such as `jsr` and `jsr_w` and exception handling.

To generate a control flow graph for the `jsr` and `jsr_w` instructions, the `ret` instructions should be linked to the next instruction of the `jsr` instruction. Figure 1 shows an example of the tree generation of subroutine instructions `jsr` and `ret`. The `jsr` instruction stores the return address which is next to the executed `jsr` instruction. When

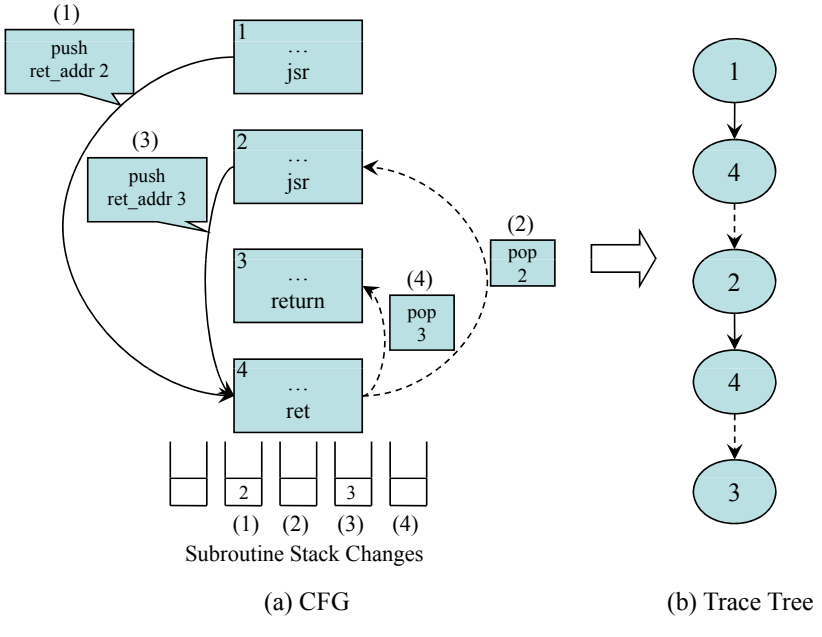


Fig. 1. An example to show the tree generation for `jsr` instruction

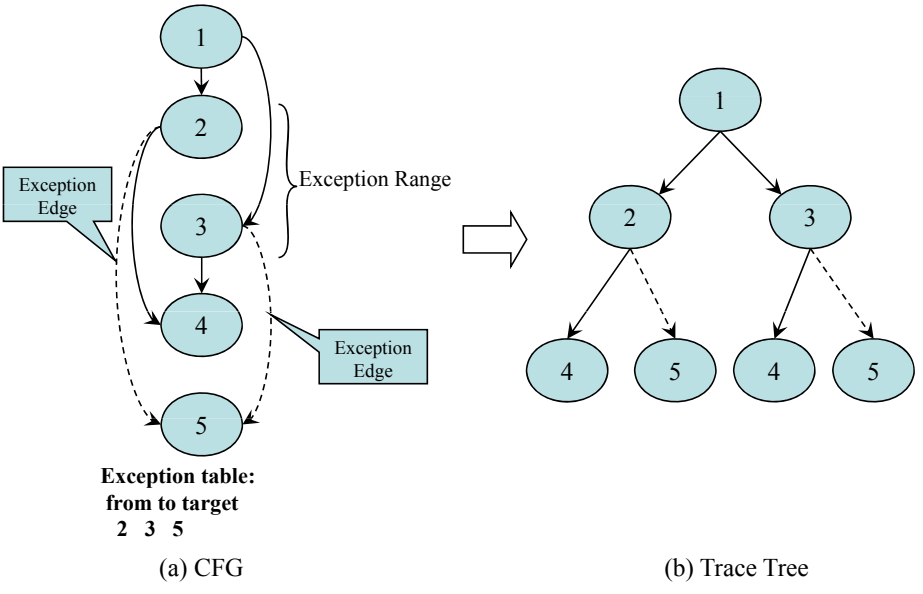


Fig. 2. An example to show the tree generation for exceptions

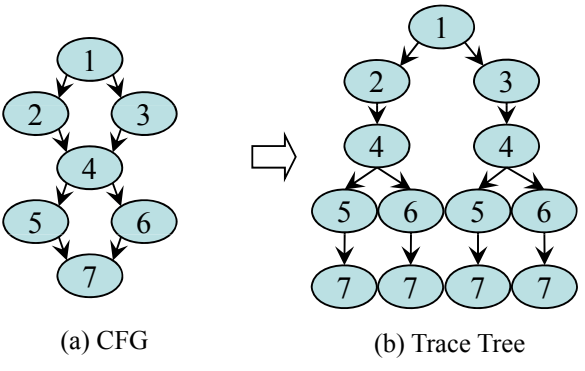


Fig. 3. An example to show the tree generation for branches

a `ret` is executed, the control flow goes to the address stored by the `jsr` instruction. We use a subroutine stack for `jsr` and `ret` instructions to build the correct tree.

For the exception handling, all instructions contained in the exception range should be linked to the exception handler. Because the number of edges generated by exceptions is huge, we add only one edge from the end of each block in the exception range to the corresponding exception handler. Figure 2 explains how a control flow graph is constructed for a exception table in Java bytecodes. Block 2 and block 3 are in the exception range. Two exception edges, which are represented as dotted lines, are linked from the blocks in the exception range to the exception handler block 5.

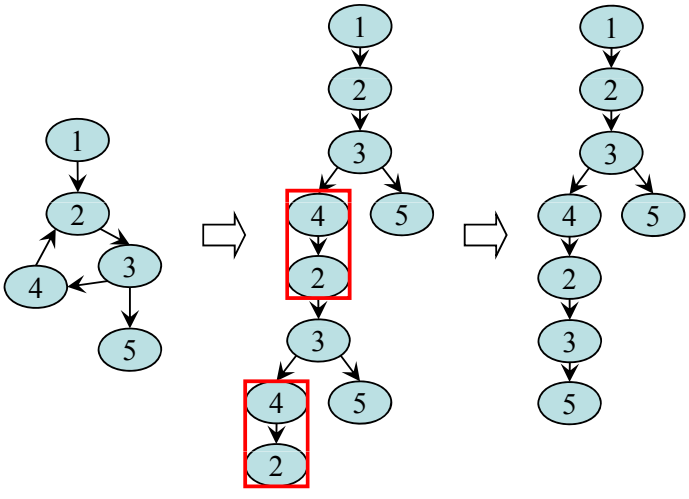


Fig. 4. An example to show the tree generation for loops

After generating the control flow graph, we extract the API flow graph. In order to make the API flow graph, all nodes which have no API calls must be removed, and we add new edges from the sources of incoming edges of the removed nodes to the targets of outgoing edges. After construction of the API flow graph, we can generate API traces. To get all possible API traces, we first construct an edge covering tree. By traversing the edge covering tree we get the edge covering traces. Figure 3 shows how the branches and their descendant nodes are duplicated. Figure 4 shows how to handle loops of API flow graphs. Because a loop can generate infinite API traces during execution, we limit the number of iterations of the loop to one cycle. One iteration cycle is sufficient to generate an edge covering API trace.

Once the API trace tree is fully constructed, traversing the whole tree generates an edge covering static API trace set.

3.3 Similarity Calculation via Semi-global Alignment

To compare two traces extracted from two programs, we utilize sequence alignment algorithms which are frequently used in Bioinformatics to compare DNA sequences. Well known sequence alignment algorithms are global alignment, local alignment, and semi-global alignment algorithms. The global alignment algorithm calculates the similarity using the whole sequences [11]. The local alignment algorithm is used to find maximum subsequence matching [12]. The semi-global alignment algorithm is based on the global alignment algorithm but compensates the score penalty caused by heading and tailing mismatches [13].

We evaluated above three alignment algorithms for our trace birthmark of Java methods. The local alignment algorithm has weakness that the similarity is sensitive to the subsequence of whole traces, which means that the result of local-alignment algorithm

is not credible. The global alignment algorithm has weakness that the similarity is sensitive to the length of the traces; if one sequence is contained in the other sequence and the difference of the lengths is high, the similarity is relatively low. We chose the semi-global alignment algorithm because it can handle the containment problem and it is more credible than the local alignment algorithm.

Definition 5. *(Global Alignment) Let the original trace be T_1, copy trace be T_2, $\sigma(i,j)$ be the match score between $T_1[i]$(i'th element of T_1) and $T_2[j]$(j'th element of T_2), and gap be the penalty score incurred by the insertion of gaps. The optimal global alignment $G(i,j)$ maximizing the alignment score of $T_1[1..i]$ and $T_2[1..j]$ is defined as*

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \sigma(i,j) \\ G(i-1,j) + gap \\ G(i,j-1) + gap. \end{cases}$$

The match, mismatch and gap penalties are

$$\sigma(i,j) = \begin{cases} 1 & \text{if } T_1[i] = T_2[j] \\ -1 & \text{if } T_1[i] \neq T_2[j] \end{cases}$$
$$gap = -1.$$

The credibility of birthmark increases in proportional to the mismatch and gap penalty. To increase the resilience of birthmark, we can decrease the penalty.

Definition 6. *(Semi-global Alignment) Given a global alignment $G(i,j)$, a semi-global alignment $SG(i,j)$ is defined as*

$$SG(i,j) = \max\{G(i,1), G(i,2), \ldots, G(i,j), 0\}$$
$$\text{where } G(0,0) = G(0,1), \ldots, G(0,j) = 0.$$

In contrast to the global alignment, the semi-global alignment does not cut scores by heading and tailing penalties.

Definition 7. *(Trace Similarity) Given traces $T_1[1..i]$ and $T_2[1..j]$, the trace similarity between T_1 and T_2 is defined as*

$$T_{sim}(T_1, T_2) = \max\left(\frac{SG(i,j)}{|T_1|}, \frac{SG(j,i)}{|T_2|}\right).$$

In Definition 7, the trace similarity considers both cases where T_1 is a copy and T_2 is original, and vice versa.

Definition 8. *(Method Similarity) Given sets of traces $TS_1[1 \ldots m]$ in method M and $TS_2[1 \ldots n]$ in method N, the method similarity between M and N is defined as*

$$M_{sim}(M,N) = \max\left(\frac{sum_{row}(M,N)}{sum_{trace}(M)}, \frac{sum_{col}(M,N)}{sum_{trace}(N)}\right)$$

where

$$
\begin{aligned}
sum_{row}(M,N) &= \sum_{i=1}^{m} row_{max}(i),\\
row_{max}(i) &= \max(SG(i,0),\ldots,SG(i,n)),\\
sum_{col}(M,N) &= \sum_{j=1}^{n} col_{max}(j),\\
col_{max}(j) &= \max(SG(0,j),\ldots,SG(m,j)),\\
sum_{trace}(M) &= \sum_{i=1}^{m} |TS_1[i]|,\\
sum_{trace}(N) &= \sum_{j=1}^{n} |TS_2[j]|.
\end{aligned}
$$

Table 1. An example SG matrix computed by the semi-global alignment algorithm

	$TS_2[1]$ $(\lvert TS_2[1]\rvert = 8)$	$TS_2[2]$ $(\lvert TS_2[2]\rvert = 7)$	$TS_2[3]$ $(\lvert TS_2[3]\rvert = 5)$
$TS_1[1]$ $(\lvert TS_1[1]\rvert = 6)$	2	3	4
$TS_1[2]$ $(\lvert TS_1[2]\rvert = 7)$	5	6	1

Table 1 is an example of SG matrix computed by the semi-global alignment algorithm. The $T_{sim}(TS_1[1], TS_2[1])$ is computed by Definition 7 as

$$T_{sim}(TS_1[1], TS_2[1]) = \max(\frac{2}{6}, \frac{2}{8}) = \frac{1}{3}.$$

The $M_{sim}(M,N)$ is computed by Definition 8 as

$$
\begin{aligned}
M_{sim}(M,N) &= \max(\frac{4+6}{6+7}, \frac{5+6+4}{8+7+5})\\
&= \max(\frac{10}{13}, \frac{15}{20}) = \frac{10}{13}.
\end{aligned}
$$

Definition 9. *(Class Similarity) Given two classes C, D and two method sets $M[1\ldots m]$ in C, $N[1\ldots n]$ in D, the class similarity between C and D is defined as*

$$C_{sim}(C,D) = \max(\frac{\sum_{i=1}^{m} map(M[i])}{\sum_{i=1}^{m} sum_{trace}(M[i])}, \frac{\sum_{j=1}^{n} map(N[j])}{\sum_{j=1}^{n} sum_{trace}(N[j])})$$

where

$$map(M[i]) = \max(sum_{row}(M[i], N[1]), \ldots, sum_{row}(M[i], N[n])),$$
$$map(N[j]) = \max(sum_{col}(M[1], N[j]), \ldots, sum_{col}(M[m], N[j])).$$

Definition 10. *(Package Similarity) Given two packages P and Q, two class sets $C[1 \ldots m]$ in P and $D[1 \ldots n]$ in Q, and method sets $M_i[1 \ldots c_i]$ in $C[i]$ and $N_j[1 \ldots d_j]$ in $D[j]$, the package similarity between P and Q is defined as*

$$P_{sim}(P, Q) = \max\left(\frac{\sum_i \sum_k map(M_i[k])}{\sum_i \sum_k sum_{trace}(M_i[k])}, \frac{\sum_j \sum_k map(N_j[k])}{\sum_j \sum_k sum_{trace}(N_j[k])}\right).$$

4 Implementation

Figure 5 shows the architecture of the proposed static API trace birthmarking system. The static API trace birthmarking system computes the package similarity between two packages. To calculate the similarity between two Java packages, we need to calculate similarities of each class in packages. In order to calculate similarity of each class,

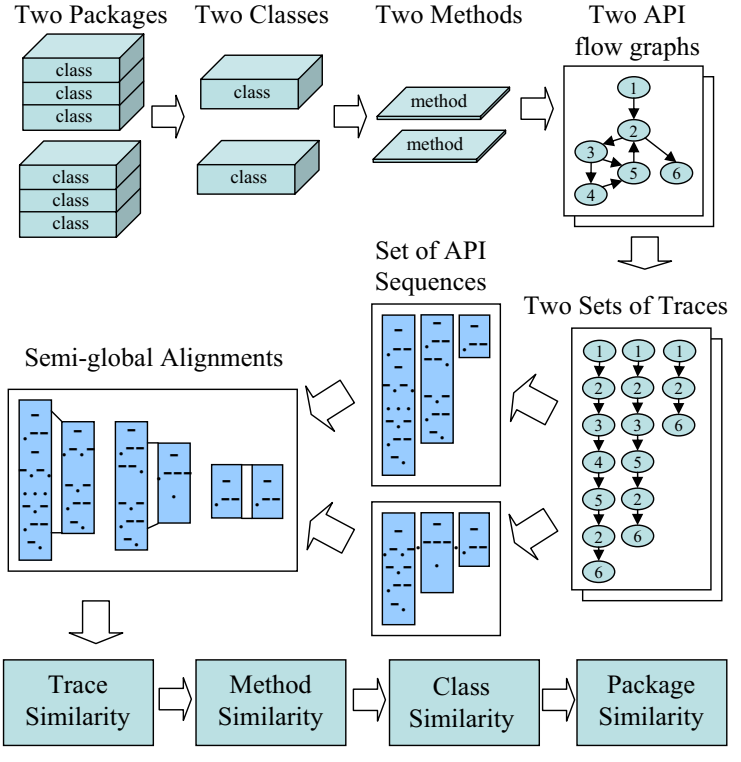


Fig. 5. Static API Trace Birthmarking System

we have to calculate the similarities of each method in classes. To calculate the similarity between two methods, two API flow graphs are generated from two methods. Nodes of the API flow graphs contain the Java API calls. All branch instructions except subroutine calls are removed from the API flow graphs. The edges are constructed by the method explained in Section 3.2. With the API flow graphs, static API trace trees are built. By traversing the trees, static API traces are generated. The API traces contain only standard Java API calls. Finally, the API trace sets are compared to get the similarity between two methods using the semi-global alignment algorithm. Because the matrix is computed using dynamic programming, the time complexity of the semi-global alignment algorithm is $O(m \cdot n)$ where m, n are the sizes of the two API traces. If we have API trace similarities, we can get method similarities, class similarities, and package similarities by Definitions 8, 9, and 10, respectively.

5 Evaluation

We evaluate the proposed birthmark by two criteria of credibility and resilience. To show the credibility and the resilience of our birthmark, we selected the benchmark programs in Table 2. These benchmark programs are well known open source XML processors which are used to evaluate the dynamic Java birthmark [8]. We also compare our birthmark with the existing static Java birthmarks: the Tamada birthmark [4] and the k-gram birthmark [5]. For the experiments, we used our birthmarking system for the static API trace birthmark and a Java birthmark toolkit *Stigmata*[1] for the Tamada birthmark and the k-gram birthmark.

As shown in Table 2, we do not include classes with API call sequences shorter than 3. We consider that such small methods hardly represent inherent characteristics of classes.

The credibility of birthmarks is measured by the similarities among the different programs. The resilience of birthmarks is measured by the similarities between the original programs and the transformed programs. For the code transformation, we used a Java obfuscator *Smokescreen*[2], and Java compilers *javac* and *jikes*[3].

Table 2. Benchmark programs

	Version	Size of Original Classes (bytes)	Number of Excluding Classes	Number of Including Classes	Size of Including Package (bytes)	Size Ratio of Including Package / Original Package
Aelfred	7.0	60,608	5	2	55,036	0.91
Crimson	1.1.3	355,230	88	57	274,735	0.77
Piccolo	1.04	323,108	49	38	227,729	0.71
XP	0.5	150,562	77	11	83,683	0.56

[1] Stigmata: Java birthmark toolkit. Available at http://stigmata.sourceforge.jp/

[2] Smokescreen Java Obfuscator. Available at http://www.leesw.com/smokescreen/

[3] Jikes Java Compiler. Available at http://jikes.sourceforge.net/

5.1 Resilience

Tables 3 and 4 show the similarities of three birthmarks between original programs and obfuscated programs. The benchmark programs are transformed with *Smokescreen* and *Jikes*, and the resulting packages are compared to the original packages. The average similarities among three birthmarks confirm that our birthmark is more resilient than the others. We inspected the transformed classes to find the reasons.

The first reason is the change of instruction order. We observed that the order of bytecodes is changed by the *SmokeScreen* obfuscator. If the original bytecodes have three load and store instruction pairs, the Smokescreen changes these pairs to three load and three store instructions. Because k-gram birthmark is vulnerable to this kind of modification, the average similarity of k-gram birthmark is lower than the others.

The second reason is the change of branch instruction and branch target. We observed that branch statements are compiled into different but semantically equivalent bytecode instructions by the *Jikes* compiler. For example, `ifgt`, which means greater than, generated by the *javac* compiler is compiled into `ifle` and the order of the following statements is exchanged by the *Jikes* compiler. The sequence of method call of the Tamada birthmark is vulnerable to the program transformations which replace instructions with the semantically equivalent instructions. In addition, Tamada birthmark cannot generate correct API call sequences in cases when the true branch and the false branch are swapped and equivalent instructions such as `ifgt` and `ifle` are used, because the Tamada birthmark follows the physically adjacent API calls rather than the control flow edges. Our static API trace birthmark can handle this type of problem because it compares two branches simultaneously by generating all API traces. Hence, our static API trace birthmark is highly resilient.

Table 3. Similarity of 3 birthmarks to evaluate the resilience to Smokescreen obfuscator

	Tamada	k-gram	API Trace
Aelfred	0.758	0.671	0.980
Crimson	0.689	0.563	0.996
Piccolo	0.733	0.614	0.998
XP	0.720	0.589	1.000
Average	0.725	0.609	0.994

Table 4. Similarity of 3 birthmarks to evaluate the resilience to Jikes compiler

	Tamada	k-gram	API Trace
Aelfred	0.857	0.892	0.975
Crimson	0.839	0.871	0.998
Piccolo	0.832	0.891	0.998
XP	0.920	0.894	0.984
Average	0.862	0.887	0.989

5.2 Credibility

Tables 5, 6, and 7 show the similarities among benchmark programs to evaluate the credibility of our static API trace birthmark.

To be a credible birthmark, the similarity between different programs should be near 0. The similarity averages of the Tamada, k-gram, and our API Trace birthmarks are 0.392, 0.230, and 0.111, respectively. The averages show that our API trace birthmark is more credible than other static Java birthmarks. However, we observed that the similarities between the *Crimson* and the *Piccolo* of three birthmarks are the biggest value among the similarities in each table. We inspected the source codes of the *Crimson* and the *Piccolo*. We found that both programs include `xml.parsers` from the Apache software foundation and `xml.sax` from Megginson Technologies. The high similarities between two programs are due to the common modules.

5.3 Module Theft

Suppose that a malicious developer include other developer's module illicitly in his program. In most cases, he tries to hide the stolen code by code obfuscation. Good birthmarks must detect code theft in this case. We evaluated module theft by comparing obfuscated versions of the *Piccolo* and the *Crimson*.

Table 8 shows the number of matched classes between the obfuscated Crimson and the Piccolo. The last row represents the number of perfectly matched classes between

Table 5. Similarity of Tamada birthmark to evaluate the credibility

	Aelfred	Crimson	Piccolo	XP
Aelfred	1.000	0.281	0.445	0.363
Crimson	-	1.000	**0.575**	0.306
Piccolo	-	-	1.000	0.380
XP	-	-	-	1.000

Table 6. Similarity of k-gram birthmark to evaluate the credibility

	Aelfred	Crimson	Piccolo	XP
Aelfred	1.000	0.261	0.224	0.184
Crimson	-	1.000	**0.456**	0.131
Piccolo	-	-	1.000	0.126
XP	-	-	-	1.000

Table 7. Similarity of our static API Trace birthmark to evaluate the credibility

	Aelfred	Crimson	Piccolo	XP
Aelfred	1.000	0.018	0.013	0.110
Crimson	-	1.000	**0.341**	0.115
Piccolo	-	-	1.000	0.068
XP	-	-	-	1.000

Table 8. The numbers of matched classes between the Piccolo and the obfuscated Crimson

Similarity interval	Number of classes included in each similarity interval					
	Tamada		k-gram		our API Trace	
	Original	Obfuscated	Original	Obfuscated	Original	Obfuscated
$0.0 \leq C_{sim} < 0.1$	0	0	4	10	7	10
$0.1 \leq C_{sim} < 0.2$	0	0	12	12	4	3
$0.2 \leq C_{sim} < 0.3$	11	12	6	1	0	1
$0.3 \leq C_{sim} < 0.4$	4	9	0	1	3	2
$0.4 \leq C_{sim} < 0.5$	7	3	1	3	1	1
$0.5 \leq C_{sim} < 0.6$	0	2	0	7	5	3
$0.6 \leq C_{sim} < 0.7$	2	3	1	2	2	2
$0.7 \leq C_{sim} < 0.8$	2	9	3	1	2	2
$0.8 \leq C_{sim} < 0.9$	2	0	2	1	2	2
$0.9 \leq C_{sim} < 1.0$	1	0	3	0	1	1
C_{sim}**=1.0**	**9**	**0**	**6**	**0**	**11**	**11**

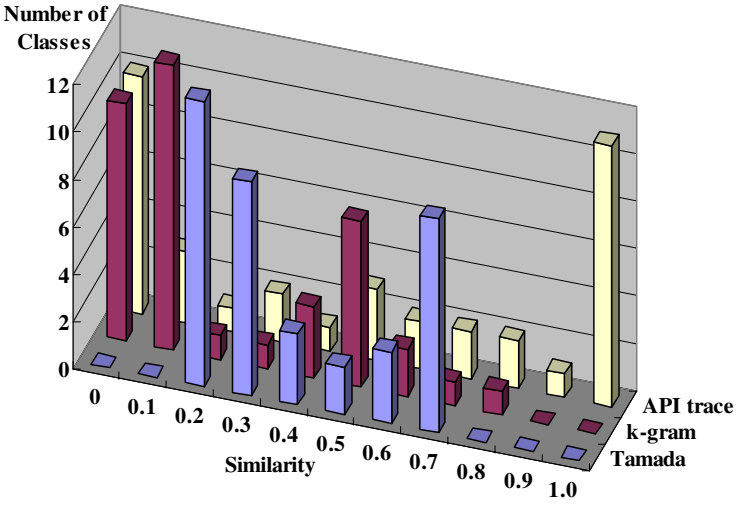


Fig. 6. Number of matched classes after code obfuscation

two programs. The perfect match means that the similarity between two classes is 1.0. Before code obfuscation, the Tamada and the k-gram birthmarks detected 9 and 6 perfectly matched classes, respectively. The Tamada and the k-gram birthmarks detected nothing perfectly matched after obfuscation, while our API trace birthmark detected 11 perfectly matched classes before and after code obfuscation.

We found 11 matched classes out of 16 common classes because two programs included different versions of classes. The *Crimson* includes the `xml.parsers` 1.1 and `xml.sax` 1.1. The *Piccolo* includes the `xml.parsers` 1.2 and `xml.sax` 1.1.1.1. However, all the 16 classes can be detected if we consider that classes are matched when similarities are greater than 0.7.

Figure 6 shows the number of matched classes after code obfuscation. It confirms that our API trace is suitable to detect module theft.

6 Conclusion and Future Work

In this paper, we propose a static API trace birthmark for Java. Because the API traces reflect the behavior of a program, our birthmark is more resilient than the existing static birthmarks. Because the API traces are extracted by static analysis, our method can be applied to library programs that dynamic birthmarks cannot handle properly. We adopted the semi-global algorithm that is widely used for comparing DNA sequences to compare two API traces. We evaluated the proposed birthmark in respect to credibility and resilience for the benchmark programs. The experimental result shows that the resilience of our static API trace birthmark is much higher than the other birthmarks, and our birthmark can detect common library modules of two packages which other birthmarks fail to detect. For future work, we plan to compare our API trace birthmark with the dynamic API birthmark [8]. We also plan to extend the comparison method by weighting API functions.

References

1. Wise, M.: YAP3: improved detection of similarities in computer program and other texts. In: Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, pp. 130–134 (1996)
2. Prechelt, L., Malpohl, G., Philippsen, M.: Finding plagiarisms among a set of programs with JPlag. Journal of Universal Computer Science 8(11), 1016–1038 (2002)
3. Schleimer, S., Wilkerson, D., Aiken, A.: Winnowing: local algorithms for document fingerprinting. In: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp. 76–85 (2003)
4. Tamada, H., Nakamura, M., Monden, A., Matsumoto, K.: Java Birthmarks-Detecting the Software Theft. IEICE Transactions on Information and Systems, 2148–2158 (2005)
5. Myles, G., Collberg, C.: K-gram based software birthmarks. In: Proceedings of the 2005 ACM symposium on Applied computing, pp. 314–318 (2005)
6. Myles, G., Collberg, C.: Detecting software theft via whole program path birthmarks. In: Information Security Conference, pp. 404–415 (2004)
7. Collberg, C., Thomborson, C., Low, D.: Manufacturing cheap, resilient, and stealthy opaque constructs. In: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 184–196 (1998)
8. Schuler, D., Dallmeier, V., Lindig, C.: A Dynamic Birthmark for Java. In: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, pp. 274–283 (2007)
9. Tamada, H., Okamoto, K., Nakamura, M., Monden, A., Matsumoto, K.: Dynamic Software Birthmarks to Detect the Theft of Windows Applications. In: International Symposium on Future Software Technology (ISFST 2004) (2004)
10. Choi, S., Park, H., Lim, H., Han, T.: A Static Birthmark of Binary Executables Based on API Call Structure. In: Cervesato, I. (ed.) ASIAN 2007. LNCS, vol. 4846, pp. 2–16. Springer, Heidelberg (2007)

11. Needleman, S., Wunsch, C.: A general method applicable to search for similarities in amino acid sequence of 2 proteins. Journal of Molecular Biology 48, 443–453 (1970)
12. Temple, F., Michael, S.: Identification of Common Molecular Subsequences. Journal of Molecular Biology 147, 195–197 (1981)
13. Brudno, M., Malde, S., Poliakov, A., Do, C., Couronne, O., Dubchak, I., Batzoglou, S.: Glocal alignment: finding rearrangements during alignment. Bioinformatics 19, 54–62 (2003)

Online Network Forensics for Automatic Repair Validation

Michael E. Locasto[1], Matthew Burnside[2], and Angelos D. Keromytis[2]

[1] Institute for Security Technology Studies, Dartmouth College
[2] Department of Computer Science, Columbia University

Abstract. Automated intrusion prevention and self-healing software are active areas of security systems research. A major hurdle for the widespread deployment of these systems is that many system administrators lack confidence in the quality of the generated fixes. Thus, a key requirement for future self-healing software is that each automatically-generated fix must be validated before deployment. Under the response rates required by self-healing systems, we believe such verification must proceed automatically. We call this process Automatic Repair Validation (ARV). We describe the design and implementation of *Bloodhound*, a system that tags and tracks information between the kernel and the application and correlates symptoms of exploits (such as memory errors) with high-level data (*e.g.*, network flows). By doing so, Bloodhound can replay the flows that triggered the repair process against the newly healed application to help show that the repair is accurate (*i.e.,* it defeats the exploit). We show through experimentation a performance impact of as little as 2.6%.

Keywords: software self-healing, automatic repair validation.

1 Introduction

Recent advances in self-healing software techniques have paved the way for autonomic intrusion reaction, but real-world deployments of such systems have lagged behind research efforts. The limits of detection technology have historically mandated that researchers address the shortcomings of intrusion detection before reaction mechanisms (*i.e.,* self-healing mechanisms) are considered – an attack must be detected before a response can be mounted. In addition, many system administrators are reluctant to allow a defense system to make unsupervised changes to the computing environment, even though (and precisely because) a machine can react much faster than a human.

1.1 Automatic Repair Validation

Automatically generated fixes must be subjected to rigorous testing in an automated fashion. This problem is the essence of Automatic Repair Validation (ARV), a new area of intrusion defense research. ARV encompasses the entire spectrum of an automated response system's functionality: attack detection, repair accuracy, repair precision, and impact on normal behavior:

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 136–151, 2008.

1. **Validation of detection** – The system must verify that the events causing an alert actually produce a compromise. In the case where the sensor is an anomaly detector, the detector's initial classification must be confirmed.
2. **Validation of a repair's accuracy** – The system must test and verify that the repair defeats at least the exploit input that triggered the detection. The rest of the paper discusses the challenges of this process and our solution to it in detail.
3. **Validation of a repair's precision** – The fix must be precise, in the sense that it blocks malicious variants of the original attack. For example, if the fix is an input filter, the system must ensure that the signature generation does not fall prey to the allergy attack [1], whereby a signature generation system is trained to correlate benign input with undesireable symptoms, thereby increasing the rate of false positives.
4. **Validation of a repair's impact on application behavior** – Behavior exhibited by the application after self-healing should be similar to the previous behavior profile of the application. Section 6 briefly discusses an approach for addressing this challenge.

Even though the other problems in this space remain important challenges, we concern ourselves with a repair's accuracy in this paper. Verifying accuracy requires the identification and replay of the attack inputs. Identifying these inputs is challenging, as they may not have been captured correctly (or at all) by the defense instrumentation. The challenge is greater if the input is contained in network traffic — data that most humans find difficult to rapidly analyze by hand. Although the general ARV problem exists for many types of systems, we propose a solution for systems that deal with network traffic. These applications remain popular targets for attack due to the relative anonymity of an attacker and the ease with which input can be sent to the system.

1.2 Correlating Across Abstraction

ARV reveals the tension between the need for abstraction during system design with the need for anti-abstraction during system healing. Since defense system components may not operate at the same layer, the underlying challenge in this problem space is correlating information across layers of abstraction. For example, a detection component may perform binary supervision, but its defense may be the generation of signatures that match network packet content. The accuracy of such signatures is suspect because the data that trips the detector is no longer contained in IP packets; rather, it resides in a memory address and may have been repeatedly transformed before detection. Without appropriate instrumentation, the detector has lost the details of how the data arrived in the memory location. This paper presents a system that contains instrumentation to address this problem.

Abstraction is a powerful tool for system design. The key assumption is that a component should be self-contained: it should not know (and therefore have a hidden dependency on) the implementation details of another component. This assumption greatly eases design cost and increases the flexibility of the system's composition. It no longer

holds when a system must self-heal. Other components must be aware of the details of the malfunctioning component so that they can respond to the failure[1].

Self-healing systems must correlate events and data across layers of abstraction. This correlation should remain flexible: if a component switches to a different implementation, the self-healing infrastructure should adapt to the internal behavior of the new component. We plan to identify a core set of anti-abstraction design patterns in follow-on work to help support this capability.

This paper considers the design, architecture, and implementation of *Bloodhound*, a system for recording and replaying attack input embedded in network flows. The primary challenge involves correlating the form of input that triggers the self-healing instrumentation with the input that originally enters the system. As such, Bloodhound correlates network flows with host-based events by marking an application's internal data structures with the ID of the flow that "taints" that data structure. Bloodhound breaks the traditional abstraction between low-level network data and high-level application data objects by placing sensors at multiple layers in the network stack and application stack. Specifically, Bloodhound runs on the OpenBSD operating system, and it uses a loadable kernel module for a pseudo-device to break the user/kernel-space abstraction.

2 Related Work

ARV is only meaningful if a system protects itself with a self-healing mechanism. Rinard *et al.* [2] have developed compiler extensions that insert code to deal with access to unallocated memory. This technique is leveraged for *failure-oblivious computing*. A related idea is that of *error virtualization* [3], which creates a mapping between the set of errors that could occur during a program's execution and the limited set of errors that are explicitly handled by the program code. The Rx system [4] improves on these approaches by performing only safe perturbations of application state to execute through a fault.

2.1 Signature Generation

Although we could leverage Bloodhound to generate exploit signatures, such a task is not our goal. In addition, recent work [5,6] has called into question the ultimate utility of exploit-based signatures, and Cui *et al.* [7] discuss using binary-level taint-tracking to construct network or filesystem level "data patches" to filter input instances related to a particular vulnerability. Newsome *et al.* suggest generating and distributing vulnerability-specific execution filters [8] based on the identification of a particular control flow path derived from tainted dataflow analysis. Many systems aim at automatically generating signatures of malicious traffic [9,10,11,12]. To generate a signature, most of these systems either examine the content or characteristics of network traffic or instrument the host to identify malicious input.

[1] Note that the problem domain we consider in relation to "self-healing" is *not* the more traditional fault-tolerant environment for distributed applications, where transparent fail-over to replicated components is the norm. In such cases, anti-abstraction serves little purpose.

Other recent work takes a hybrid approach and performs host-type processing on network data. Abstract Payload Execution (APE) [13] identifies network traffic that contains malicious code by treating the content of a packet as machine instructions. Instruction decoding of packets can identify the sequence of instructions in an exploit whose purpose is to guide the program counter to the exploit code. Kruegel *et al.* [14] detect polymorphic worms by learning a control flow graph for the worm binary with similar techniques. *Convergent static analysis* [15] also aims at revealing the control flow of malcode.

DIRA [16] is a compiler extension that adds instrumentation to keep track of memory operations and check the integrity of control flow transfers. It creates a string-based signature for filtering further exploit instances. Liang and Sekar [12] and Xu *et al.* [17] concurrently proposed using address space randomization to drive the detection of memory corruption vulnerabilities and create a signature to block further exploits. These systems operate in a similar fashion to Bloodhound in that they trace back from a memory error to network data. The work of King and Chen [18] utilizes virtual machine logging and replay to step back through checkpoints of a system to identify the ultimate source of a compromise.

Newsome *et al.* propose dynamic taint analysis [19] to detect exploited vulnerabilities. The Vigilante [20] system uses similar analysis for detection and defines an architecture for production and verification of Self-Certifying Alerts (SCAs), a data structure for exchanging information about newly discovered vulnerabilities. The verification step is an example of the form of ARV mentioned in Section 1: Vigilante verifies the control flow path that forms the basis for the alert actually causes an exploit to occur. While Bloodhound uses tainted dataflow analysis, it is not a replacement for such systems. Instead, it augments dataflow analysis systems by considering how tainted data flows through the kernel as well as userspace memory.

2.2 Replaying Traffic

Bloodhound archives all flows consumed by an application and replays only those flows that were related to the exploit in question. Replaying application protocol dialogs is a crucial aspect of an ARV system, and it proves useful in a number of situations (*e.g.,* application or protocol debugging). Traffic can be reproduced in two major ways. First, the raw packets can be recorded and replayed, but this approach may require a large amount of storage and further packet processing. The second approach builds an analytical model of traffic and then generates traffic matching these characteristics.

TCPopera [21] can interactively replay network traffic. It is broadly applicable to problems that require producing large amounts of realistic network data. Roleplayer by Cui *et al.* [22] and the ScriptGen system by Leita *et al.* [23] attempt to reconstruct and replay application-level messages from network flows with little contextual data and a few guiding heuristics.

The Replayer system [24] formalizes the problem of application replay. Replayer describes a sound approach (that is, one not based on heuristics) to generate and issue an input that directs Host Bob to reach the same state as Host Alice (as determined by some post-condition test). Although our replay problem is somewhat different, the Replayer system is the most closely related research effort to Bloodhound, and we found the

syntax of Replayer's formal model of application protocol replay useful to help frame Bloodhound's task.

The most significant difference between Bloodhound and Replayer is that Bloodhound focuses on identifying whether a particular network flow in the database of stored flows contains malicious input. In addition, the underlying problem differs; Replayer is designed to suitably transform a traffic trace or input so that a second host reaches a state equivalent to the first host. Replayer does this by, for example, changing parts of the input to reflect the appropriate hostname or cookie value based on the post-condition constraint. In contrast, Bloodhound leverages taint analysis to identify the set of network packets involved in an exploit. We designed Bloodhound to work with a self-healing system. Since the host is modified via self-healing, the target state explicitly differs from the initial observed state (*i.e.,* the application reaches a new "healed" state rather than the same corrupted state).

One interesting avenue of research would be to combine Replayer and Bloodhound so that stored flows that Bloodhound has selected for replay are consistent with the state of the application and external world. The major challenge with this approach is to ensure that fields critical to the exploit are not changed in a way that interferes with the exploit's efficacy, thus introducing a false negative into the validation process (*i.e.,* the systems believes the exploit was defeated by the self-healing when it was simply broken by the replay engine).

3 Design Space

ARV consists of automatically validating each step in the process toward a repair. We focus our discussion here on how to determine the accuracy of a repair for network applications. Bloodhound's tasks include (a) preferentially recording network traffic, (b) searching through these flows after the application has been healed, and (c) replaying the relevant flows to test this repair. Bloodhound provides evidence that an automated response protects against the input that triggered the self-healing mechanism.

3.1 ARV Replay Definition

ARV replay involves selecting a series of packets to transmit to a modified version of an application to test whether it survives the act of consuming those packets. An ARV replay system, in effect, reprises the role of the attacker. Consider a program P and a set of network flows F. Some subset of those network flows are exploit flows e.

$$\{e_0, e_1, \ldots, e_n\} \subseteq F \tag{1}$$

When P runs and receives input F containing exploit flows, it enters an exploited or error state σ (according to some detection mechanism).

$$\text{Run}(P, F) \rightarrow \sigma \tag{2}$$

After P consumes F and reaches the exploited or error state σ, a self-healing function H operates on P and σ to produce a "healed" program P', optionally replacing σ or other states with correct or healed versions according to the repair strategy in use.

$$H(P, \sigma) \rightarrow P' \tag{3}$$

The ARV accuracy test is to identify the subset $e_0, e_1, \ldots, e_n$ of F and verify that:

$$\text{Run}(P', \{e_0, e_1, \ldots, e_n\}) \nrightarrow \sigma \tag{4}$$

that is, to determine whether the healed version of the application enters an error or exploit state on replay of the attack traffic.

During normal operation of P, (*i.e.,* when it has not entered σ), the system records a database of flows F where each flow f_i consists of incoming and outgoing packets. Each packet of f_i causes changes in user and kernel memory and expresses a particular control flow path in P. Instrumenting memory accesses to observe changes derived from each f_i results in a directed graph D. Each node in D is a memory address, and each edge in D indicates that the source node "taints" the destination node. We label each edge with the instruction responsible for propagating the taint.

While creating D seems fairly straightforward, D — as described — does not contain enough information to support precise traceback. Traceback becomes imprecise when the OS or application reuses a memory location to hold data derived from both malicious and non-malicious flows. At such "common point" memory locations (like a buffer that stores incoming requests), D does not prevent the traceback routine from mistakenly expanding its scope. Common points have a fan-in from multiple flows that becomes a fan-out during traceback.

Table 1. *Forward-Marking Supports Precise Flow Traceback.* The routine adds a node to D, labels the transition, and labels the new (or existing) target node. Traceback iterates over E to collect the malicious flow subsets from the corresponding nodes of D.

```
ROUTINE PROPAGATE(MEM SRC, MEM DST, FLOW F)
    create new edge in D from src to dst
    addFlow(dst, f)

ROUTINE TRACEBACK(EXPLOITMEM E, TAINTGRAPH D)
    foreach memory address x in E
        foreach node y in D
            if x = y
                maliciousFlows ← getFlows(y)
    return maliciousFlows
```

Our solution adopts a simplified traceback approach based on a *forward marking* scheme that labels each node (memory address) in D with the flow ID responsible for the current change in the node's data. Once P enters σ, the detector generates a set of memory addresses E that are involved in the exploit (*e.g.,* an address whose contents enters the instruction register). Forward marking continuously maintains the information necessary to quickly derive $\{e_0, e_1, \ldots, e_n\}$ (the flows responsible for the

exploit) given E. At that point, Bloodhound performs the accuracy test of Equation 4 to determine if P' represents a viable candidate to replace P in production service.

3.2 Traffic Recording

Many options exist for preferentially recording flows. The simplest approach archives all network traffic and replays the entire archive on demand. This approach seems untenable; a system can only store a finite volume of traffic, and replaying or searching an arbitrarily large archive is infeasible in a timing-dependent domain like self-healing software. The flow archive's size must result in a reasonable duration for searching and replaying flows. We consider some heuristics for choosing which flows to store:

- *Save a sliding window of the last* n *days worth of traffic.* This heuristic is simple to implement, and it is simple to tune the size of the archive to optimize storage or replay duration. To test a patch against the archive, simply replay all stored flows. The obvious downside to this heuristic is that it fails against attacks that last longer than, or started before, the n days stored in the archive.
- *Save a probabilistic window.* Rather than using a window with a fixed horizon, probabilistically eject flows from the archive as they age. Aging policies can range from FIFO to LRU to random ejection. Regardless of the aging policy chosen, the self-healing software can only make probabilistic statements about its confidence in a particular patch, based on the likelihood that the entire exploit was contained in the archive.
- *Archive flows flagged by a signature-based misuse detector (e.g., Snort).* This heuristic has the advantage of simplicity, but it is a poor match for a self-healing system that can patch against new or `0`-day attacks. The self-healing system can patch against never-before-seen attacks, but the testing framework would could only replay flows for which a signature already exists.
- *Archive those flows identified by a payload-based anomaly detection system such as Anagram [25].* This heuristic supports the detection of suspect flows that have never been seen before, unlike the previous example. The viability of this heuristic depends entirely on the abilities of the anomaly detection system. If the anomaly detection system has a low false negative rate, then the likelihood that the entire exploit package is archived is very high.
- *Archive flows based on tainted dataflow analysis.* This heuristic focuses on reducing the duration of testing, rather than reducing storage space. As an application and the OS handle each flow, the flow will taint various user and kernel data structures. If each flow is indexed based on the data structures it taints, then during testing only those flows which taint the data structures involved in the patch under question require replay.

These heuristics may also be combined to achieve various points on the trade-off curve between archive size, replay duration, and confidence. While the choices of recording strategies listed above all involve tradeoffs, we choose to employ a solution that does not admit false negatives during detection. That is, during replay, we prefer to replay flows we *know* to be malicious. Thus, Bloodhound uses a form of host-based taint analysis to help identify flows that are involved in a particular exploit.

3.3 Classes of Attack

We classify attacks into several broad categories to make it easier to illustrate the potential complexity of the recording and replay task. The simplest scenario occurs when a single TCP flow or UDP packet encompasses the entire attack (*e.g.,* worms like Slammer or Code Red). The attacker initiates a connection, transmits the exploit code, and then closes the connection. Testing must replay the entire flow. In a slightly more complex version, the attack may be a subset of a larger, innocent flow. Consider an SSH connection where the attacker behaves innocently for several hours and then runs an exploit. We differentiate this attack class from the previous scenario because it may not be desirable to store all of a long-duration flow if only a portion is suspicious.

An attack can be distributed across multiple TCP flows, each of which taken alone appears innocent. Consider a hypothetical attack where one flow corrupts a buffer and a second one delivers the rest of the exploit. To complicate matters, an attack may take advantage of the timing relationship between multiple flows or the timing relationship between packets of a single flow. Attacks of this form exploit race conditions in multi-threaded code, and recreating the circumstances of race conditions is notoriously difficult. At the very least, the timing relationship between the packets or flows must be preserved for testing. This requirement presents difficulties for a *rapid* automatic response, as deployment time is constrained by characteristics of the attack – parameters that the attacker controls. An attacker can potentially distribute exploits from both these scenarios over arbitrarily many flows.

The attack may depend on innocent user action. Consider an exploit where the adversary sends an attack packet, followed by N innocent requests, the combination of which triggers the exploit. Any replay of the flows must include the attack packet *and* all N innocent requests. This class is particularly difficult because, to the untrained eye, the N^{th} packet is highly suspect, while the true attack packet does not necessarily stand out. Worst of all, regardless of the validity of a particular patch, testing against the N^{th} packet almost always succeeds because it is an innocent request.

Finally, an attack may be polymorphic. That is, the algorithm for generating the exploit code may use cryptographic or other heuristics to change the form of the attack over time. As a result, searching a flow or flows for particular bit patterns cannot necessarily identify attacks of this form. In the most pathological case, an attack may consist of any combination of the above attack classes: an attack may be polymorphic, spread across multiple TCP flows, *and* depend on innocent user action. While we do not expect this situation to dominate, it is useful to illustrate the extent of the design space.

4 Bloodhound Implementation

Based on the constraints we present in Section 3, we cover the implementation of Bloodhound's major components, including the structure of our data storage component as well as the implementation of a pseudo-device and its communication protocol with the user space data management framework. We discuss ways to optimize traffic recording through the use of a network content anomaly detector in Section 6.

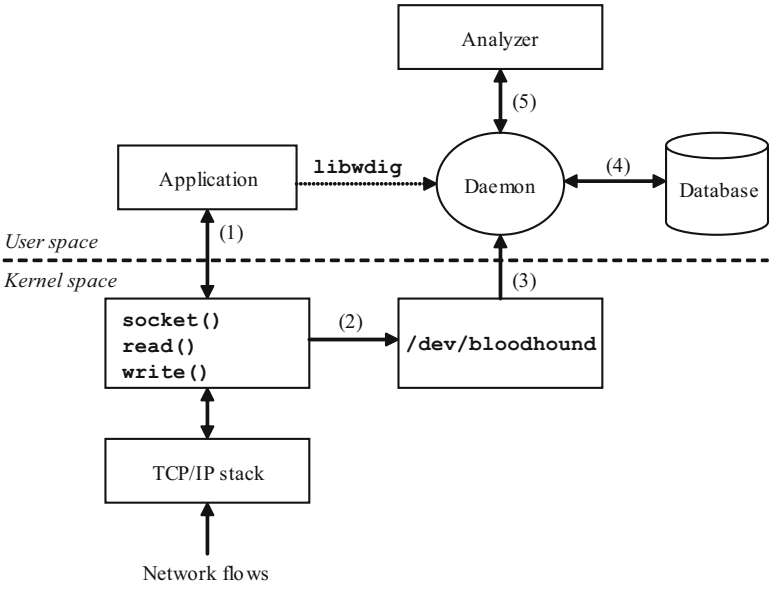


Fig. 1. *Bloodhound Architecture*. In order to track network flows from the operating system to user applications, Bloodhound must break the user space/kernel abstraction. When an application issues a network I/O system call (1), Bloodhound copies the network flow information to a pseudo-device `/dev/bloodhound` (2). A daemon (3) regularly polls the pseudo-device and then stores the network flow data in a database (4). After a self-healing repair has been effected, the analyzer can issue queries to the database (5) to discover which flow caused the exploit. The flow can then be replayed against the newly healed application to test the efficacy of the repair.

4.1 Architecture and Operation

Bloodhound must store flow information and enough structured forensic information to link flows with kernel and application data items, so Bloodhound's core components are distributed across the kernel and user space as shown in Figure 1.

The `/dev/bloodhound` pseudo-device is implemented as an OpenBSD Loadable Kernel Module. The pseudo-device supports an `ioctl()` call `BH_PID` which indicates the process ID number of specific process to observe. On loading, the module hooks into the I/O system calls to intercept network data before it is copied to user space. Each flow associated with the observed process is copied to an in-kernel buffer which is read through standard `read()` calls on `/dev/bloodhound`.

Data is read out of `/dev/bloodhound` by a simple daemon process which uses a Berkeley DB 4.5 database to store flows and related information. The database maintains two primary relationships: a mapping from flow IDs to flow objects (a data struture that we define), and a mapping from tainted memory addresses to flow IDs. We created a library (`libwdig`) to provide access to the data store and define our "flow" data type. The library contains the following core procedures:

- **Register Flow** – This procedure creates a message that encodes a flow and sends it via the protocol described below to the daemon. The daemon parses the message and inserts the flow into the database indexed by the flow ID.
- **Register Binding** – This procedure creates a message that contains a memory address and a flow ID. The daemon inserts this mapping into the database.
- **Retrieve Flow** – This procedure provides a physical memory address to the daemon. If the address points to a flow, the daemon returns the associated flow ID.
- **Retrieve Location** – This procedure supplies the daemon with a flow ID. As a result, the daemon traverses the bindings of memory addresses to flow IDs and returns a list of memory addresses that are associated with the given flow ID.

The user space daemon employs `libwdig` to mediate an application's access to the database. User applications can also call `libwdig`'s functions directly to "manually" store trace information. We plan to investigate methods of automatically performing this dataflow analysis using binary rewriting to inject calls to `libwdig` functions.

The daemon serves as the hub of the system. It manages communication with three different components. First, it intercepts any user space applications that use the `libwdig` library directly. The daemon receives requests for flows or memory binding insertions into the database and invokes the appropriate database functions to handle the request. Second, the daemon communicates with the kernel through the `bloodhound` device. The daemon periodically polls the bloodhound device for messages. The daemon retrieves the messages and checks to see whether the flow IDs associated with the messages have been entered in the database. If they have not, it registers new flows in the database with the new flow IDs. Finally, the daemon helps the Analyzer perform forensic operations by receiving requests for memory address or flow ID lookups and passing them to the database. The daemon uses a TCP-based protocol to communicate with the Analyzer, as described in Table 2.

Table 2. *Message Structure for Daemon Communication.* A message consists of two control bytes: the first indicates either query `Q` or response `R`, and the second distinguishes between a memory location `L`, a flow `F`, or a flow ID `I`. The rest of the message encodes a serialized representation of the memory location or flow.

Direction	Message Format	Meaning
To daemon	`Q, I, flow_id, flow`	Register `flow`
	`Q, L, loc, flow_id`	Register a binding
	`Q, L, loc`	Retrieve `flow_id` at location `loc`
	`Q, I, flow_id`	Retrieve `flow_id`
To client	`R, F, flow`	Response to retrieve `flow`
	`R, I, flow_id`	Response to register `flow ID`

5 Evaluation

We evaluate Bloodhound by examining three aspects of the system. First, we discover what impact Bloodhound has on the normal, *i.e.,* non-repair-time, performance of an

application by characterizing the slowdown due to system call interposition of network I/O related calls. Second, we employ an end-to-end test of the system to show how Bloodhound's forensic capability works for a synthetic vulnerability. Finally, we consider the utility of a possible optimization that employs either signature or statistical content anomaly approaches to pre-classify (and thus limit) the number of flows that Bloodhound would have to store for any given process.

5.1 Performance

To understand the performance impact of Bloodhound, we instructed the `bloodhound` device to observe a single-process instance of the Apache web server. The web server was running on a Dell PowerEdge 2650 with a 2.8GHz Intel Xeon processor and 1GB of RAM. We used a second, identical, Dell PowerEdge 2650, connected to the first over Gigabit Ethernet to download a collection of files. The collection of files we chose was the Apache web server manual, as distributed with the OpenBSD operating system. It consists of 163 text and graphics files totalling 2.1M of data. The manual was downloaded 25 times with `/dev/bloodhound` inactive, followed by another 25 downloads with `/dev/bloodhound` activated. On average a download with `/dev/bloodhound` inactivate took 4.19s, while a download with `/dev/bloodhound` active took 4.31s, for a general performance impact of 2.6%.

5.2 Efficacy

The purpose of our end-to-end efficacy test is to illustrate how Bloodhound can work back from a memory error or other indication of a detected attack to the network flow that contained the exploit input. Memory errors manifested in signals like SIGSEGV are common symptoms of detected exploits, especially when protection mechanisms like Stackguard, address space layout randomizationm or instruction set randomization are employed. In any case, Bloodhound assumes that such a signal will be raised, and that the Analyzer can obtain a memory address to start working backward from. In future work, we plan to use a binary rewriting tool to track tainted data between application-level data structures.

We test the traceback process for a synthetic vulnerability. Our hypothesis is that we can use Bloodhound's kernel instrumentation to track an attack back from a memory error to the flow causing the error. Our experiment is based on a function containing a stack-based buffer overflow in an echo server. The echo server reads user input into a small buffer and echos it back to the user. An exploit script sends a long string of input characters to the server, causing an overwrite of the stack. Each time the `read` system call is used, it uses `/dev/bloodhound` to notify the database of the contents boundaries of the data transfer.

When an overflow occurs and the function returns, the OpenBSD stack-smash protection is triggered and causes the program to crash, dumping core. A script loads the core and determines the location of the RET value on the stack. This value is used as the search value in the database, identifying the dataset that triggered the overflow.

6 Discussion

Automatically validating a repair is a rich field for future work, especially for techniques to ensure that the behavior of an application after repair matches a profile of behavior known to be "good" or clean of malicious influence. A number of considerations exist in this space that parallel the challenges that Bloodhound faces. In particular, the choice of behavior aspects to record and analyze is a key to balancing the tradeoff between the amount of information retained and the ability to confirm with enough specificity that the system can automatically distinguish between known-good tested behaviors and anomalous or malicious behaviors. We believe that a promising approach would start with the rich set of techniques previously proposed for system call anomaly detection [26,27,28]. Capturing aspects of both data and control flow [29], including library, application, and system calls as well as function return values and arguments [30] seems like it would provide a solid profile with enough information to distinguish between these behaviors.

6.1 Limitations

Bloodhound's implementation can be improved along three lines. First, we do not deal with taint-tracking through the application itself. Such taint-tracking can be accomplished by a programmer making direct calls into our taint-tracking library. We can also explore the combination of our kernel-level taint-tracking with existing binary-level tainted dataflow analysis. Second, we plan to incorporate behavior profiling (as described above) into Bloodhound so that it can verify an application's post-healing profile. We are currently porting our OpenBSD implementation to Linux to support these capabilities.

A third area of future work deals with improving Bloodhound's playback capabilities to handle some of the more advanced classes of attack we list in Section 3.3. For example, while other "innocent" user packets may have set up the attack (*e.g.*, by causing some limit to be exceeded), Bloodhound does not necessarily identify them as involved in the exploit. We defer research on these types of attacks; the goal of our current research and development is to provide an infrastructure — currently absent — for addressing them. We believe, however, that repair validation can proceed in the presence of these types of attacks, and we offer a sketch of one possible way forward: the use of continuous hypothesis testing that proceeds through each level of attack. If the cost of applying the fix to a production system can be kept relatively low, Bloodhound can validate the repair in stages, where each stage assumes that the attack was more sophisiticated and constructs appropriate playback scenarios as needed. This incremental process of generating and validating fixes allows Bloodhound to both quickly validate a fix for a specific version of the exploit and continuously ensure that the fix works against more advanced attack scenarios. The exploration of this type of intrusion defense system seems valuable, and Bloodhound's provides a framework so that researchers can implement more intelligent playback and testing strategies.

One potential criticism of Bloodhound is that it appears to be exploit-specific and therefore does not provide protection that vulnerability-specific systems might. While

Bloodhound focuses on identifying a particular exploit input, its task does not conflict with the goals of vulnerability-specific defense systems [6,7,8,20] and related analysis [5,31]. Instead, Bloodhound can provide these systems with more confidence that the fix is correct and blocks *at the very least* the malicious input that triggered the instrumentation. Future work can use the input traffic as a template to generate other semantically correct instances of the flow so that the fix can be tested against a variety of inputs that exploit the same vulnerability. Systems similar to RolePlayer [22] or Replayer [24] seem well suited to this task. Cui *et al.* [7] illustrate the process of deriving practical signatures of previously unknown vulnerabilities. These "data patches" are generated in part by binary-level taint-tracking and help filter related exploit inputs.

6.2 Further Optimizations

We have performed several preliminary experiments to determine the feasibility of reducing the amount of traffic that ARV systems like Bloodhound need to store. ARV is essentially a problem of search; performing an online search, even of an indexed corpus, can be sped up if the size of the corpus is reduced. A system that only considers packets relevant to the current vulnerability or exploit would prove useful. Bloodhound uses taint propagation to achieve this measure. However, an ARV system can complement this dataflow analysis with packet classification schemes, including payload anomaly detection. We refer the interested reader to our technical report, which has the details [32].

7 Conclusions

Most attacks occur rapidly enough to frustrate manual defense or repair. It appears that defense systems must include some degree of autonomy. Recent advances have led to an emerging interest in self-healing software as a solution to this problem. System owners, however, are understandably reluctant to permit automated changes to their environment and applications in response to attacks. Testing an automatic repair helps raise the confidence level in self-healing systems. One critical part of such testing is the verification that the changes made by the self-healing mechanism actually defeat the original attack or close variations thereof.

This paper identifies the important challenge of Automatic Repair Validation (ARV): using audit information to test the resilience and efficacy of a self-healing repair. We present *Bloodhound,* a system for recording and replaying network flows related to the exercise of a particular vulnerability. The design process reveals a number of challenging problems that the research community needs to address in order to make self-securing systems a reality. Our implementation and experiments illustrate that the problem is surmountable; the performance impact on normal operation due to monitoring seems reasonable, and the system can trace back to the flow at fault. In the future, we plan to deploy Bloodhound in a testbed like DETER to test how well Bloodhound can provide an audit service to other nodes.

Acknowledgments

We encountered the problem of automated verification of repairs when we built a system capable of self-healing network server applications against buffer overflow attacks. We would like to recognize the hard work of the other developers, Stelios Sidiroglou and Gabriela Cretu. In addition, we would like to thank Felix Wu and Seung-Sun Hong for sharing TCPopera with us. Finally, Ke Wang provided us with Anagram and was instrumental in helping us to run it.

This paper reports on work that was supported in part by ARO/DHS contract DA W911NF-04-1-0442, USAF/AFRL contract FA9550-07-1-0527, and NSF Grant 06-27473, with additional support from Intel Corporation. The content of this work is the responsibility of the authors and should not be taken to represent the views or practices of the U.S. Government or its agencies.

References

1. Chung, S.P., Mok, A.K.: Allergy Attack Against Automatic Signature Generation. In: Zamboni, D., Krügel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 61–80. Springer, Heidelberg (2006)
2. Rinard, M., Cadar, C., Dumitran, D., Roy, D., Leu, T., Beebee, J.W.: Enhancing Server Availability and Security Through Failure-Oblivious Computing. In: Proceedings 6^{th} Symposium on Operating Systems Design and Implementation (OSDI) (December 2004)
3. Sidiroglou, S., Locasto, M.E., Boyd, S.W., Keromytis, A.D.: Building a Reactive Immune System for Software Services. In: Proceedings of the USENIX Annual Technical Conference, April 2005, pp. 149–161 (2005)
4. Qin, F., Tucek, J., Sundaresan, J., Zhou, Y.: Rx: Treating Bugs as Allergies – A Safe Method to Survive Software Failures. In: Proceedings of the Symposium on Systems and Operating Systems Principles (SOSP) (2005)
5. Brumley, D., Newsome, J., Song, D., Wang, H., Jha, S.: Towards Automatic Generation of Vulnerability-Based Signatures. In: Proceedings of the IEEE Symposium on Security and Privacy (2006)
6. Wang, H.J., Guo, C., Simon, D.R., Zugenmaier, A.: Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits. In: Proceedings of the ACM SIGCOMM (August 2004)
7. Cui, W., Peinado, M., Wang, H.J., Locasto, M.E.: ShieldGen: Automated Data Patch Generation for Unknown Vulnerabilities with Informed Probing. In: Proceedings of the IEEE Symposium on Security and Privacy (May 2007)
8. Newsome, J., Brumley, D., Song, D.: Vulnerability–Specific Execution Filtering for Exploit Prevention on Commodity Software. In: Proceedings of the 13^{th} Symposium on Network and Distributed System Security (NDSS 2006) (February 2006)
9. Kim, H.A., Karp, B.: Autograph: Toward Automated, Distributed Worm Signature Detection. In: Proceedings of the USENIX Security Conference (2004)
10. Singh, S., Estan, C., Varghese, G., Savage, S.: Automated Worm Fingerprinting. In: Proceedings of Symposium on Operating Systems Design and Implementation (OSDI) (2004)
11. Newsome, J., Karp, B., Song, D.: Polygraph: Automatically Generating Signatures for Polymorphic Worms. In: Proceedings of the IEEE Symposium on Security and Privacy (May 2005)

12. Liang, Z., Sekar, R.: Fast and Automated Generation of Attack Signatures: A Basis for Building Self-Protecting Servers. In: Proceedings of the 12^{th} ACM Conference on Computer and Communications Security (CCS) (November 2005)
13. Toth, T., Kruegel, C.: Accurate Buffer Overflow Detection via Abstract Payload Execution. In: Wespi, A., Vigna, G., Deri, L. (eds.) RAID 2002. LNCS, vol. 2516, pp. 274–291. Springer, Heidelberg (2002)
14. Kruegel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G.: Polymorphic Worm Detection Using Structural Information of Executables. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 207–226. Springer, Heidelberg (2006)
15. Chinchani, R., van den Berg, E.: A Fast Static Analysis Approach to Detect Exploit Code Inside Network Flows. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 284–308. Springer, Heidelberg (2006)
16. Smirnov, A., Chiueh, T.: DIRA: Automatic Detection, Identification, and Repair of Control-Hijacking Attacks. In: Proceedings of the 12^{th} Symposium on Network and Distributed System Security (NDSS) (February 2005)
17. Xu, J., Ning, P., Kil, C., Zhai, Y., Bookholt, C.: Automatic Diagnosis and Response to Memory Corruption Vulnerabilities. In: Proceedings of the 12^{th} ACM Conference on Computer and Communications Security (CCS) (November 2005)
18. King, S.T., Chen, P.M.: Backtracking Intrusions. In: 19^{th} ACM Symposium on Operating Systems Principles (SOSP) (October 2003)
19. Newsome, J., Song, D.: Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In: Proceedings of the 12^{th} Symposium on Network and Distributed System Security (NDSS) (February 2005)
20. Costa, M., Crowcroft, J., Castro, M., Rowstron, A.: Vigilante: End-to-End Containment of Internet Worms. In: Proceedings of the Symposium on Systems and Operating Systems Principles (SOSP) (2005)
21. Hong, S.S., Wu, S.F.: On Interactive Internet Traffic Replay. In: Proceedings of the 8^{th} International Symposium on Recent Advances in Intrusion Detection (RAID), September 2005, pp. 247–264 (2005)
22. Cui, W., Paxson, V., Weaver, N.C., Katz, R.H.: Protocol-Independent Adatpive Replay of Application Dialog. In: Proceedings of the 13^{th} Symposium on Network and Distributed System Security (NDSS 2006) (February 2006)
23. Leita, C., Mermoud, K., Dacier, M.: ScriptGen: an automated script generation tool for honeyd. In: ACSA 2005, 21st Annual Computer Security Applications Conference, Tucson, USA, December 5-9 (2005)
24. Newsome, J., Brumley, D., Franklin, J., Song, D.: Replayer: Automatic Protocol Replay by Binary Analysis. In: Proceedings of the 13^{th} ACM Conference on Computer and Communications Security (CCS), pp. 311–321 (2006)
25. Wang, K., Parekh, J.J., Stolfo, S.J.: ANAGRAM: A Content Anomaly Detector Resistant To Mimicry Attack. In: Zamboni, D., Krügel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 226–248. Springer, Heidelberg (2006)
26. Gao, D., Reiter, M.K., Song, D.: Gray-Box Extraction of Execution Graphs for Anomaly Detection. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2004)
27. Feng, H.H., Kolesnikov, O., Fogla, P., Lee, W., Gong, W.: Anomaly Detection Using Call Stack Information. In: Proceedings of the 2003 IEEE Symposium on Security and Privacy (May 2003)
28. Giffin, J.T., Dagon, D., Jha, S., Lee, W., Miller, B.P.: Environment-Sensitive Intrusion Detection. In: Proceedings of the 8^{th} International Symposium on Recent Advances in Intrusion Detection (RAID) (September 2005)

29. Bhatkar, S., Chaturvedi, A., Sekar, R.: Improving Attack Detection in Host-Based IDS by Learning Properties of System Call Arguments. In: Proceedings of the IEEE Symposium on Security and Privacy (2006)
30. Mutz, D., Valeur, F., Vigna, G., Kruegel, C.: Anomalous System Call Detection. ACM Transactions on Information and System Security 9(1), 61–93 (2006)
31. Crandall, J.R., Su, Z., Wu, S.F., Chong, F.T.: On Deriving Unknown Vulnerabilities from Zero-Day Polymorphic and Metamorphic Worm Exploits. In: Proceedings of the 12^{th} ACM Conference on Computer and Communications Security (CCS) (November 2005)
32. Anonymous: Anonymized. Technical Report

Return Value Predictability Profiles for Self–healing

Michael E. Locasto[1], Angelos Stavrou[2], Gabriela F. Cretu[3], Angelos D. Keromytis[3], and Salvatore J. Stolfo[3]

[1] Institute for Security Technology Studies, Dartmouth College
[2] Department of Computer Science, George Mason University
[3] Department of Computer Science, Columbia University

Abstract. Current embryonic attempts at software self–healing produce mechanisms that are often oblivious to the semantics of the code they supervise. We believe that, in order to help inform runtime repair strategies, such systems require a more detailed analysis of dynamic application behavior. We describe how to profile an application by analyzing all function calls (including library and system) made by a process. We create predictability profiles of the return values of those function calls. Self–healing mechanisms that rely on a transactional approach to repair (that is, rolling back execution to a known safe point in control flow or slicing off the current function sequence) can benefit from these return value predictability profiles. Profiles built for the applications we tested can predict behavior with 97% accuracy given a context window of 15 functions. We also present a survey of the distribution of actual return values for real software as well as a novel way of visualizing both the macro and micro structure of the return value distributions. Our system helps demonstrate the feasibility of combining binary–level behavior profiling with self–healing repairs.

Keywords: behavior profiling, anonamly detection, self-healing.

1 Introduction

The growing sophistication of software attacks has created the need for increasingly finer-grained intrusion detection systems to drive the process of automated response and intrusion prevention. Because such fine-grained mechanisms are currently perceived as too expensive in terms of their performance impact, questions relating to the feasibility and value of such analysis remain unexplored. In particular, it seems that self–healing mechanisms would benefit from a detailed behavior profile.

This paper demonstrates the efficacy and feasibility of building profiles of application behavior at a fine-grained level of detail. We focus on the use of function return values as the main feature of these profiles. We do so because return values can help drive control flow decisions after a self–healing repair. In this paper, we show how to build profiles that contain this information at the binary level — that is, without making changes to the application's source, the OS, or the compiler.

1.1 Observing Program Behavior

A popular approach to observing program behavior utilizes anomaly detection on a profile derived from system call sequences [1, 2, 3, 4, 5]. Relatively little attention has

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 152–166, 2008.

been paid to the question of building profiles — in a non-invasive fashion — at a level of detail that includes the application's *internal* behavior. In contrast, typical system call profiling techniques characterize the application's interaction with the operating system. Because these approaches treat the application as a black box, they are generally susceptible[1] to mimicry attacks [7]. Furthermore, the increasing sophistication of software attacks [8] calls into question the ability to protect an application while remaining at this level of abstraction (*i.e.,* system call interface). Finally, most other previous approaches instrument the application's source code or perform static analysis. In contrast, we constructed a non–invasive return value collector tool using the Pin [9] dynamic binary rewriting framework to gather profile information.

1.2 Self–healing

Various approaches to software self–healing [10, 11, 12, 13] concentrate on a transactional approach in which the current sequence of functions is rolled back to some known safe point in execution [11, 14, 15], and the calculations done by the aborted "transactions" are undone. Such approaches require a profiling mechanism to both guide the selection of "known safe points" and set appropriate state at those points.

For example, the concepts of *error virtualization* [12] and *failure–oblivious computing* [10] are representative of approaches that attempt to execute through faults (*e.g.,* memory corruption due to an exploit) by manufacturing information that helps control subsequent execution. Failure–oblivious computing manufactures values for read operations and silently expands or truncates memory overwrites. In error virtualization, a heuristic helps determine the return value for an aborted function; the hope is that the rest of the software will gracefully handle this manufactured error return value and continue executing, albeit without the influence of the attacker.

Determining these return values employs source code analysis on the return type of the function in question. This approach is somewhat unsatisfactory; it seems as if it should be possible to dynamically and automatically collect enough information to determine appropriate error virtualization values. This paper addresses the problem of how to automatically extract enough information from program execution to accurately characterize program behavior in terms of return values to support self–healing.

Behavior profiling has been used to create policies for detection [16, 17]. In contrast, we suggest using this information to help automatically generate templates for repair policies [18]. In addition, this information can drive the selection of "rescue points" for the ASSURE system [15]. One goal of this paper is to provide systems like SEAD and ASSURE with a profiling mechanism.

1.3 Caveats and Limitations

Binary-level function profiling proves more difficult than may initially be expected. Functions are source level artifacts that have only rough analogues at the machine level. Since a compiler can arbitrarily transform the source–level representation of a function

[1] Gao *et al.* [6] discuss a measure of behavioral distance where sequences of system calls across heterogeneous hosts are correlated to help avoid mimicry attacks.

or signal handling can interrupt control flow, it is difficult to cover all cases of function entry and exit. We rely on Pin [9] to detect these events, although it can fail to do so in the presence of tail recursion or aggressive function inlining by the compiler. Finally, because the profile is dependent on a particular binary, our system must recognize when an older profile is no longer applicable *e.g.,* as a result of a new version of the application being rolled out, or due a patch. We can detect this in several ways, including the modification time of the program image on disk.

1.4 Contributions

Overall, we demonstrate the utility of fine-grained application modeling to support self–healing repairs. Our work differs from related work (Section 2) on anomaly detection and self–healing software in two important respects: (1) the structure and granularity of our profiles, and (2) the focus on repair rather than detection.

We create a new model of program behavior extracted *dynamically* from the execution of the program binary without instrumenting the source code, modifying the compiler, or altering the OS. We condition this model based on a feature set that includes a mixture of parent functions and previous sibling functions. Prior approaches look at the call stack, thus ignoring previous siblings, which have already completed execution and so are no longer part of the call stack. This model can help select appropriate error virtualization values, inform the choice of rescue points, or drive the creation of repair policy templates. In addition, we provide a survey of return values used in real software. Finally, we propose *relative scaled k-means clusters*, a new way to simultaneously visualize both the micro and macro structure of feature-frequency behavior models. Details on our profiling experiments and results can be found in Section 4. Section 5 characterizes the return value content of the profiles.

2 Related Work

Our work provides a mechanism to describe application behavior. Thus, our modeling algorithm draws from a rich literature on host–based anomaly detection schemes. While this area is well–mined, we believe it is worthwhile to revisit previous efforts to validate and potentially improve on them. Most significantly, we focus on the utility of behavior profiles for post-attack repair rather than pre-attack detection.

Anomaly Detection. Host-based anomaly detection is not a new topic. The seminal work of Hofmeyr, Somayaji, and Forrest [3, 19] helped initiate application behavior profiling at the system call level. Feng *et al.* [4] and Bhatkar *et al.* [20] contain good overviews of the literature in this space. Most approaches to host-based intrusion detection perform anomaly detection [2, 5, 16, 21] on sequences of system calls and their arguments [22] because the system call interface represents the services that user–level malcode, once activated, must use to effect persistent state changes and other forms of I/O. System call information is easy to collect; the `strace(1)` and `ltrace(1)` tools for Linux are built to do exactly that. The closest work to our building of behavior profiles is the work by Mutz *et al.* [1] and Feng *et al.* [4]; the most significant differences in our model building is that we employ sibling functions when building profiles,

and we examine the return values (rather than arguments). The most significant overall differences between our current work and the general space of system call AD is that we consider how to use this profile in the process of self–healing repairs.

Profiling for Self–Healing. The key assumption underlying error virtualization [12] is that a mapping can be created between the set of errors that *could* occur during a program's execution and the limited set of errors that are explicitly handled by the existing program code. By virtualizing the errors, an application can continue execution through a fault or exploited vulnerability by nullifying the effects of such a fault or exploit and using a manufactured return value for the function where the fault occurred.

ASSURE [15] attempts to minimize the likelihood of a semantically incorrect response to a fault or attack by identifying *error virtualization rescue points*: program locations that are known (or at least conjectured, according to a behavior profile) to successfully propagate errors and recover execution. The key insight is that a program should respond to malformed input differently than benign input; locations in the code that successfully handle these sorts of anticipated input "faults" are good candidates for recovering to a safe execution flow. We view our behavior profiling as a service provider to ASSURE's rescue point selection; ASSURE provides an input training set and handles the details of "teleporting" a failure to the appropriate rescue point.

3 Profile Structure

We define a profile structure that allows us to predict function return values based on the preceding context [1, 23] (functions that have just finished executing). Our system is a hybrid that captures aspects of both control flow (via the execution context) and portions of the data flow (via function return values). We construct an "execution context" for each function based on the application's behavior in terms of both control (predecessor function calls) and data (return values) flow. This context helps collapse *occurrences* of a function into an *instance* of a function to avoid under-fitting or over-fitting the model.

A behavior profile is a graph of execution history records. Each record contains an identifier, a return value, a set of arguments, and a context. Function names serve as identifiers (although callsite addresses are sometimes substituted). Parent and previous sibling functions compose the context. Argument and return values correspond to the values at function entrance and exit, respectively. The purpose of each item is to help identify an instance of a function. For example, considering every occurrence of `printf()` as the *same* instance reduces our ability to make predictions about its behavior. Likewise, considering all occurrences of `printf()` to be *distinct* instances reduces our ability to make predictions in a reasonable amount of time.

We adopt a mixture of parents and siblings to define a context for two reasons. First, a flat or nil context contains very little information to base a return value prediction on. Second, previous work focuses on the state of a call stack, which consists solely of parent functions. As our results in Section 4 demonstrate, the combination of parents and siblings is a powerful predictor of return values. The window size determines, for each function whose profile is being constructed, the number of functions preceding it in the execution trace that will be used in constructing that profile. Figure 1 provides

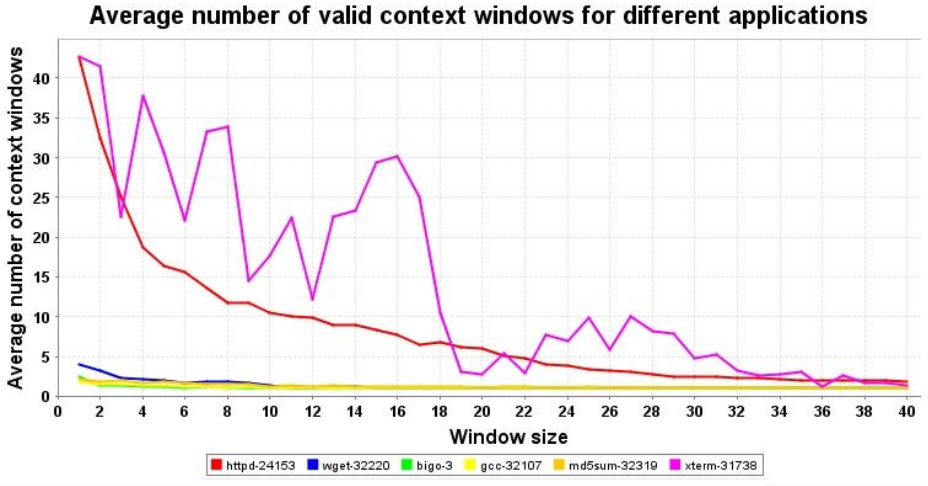


Fig. 1. *Drop in Average Valid Window Context*. This graph shows that the amount of unique execution contexts we need to store to detect changes in control flow decreases as window size increases. `xterm` is a special case because it executes a number of other applications. If we consider the ratio of valid context windows to all possible permutations of functions, then we would see an even sharper decrease.

insight: it shows that the amount of unique execution contexts drops as the window size increases. In contrast, if there were a large amount of valid windows, our detection ability would be diminished.

4 Evaluating Profile Generation

We start by assessing the feasibility of generating profiles that can predict the return values of functions. This section considers how to generate reliable profiles of application execution behavior for both server programs and command line utilities. These profiles are based on the binary call graph features combined with the return values of each function instance. We test and analyze applications that are representative of the software that runs on current server and desktop Unix environments, including: `xterm` (X.Org 6.7.0), `gcc` (GNU v3.4.4), `md5sum` (v5.2.1), `wget` (GNU v1.10.2), the `ssh` client (OpenSSH 3.9p1) and `httpd` (Apache/2.0.53). We also employ some crafted test applications to verify that both the data and the methods used to process them are correct. We include only one of these applications (`bigo`) here because it is relatively small, simple to understand, and can easily be compared against profiles obtained from the other applications. The number of unique functions for all these applications is: xterm, 2111; gcc, 294; md5sum, 239; wget, 846; ssh, 1362; httpd, 1123; and bigo, 129.

Return Value Prediction. Finding a suitable repair for a function, in the context of the self–healing systems we have been discussing, entails examining the range of return values that the function produces. As Section 3 explains, the notion of return value

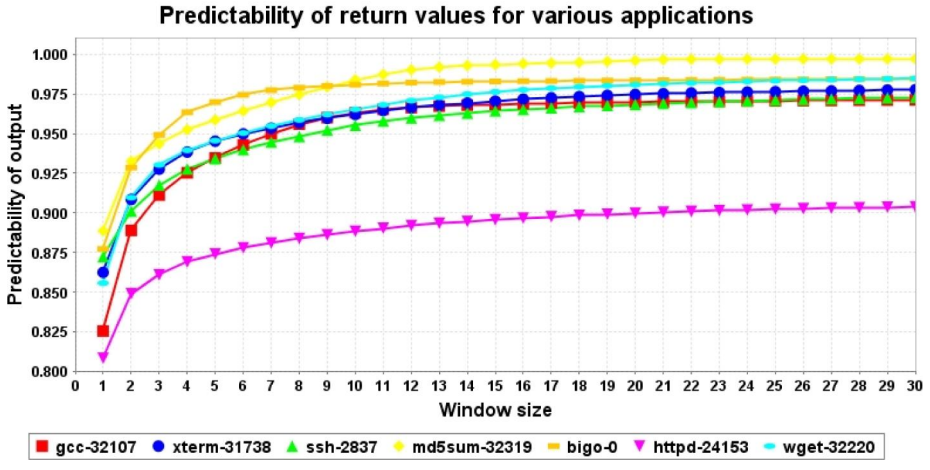


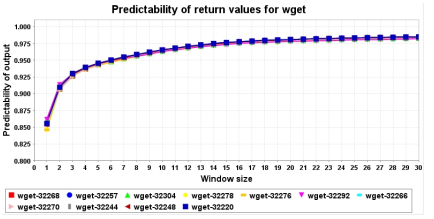
Fig. 2. *Average Predictability of Return Values*. Return value prediction for various applications against a varying context window size. Window sizes of 15 or more achieve an average prediction of 97% or more for all applications other than `httpd` (with a rate of about 90%).

"predictability" is defined as a value from 0..1 for a specific context window size. A predictability value of 1 indicates that we can fully predict the function's return value for a given context window.

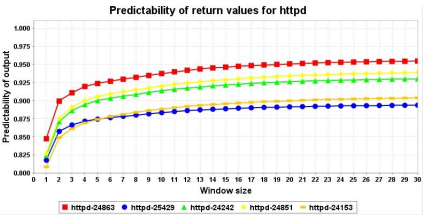
Figure 2 shows the average predictability for the set of examined applications. It presents a snapshot of our ability to predict the return value for various applications when we vary the context window size (*i.e.,* the history of execution). Using window sizes of more than 15 can achieve an average prediction rate of more than 97% for all applications other than `httpd`. For `httpd`, prediction rates are around 90%. This rate is mainly caused by Apache's use of a large number of custom functions that duplicate the behavior of standard library functions. Moreover, Apache is larger and more complex than the other applications and has the potential for more divergent behavior. Of course, this first data set is only a bird's eye view of an overall trend since it is based on the behavior of the average of our ability to predict function return values.

To better understand how our predictions perform, we need to more closely examine the measurements for different runs of the same application. In Figure 3(b) and Figure 3(a) we present results for different runs of `httpd` and `wget`. The `wget` utility was executed using different command line arguments and target sites. Apache was used as a daemon with the default configuration file but exposed to a different set of requests for each of the runs. As we expected, `wget` has similar behavior between different runs: both the function call graph and the generated return values are almost identical. On the other hand, Apache has runs that appear to have small but noticeable differences. As reflected in the average plots, however, all runs still have high predictability.

Some questions remain, including how effective our method is at predicting return values of individual functions. Also, if there are any function that we cannot predict well, how many functions of this type are there, and is there some common feature of these functions that defies prediction? Answering these questions requires measurements for

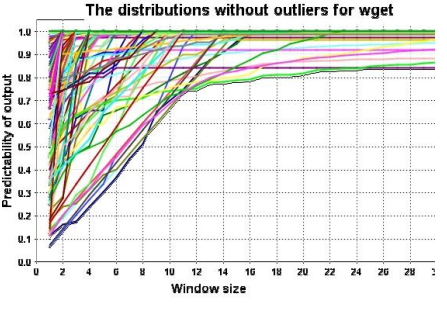


(a) *Average Predictability of Return Values for Different Runs of* `wget`. Although there are 11 runs for `wget`, each individual run is both highly predictable (>98%) and very similar to the others' behavior for different window sizes.

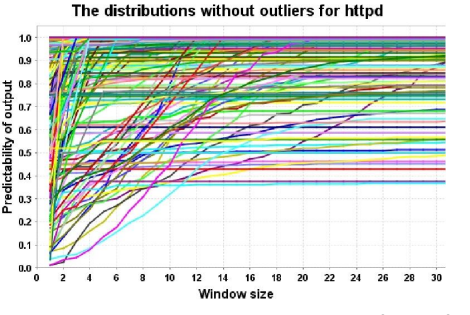


(b) *Average Predictability of Return Values for* `httpd` *and for Different Runs*. Although return value prediction remains high (>90%) for `httpd`, some variations are observable between the different runs. This phenomena is encouraging because it suggests that the profile can be specialized to an application's use at a particular site.

Fig. 3. Return Value Predictability for both `wget` and `httpd` with Different Window Sizes



(a) *Predictability of Return Values for* `wget` *Functions*. Each line represents a function and its predictability evolution as context window size increases. Most functions stabilize after a window size of ten. This graph excludes a small set (Table 1) of outlier functions (functions that are two standard deviations from the average).



(b) *Predictability of Return Values for* `httpd` *Functions*. Each line represents a function and its predictability evolution as context window size increases. As expected, `httpd` has more functions that diverge from the average. It also has more outliers, as shown by Table 1.

Fig. 4. Per-Function Return Value Predictability for both `wget` and `httpd`

individual function predictability. To visually clarify these measurements, in Figures 4(a) and 4(b), we remove functions that have a prediction of two or more standard deviations from the average. The evolution of predictability for `wget` and `httpd` is illustrated in Figure 4(a) and Figure 4(b). This evolution is consistent with what we observe in Figure 2: most functions are predictable — and for small context windows.

A small percentage of the functions, however, produce return values which are highly unpredictable. This situation is completely natural: we cannot expect to predict return values that depend on runtime information such as memory addresses. Additionally, there are some functions that we *expect* to return a non-predictable value: a random

Table 1. *Percentage of Unpredictable (Outlier) Functions*. We illustrate the nature of the overlap in behavior profiles by examining which functions are outliers both within and across programs.

Application	% outliers	% common outliers	% common functions
gcc	5	53	51
md5sum	5	87	84
wget	6	62	56
xterm	6	39	17
ssh	5	35	33
httpd	10	10	28

number generator is a simple example. In practice, as we can deduce from our experiments (see Table 1), the number of such "outlier" functions is rather small in comparison to the total number of well–behaved, predictable functions.

In Table 1, each column represents the percentage (out of all functions in each program) of common or outlier functions. The first column presents the percentage of functions that are outliers *within* a program: that is, functions that deviate from the average profile by more than two standard deviations for all windows of size >10. For each program, there are relatively few "outlier" functions. The second column examines the percentage of common outliers: outliers that appear in two or more applications. We can see that the functions that are unpredictable are consistent and can be accounted for when creating profiles. The third column displays non-outlier functions that are common across applications. These common and predictable functions help show that some aspects of program behavior are consistent across programs.

5 Return Value Characteristics

While Section 4 shows how well we can predict return values, this section focuses on *what* return values actually form part of the execution profile of real software, and how those values are embedded throughout the model structure. We wrote a Pin tool to capture the frequency distribution of return values occurring in a selection of real software, and we created distributions of these values. These distributions provide insight into both the micro and macro structure of a return value behavior profile. Our data visualization technique scales the height and width of each cluster to simultaneously display both the intra- and inter-cluster structure. Our analysis aims to show that return values can reliably classify similar runs of a program as the same program as well as distinguish between execution models of different programs.

Return Value Frequency Models. Our Pin tool intercepts the execution of the monitored process to record each function's return value. The tool builds a table of return value frequencies. After the run of the program completes, we feed this data to MATLAB for a further evaluation that leads to a final return value frequency model. As a proof of concept, a model for a particular monitored process is simple, consisting of the average frequency distribution over multiple runs of the same program. Intuitively, we expect several types of clusters to emerge out of the average frequency distribution. We

Table 2. *Manhattan Distance Within and Between Models.* The diagonal (shown in italics) displays the average distance between each trace and the behavior profile derived from each trace of that program. All other entries display the distance between the execution models for each program. We omit the lower entries because the table is symmetric. Note the difference between gzip and gunzip as well as the similarity of gzip to itself.

	date	**echo**	**gzip**	**gunzip**	**md5sum**	**sha1sum**	**sort**
date	*3.03e+03*	3.72e+03	1.61e+07	1.87e+06	6.46e+04	6.47e+04	5.45e+03
echo	-	*548*	1.61e+07	1.87e+06	6.41e+04	6.42e+04	5.43e+03
gzip	-	-	*212.4*	1.79e+07	1.61e+07	1.61e+07	1.61e+07
gunzip	-	-	-	*1.91e+04*	1.92e+06	1.92e+06	1.87e+06
md5sum	-	-	-	-	*3.03e+04*	3.38e+04	6.56e+04
sha1sum	-	-	-	-	-	*1.67e+04*	6.57e+04
sort	-	-	-	-	-	-	*4.24e+03*

anticipate clusters that contain very high frequency return values, such as -1, 0, and 1 (standard error or success values as well as standard output handles). We expect a larger, more dispersed cluster that records pointer values as well as a cluster containing more "data" values such as ASCII data processed by character or string handling routines.

We examine three hypothesis dealing with the efficacy of execution behavior profiles based on return value frequency:

1. traces of the same program under the same input conditions will be correlated with their model
2. the model of all traces of one program can be distinguished from the model of all traces of another program
3. we can make the structure of the return value frequency models apparent using k-means clustering

For the first hypothesis, we use Manhattan distance as a similarity metric in order to compare each trace of the same process with the return value model of that process. In effect, we compare each return value frequency to the corresponding average frequency among all traces of that program. To evaluate the second hypothesis, we use the Manhattan distance between each process model. The base set for the return values consists of all return values exhibited by all processes that are analyzed over all their runs. Table 2 shows how each model for a variety of program types (we include a variety of programs, like sorting, hashing, simple I/O, and compression) stays consistent with itself under the same input conditions (smaller Manhattan distance) and different from models for each other program (larger Manhattan distance).

Each process has a particular variance with each trace quantified in the similarity value between its model and the trace itself, but when compared against the rest of the processes it can be easily distinguished. We ran each program ten times under the same input profile to collect the traces and generate the model for each program. We used as input profiles generic files/strings that can be easily replicated (in some cases no input was needed): date - N/A, echo - "Hello World!", gzip - *httpd-2.2.8.tar*, gunzip - *httpd-2.2.8.tar.gz*, md5sum *httpd-2.2.8.tar*, sha1sum - *httpd-2.2.8.tar* and sort - *httpd.conf* (the unmodified config file for httpd-2.2.8).

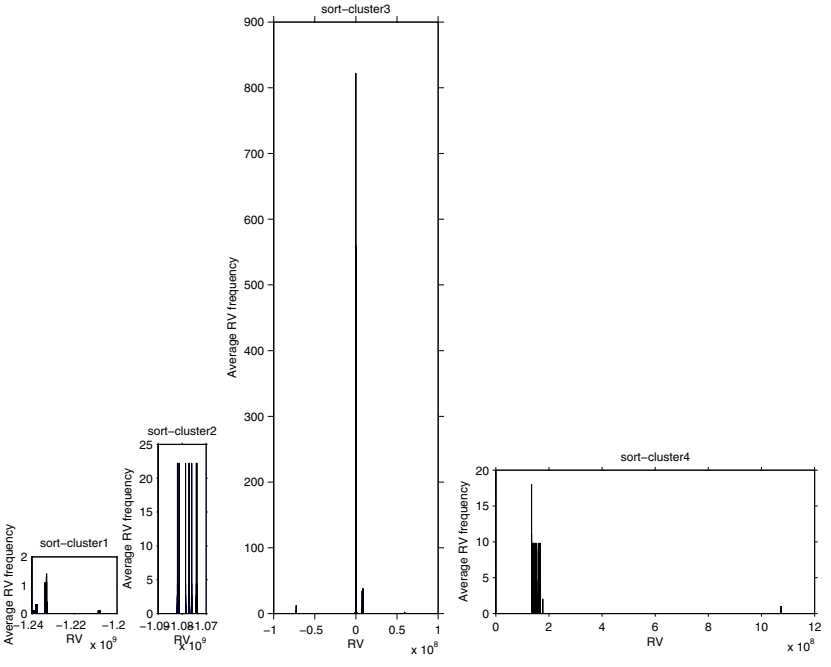
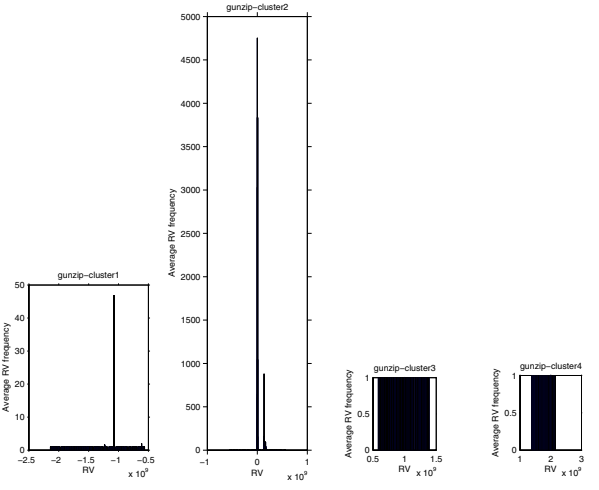
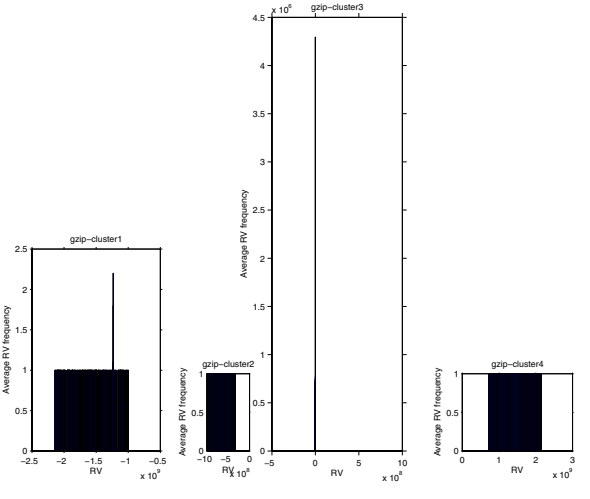


Fig. 5. *Relatively Scaled k-means Clusters for* `sort`. Note how each component of the model is scaled relative to the others while displaying the distribution of similar-frequency values internally; this technique clearly displays the differences between the high frequency return values (cluster 3) and the frequent but more widespread parts of the model (cluster 4) as well as the behavior of values within each component. Both the vertical and horizontal dimensions of each cluster are scaled. We display the clusters in increasing order from left to right determined by the lower end of the horizontal axis range.

Clustering with k-means. Section 4 shows how to build profiles that are useful in predicting return values. The analysis here aims to achieve a better idea of what those actual return values are and how frequently real applications use them. We cluster the return values into frequency classes, and we chose the k-means method to accomplish the clustering. The large disparity in the magnitude of return value frequencies can reduce our ability to convey information about the overall structure of the model if displayed in a simple histogram. Accordingly, we found a new way to simultaneously display both the internal structure of each cluster as well as the external relationships between the clusters. Our clustering method captures the localized view of return value frequency per RV region and our visualization method provides insight into the relative coverage of a particular RV region. Figures 5, 6(a), 6(b), 7(a), 7(b), 8(a), and 8(b) present the clusters obtained for each of the analyzed processes. Each model has a predominant cluster (*e.g.,* cluster2 for data, cluster2 for gzip, *etc.*) which contains the discriminative return value frequencies and has coverage. The remaining clusters contain the lower



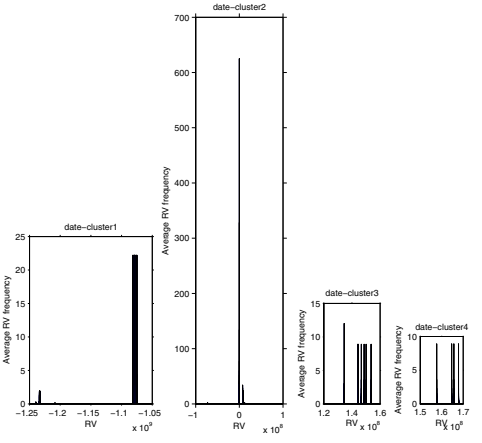
(a) *k-means cluster for* `gunzip` *return values.*



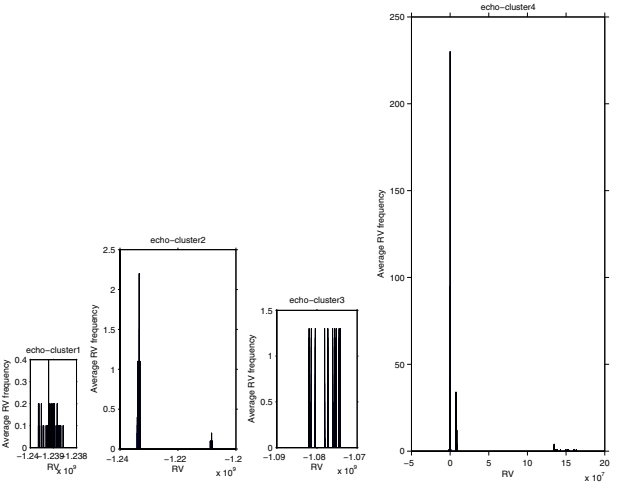
(b) *k-means cluster for* `gzip` *return values.*

Fig. 6. Return Value Frequency Distributions for a Compression Program

frequency return values. We conjecture that by increasing the number of cluster we can achieve better granularity that can distinguish between different types of return values and classify them accordingly.

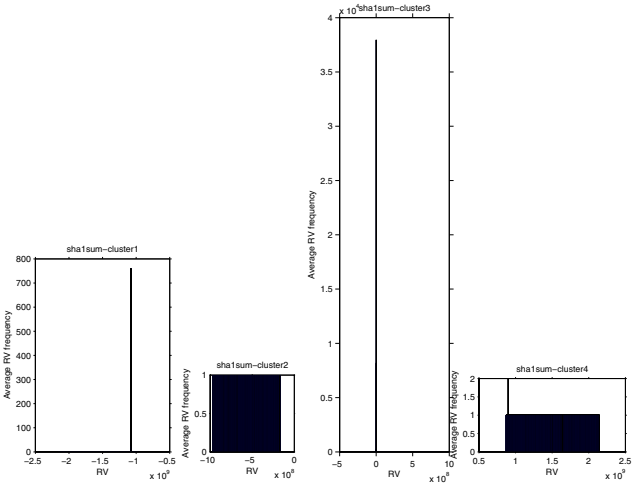


(a) *k-means cluster for* `date` *return values*.

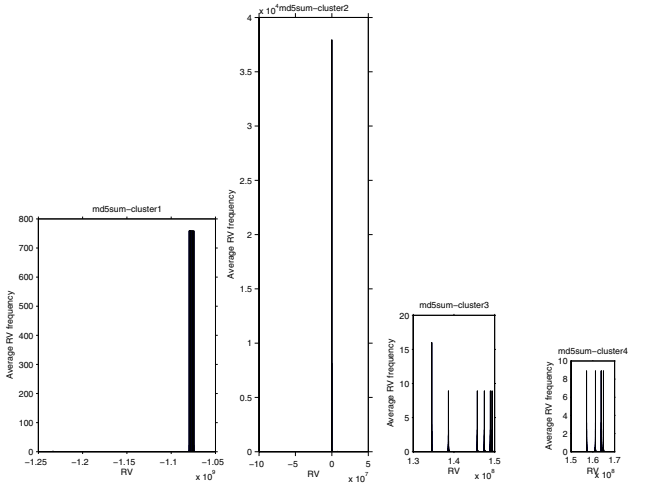


(b) *k-means cluster for* `echo` *return values*.

Fig. 7. Return Value Frequency Distributions for Output Programs



(a) *k-means cluster for* `sha1sum` *return values.*



(b) *k-means cluster for* `md5sum` *return values.*

Fig. 8. Return Value Frequency Distributions for Hash Programs

6 Conclusion

We propose a novel approach to dynamically profiling application execution behavior: modeling the return values of internal functions. Our return value sniffer is available

under the GNU GPL at our website[2]. We show that using a window of return values, including values returned by sibling and parent functions, can make return value prediction as accurate as 97%. We also introduce a novel visualization method for conveying both the micro and macro structure of the return value frequency model components. We intend to investigate models that operate independently of the input profile. We intend to investigate how our behavior profiling mechanism can be used to create repair policy and assist other self–healing systems select an appropriate response.

Acknowledgments

This paper reports on work that was supported in part by ARO/DHS contract DA W911NF-04-1-0442, USAF/AFRL contract FA9550-07-1-0527, and NSF Grant 06-27473. The content of this work is the responsibility of the authors and should not be taken to represent the views or practices of the U.S. Government or its agencies.

References

1. Mutz, D., Robertson, W., Vigna, G., Kemmerer, R.: Exploiting Execution Context for the Detection of Anomalous System Calls. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 1–20. Springer, Heidelberg (2007)
2. Chari, S.N., Cheng, P.C.: BlueBoX: A Policy–driven, Host–Based Intrusion Detection System. In: Proceedings of the 9^{th} Symposium on Network and Distributed Systems Security (NDSS 2002) (2002)
3. Somayaji, A., Forrest, S.: Automated Response Using System-Call Delays. In: Proceedings of the 9^{th} USENIX Security Symposium (August 2000)
4. Feng, H.H., Kolesnikov, O., Fogla, P., Lee, W., Gong, W.: Anomaly Detection Using Call Stack Information. In: Proceedings of the 2003 IEEE Symposium on Security and Privacy (May 2003)
5. Gao, D., Reiter, M.K., Song, D.: Gray-Box Extraction of Execution Graphs for Anomaly Detection. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2004)
6. Gao, D., Reiter, M.K., Song, D.: Behavioral Distance for Intrusion Detection. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 63–81. Springer, Heidelberg (2006)
7. Wagner, D., Soto, P.: Mimicry Attacks on Host-Based Intrusion Detection Systems. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (November 2002)
8. Chen, S., Xu, J., Sezer, E.C., Gauriar, P., Iyer, R.K.: Non-Control-Data Attacks Are Realistic Threats. In: Proceedings of the 14^{th} USENIX Security Symposium, August 2005, pp. 177–191 (2005)
9. 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 Programming Language Design and Implementation (PLDI) (June 2005)
10. Rinard, M., Cadar, C., Dumitran, D., Roy, D., Leu, T.: Enhancing Server Availability and Security Through Failure-Oblivious Computing. In: Proceedings 6^{th} Symposium on Operating Systems Design and Implementation (OSDI) (December 2004)

[2] `http://www.cs.dartmouth.edu/~locasto/research/rval/`

11. Qin, F., Tucek, J., Sundaresan, J., Zhou, Y.: Rx: Treating Bugs as Allergies – A Safe Method to Survive Software Failures. In: Proceedings of the Symposium on Systems and Operating Systems Principles (SOSP) (2005)
12. Sidiroglou, S., Locasto, M.E., Boyd, S.W., Keromytis, A.D.: Building a Reactive Immune System for Software Services. In: Proceedings of the USENIX Annual Technical Conference, April 2005, pp. 149–161 (2005)
13. Smirnov, A., Chiueh, T.: DIRA: Automatic Detection, Identification, and Repair of Control-Hijacking Attacks. In: Proceedings of the 12^{th} Symposium on Network and Distributed System Security (NDSS) (February 2005)
14. Brown, A., Patterson, D.A.: Rewind, Repair, Replay: Three R's to dependability. In: 10^{th} ACM SIGOPS European Workshop, Saint-Emilion, France (September 2002)
15. Sidiroglou, S., Laadan, O., Keromytis, A.D., Nieh, J.: Using Rescue Points to Navigate Software Recovery (Short Paper). In: Proceedings of the IEEE Symposium on Security and Privacy (May 2007)
16. Provos, N.: Improving Host Security with System Call Policies. In: Proceedings of the 12^{th} USENIX Security Symposium, August 2003, pp. 207–225 (2003)
17. Lam, L.C., Cker Chiueh, T.: Automatic Extraction of Accurate Application-Specific Sandboxing Policy. In: Proceedings of the 7^{th} International Symposium on Recent Advances in Intrusion Detection (September 2004)
18. Locasto, M.E., Stavrou, A., Cretu, G.F., Keromytis, A.D.: From STEM to SEAD: Speculative Execution for Automatic Defense. In: Proceedings of the USENIX Annual Technical Conference, June 2007, pp. 219–232 (2007)
19. Hofmeyr, S.A., Somayaji, A., Forrest, S.: Intrusion Detection System Using Sequences of System Calls. Journal of Computer Security 6(3), 151–180 (1998)
20. Bhatkar, S., Chaturvedi, A., Sekar, R.: Improving Attack Detection in Host-Based IDS by Learning Properties of System Call Arguments. In: Proceedings of the IEEE Symposium on Security and Privacy (2006)
21. Giffin, J.T., Dagon, D., Jha, S., Lee, W., Miller, B.P.: Environment-Sensitive Intrusion Detection. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 185–206. Springer, Heidelberg (2006)
22. Mutz, D., Valeur, F., Vigna, G., Kruegel, C.: Anomalous System Call Detection. ACM Transactions on Information and System Security 9(1), 61–93 (2006)
23. Eskin, E., Lee, W., Stolfo, S.J.: Modeling System Calls for Intrusion Detection with Dynamic Window Sizes. In: Proceedings of DARPA Information Survivabilty Conference and Exposition II (DISCEX II) (June 2001)

Involuntary Information Leakage in Social Network Services*

Ieng-Fat Lam, Kuan-Ta Chen, and Ling-Jyh Chen

Institute of Information Science, Academia Sinica
{iengfat,ktchen,cclljj}@iis.sinica.edu.tw

Abstract. Disclosing personal information in online social network services is a double-edged sword. Information exposure is usually a plus, even a must, if people want to participate in social communities; however, leakage of personal information, especially one's identity, may invite malicious attacks from the real world and cyberspace, such as stalking, reputation slander, personalized spamming and phishing.

Even if people do not reveal their personal information online, others may do so. In this paper, we consider the problem of *involuntary information leakage* in social network services and demonstrate its seriousness with a case study of Wretch, the biggest social network site in Taiwan. Wretch allows users to *annotate* their friends' profiles with a one-line description, from which a friend's private information, such as *real name, age, and school attendance records*, may be inferred without the information owner's knowledge. Our analysis results show that users' efforts to protect their privacy cannot prevent their personal information from being revealed online. In 592,548 effective profiles that we collected, the first name of 72% of the accounts and the full name of 30% of the accounts could be easily inferred by using a number of heuristics. The age of 15% of the account holders and at least one school attended by 42% of the holders could also be inferred. We discuss several potential means of mitigating the identified involuntary information leakage problem.

1 Introduction

Social network services (SNS) represent one of the most important applications of the Internet in recent years, with some SNSs hosting millions of profiles, for example, Myspace, Facebook, Flickr, Orkut, and Yahoo! 360. Such services provide a virtual playground for participants to meet new friends, maintain contact with acquaintances, and share resources with others over the Internet. To let others know about themselves, users normally publish personal information

* This work was supported in part by Taiwan Information Security Center (TWISC), National Science Council under the grants NSC 97-2219-E-001-001 and NSC 97-2219-E-011-006. It was also supported in part by Taiwan E-learning and Digital Archives Programs (TELDAP) sponsored by the National Science Council of Taiwan under NSC Grants: NSC 96-3113-H-001-010, NSC 96-3113-H-001-011 and NSC 96-3113-H-001-012.

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 167–183, 2008.

online, such as their appearance, nationality, school attendance records, work experience, and hobbies. The information not only lets people know more about a person, but also enables others to find the user through web searches. Thus, users are normally encouraged to disclose personal information in order to receive higher exposure (more "eyeball counts" in Internet jargon) in the community.

Disclosing personal information online is a double-edged sword. Information exposure is usually a plus, even a must, if people want to join certain types of social communities; however, leakage of personal information, especially one's identity, may invite malicious attacks from the real world and cyberspace. Many studies have addressed the problems of privacy invasion and security threats raised by information exposure online, e.g., stalking, reputation slander, personalized spamming and phishing. We discuss some of the threats in Section 6.1 and refer interested readers to [1].

Even if people do not reveal their personal information, *others may do so.* We illustrate how this could happen with the following example. Suppose Alice uses the pseudonym `boulder_987` to protect her identity. Bob, a friend of Alice, may reveal Alice's age, occupation, and even her real name during their interaction in the following ways:

- Bob may recommend[1] Alice as "`the best steak chef in Boston`" on his own page, and thereby inadvertently reveal Alice's occupation and the city she lives in.
- Suppose Bob also uploaded a photo taken with Alice to his online albums. He may annotate the photo with "`Bob and Alice Lewis at George's Wedding`" and link the photo to Alice's profile. This action would reveal Alice's full name and the fact that Bob, Alice, and George know each other.

Thus, Bob may unintentionally reveal a great deal of information about Alice without her knowledge. In other words, Alice's efforts to protect her identity may be easily nullified by others' behavior. Moreover, it is difficult to detect occurrences of such leakages due to their *distributed* nature. This problem, which we call *involuntary information leakage*, is becoming a serious threat to privacy because of the popularity of social network services.

In this paper, we investigate the extent of *involuntary information leakage in social network services.* We analyze data gathered from Wretch, the biggest social network site in Taiwan. The data set contains 592,548 effective profiles, the social connections between the profiles, and annotations describing the social connections. To quantify the degree of such leakages, we attempt to infer the real name, age, and school attendance records of each user based on annotations made by friends. Our results show that the first name of 72% of users and the full name of 30% of users can be easily inferred by a number of heuristics. The age of 15% of users and at least one school of 42% of users can also be inferred. The high ratio of information leakage evidences that users tend to annotate their friends by using real names and by describing their offline relationships. Based on our

[1] Many SNSs provide a recommendation/endorsement system in which a user can "recommend" another user to the public.

analysis results, we consider several possible ways of mitigating the identified involuntary information leakage problem.

The remainder of this paper is organized as follows. In Section 2 we provide an overview of earlier studies related to social networks and online privacy. In Section 3, we describe our data collection procedures and examine the demography and levels of self-disclosure. We investigate the leakage of real names in Section 4 and the leakage of age and school attendance records in Section 5. Section 6 considers threats and risks that may occur due to information leakage and potential solutions to the problem. Then, in Section 7 we present our conclusions.

2 Related Work

Online social network services have attracted the attention of both entrepreneurs and researchers in recent years [2]. From an academic perspective, the services provide the research community with an unprecedented opportunity to analyze the structure and properties of online social networks on a large scale.

Ahn et al. analyzed the structure of online social networks and found that they have many similarities with offline social networks [3]. Mislove et al. conducted a large-scale study of snapshot graph structures of online social networks. They compared the structures of online social networks and the Web, and validated online social networks with the structural properties of offline social networks [4]. Kumar et al. analyzed the evolution of two popular online social networks, and identified the high-connectivity core and the star structure of each network [5]. O'Murchu et al. compared and classified different categories of social and business networking communities [6].

User interaction and relationships in social network services have also been investigated by researchers. Boyd analyzed social network services from the perspective of human factors, and found that knowledge of, or trust between, users is not required to establish online relationships [7]. Moreover, it has been shown that online social networks engender much weaker ties between users than their offline counterparts [2].

Based on 4,000 profiles gathered from Facebook.com, Gross et al. analyzed the degree of information disclosure and the subsequent risks. They found that *"personal data is generously provided, and limiting privacy preferences are hardly used"* [2], while Acquisti observed that *"technology alone or awareness alone may not address the heart of the privacy problem"* [8]. The privacy issues raised by social network services present a difficult challenge to both information technology and social science researchers.

3 Data Description

In this section, we begin with an introduction to Wretch, the social network service we studied, and then describe our data collection procedures.

3.1 Wretch

Wretch (`http://www.wretch.cc`) was established in 1999 and acquired by Yahoo! Taiwan in 2006. It is currently the most popular social networking site in Taiwan. At the time of writing (Feb 2008), it hosted about 4 million profiles. Like other social network systems, Wretch provides an array of services, including albums, blogs, a bulletin board system (BBS), video sharing, and a discussion forum. Anyone can freely browse all the profiles on Wretch without an account. Joining the service as a registered member is free. Members can upgrade their service levels with a yearly subscription to obtain a larger storage space and more functionalities.

3.2 Data Collection

To collect users' profiles and information about their social relations, we developed a crawler program to fetch profile pages from Wretch. We began the data crawling with an initial set of accounts. In each round, the crawler fetched an account's profile and its friend list; and HTML processing techniques were employed to extract the desired information. If the friend list contained an account the crawler had not seen before, it was added to the job queue. At the end of each round, the crawler randomly selected an account from the queue, and targeted that account in the next round. The crawler continued fetching users' data until the job queue was empty.

We collected the data in September 2007. In sum, we fetched 766, 972 profiles, which constituted 20% of Wretch's population at the time. As we focus on name leakages caused by friends, profiles with an empty friend list are irrelevant to our study. Therefore, we removed profiles that did not have any outgoing friend connections. This yielded a reduced set of 592, 548 profiles, corresponding to 15% of the population. Our data set is summarized in Table 1.

Table 1. Overview of crawled data

Wretch Data	
Number of users	766, 972 (20%)
Number of Effective users	592, 548 (15%)
Number of Connections	7, 619, 212
Avg Connections per user	11.5

3.3 Self-information Disclosure

Self-disclosure is defined as "*telling others previously unknown knowledge so that it becomes shared knowledge*" [9]. It is normally intended to increase

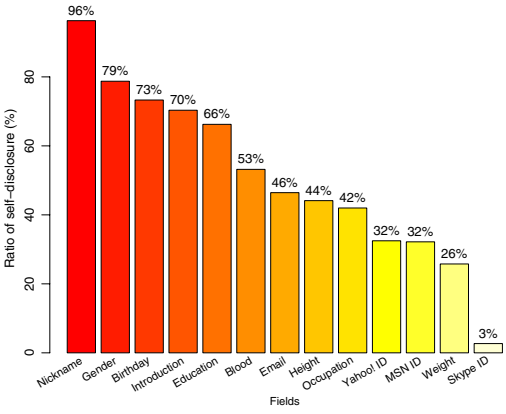


Fig. 1. Ratio of self-disclosure. The gender and birthday fields have a high disclosure ratio.

understanding between people, build trust, strengthen the ties between people, and bind romantic relationships or friendships [10].

To determine the degree that Wretch users reveal information about themselves, we summarize the disclosure ratio of personal statistics in Fig. 1. We observe that most users provide gender and birthday information, and many list their email addresses. In addition, details of instant messaging accounts (e.g., MSN Messenger and Yahoo Messenger) are often given, with a disclosure ratio higher than 30%. This is not surprising as real-time messaging applications are now one of the main communication methods used by young people [11].

We define the *degree of self-disclosure* (DSD) in order to quantify a user's tendency to disclose his/her own personal information. A user's DSD is defined as follows:

$$DSD = \sum_{i=1}^{n} F_i \times W_i, \tag{1}$$

where n is the total number of fields, F_i is a binary value indicating whether field i has been completed, and W_i is the weight of field i. We compute W_i by the following equation:

$$W_i = 1 - \frac{R_i}{\sum_{j=1}^{n} R_j}, \tag{2}$$

where R_i is the *disclosure ratio* of field i. The ratio is computed by dividing the number of users who complete field i by the total number of users. For example, if there are $1,000$ users, and 100 of them provided the weight information, the disclosure ratio of the field "weight" would be $100/1000 = 0.1$. We compute the DSD for all fields, except nicknames, introductions, and instant messaging accounts, as the account fields may not be applicable to every user.

4 Involuntary Name Leakage

On Wretch, a user can provide a free-form text (limited to one line) to annotate a friend's profile, which we call *friend annotations* (or *descriptions*). However, people invent their own ways of using this field. After viewing thousands of annotations, we identified some typical patterns and found that a typical annotation is comprised of three parts: 1) a tag, which is used for *classification*; 2) the *real name or nickname* of the friend to be annotated; and 3) a description of the friend's features or the *relationship* between the two friends. Some examples are "∗Beauty∗ Cathy Brown -- The hottest girl of Nightingale High School" and "[[School Mate]] Tony MY BUDDY." In these ways, a great deal of personal information could be revealed through friend annotations without the information owner's knowledge. (Hereafter, we refer to the information owner as the *annotatee*.)

For example, if we find 5 incoming annotations for a user contain the substring "Jane Garcia" and 10 descriptions contain "Jane," then we may safely guess that the annotatee's real name is Jane Garcia.

4.1 Inference Methodology

To infer the real name of a profile, we first collect all of its incoming annotations, i.e., those that the user's friends compile for him/her. Then, from each description, we extract name candidate tokens from the text as follows:

1. We break the text into disconnected tokens by using several types of delimiters, for example:
   (a) symbols: <SPACE>, <TAB>, #
   (b) punctuation marks: ' " , . ( ) [ ]
2. As most Wretch clients use Chinese, we employ Chinese-specific naming rules to determine whether a delimited token is potentially a Chinese name. A Chinese name is usually composed of two or three characters[2] — a one-character family name, e.g., Chen, Wang, Lin, and a one- or two-character first (given) name, e.g., Xin, Kuan-Ta, Ieng-Fat. Thus, if a token contains two or three characters, it could be a Chinese first name or full name, and we consider it as a real name candidate token.
3. We associate each name candidate token with a *duplication count*. For example, if a name candidate token C_1 appears in annotations from three friends, the duplication count of C_1 will be 2.

We summarize the statistics of friend annotations and extracted name candidate tokens in Table 2. In our data set, we find that, on average, a user has 7 online friends and 6.8 incoming annotations from them.

Although the friend description is not a required field, 96% of the connections are annotated. In addition, 49% of the incoming annotations for a user contain

[2] Some rare Chinese family names are comprised of two characters, e.g., Ouhyoung; thus, a four-character full name is possible. However, to avoid false identification of real names, we only consider two- or three-character names.

Table 2. Summary statistics of friend annotations and extracted name candidate tokens

Friend Annotations and Name Candidates	
Avg In-Degree	7.10
Avg In-Degree with Annot.	6.81 (96%)
Avg In-Degree with Dup. Tokens	3.46 (49%)
Avg # Unique Name Candidates	3.81

at least one name candidate token that appears in other annotations for the same user. For each user, we extract an average of 3.8 unique name candidate tokens, which will serve as our input for real name inference.

Inference of Full Names. Given the extracted name candidate tokens for a user, we apply the following heuristic rules to infer the user's full real name.

1. **Common Family Name:** We consider a token as a full real name if 1) its first character is a common family name (according to the 100 family names listed in [12]); and 2) its duplication count is greater than 1.
2. **First Name as a Substring of the Full Name:** We consider a token C_i as a full real name if 1) there is another token C_j equal to C_i without its first character; and 2) the duplication count of C_i is greater than 1. For example, if C_i is "Wang Ta-Ming," C_j is "Ta-Ming," and C_i appears more than once, then we consider that C_i is probably the full real name of the user.
3. **Common Full Name:** We consider a token as a full real name if it was one of the 574,010 names on the enrollment list for the national college exams for the years 1994 to 2007 [13].
4. **Nickname Decomposition:** In Chinese, it is common for a person to have a nickname that is derived from his/her family or given names. For example, a man called Wang Ta-Ming may have a nickname like "Old Wang," "Bro Wang," "Bro Ta," "Little Ming," or "Bro Ta-Ming." Generally, for a person with a name in the format "FN GN1-GN2," some possible nicknames could be:
   (a) $prefix + X$,
   (b) $prefix + X + X$,
   (c) $X + postfix$,
   where X could be FN, GN1, GN2, or GN1-GN2. We examined the nicknames that appeared in our data set and manually picked 38 common prefixes and postfixes for nickname composition. We consider that a token C_i is a full real name if 1) it contains the predefined nickname prefixes or postfixes as specified; and 2) after removing the corresponding prefix or postfix, the remaining part is the same as other name candidate tokens.
5. **Common Word Removal:** If we do not find any matched name candidate based on the above rules, a name candidate token is considered as a full real name if 1) its duplication count is greater than 1; 2) it does not contain

any nickname-composition prefix or postfix; and 3) it does not contain any general word that people use in daily life, e.g., he, she, friend, lover, classmate, good, or bad. The removal of common words is based on a dictionary containing 100,511 words [14]. If more than one name candidate matches these criteria, then the one with the highest duplication count is deemed the real name of the user.

Inference of First Names. The method used to infer first names is the same as that used for full names, except for the following three points. 1) We use first name candidates (two characters) instead of real name candidates (three characters). 2) Because heuristics 1 and 2 are used specifically for full name inference, we only use heuristics 3, 4, and 5 for the first name. 3) A common first name table (comprising 208,581 names) is used in heuristic 3 instead of the common full name table [12]. Also, because users may include the names of other people, such as "Dolly's sister," we only apply rule 3 to name candidate tokens that have a duplication count greater than 1.

4.2 Inference Results

Here, we summarize the results of the name inference procedures. Table 3 lists the ratio of users whose real names we were able to infer. We successfully inferred first names for 72% of users, and full names for 30% of users. If we count both first names and full names, totally 78% of users are subject to the risk of name leakage. In addition, we detail the inference success ratio of each heuristic rule for real names and first names in Table 4.

Table 3. Ratios of Correctly Inferred Names

Type of name	Ratio of Name Inference
Nickname	60%
Real name	30%
First name	72%
Real name or first name	78%

Table 4. Ratio of Users Whose Names Can be Inferred, by Different Heuristic Rules

Method	Real Name	First Name
Common family name	3%	N/A
Relation of first name	11%	N/A
Common full/first name	9%	57%
Relation of nickname	2%	14%
Removal of common words	27%	69%

4.3 Validation

A complete validation of our name inference results was not possible because we did not know the true real names of the users. Therefore, we employed manual validation. By randomly selecting 1,000 profiles, and examining the real names we inferred for them. We believe that at least 738 of the selected names are correct. The majority of cases of incorrectly inferred names are caused by mistaking a user's nickname for the real name, as we cannot enumerate all possible prefixes and postfixes for nicknames derived from real names. Moreover, our methods cannot distinguish a nickname from a real name if the former is not a literal derivative of the latter.

We acknowledge that our proposed heuristics cannot always derive correct real names. However, exact real name inference is not our primary goal. Instead, we seek to verify the qualitative fact that "*involuntary real name leakage occurs in real-life social network systems, and the degree of leakage is significant.*" In this way, our inference results, though not very accurate, are sufficient to support our conjecture.

4.4 Demographic Analysis

Fig. 2 shows that males are susceptible to higher full name leakage risks than females. However, the first names of males are less likely to be released involuntarily than those of females. This interesting finding implies that people are more likely to use full names to describe a male friend and first names to describe a female friend. Fig. 3 shows that the risk of name leakage is lower for older users. This phenomenon can be simply explained by the number of friends that people have in the social network. Because the 16–25 age group constitutes the core population of Wretch and social connections are mostly between users of similar age, younger people tend to have more online friends than other age groups. Therefore, they have more incoming friend annotations and a higher risk of name leakage than users in other age groups.

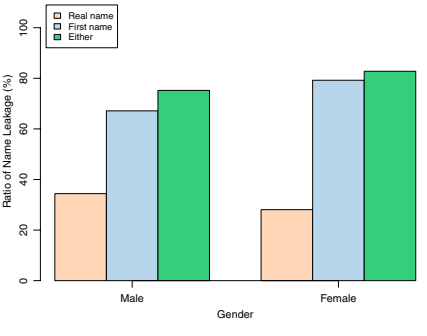


Fig. 2. Ratio of name leakage based on gender

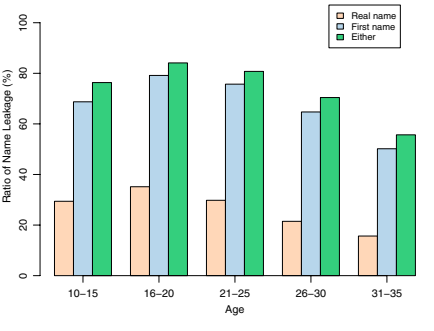


Fig. 3. Ratio of name leakage based on age

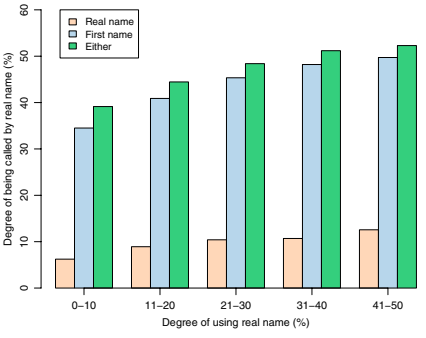


Fig. 4. DUR vs DCR. DCR has a positive relation to DUR

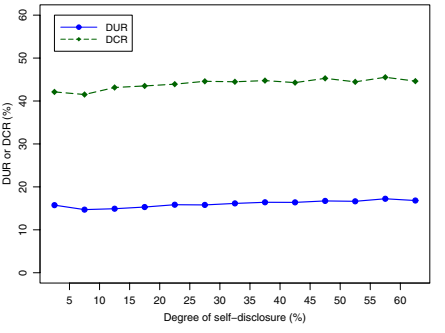


Fig. 5. DCR and DUR distribution over DSD

4.5 Risk Analysis

To confirm that identity leakage is definitely *involuntary*, we checked whether users whose names were inferred correctly had disclosed their real names in their profiles. We found that less than 0.1% of those users had voluntarily revealed their real names (either their full name or first name) in their profiles. This supports our contention that name leakage is caused by annotations made by online friends, rather than self-disclosure.

To determine the source of name leakage, i.e., who reveals the identities of other users, we investigate the usage of real names in friend annotations. We define two metrics to quantify the tendency to use real names when describing online friends:

- **Degree of Using Real Name (DUR):** DUR quantifies a person's tendency to use real names when annotating his/her friends' profiles. It is computed as the ratio of that person's outgoing annotations that contain the annotatee's real name.
- **Degree of Being Called by Real Name (DCR):** DCR quantifies the extent that a user is annotated with his/her real name by online friends. It is computed as the ratio of incoming annotations containing the user's real name.

We first investigate whether real name use behavior is *symmetric*, i.e., are DCR and DUR positively correlated? Fig. 4 suggests that there is a consistent positive correlation between the two metrics. The result indicates that users who receive annotations containing their real names also tend to use real names when sending friend annotations. We also consider the impact of the degree of self-disclosure (DSD) on DCR and DUR, as shown in Fig. 5. Even though both DCR and DUR have a slight positive relationship with DSD, the correlation is insignificant, which indicates that name leakage is not strongly related to self-disclosure. That is, friends may still reveal a user's real name unintentionally no matter how much the user tries to protect his/her identity online.

5 Involuntary Leakage of Age and Education Records

In online social networks, the connections between people are usually a duplicate of their *offline relationships*, which suggests that people have *common attributes*. For example, if the relationship between two people is described as "classmates," they should study in the same school, live in nearby locations, and be a similar age; likewise, the relationship "colleague" implies that two people work in the same organization and probably have similar professions.

If we know two users are classmates in a primary school and one of them disclosed his/her age in the profile, we can infer that the other one is a similar age. To assess the risk of personal information being revealed involuntarily, in the following, we infer the age and education records of users who did not provide this information. In our data set, 56% (331, 827) of users disclosed their ages, but only 10% (59, 255) disclosed their current schools in their profiles. By applying a heuristic inference method, we successfully inferred the ages for 18% of those who did not disclose their ages (15% of the total number of accounts), and inferred at least one school for each of 42% of users in our data set.

5.1 Inference Methodology

Inferring Age. Our inference procedures are executed in a round-based manner. Initially, a job queue is filled with all the accounts that provide age information. In each round, we fetch an account X from the queue, and check all of its incoming and outgoing friend annotations. If we determine that a user Y on X's friend list should be the same age, we set the age field of Y equal to that of X and add Y to the job queue. Our heuristic rule for "same age" is based on the offline relationships of "classmate" or "schoolmate in the same year." We identify such relationships by searching with keywords like "classmate", "class leader", "same class," and "same grade" in friend annotations. The inference procedure continues until the job queue is empty.

Inferring Education Records. The inference methodology for education records is similar to that for age inference except for three points. First, while a person's age is unique, his/her education records will include information about attending several schools. For simplicity, we assume that each user attended at most one school in each of the four education levels, namely, elementary school, junior high school, senior high school, and college. Second, while users are unlikely to specify a friend's age in annotations, they may use a school's name as a category name. Thus, we can infer not only if two users attended the same school, but also the name of a school the annotatee ever enrolled in. Third, in addition to school names and offline relationships like "classmate" and "schoolmate," we can determine if two users ever studied in the same school by keywords like "same school," "same college," or "*some* department."

5.2 Inference Results

Here we summarize the inference results of age and education records. The upper graph of Fig. 6 shows that the success ratio for age inference is higher for

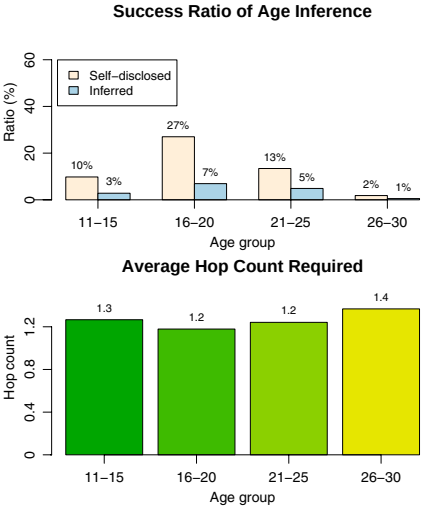


Fig. 6. Inference results for users' ages

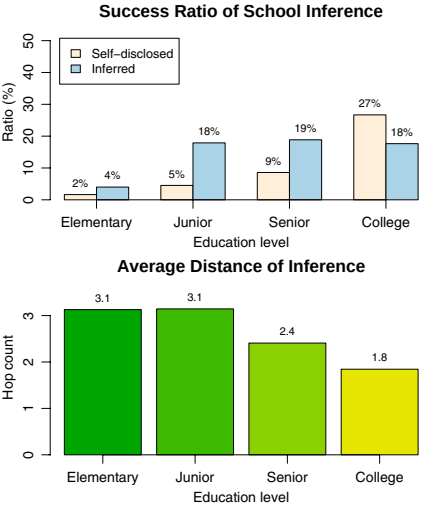


Fig. 7. Inference results for users' education records

users in the 16–25 year age group. In sum, we successfully inferred the ages of 15% of users, which corresponds to 18% of the users who did not provide age information. The lower graph suggests that the average hop count required for inference is about 1.2, which indicates that, in most cases, the inferred age of a user is directly propagated the profile of a user who discloses his/her own age.

Fig. 7 shows the inference results for users' education records. The success ratios for school inference were approximately 20% for all education levels, except elementary school. Totally, we inferred at least one school for 42% of users in the data set. This success ratio is reasonably high, as only 5% and 9% of users self-disclosed their respective junior high and senior high schools. The results suggest that many Wretch users are linked by relationships established in high school. The lower graph in Fig. 7 shows that the average hop count required for inference decreases by education level, which implies that users at higher education levels have stronger connections with their schoolmates online.

5.3 Validation

We apply a cross-validation approach to verify the inferred ages and education records. That is, we verify the inferred ages based on the self-disclosed school information, and verify the inferred school names based on the self-disclosed ages.

To verify if the inferred age information is correct, we find all the users who satisfy the following three criteria: 1) they disclose their current schools, 2) they do not disclose their ages, and 3) we inferred their ages by the above methodology. Intuitively, if two people currently attend the same school, their

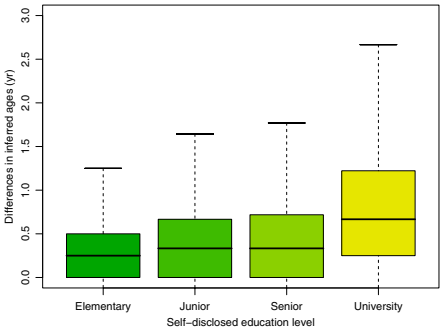


Fig. 8. The inferred age differences between pairs of self-disclosed schoolmates in the four education levels

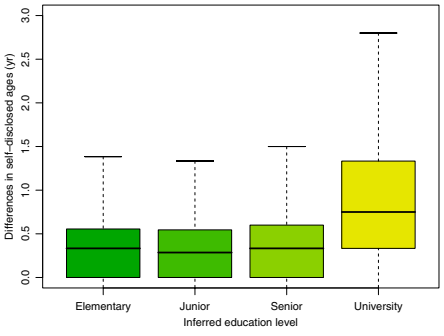


Fig. 9. The self-disclosed age differences between pairs of inferred schoolmates in the four education levels

ages should be similar. Thus, we computed the age differences between pairs of schoolmates, and identified a total of 208,086 valid pairs. We summarize the age differences between pairs of users at different education levels in Fig. 8.

We verify the correctness of the inferred education records based on the same intuition. However, the users being compared should satisfy the following criteria: 1) they disclose their ages, 2) they do not disclose their current schools, and 3) we inferred their education records. In this case, 42,896 pairs of users who described their relationship as "schoolmates" were identified. The distributions of the age difference between each pair of users are shown in Fig. 9. Both plots show that the average age differences between pairs of users in the four education levels are generally less than 2 years. This result suggests that our inference results for users' ages and education records are accurate. It also confirms our conjecture that significant involuntary information leakage occurs in real-world social network services.

6 Discussion

In this section, we discuss the problems that may be caused by identity leakage, such as personalized email spams and spear phishing, and suggest several ways to mitigate the involuntary identity leakage problem.

6.1 Threats Caused by Name Leakage

Spamming. Spam mail has become a major problem in recent years. It is actually a business activity whereby spammers send emails with product information based on massive email address lists. In effect, anyone who has an email address is regarded as a potential customer [15].

To protect users from spamming, academia and industry have developed a number of anti-spam mechanisms. One method of protection against spam involves using a white list [16, 17, 18], so that emails from trusted parties will

not be mistaken for spam. However, as spamming is profitable, spammers are always devising new ways to penetrate spam filters. To combat the white list approach, spammers collect email addresses and information about social relationships from social network services [15]. In our data set, 46% of users disclose a well-formed email address that spammers may use to send spam mail as if it has been sent by one of the target's friends. In this way, spam mail can bypass the filter and be delivered to the target's mailbox.

Spammers may make use of inferred real names in two ways. 1) They may use the recipient's real name in the mail's content. 2) They may make spam mails look like they have been sent by a friend of the target by using the friend's real name. Consider an email containing the recipient's full real name, where the sender is specified as a friend with his/her correct email address and full real name. The user may have difficulty verifying the email's authenticity.

Phishing. Similar difficulties also exist with respect to phishing detection. Phishers normally send emails, which contain a link to a forged web page, to obtain people's sensitive information, such as an account ID, social security number or credit card number [19]. Some phishing-targeted companies, including eBay and PayPal, and Internet security vendors provide guidelines for recognizing phishing emails. Common rules include "*checking if the email includes your real name because phishers do not have personal information*" [20,21]. However, the assumption that phishers do not have personal information might be incorrect as self-disclosure is becoming more frequent in social network services. The involuntary name leakage problem will exacerbate the problem further, as it will be more difficult for users and phishing detection mechanisms [19, 22, 23] to verify the authenticity of a web page.

6.2 Potential Solutions

We consider three possible ways to mitigate the problem of involuntary name leakage in social networking services.

A. Personal Privacy Settings. Social network service providers should provide the following preference settings for every account, no matter whether it is free or not:

1. options to hide personal information;
2. options to hide social connections (the level of connections to be hidden should be configurable, e.g., direct friends or friends of friends);
3. options to prevent a user's friends annotating the user's profile with certain words, or deny any incoming annotation completely.
4. options to deny specific people access to a user's incoming and/or outgoing annotations.

B. Browsing Scope Settings. It is recommended that social network service providers limit the profile browsing scope of users. For example, a user could be restricted to browsing friend annotations made by users who are at most

two degrees away. One common way to limit browsing scope is through group partitioning, i.e., only users belonging to a certain group, where the action of joining the group requires the approval of a moderator, can access more sensitive information about users in the same group. Such mechanisms could prevent malicious parties from downloading users' personal information, social connections, and annotations in an automated way. We believe that this is the key to solving the problem of large-scale information leakage in social network services.

C. Owner's Confirmation. Every operation involving user information should be confirmed by the information owner. For example, friend annotations should only be shown if the annotatee agrees. This would at least prevent unintentional personal information leakage by the user's friends.

7 Conclusion

In this paper, we consider the involuntary information leakage problem in social network services. To assess the seriousness of the problem, we conduct a case study of Wretch, the most popular social network service in Taiwan. Because Wretch users are allowed to annotate their friends' profiles with a one-line, free-form description, sensitive information, such as a person's real name, can be disclosed without the annotatee's knowledge. We show that 78% of users in our data set were subject to involuntary name leakage. In addition, the ages of 15% of users and the school attendance records (partial or complete) of 42% of users could be easily inferred using a simple heuristic. We also show that the risk of information leakage is not related to the degree of self-disclosure; thus, telling a user not to disclose his/her personal information is not an effective way to reduce the risk that his/her identity could be revealed by online friends.

We discuss three possible ways to mitigate the identified leakage problem, namely, providing personal privacy settings, regulating the browsing scope, and requiring an owner's authorization to release personal information. However, as most users do not change the default settings, we believe that the default mechanisms provided by the services are the most effective methods available. We suggest operators should at least mandate that annotations do not contain any sensitive information (as judged by the annotatee) unless the annotatee agrees.

References

1. ENISA: Enisa position paper no.1, security issues and recommendations for online social networks (October 2007), `http://www.enisa.europa.eu/doc/pdf/deliverables/enisa_pp_social_networks.pdf`
2. Gross, R., Acquisti, A., Heinz III, H.: Information revelation and privacy in online social networks. In: Proceedings of the 2005 ACM workshop on Privacy in the electronic society, pp. 71–80. ACM Press, New York (2005)

3. Ahn, Y., Han, S., Kwak, H., Moon, S., Jeong, H.: Analysis of topological characteristics of huge online social networking services. In: Proceedings of the 16th international conference on World Wide Web, pp. 835–844. ACM Press, New York (2007)
4. Mislove, A., Marcon, M., Gummadi, K., Druschel, P., Bhattacharjee, B.: Measurement and analysis of online social networks. In: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, pp. 29–42. ACM, New York (2007)
5. Kumar, R., Novak, J., Tomkins, A.: Structure and evolution of online social networks. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 611–617. ACM Press, New York (2006)
6. OMurchu, I., Breslin, J., Decker, S.: Online social and business networking communities. In: Proceedings of ECAI 2004 Workshop on Application of Semantic Web Technologies to Web Communities (2004)
7. Boyd, D.: Friendster and publicly articulated social networks. In: Conference on Human Factors and Computing Systems (CHI 2004), Vienna, Austria, April, pp. 24–29 (2004)
8. Acquisti, A.: Privacy in electronic commerce and the economics of immediate gratification. In: Proceedings of the 5th ACM conference on Electronic commerce, pp. 21–29. ACM Press, New York (2004)
9. Jourard, S., Lasakow, P.: Some factors in self-disclosure. Journal of Abnormal and Social Psychology 56(1), 91–98 (1958)
10. Joinson, A.N., Paine Schofield, C.: Oxford Handbook of Internet Psychology. In: Self-Disclosure, Privacy and the Internet, pp. 237–252. Oxford University Press, Oxford (2007)
11. Farmer, R.: Instant messaging–collaborative tool or educator's nightmare. In: The North American Web-based Learning Conference (NAWeb 2003) (2003)
12. Tsai, C.H.: Common chinese names, `http://technology.chtsai.org/namefreq/`
13. Tsai, C.H.: A list of chinese names, `http://technology.chtsai.org/namelist/`
14. Tsai, C.H.: A review of chinese word lists accessible on the internet, `http://technology.chtsai.org/wordlist/`
15. Judge, P., Alperovitch, D., Yang, W.: Understanding and reversing the profit model of spam. In: Workshop on Economics of Information Security 2005 (WEIS 2005) (June 2005)
16. Oscar, P., Vwani, R.: Personal Email Networks: An Effective Anti-Spam Tool. IEEE Computer 38(4), 61–68 (2005)
17. Seigneur, J., Dimmock, N., Bryce, C., Jensen, C.: Combating spam with TEA (trustworthy email addresses). In: Proceedings of the Second Annual Conference on Privacy, Security and Trust (PST 2004), pp. 47–58 (2004)
18. Garcia, F., Hoepman, J., van Nieuwenhuizen, J.: Spam Filter Analysis. In: Proceedings of 19th IFIP International Information Security Conference, WCC 2004-SEC. Kluwer Academic Publishers, Dordrecht (2004)
19. Zhang, Y., Egelman, S., Cranor, L., Hong, J.: Phinding phish: Evaluating anti-phishing tools. In: Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS 2007). (2007)
20. Microsoft.com: Recognize phishing scams and fraudulent e-mails, `http://www.microsoft.com/athome/security/email/phishing.mspx`

21. PayPal: Phishing guide part 2, https://www.paypal.com/us/cgi-bin/webscr?cmd=xpt/cps/securitycenter/general/RecognizePhishing-outside
22. Wu, M., Miller, R., Garfinkel, S.: Do security toolbars actually prevent phishing attacks? In: Proceedings of the SIGCHI conference on Human Factors in computing systems, pp. 601–610. ACM Press, New York (2006)
23. Florêncio, D.A.F., Herley, C.: Analysis and improvement of anti-phishing schemes. In: SEC 2006, pp. 148–157 (2006)

Privacy Preserving Computations without Public Key Cryptographic Operation

Koji Chida and Katsumi Takahashi

NTT Corporation, Tokyo, Japan
chida.koji@lab.ntt.co.jp

Abstract. We develop the privacy preserving computation protocol presented by Naor, Pinkas and Sumner at ACM Conference on Electronic Commerce in 1999 into a more efficient one. Their protocol is based on the Yao's two-party secure function evaluation and can be used to implement any combinatorial circuit for input clients' data without disclosing them. In this paper we propose three types of protocol as variants of the Naor-Pinkas-Sumner protocol in each different framework. The first protocol is the almost same framework as theirs but requires no public key cryptographic operations for clients unlike their protocol. The second protocol furthermore eliminates an oblivious transfer from the two-party operation in their protocol and the first protocol by adding a new entity named "mediator" into the Yao's two-party setting. In the new three-party setting, it is assumed that no party colludes with any other parties to retain the secrecy of the clients' data. The last protocol removes the mediator from the second protocol in return for clients' some additional burden. Since an oblivious transfer used in the Naor-Pinkas-Sumner protocol and the first protocol is the dominant step in each protocol, the second and last protocols are expected to be much faster than the others.

1 Introduction

The recognition of privacy by consumers or citizens is increasing due to the current cases of personal information leakage, though some personal information is useful for sales promotion, national statistical research, biometric authentication, and so on. One reason for its phenomenon would be that a personal data once leaked out to the Internet is basically difficult to delete and widely propagated for a long time. Therefore, privacy protection by organizations that retain personal information regarding their customers or citizens is becoming an important business and social topic.

A simple measure to prevent the leakage of information would keep personal information encrypted except the case using it, however, there still has not yet been developed a satisfactory solution when considering the repeated internal threats over the past few years. To prevent internal illegal acts, various approaches have been put to practical use. The *Oracle(R) Database Vault* [9] protects customers' information from internal threats by implementing separation-of-duty mandates that require more than one person to complete a sensitive task.

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 184–200, 2008.

However, the computer environment in which the access (or decryption) key is restored is not necessarily fail-safe because various threats such as key loggers, computer viruses, or fraudulent reverse engineering still exist.

A cryptographic application technology such as a threshold cryptosystem (e.g., [5]) or multiparty protocol (e.g., [7]) has potential to strongly support privacy protection. Various improvements and extensions have been studied regarding these technologies. Due to a threshold cryptosystem, manager(s) can decrypt only a specific ciphertext without restoring the decryption key on any computer. Even if the managers of less than the threshold conspire illegally, they cannot maliciously retrieve significant information for personal data from the encrypted data set. Furthermore, the existing multiparty protocols including specific ones for two-party such as in [13] enable various computations of secret data without leaking it, though versatile multiparty protocols generally tend to require a significant amount of computation time and cause heavy network traffic.

Scenario

We consider a scenario in which privacy-protected personal data of clients are sent to server(s) through a monitoring device and the server obtains some knowledge concerning the data with keeping the privacy. This scenario includes a private biometric authentication that aims to authenticate clients based on their physical characteristics such as fingerprint or iris without revealing any information regarding these personal characteristics to the verifier or an attacker [12]. The location and physical data of clients obtained from a GPS, RFID, sensor, etc., may also be sensitive but valuable for the purpose of sales promotion or statistical research for public interest. Of course, the computational effort of servers should be reduced as much as possible from a viewpoint of improvements in service quality and cost saving of capital investment.

On the other hand, a cell-phone is expected to be one of most likely monitoring devices because most of recent cell-phones are available in high-bandwidth network and equipped with a GPS. Besides, some types of cell-phone have a function of biometric authentication or RFID reader.

Our Contribution

We propose three protocols for extensively achieving the scenario as stated previously in multiparty (including two-party-specific) setting in order to securely utilize various personal data. Multiparty protocols typically have the property that multiple parties engage in the execution for an objective function with keeping the input data secret unless parties more than a threshold number conspire. All the proposed protocols support any combinatorial circuit as well as the Naor-Pinkas-Sumner protocol.

The first protocol (Protocol I) is a client-friendly one. In some previous approaches each monitoring device has to encrypt a personal data per bit by a public key cryptographic operation or place an additional heavy burden on the servers. The detail will be introduced in Sect. 2. However, in the first protocol

Table 1. Feature Comparison among Proposed Protocols

	Protocol I	Protocol II	Protocol III
Clients' Computational Complexity	○	○	○
Clients' Communication Cost	○	○	×
Servers' Computational Complexity	×	○	○
Number of Required Servers	2	3	2

each monitoring device requires an XOR operation just one time for preserving the privacy of a personal data with a small additional burden on the servers. This would be helpful for a low-performance client's device. In the first protocol the server-side operation needs to be performed in two-party-specific setting.

The second protocol (Protocol II) furthermore eliminates a public key cryptographic operation, more precisely an oblivious transfer [6], which is used in the server-side operation of the first protocol and the Naor-Pinkas-Sumner protocol, by adding a new entity named "mediator" into the first protocol. Besides, the basic idea of the second protocol can be applied to the original Yao's two-party protocol [13]. We will describe the specification in Sect. 3.2.

The last protocol (Protocol III) removes the mediator from the second protocol in return for clients' some additional burden. This can be achieved by replacing the mediator's operation with each client's one dispersively.

Table 1 roughly compares some features of the proposed protocols.

A trick of the first protocol is to share a personal data into two fragments and embed one fragment into the *garbled* circuit [13] corresponding to an objective function. The remaining fragment is sent to a server through a secure channel as illustrated in Figure 4. The garbled circuit is generated by a proxy. Finally, the server obtains the outcome of the objective function for input the personal data in cooperation with the proxy by performing the garbled circuit with oblivious transfer.

On the other hand, in the second protocol each client sends the remaining fragment not the server but the mediator as illustrated in Figure 5. The mediator generates the garbled values corresponding to 0 and 1 for every input wires of the circuit. Then, the mediator sends all of them to the proxy while it sends only the garbled values corresponding to the remaining fragment to the server. Finally, the server performs a garbled circuit using the garbled values received from the mediator. The second protocol requires no oblivious transfer, which is the dominant task of the Yao's two-party protocol, the Naor-Pinkas-Sumner protocol, and the first protocol, thereby is expected to be much faster than the others.

In the last protocol each client generates the garbled values corresponding to 0 and 1 just for the wires to which its personal data are input. Then it sends all of them to the proxy while it sends only the garbled values corresponding to the remaining fragment to the server.

2 Previous Work

Potential solutions in multiparty setting for extensively achieving our scenario can be classified into three classes. First is using a versatile multiparty protocol such as in [1,2,3,7,11] (Class I). For example, in the protocols presented in [3,11] each client encrypts its personal data per bit using a public key and publishes the ciphertexts. After that, multiple servers jointly perform a multiparty protocol for input the ciphertexts for each gate that comprises the circuit corresponding to an objective function.

Second is reducing the computational effort for clients using the *decomposition* technique presented in [4,12] for Class I (Class II). In [12] each client encrypts its personal data as a lump using a public key and then multiple servers decompose the ciphertext into each bit of the personal data encrypted by the same public key without disclosing the personal data before the execution of multiparty protocol. This means the computational effort for each server increases as compared to that of Class I, though the burden for clients' device becomes smaller.

The last is the privacy preserving computation protocol presented by Naor, Pinkas and Sumner [8] and its variations (Class III). Their protocol is based on the Yao's two-party secure function evaluation [13] and was developed to implement any mechanism design including auctions without disclosing the input data. A key technology of the Yao's two-party secure function evaluation is the 1-out-of-2 oblivious transfer. The notion of oblivious transfer was suggested by Rabin [10]. Later, it was developed as the 1-out-of-2 oblivious transfer by Even, Goldreich and Lempel [6]. The 1-out-of-2 oblivious transfer consists of a server that knows two secret values (m_0, m_1) and a client that wishes to get m_σ for $\sigma \in \{0, 1\}$. The goal is that the client gets m_σ without any information concerning $m_{1-\sigma}$ while the server gains no information concerning σ. The traditional 1-out-of-2 oblivious transfers basically need public key operations for both the client and server.

We here introduce the Naor-Pinkas-Sumner protocol [8] in detail as it is the underlying protocol of ours. The entities are clients that provide its personal data to a server without disclosing the data, an proxy that generates the garbled circuit corresponding to an objective function such as for statistics or authentication, and the server that performs the garbled circuit for input the privacy-protected personal data and obtains some knowledge concerning the data with keeping the privacy. It is assumed that each client and the server are connected via a secure channel while the others allow a publication channel. Figure 1 shows their system. We can alternatively use an end-to-end cipher communication between each client and the server through the proxy as stated in [8]. In this case, it eliminates the need for the direct channel between each client and the server.

In the Naor-Pinkas-Sumner protocol the server obtains $f(\alpha_1, \ldots, \alpha_n)$ with the help of the proxy without knowing $\alpha_1, \ldots, \alpha_n$, where f is an objective function and α_ν is C_ν's data. For example, α_ν and f will be the bidding price chosen by C_ν and the maximum function for input $\alpha_1, \ldots, \alpha_n$, respectively, in a sealed-bid auction. In such a case, the Naor-Pinkas-Sumner protocol keeps bidding prices

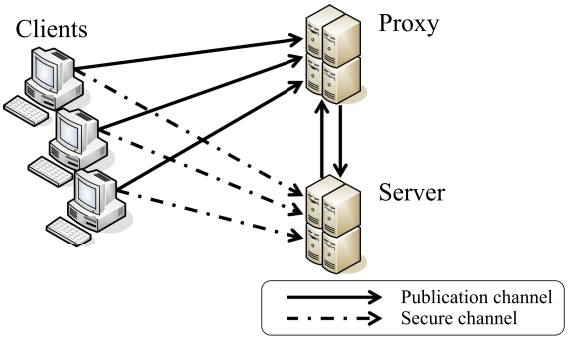


Fig. 1. A system to run the Naor-Pinkas-Sumner protocol

$\alpha_1, \ldots, \alpha_n$ except the winning bid secret even after the bid opening unless the server and proxy conspire.

We give a brief description of their protocol below.

1. The proxy constructs the garbled circuit corresponding to an objective function based on the Yao's algorithm [13] and sends it to the server.
2. Each client engages in a variant of 1-out-of-2 oblivious transfer with the proxy and server. At the end of this step, the server obtains the garbled values corresponding to $\alpha_1, \ldots, \alpha_n$.
3. The server evaluates $f(\alpha_1, \ldots, \alpha_n)$ using the garbles of circuit and input values.

Next, we describe two key components of the above-mentioned protocol, the Yao's two party secure function evaluation [13] and the 1-out-of-2 *proxy* oblivious transfer [8]. The former component is used in Steps 1 and 3. The latter is used in Step 2.

Yao's Two Party Secure Function Evaluation

It garbles the circuit composed of Boolean gates such as AND, OR and NOT of an objective function. We review an example of it followed by [8].

It is run between a client and server. The client's input is value α and the server's input is function f. At the end of the protocol, the client learns $f(\alpha)$ without any information about f except the value $f(\alpha)$. The server learns nothing about α. The function f is assumed as a combinatorial circuit composed of primitive gates g: $\{0,1\} \times \{0,1\} \to \{0,1\}$.

An example of the procedure is as follows.

Client's Input: α
Server's Input: f
Client's Output: $f(\alpha)$

1. The server performs the following steps.
   (a) Assign to each wire i of the circuit two random values $W_i^0, W_i^1 \in \{0,1\}^\kappa$ corresponding to 0 and 1, respectively, and a random bit c_i, where κ is security parameter.
   (b) Set $(\langle W_i^0, c_i\rangle, \langle W_i^1, \bar{c}_i\rangle)$ as the garbled value of wire i, where $\bar{c}_i = 1 - c_i$.
   (c) For each gate g with two input wires i and j and the output wire k, make table T_g composed of the four entries of the form,

$$c_i, c_j : (W_k^{g(0,0)}, g(0,0) \oplus c_k) \oplus F_{W_i^0}(c_j) \oplus F_{W_j^0}(c_i),$$
$$c_i, \bar{c}_j : (W_k^{g(0,1)}, g(0,1) \oplus c_k) \oplus F_{W_i^0}(\bar{c}_j) \oplus F_{W_j^1}(c_i),$$
$$\bar{c}_i, c_j : (W_k^{g(1,0)}, g(1,0) \oplus c_k) \oplus F_{W_i^1}(c_j) \oplus F_{W_j^0}(\bar{c}_i),$$
$$\bar{c}_i, \bar{c}_j : (W_k^{g(1,1)}, g(1,1) \oplus c_k) \oplus F_{W_i^1}(\bar{c}_j) \oplus F_{W_j^1}(\bar{c}_i),$$

   with random order, where F_W: $\{0,1\}^{\kappa+1} \to \{0,1\}^{\kappa+1}$ is a pseudo-random permutation with seed W. However, set $c_k = 0$ if g is the final step of circuit. This is because the server obtains the final output of the circuit, that is, the bit sequence of $f(\alpha)$.
   (d) Send the garbled circuit composed of T_g to the client.
2. For each bit of α input to wire i, b, the server and client engage in a 1-out-of-2 oblivious transfer. At the end of this step, the client obtains $\langle W_i^b, b \oplus c_i\rangle$.
3. The client evaluates the garbled circuit for input $\langle W_i^b, b \oplus c_i\rangle$ and obtains $f(\alpha)$.

Suppose that the client has the garbles of bits input to g with input wires i and j and wishes to get the garble corresponding to the output wire k. Then, the garbles can be represented as $\langle W_i^b, b \oplus c_i\rangle$ and $\langle W_j^{b'}, b' \oplus c_j\rangle$ for input bits b and b'. In this setting, we see that the client can extract the garble of $g(b, b')$, that is, $\langle W_k^{g(b,b')}, g(b,b') \oplus c_k\rangle$, from T_g. By iterating this extraction for T_g sequentially, it can obtain the outcome of $f(\alpha)$. Note that the client gains no information concerning $g(b, \bar{b}')$, $g(\bar{b}, b')$ and $g(\bar{b}, \bar{b}')$ through the execution of protocol.

1-out-of-2 proxy oblivious transfer

It is an extension of 1-out-of 2 oblivious transfer. The goal of the protocol is that the server obtains a secret value of the proxy according to the client's choice instead of the client. Prior to the description, we review the El Gamal cryptosystem to introduce some symbols.

Consider an additive cyclic group $\mathcal{G}_q$ of order q generated by G where the Decisional Diffie-Hellman problem is difficult. Consider the El Gamal public key, $PK = (G, H = uG) \in \mathcal{G}_q^2$, and its private key, u. An encryption of $M \in \mathcal{G}_q$ from the El Gamal cryptosystem is represented as $E_{PK}(M) = (rG, rH + M) \in \mathcal{G}_q^2$, where $r \in_R \mathbb{Z}/q\mathbb{Z}$. Plaintext M can be obtained from $E_{PK}(M)$ by subtracting urG $(= rH)$ from $rH + M$.

Besides, the 1-out-of-2 proxy oblivious transfer uses an element of $\mathcal{G}_q$, X, where it is assumed to be hard to solve the discrete logarithm to base G of X for all entities. A procedure of the 1-out-of-2 proxy oblivious transfer is as follows.

Common Input: (G, q, X)
Client's Input: $\sigma \in \{0, 1\}$
Proxy's Input: (m_0, m_1)
Server's Output: m_σ

1. A client chooses $u \in_R \mathbb{Z}/q\mathbb{Z}$ and computes $PK_\sigma = uG$ and $PK_{1-\sigma} = X - PK_\sigma$.
2. The client sends PK_0 and u to the proxy and server, respectively.
3. The proxy computes $(E_{PK_0}(\mathcal{C}(m_0)), E_{PK_1}(\mathcal{C}(m_1)))$ and sends it to the server after computing $PK_1 = X - PK_0$, where $\mathcal{C}$ is a good error detecting code.
4. The server tries to decrypt both $E_{PK_0}(\mathcal{C}(m_0))$ and $E_{PK_1}(\mathcal{C}(m_1))$ using u. Due to the error detecting code, it can determine which one was decrypted correctly, and thereby obtain m_σ.

The Naor-Pinkas-Sumner protocol invokes the 1-out-of-2 proxy oblivious transfer described above $|x_i|$ times for client C_i, where $|x_i|$ is the bit length of x_i.

3 Proposed Protocol

We propose three protocols as variants of the Naor-Pinkas-Sumner protocol in each different framework. The first protocol (Protocol I) reduces the computational effort for clients. The second protocol (Protocol II) furthermore eliminates an oblivious transfer from among the server-side operations in Protocol I by adding a new entity into the server-side. The last protocol (Protocol III) removes the new entity from the second protocol in return for clients' some additional burden.

All the proposed protocols and the Naor-Pinkas-Sumner protocol are in different security conditions. In Sect. 5, we will analyze the proposed protocols based on the notion described in Sect. 4.

3.1 Protocol I

As stated in Sect. 2, when the Naor-Pinkas-Sumner protocol is used in our scenario, each monitoring device has to perform the 1-out-of-2 proxy oblivious transfer $|\alpha|$ times to keep personal data α secret. However, in Protocol I each monitoring device requires an XOR operation just one time for preserving the privacy of a personal data with a small additional burden on the servers.

An aspect of Protocol I is to share a personal data $\alpha \in \{0, 1\}^\ell$ into two fragments $s \in_R \{0, 1\}^\ell$ and $t = \alpha \oplus s$ and embed s into the garbled circuit corresponding to an objective function, where ℓ is a positive integer. Intuitively, Protocol I converts every gate that comprises the first step from the top of a circuit, $g(x, y)$, and its input bits a and b into the translations using random bits v and w generated by a client as illustrated in Figure 2. By this translation, the truth table of g changes to the new one illustrated in Figure 3. After that, a proxy generates the garbled gate of $g(x \oplus v, y \oplus w)$ and a server receives $a \oplus v$ and $b \oplus w$ from the client. Finally, the server evaluates $g(a, b)$ for input $a \oplus v$ and

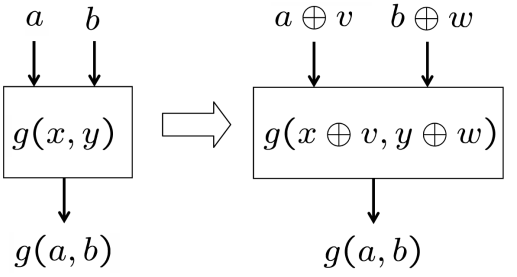


Fig. 2. An original gate and the translation using random bits

$a\backslash b$	0	1
0	$g(0,0)$	$g(0,1)$
1	$g(1,0)$	$g(1,1)$

$a\backslash b$	w	$1-w$
v	$g(0,0)$	$g(0,1)$
$1-v$	$g(1,0)$	$g(1,1)$

Fig. 3. Truth tables of an original gate and the translation

$b \oplus w$ to the garbled gate of $g(x \oplus v, y \oplus w)$. The server obtains neither a nor b through the execution due to random bit v and w, respectively.

Alternatively, we can embed the random bits into not a gate but the input wires of the gate. This alternative makes a procedure more simple, so we adopt the alternative as a component of the procedure (Step 2.(b)) described later.

Figure 4 shows a system to run Protocol I. An overall procedure of Protocol I is as follows.

1. Each client C_ν $(1 \leq \nu \leq n)$ shares its personal data α_ν into two fragments $s_\nu \in_R \{0,1\}^\ell$ and $t_\nu = \alpha_\nu \oplus s_\nu$ and sends s_ν and t_ν to the proxy and the server, respectively. Denote the $(m+1)$-st LSB of α_ν, s_ν and t_ν as $\alpha_{\nu,m}$, $s_{\nu,m}$ and $t_{\nu,m}$, respectively.
2. The proxy constructs the garbled circuit corresponding to objective function f embedded with s_ν as follows:
   (a) Assign to each wire i of the circuit two random values $W_i^0, W_i^1 \in \{0,1\}^\kappa$ corresponding to 0 and 1, respectively, and a random bit c_i, where κ is security parameter.
   (b) Set $(\langle W_i^0, c_i\rangle, \langle W_i^1, \bar{c}_i\rangle)$ as the garbled value of wire i. However, reset it to $(\langle W_i^{s_{\nu,m}}, s_{\nu,m} \oplus c_i\rangle, \langle W_i^{\bar{s}_{\nu,m}}, \bar{s}_{\nu,m} \oplus c_i\rangle)$ if wire i takes α_ν's bit $\alpha_{\nu,m}$.
   (c) For each gate g with input wires i and j and the output wire k, make the garble of g, T_g, followed by Step 1.(c) in the Yao's two party secure function evaluation described in Sect. 2.
   (d) Send the garbled circuit composed of T_g to the server.
3. For each bit of t_ν input to wire i, b, the server engages in a 1-out-of-2 oblivious transfer with the proxy. At the end of this step, the server obtains $\langle W_i^b, b \oplus c_i\rangle$.
4. The server evaluates the garbled circuit from T_g and $\langle W_i^b, b \oplus c_i\rangle$ and obtains $f(\alpha_1, \ldots, \alpha_n)$.

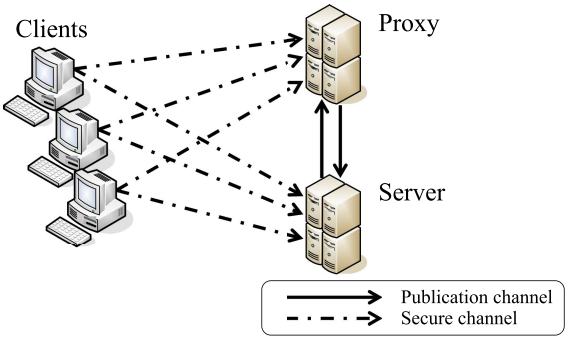


Fig. 4. A system to run Protocols I and III

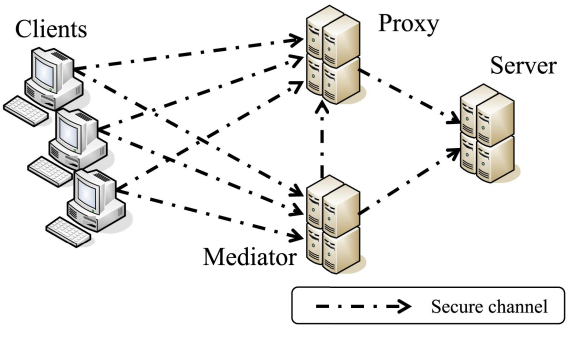


Fig. 5. A system to run Protocol II

3.2 Protocol II

Protocol II is a simple extension of Protocol I. It requires a mediator that generates garbled values input to the garbled circuit and receives the remaining fragment from clients in order to send the garbled values corresponding to the remaining fragment to the server.

Figure 5 shows a system to run Protocol II. An overall protocol is as follows.

1. The mediator generates random bit c_i and two random values W_i^0, $W_i^1 \in \{0,1\}^\kappa$ corresponding to 0 and 1 of each input wire i of the circuit and sends them to the proxy.
2. Each client C_ν shares its personal data $\alpha_\nu \in \{0,1\}^\ell$ into two fragments $s_\nu \in_R \{0,1\}^\ell$ and $t_\nu = \alpha_\nu \oplus s_\nu$ and sends s_ν and t_ν to the proxy and the mediator, respectively.
3. The proxy constructs the garbled circuit composed of T_g corresponding to objective function f embedding s_ν, followed by the procedure described in Sect. 2, and sends the garbled circuit to the server.

4. The mediator sends $\langle W_i^b, b \oplus c_i \rangle$ corresponding to t_ν to the server, where b is each bit of t_ν input to wire i.
5. The server evaluates $f(\alpha_1, \ldots, \alpha_n)$ from T_g and $\langle W_i^b, b \oplus c_i \rangle$.

The procedure above can be applied to the original Yao's two-party protocol by replacing clients' operation in Step 2 with proxy's one.

3.3 Protocol III

Protocol III delegates the task of mediator to clients dispersively and thereby removes the mediator from Protocol II in return for clients' some additional burden.

The system configuration to run Protocol III can be the same as that of Protocol I. An overall protocol is as follows.

1. Each client C_ν performs the following steps.
   (a) Share its personal data $\alpha_\nu \in \{0,1\}^\ell$ into two fragments $s_\nu \in_R \{0,1\}^\ell$ and $t_\nu = \alpha_\nu \oplus s_\nu$.
   (b) Generate random bit c_i and two random values W_i^0, $W_i^1 \in \{0,1\}^\kappa$ corresponding to 0 and 1 of the specific wires i to which C_ν's data are input.
   (c) Send $(s_\nu, (c_i, W_i^0, W_i^1))$ and $\langle W_i^b, b \oplus c_i \rangle$ to the proxy and the server, respectively, where b is each bit of t_ν input to wire i.
2. The proxy constructs the garbled circuit composed of T_g corresponding to objective function f embedding s_ν, followed by the procedure described in Sect. 2, using (c_i, W_i^0, W_i^1) and sends the garbled circuit to the server.
3. The server evaluates $f(\alpha_1, \ldots, \alpha_n)$ from T_g and $\langle W_i^b, b \oplus c_i \rangle$.

4 Model and Security Requirement

The three classes described in Sect. 2 have different aspects and security conditions. The mandatory requirements for multiparty privacy preserving computations (MPPCs) are *completeness* and *secrecy*. We call an MPPC protocol complete if the outcome of the protocol is correct whenever all the parties and equipments used in the protocol work accurately followed by the specification. Also, we say that an MPPC protocol has secrecy if no party gains information concerning the clients' data input to an objective function in the execution in a certain condition.

On the other hand, some applications might have to check the *correctness* of the outcome of an MPPC protocol. Voting and auction would be representative applications. This function of correctness is frequently considered as *verifiability*. That is, a verifiable MPPC protocol has a function that a verifier can check validity of the data output from each entities. Also, the *robustness*, which is the property if an MPPC system works accurately in the presence of a untrusted entity, might significantly affect some applications.

Most of the traditional protocols achieving Class I or II have completeness, secrecy, and verifiability in a certain condition. Some of them are robust as long as more than $N/2$ parties are uncorrupted, where N is the total number of parties engaged in a multiparty protocol. However, the basic protocol described in Sect. 2 as Class III does not have verifiability, not to mention robustness. For the purpose of verifiability, a non-malleable encryption for clients and a cut-and-choose technique are additionally required as described in [8].

5 Security Analysis

In this section we consider completeness and the secrecy of personal data for the proposed protocols based on the notion described in Sect. 4. We leave the verifiability of server-side entities (that is, a server, proxy, and mediator) and robustness issue of the proposed protocols assuming that the entities are *semi-trusted*. In the assumption, all the server-side entities perform their task correctly but some of them may try a security breach, in other words, they may be honest-but-curious.

For the completeness concerning Protocol I, we focus on the relation between the garbled data sent to the server and the garbled gates corresponding to the first step from the top of circuit. The completeness of the remaining part is obvious. Let $\alpha_{\nu,v}$, $s_{\nu,v}$ and $t_{\nu,v}$ be the $(v+1)$-st LSB of α_ν, s_ν and t_ν, respectively. For simplicity, we suppose $\alpha_{\nu,0}$ and $\alpha_{\nu,1}$ are input to gate $g = g(y, z)$ in the original execution of circuit. Then, the input to the garbled gate, $\tilde{g} = g(y \oplus s_{\nu,0}, z \oplus s_{\nu,1})$, are $t_{\nu,0}$ and $t_{\nu,1}$ instead of $\alpha_{\nu,0}$ and $\alpha_{\nu,1}$, respectively. Obviously, $g(t_{\nu,0} \oplus s_{\nu,0}, t_{\nu,1} \oplus s_{\nu,1}) = g(\alpha_{\nu,0}, \alpha_{\nu,1})$ hold, and therefore $\tilde{g}$ works as an alternative of g.

For the secrecy concerning Protocol I, we consider whether the server-side entities (i.e., the server and proxy) can violate the privacy from information obtained through the execution of Protocol I. The information the server obtains is a garbled circuit, t_ν $(= \alpha_\nu \oplus s_\nu)$, the garbled values corresponding to t_ν, and the communication log in 1-out-of-2 oblivious transfer. This is the same condition of the client in the Yao's two party secure function evaluation described in Sect. 2. We see that the proxy gains no information about α_ν from its input, s_ν, assuming the security of 1-out-of-2 oblivious transfer. Therefore the secrecy of personal data is assured unless the server and proxy conspire.

We next consider the completeness and secrecy concerning Protocol II. The completeness can be easily obtained from the observation for Protocol I. For the secrecy against the mediator, it gains no information about α_ν from $t_\nu = \alpha_\nu \oplus s_\nu$ due to the random s_ν. The information the server obtains is only the garbles of the circuit and its input data. The proxy obtains $s_\nu = \alpha_\nu \oplus t_\nu$ and (W_i^0, W_i^1). The former information does not leak α_ν unless the mediator gives t_ν. Of course, the latter garbles give nothing about α_ν. On the other hand, it is obvious that Protocol II is vulnerable if the server conspires with either of the proxy or mediator. Therefore the secrecy of personal data is assured as long as they keep their information secret.

Next is about the completeness and secrecy concerning Protocol III. The completeness can also be easily obtained from the observation for Protocol I. For the secrecy concerning Protocol III, we consider whether the server and proxy can violate the privacy from information obtained through the execution of Protocol III. The information the server obtains is a garbled circuit and the garbled values corresponding to t_ν. So the server gains no information about α_ν. The information the proxy obtains is s_ν and (c_i, W_i^0, W_i^1). Since it is obvious (c_i, W_i^0, W_i^1) includes information about neither α_ν nor t_ν, the proxy gains no information about α_ν. Therefore the secrecy of personal data is assured unless the server and proxy conspire.

We finally consider a fraud by client. Protocols I and II remain nearly unaffected by it because in the protocols each client only sends two fragments of a data to the server-side entities. Even if the fragments are incorrect, its data is just taken as the exclusive OR of the incorrect fragments. On the other hand, in Protocol III a malicious client has a chance of confusing the execution of the server-side operation by sending an incorrect value to the proxy or server. More precisely, it is considered the following two incorrect cases for $(s_\nu, (c_i, W_i^0, W_i^1))$ and $\langle \hat{W}_i^b, \hat{b} \oplus c_i \rangle$ respectively sent to the proxy and the server.

(i) $\hat{W}_i^b \notin \{W_i^0, W_i^1\}$,
(ii) $\hat{W}_i^b \in \{W_i^0, W_i^1\}$ but $b \neq \hat{b}$.

For Case (i), an incorrect garbled value may be output from the output wire. To prevent such a trouble, it is desirable that the proxy provides a good error detecting code to each output garbles in order to make the proxy aware the fraud in the execution. Case (ii) is essentially the same as a unworthy fraud against Protocols I and II described previously.

6 Comparison

We compare the efficiency and security level of the proposed protocols and some existing protocols based on the survey in Sect. 2.

All the protocols that belong to Class I, II or III can be divided into three phases. The first phase is the generation of privacy-protected personal data by clients (Phase 1). The second phase is the transmission of the privacy-protected personal data (Phase 2). The last phase is the evaluation of an objective function for input the personal data without disclosing the data (Phase 3).

The original personal data is represented as α. Denote the numbers of clients and server-side entities as n and N, respectively. Assume every personal data of client is ℓ bits integer. Also assume any ℓ bits integer is accepted by an objective function. Denote the numbers of gates and input wires of a circuit corresponding to an objective function as $z_0 = z_0(n, \ell)$ and $z = z(n, \ell)$, respectively. Denote the depth of the circuit as $d = d(n, \ell)$.

6.1 Class I

[7] is a cryptographic approach using oblivious transfer. In Phase 1 each client divides a personal data into N fragments by the exclusive OR once and generating $N-1$ pseudo-random numbers. In Phase 2 each fragment is sent to a server among N servers via a secure channel. The integer N can be arbitrarily chosen if $N \geq 2$ hold. In Phase 3 a 1-out-of-4 oblivious transfer, which is invoked $O(zN)$ times for each server, with a zero-knowledge protocol is considered as the dominant task among the servers. It requires at least $\lfloor N/2+1 \rfloor$ servers' corporation with $O(d)$ round complexities. [7] has secrecy, correctness, and robustness when assuming at least $\lfloor N/2+1 \rfloor$ servers are honest. If the zero-knowledge protocol used in the 1-out-of-4 oblivious transfer has a trouble, not only the correctness but also the secrecy are not guaranteed.

[1,2] are an information theoretic approach using secret sharing. In Phase 1 each client generates a random polynomial with degree $N'-1$, $P(x)=\sum_{i=0}^{N'-1} a_i x^i$, in a finite field and sets $a_0 = \alpha$. Then it computes a specific coordinate, $(j, P(j))$, for $j = 1, \ldots, N$. In Phase 2 each coordinate $(j, P(j))$ is sent to j-th server via a secure channel. The integers N and N' can be chosen such that $2N'-1 \leq N < 3N'-1$ and $N' \geq 2$ hold. In Phase 3 a polynomial interpolation, which is invoked $O(zN)$ times for each server, with a zero-knowledge protocol is considered as the dominant task among the servers. It requires $2N'-1$ servers' corporation with $O(d)$ round complexities. [1,2] have secrecy, correctness, and robustness when assuming at least $2N'-1$ servers are honest. We do not know whether the zero-knowledge protocol for polynomial interpolation affects the secrecy of α.

[3,11] are a cryptographic approach using homomorphic encryption. In Phase 1 each client encrypts its personal data per bit using a public key. In Phase 2 all the ciphertexts of bits of personal data are sent to N servers via a publication channel. The integer N can be arbitrarily chosen if $N \geq 2$ hold. In Phase 3 an interactive cryptographic protocol cleverly using homomorphicity of the ciphertexts, which is invoked $O(z)$ times for each server, with a zero-knowledge protocol is considered as the dominant task among the servers. It requires at least $\lfloor N/2+1 \rfloor$ servers' corporation with $O(d)$ and $O(dN)$ round complexities for [3] and [11], respectively. Note that the computational effort for each server in [11] is slightly lighter than that in [3]. [3,11] have secrecy, correctness, and robustness when assuming at least $\lfloor N/2+1 \rfloor$ servers are honest. If the zero-knowledge protocol used in the interactive cryptographic protocol has a trouble, not only the correctness but also the secrecy are not guaranteed.

6.2 Class II

[12] is a cryptographic approach that can be applied to [3]. In Phase 1 each client encrypts its personal data as a lump using a public key. In Phase 2 the ciphertext of personal data is sent to N servers via a publication channel. The integer N can be arbitrarily chosen if $N \geq 2$ hold. In Phase 3 a decomposition protocol with a zero-knowledge protocol is performed for each ciphertext at first. It requires at least $\lfloor N/2+1 \rfloor$ servers' corporation with $O(1)$ round complexities.

Then, the interactive cryptographic protocol of [3] is performed as described previously. [3] with [12] has secrecy, correctness, and robustness when assuming at least $\lfloor N/2 + 1 \rfloor$ servers are honest. If the zero-knowledge protocol used in the decomposition protocol has a trouble, not only the correctness but also the secrecy are not guaranteed.

6.3 Class III

[8] is a cryptographic approach using oblivious transfer. The system is limited to $N = 2$ (a server and proxy). In Phase 1 each client generates a public and private key pair and then encrypts its personal data per bit using the public key in the execution of the 1-out-of-2 proxy oblivious transfer described in Sect. 2. In Phase 2 the private key is sent to the server via a secure channel while the ciphertexts of bits of personal data are sent to the proxy via a publication channel. In Phase 3 the 1-out-of-2 proxy oblivious transfer, which is invoked z_0 times in this phase, with a zero-knowledge protocol is considered as the dominant task among the server and proxy. Other tasks include operations for pseudo-random permutation, exclusive OR, pseudo-random number generation for the server and proxy $O(z)$ times. In this phase the server and proxy communicate within a constant round. [8] has secrecy if the server and proxy does not conspire. The correctness and robustness can be violated if either of the server and proxy acts illegally. However, the correctness violation by proxy can be prevented to some degree by using a cut-and-choose technique, though the burden of the server becomes larger. The cut-and-choose technique may also be applied to Protocols I, II, and III in a natural way, however, we do not mention such a case any more from a practical viewpoint. Of course, the server has to perform its task correctly if it utilizes the outcome itself. If the zero-knowledge protocol used in the 1-out-of-2 proxy oblivious transfer has a trouble, even the secrecy is not guaranteed.

Protocol I is a cryptographic approach using oblivious transfer as well as [8]. The system is limited to $N = 2$ (a server and proxy). In Phase 1 each client divides a personal data into two fragments by the exclusive OR once and generating a pseudo-random number. In Phase 2 each fragment is sent to the server and proxy via a secure channel. In Phase 3 a 1-out-of-2 oblivious transfer, which is invoked z_0 times in this phase, with a zero-knowledge protocol is considered as the dominant task among the server and proxy. Other tasks include operations for pseudo-random permutation, exclusive OR, pseudo-random number generation for the server and proxy $O(z)$ times. In this phase the server and proxy communicate within a constant round. Protocol I has secrecy if the server and proxy does not conspire. The correctness and robustness are violated if either of the server and proxy acts illegally. If the zero-knowledge protocol used in the 1-out-of-2 oblivious transfer has a trouble, even the secrecy is not guaranteed.

Protocol II is an information theoretic approach unlike Protocol I. The system is limited to $N = 3$ (a server, proxy, and mediator). In Phase 1 each client divides a personal data into two fragments by the exclusive OR once and generating a pseudo-random number. This is the same as Protocol I. In Phase 2

each fragment is sent to the mediator and proxy via a secure channel. In Phase 3 the primary tasks are operations for pseudo-random number generation $O(z_0)$ times for the mediator and pseudo-random permutation, exclusive OR, pseudo-random number generation for the server and proxy. All the computational effort for the server and proxy are within $O(z)$. In this phase the server, proxy, and mediator communicate within a constant round. Protocol II has secrecy if none of the server, proxy, and mediator colludes with others. The correctness and robustness are violated if either of the server, proxy, and mediator acts illegally.

Protocol III is an information theoretic approach as well as Protocol II. The system is limited to $N = 2$ (a server and proxy). In Phase 1 each client divides a personal data into two fragments by the exclusive OR once and generating a pseudo-random number $O(z_0')$ times, where z_0' is a number of the input wires to which the data of the client are input. Obviously, $z_0' \leq z_0$ hold. In Phase 2 each fragment is sent to the server and proxy via a secure channel. In addition, all the garbled values generated from the pseudo-random numbers are sent to the proxy while only the garbled values corresponding to the fragment for the server are sent to the server. In Phase 3 the primary tasks are operations for pseudo-random permutation, exclusive OR, pseudo-random number generation for the server and proxy. All the computational effort for the server and proxy are within $O(z)$. In this phase the server and proxy communicate within a constant round. Protocol III has secrecy if the server and proxy does not conspire assuming that a good error detecting code is used to each output garbles as described in Sect. 5. The correctness and robustness are violated if either of the server and proxy acts illegally.

6.4 Discussion

For Class I, [3,7,11], which are a cryptographic approach, burden each client with a heavy computational effort for it, though they provide the highest level of security among the protocols that belong to Class I, II, or III. On the other hand, [1,2], which are an information theoretic approach, provides a small computational effort for each client. It also provides one of the lowest computational effort for servers among the protocols that belong to Class I, II, or III if a zero-knowledge protocol is never used in the entire protocol. However, for maintaining secrecy, it requires at least three servers and a consideration of impact in case that a zero-knowledge protocol is never used in the entire protocol.

For Class II, although [3] with [12] reduces the computational effort for each client as compared to [3], the computational effort for each server becomes higher.

For Class III, [8] burdens each client with a heavy computational effort for it. Protocol I reduces the effort substantially, though the computational effort for each server is higher than that of [1,2] except a zero-knowledge protocol. The advantage of Protocol II looks very similar to [1,2] except a zero-knowledge protocol. Their computational efforts for clients and servers are the almost same level. They do not support correctness. The differences between them include,

– a number of servers in [1,2] except a zero-knowledge protocol can be any number more than two while that in Protocol II is restricted to three, and

– the round complexity for servers in Protocol II becomes constant while that in [1,2] except a zero-knowledge protocol becomes $O(d)$.

Furthermore, in return for clients' some additional burden, Protocol III runs in the two-party setting of servers unlike Protocol II and [1,2]. This aspect not only increases the level of secrecy but also has a potential to extend applications.

7 Conclusion

We proposed three privacy preserving computation protocols suitable for acquisition of personal information using a low-spec computer within monitoring device. The first protocol is based on the Yao's two party secure function evaluation and requires an XOR operation just one time for preserving the privacy of a personal data for each client (monitoring device) while the traditional protocol has to perform a public key cryptographic operation per bit of the personal information for preserving the privacy. Furthermore, the second and third protocol achieves the privacy preserving computation without oblivious transfer. The second protocol requires three parties unlike the first, third and traditional protocols. The secrecy of input data in the second protocol is assured unless any two parties among three conspire. The third protocol requires clients' some additional burden in order to remove the third party from the second protocol. In particular, the second and third protocols are expected to be much faster than the traditional and first protocols because the oblivious transfers used in the traditional and first protocols are the dominant step in each protocol.

References

1. Ben-Or, M., Goldreich, O., Wigderson, A.: Completeness theorems for non-cryptographic fault tolerant distributed computation. In: Proc. of STOC 1988, pp. 1–10. ACM Press, New York (1988)
2. Chaum, D., Crepeau, C., Damgård, I.: Multiparty unconditionally secure protocol. In: Proc. of STOC 1988, pp. 11–19. ACM Press, New York (1988)
3. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–300. Springer, Heidelberg (2001)
4. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 285–304. Springer, Heidelberg (2006)
5. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 307–315. Springer, Heidelberg (1990)
6. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Communications of the ACM 28(6), 637–647 (1985)
7. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game, or a completeness theorem for protocols with honest majority. In: Proc. of STOC 1987, pp. 218–229. ACM Press, New York (1987)

8. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Proc. of ACM EC 1999, pp. 129–139. ACM Press, New York (1999)
9. Oracle Press Release (April 26, 2006), http://www.oracle.com/corporate/press/2006_apr/oracle-database-vault.html
10. Rabin, M.O.: How to exchange secrets by oblivious transfer, Technical Report TR-81, Harvard Aiken Computation Laboratory (1981)
11. Schoenmakers, B., Tuyls, P.: Practical two-party computation based on the conditional gate. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 119–136. Springer, Heidelberg (2004)
12. Schoenmakers, B., Tuyls, P.: Efficient binary conversion for Paillier encrypted values. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 522–537. Springer, Heidelberg (2006)
13. Yao, A.C.: How to generate and exchange secrets. In: Proc. of FOCS 1986, pp. 162–167. IEEE Press, Los Alamitos (1986)

A Novel Framework for Watermarking: The Data-Abstracted Approach

Cyril Bazin, Jean-Marie Le Bars, and Jacques Madelaine

GREYC - CNRS, Dept. Informatique
Université de Caen Basse-Normandie
6, Bd maréchal Juin, Caen, France
cyril.bazin@info.unicaen.fr, jacques@info.unicaen.fr,
lebars@info.unicaen.fr

Abstract. This paper introduces a data-abstracted zero-bit watermarking scheme based on the document quality. It can be applied to various kinds of structured documents and guarantees that the document quality is preserved by the watermarking process. The scheme is blind, robust and fast. It is described as local manipulations of small parts of the document: the so-called *sites*. As a proof of concept, it is applied to geographical documents and relational databases, extending two existing methods. The general scheme allows to compute the resilience level in terms of sites modifications. It is then possible to tune the sites definition and manipulations in order to obtain a given confidence depending on the document type and on the transformations the watermark must resist to.

1 Introduction

Today, data exchange over the Internet is accessible for anyone and it is easy to copy document using peer-to-peer with no respect to the copyright. Robust digital watermarking [1] [2] is a way to preserve the document copyright. A watermarking algorithm amends the document by inserting a mark in such a way that its usability remains essentially the same. Afterwards, a detection algorithm can check if a mark has been inserted in the document. The mark depends upon a key that is only known by the one who applies the watermarking algorithm and allows the original owner identification; this is called zero-bit watermarking [1], as no specific message is embedded.

Our purpose is to define an abstract zero-bit watermarking scheme applicable to various kind of data. It will allow to unify the presentation of various known schemes and helps to define new ones.

Abundant literature exists for watermarking raster images; nevertheless, these schemes seem not easily generalizable to other data types. We investigate geographical data watermarking methods that appear to be generalizable at least to other structured document types. A survey of digital vector map watermarking [3] shows a comparison table between the schemes properties such as blindness, robustness and map qualities preservation. Ohbuchi [4] presents a digital map watermarking scheme using spectral domain modification of a mesh based on Delaunay Triangulation. This watermarking scheme is robust against many

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 201–217, 2008.

transformations. For detection purposes, both original and watermarked documents are needed; such an algorithm is call *well informed*. On the contrary, we don't want to use the original during the detection step leading to a so-called *blind* scheme. Many other blind watermarking schemes of geographical data were presented since the survey publication [5], [6]. Lafaye [5] provides a watermarking derived from the work of Agrawal [7] on relational databases. The watermarking is done locally by modifying the shapes of each geographical object. A blind watermarking scheme for digital vector maps using also the concept of Delaunay Triangulation is presented in [6]. The main idea of this paper is to split the document in many small and local parts: the *sites*. Then, the watermarking algorithm modifies some of the sites chosen using a secret key introducing a statistical bias in the document. How to choose and to modify the sites depends only on their local properties. This scheme respects the topology of the vector digital map and the modifications of the vertices positions can be bounded. It exploits also an interesting idea: the whole document quality is preserved by preserving each site local quality. As the sites are defined only on local properties, by construction, the scheme is robust again cropping, data reordering and various geometrical transformations. Furthermore, as it uses a statistical bias, the scheme is robust again low noise addition. Finally, the watermarking and detection algorithms are fast as their complexity behaves quasilinearly with the number of map vertices.

The main goal of this paper is to abstract this method from geographical data and to show how it still applies for geographical data and how it applies for example for databases watermarking. This abstraction uses the notion of *document quality* that is proper to each data type. This notion allows to isolate the data type specific parts of the algorithm. Thus, it is possible to define a watermarking scheme that can be expressed only with the notion of document quality and that does not depend directly on the data type specificity. Thanks to this abstraction, the way to define a watermarking scheme for a new type of document becomes easier. In term of software engineering, it becomes possible to create a templated library based on the watermarking scheme which is reusable on different kinds of data.

Section 2 gives general definitions for watermarking, and precision on the types of documents we watermark. In Section 3, we introduce the abstracted watermarking scheme. We introduce the data-specific parts of the algorithm and define the constraints they must check. As a proof of concept, we show an example of the instantiation of these parts in case of geographical data watermarking and in case of databases watermarking. Section 4 discusses the watermark robustness under various hypothesis.

2 Basic Definitions

2.1 Watermarking Scheme

A watermarking scheme defines two functions: a watermark application function $\mathscr{W}$ and a watermark detection function $\mathscr{D}$. Each of these functions relies on a secret key k chosen in a keys set $\mathcal{K}$. The key is kept secret by the owner of the

watermarked document. Note that a watermarking scheme depends upon the kind of document to watermark. We introduce $\mathcal{D}$ as the finite set of documents to watermark.

A watermarked document must keep its usages. Instead of formalizing directly the usages, we define a property, called *quality* that should remain invariant all through the process. The quality is defined such that of if the quality of a document is preserved by the watermarking process, the document preserves its usages. Thus, we don't need the definition for a single document quality, but only to be able to check quality modifications between two documents.

Definition 1 (Document quality function $Q_{\mathcal{D}}$). *We define $Q_{\mathcal{D}} : \mathcal{D} \times \mathcal{D} \rightarrow \{0,1\}$ so that for any $(d, d') \in \mathcal{D}^2$, $Q_{\mathcal{D}}(d, d') = 1$ if and only if, d' has the same quality as d.*

Note that the quality of a document is relative to another document. Hence, it is possible to check if a document modification preserves the original document quality. The watermark application algorithm must preserve the quality of the original document. Thus for any document $d \in \mathcal{D}$ and any key $k \in \mathcal{K}$, the function $\mathscr{W}$ has to check the relation $Q_{\mathcal{D}}(d, \mathscr{W}(d,k)) = 1$ in order for the watermarking scheme to not degrade the document quality.

Definition 2 (Watermarking Scheme). *The functions $\mathscr{W} : \mathcal{D} \times \mathcal{K} \rightarrow \mathcal{D}$ and $\mathscr{D} : \mathcal{D} \times \mathcal{K} \rightarrow \{0,1\}$ must be defined such that given a random variable D uniformly distributed on $\mathcal{D}$, for all keys $k \in \mathcal{K}$, the scheme satisfies:*

$$\Pr(\mathscr{D}(D,k) = 1 | \nexists d' \in \mathcal{D} \quad D = \mathscr{W}(d',k)) \leq \varepsilon^+$$
$$\Pr(\mathscr{D}(D,k) = 0 | \exists d' \in \mathcal{D} \quad D = \mathscr{W}(d',k)) \leq \varepsilon^-$$

In addition, for any document $d \in \mathcal{D}$ and any key $k \in \mathcal{K}$, the function $\mathscr{W}$ has to check the relation $Q_{\mathcal{D}}(d, \mathscr{W}(d,k)) = 1$

The constant ε^+ represents the probability to detect a watermark in a non-watermarked document, it represents the false-positive rate. ε^- represents the false-negative rate: the probability to not check the presence of a watermark in a watermarked document. The smaller the values of the constants ε^+ and ε^- are, the better the quality of the scheme is.

The following definitions that uses the notion of document quality formalize what is called a robust watermarking scheme.

Definition 3 (Transformation). *We consider here only the transformations that preserve the document quality, i.e. functions $\mathscr{T}$ such that : for any document $d \in \mathcal{D}$, $Q_{\mathcal{D}}(d, \mathscr{T}(d)) = 1$.*

Definition 4 (Robustness). *A watermarking scheme $(\mathscr{W}, \mathscr{D})$ is robust against a transformation $\mathscr{T}$ if given a random variable D uniformly distributed on $\mathcal{D}$:*

$$\Pr\left(\mathscr{D}(\mathscr{T}(D),k) = 0 \middle| \exists d' \in \mathcal{D} \quad D = \mathscr{W}(d',k)\right) \leq \varepsilon^-$$

2.2 Geographical Document

A geographical document is a composition of geographical objects (roads, regions, towns, etc.). Each of these objects is defined with geometries and textual data. A geometry is a collection of either polygons, polylines or points as standardized by the Open Geospatial Consortium[8]. The textual data are the attributes values of the object (roads width, towns populations, etc.). Geographical documents are given with the coordinate systems of the objects (Lambert2, UTM, etc.) and the data accuracy δ. The accuracy is a measure of the maximum errors permitted in horizontal positions and elevations. In this work, we only consider the data parts that defines the documents objects geometries. $\mathcal{D}$ represents the geographical documents set from which it is possible to extract a graph $G = (V, E)$ and an accuracy δ. The graph G is computed using the edges and the vertices of the objects geometries. The graph edges are non-oriented and each graph vertex G has for coordinates (x, y).

The quality $Q_{\mathcal{D}}$ we want to preserve in the geographical document is the graph topology. We preserve this topology by keeping the Delaunay mesh associated to the graph unaltered. Furthermore, we want to bound the loss of accuracy of the watermarked document to $\delta' > 0$ which is a scheme parameter. Thus, the function $Q_{\mathcal{D}}$ is satisfied if and only if the topologies of both documents are equivalents and the maximum displacement between two vertices representing the same information in both documents is lesser than δ'. Finally, the watermarked document accuracy is bounded with $\delta + \delta'$.

The transformations $\mathscr{T}$ we consider are: objects order shuffling, map rotation and map cropping. The objects order shuffling means the modification of the geographical objects order in the document or the modification of the order of the vertices in the geometries. The last attack we consider is the cropping of the document according to a bounding box. These transformations do not alter the document quality $Q_{\mathcal{D}}$.

2.3 Delaunay Mesh

In the geographical document watermarking scheme, we use the mesh built on the vertices extracted from the geographical document. The mesh is known as the Delaunay triangulation [9]. The triangles of the mesh are such that no vertex is inside the circumcircle of any triangle of the mesh. This tends to avoid sliver triangles preferring equilateral triangle. Our watermarking algorithm preserves this triangulation. We consider that preserving the Delaunay triangulation, leads to preserve the document topology. We use the computational geometry library CGAL[10] to compute the Delaunay Triangulation.

2.4 Databases

In this work, we also apply the watermarking scheme on relational databases. We solve the same watermarking problem as the one presented in the paper of Gross-Amblard [11]. The problem consists in watermarking a database relation

while keeping a weight independent sum constraint unchanged. For example, the relation represents the production of many factories. We want to preserve the total production of each factory. But, we allow a small and absolute modification of each production. The global sum request to preserve and distortion allowed might be described using a formal description [12].

Here, the documents set $\mathcal{D}$ represents a vector of tuples indexed by their identifiers. In order to simplify the scheme presentation, we consider that each tuple is a couple (i, b) where i and b represents respectively the identifier and the modifiable bit of the tuple. We assume that $b = 1$ with a probability 0.5. The method is straightforwardly extensible to tuples having several modifiable bits. In the database watermarking scheme we present, the document quality $Q_{\mathcal{D}}$ to preserve is a sum request. In addition, tuple identifiers must be kept unchanged.

3 Abstract Watermarking Scheme Description

3.1 Overview

The main idea of our scheme is to choose small parts of the document (that we call *sites*) using a key and to modify a property local to these parts. A site can be considered as a sub-part of the document.

The section 3.2 defines the notion of site. The first step of our scheme, described in the section 3.3, is to extract the *sites* from the document. Then, we select the sites to be modified using the secret key and a code computed with the site (section 3.4). The watermark will be a modification of the selected sites. In fact, we enforce the sites to satisfy a property Φ_x. The section 3.5 presents how the selected sites are modified. The modified sites are replaced in the document using the replacement function presented in the section 3.6. A site checks the property Φ_x with a given probability. The watermarking algorithm, presented in the section 3.7, modifies this probability. This modification introduces a statistical bias in the selected parts of the document. Thus, with the key, it is possible to check if this statistical bias has been introduced in the document, as presented in the section 3.8. On the counterpart, without the key, it is impossible to check for the watermark presence.

We introduce a notion of quality for a site such that if the sites of a document are modified, as long as the quality of each site is preserved, the quality of the document is preserved. The site modification algorithm ensures that the modification preserves the site quality. Hence, the watermarking algorithm ensures that the whole document quality is preserved.

3.2 Definition of the Notion of *Site*

The proposed watermarking scheme aims to preserve the global document quality by preserving the quality of each site of the document. We introduce the notions of sites, and site quality. The site extraction function will be presented in the section 3.3.

Definition 5 (The sites sets $\mathcal{S}_d$ and $\mathcal{S}$). *The sites set of a document $d \in \mathcal{D}$ is denoted by $\mathcal{S}_d$. The sites set of all documents is denoted by $\mathcal{S} = \bigcup_{d \in \mathcal{D}} \mathcal{S}_d$.*

Definition 6 (The site quality $Q_{\mathcal{S}}$). *The site quality is given by a boolean function. $Q_{\mathcal{S}} : \mathcal{S} \times \mathcal{S} \to \{0, 1\}$ is such that $Q_{\mathcal{S}}(s, s') = 1$ if s and s' have the same quality, $Q_{\mathcal{S}}(s, s') = 0$ otherwise.*

Application to geographical documents. For geographical documents, we define a site for each vertex of the map. A site is composed of a vertex, the neighbors of this vertex in the Delaunay mesh and the edges between all those vertices. More formerly a site is a tuple $(c, (n_1, \dots, n_N), (m_1, \dots, m_N), E_c)$ where:

- $c \in V$, a vertex of the graph.
- $(n_1, \dots, n_N) \subseteq V$, the counterclockwise-ordered neighbors of c in the Delaunay mesh of G.
- $(m_1, \dots, m_N) \subseteq V$, the counterclockwise-ordered mirrors vertices of c in the Delaunay mesh of G. m_i is the mirror vertex of c by the face (c, n_i, n_{i+1}). These vertices represents the "second belt" of vertices around c.
- $E_c \subseteq E$, the edges of E connecting $c, n_1, \dots, n_N$.

Algorithm 1. The function $Q_{\mathcal{S}}$ for the geographical data watermarking scheme

Data: $s = (c, (n_1, \dots, n_N), (m_1, \dots, m_N), E_c) \in \mathcal{S}$: a site
Data: $s' = (c', (n'_1, \dots, n'_{N'}), (m'_1, \dots, m'_{N'}), E_{c'}) \in \mathcal{S}$: the site to compare to s
Result: 1 if the s' as the same quality as s, 0 otherwise
begin
  `// test if the center are not close enough`
  if *distance between c and $c' > \delta'$* **then return** 0 ;
  `// Test if the neighborhood of the sites are the same`
  if $N \neq N'$ **then return** 0 ;
  if $E_c \neq E_{c'}$ **then return** 0 ;
  `// Test if the Delaunay triangulation is respected`
  foreach $i \in \{1, \dots, N\}$ **do**
    if *c' is outside of the circumcircle of n_i, n_{i+1}, m_i* **then return** 0 ;
    if *c' is inside of the circumcircle of n_i, n_{i+1}, n_{i+2}* **then return** 0 ;
  return *1*
end

Algorithm 2. The function $Q_{\mathcal{S}}$ for the databases watermarking scheme

Data: $s = ((i_1, b_1), (i_2, b_2)) \in \mathcal{S}$: a site
Data: $s' = ((i'_1, b'_1), (i'_2, b'_2)) \in \mathcal{S}$: the site to compare to s
Result: 1 if the s' as the same quality as s, 0 otherwise
begin
  return $i_1 = i'_1$ *and* $i_2 = i'_2$ *and* $b_1 + b_2 = b'_1 + b'_2$;
end

The function $Q_{\mathcal{S}}$ is implemented by the algorithm 1. this algorithm compares two sites: s and s'. It checks if the distance between the central vertex of s and s' is not beyond the maximum authorized precision loss δ'. Then, the algorithm checks if both sites have the same topology. Finally, it checks if s' has the same local Delaunay triangulation as s. To test if the Delaunay triangulations are different, we check if the central vertex of s' is inside or outside circumcircles computed using s [13].

Application to databases watermarking. For the database watermarking scheme, the sites set consists in all the possible combinations of two tuples $((i_1, b_1), (i_2, b_2))$ such that $i_1 > i_2$. The idea behind this definition is taken from [11]. This way it is possible to balance the perturbations introduced into a site. It is clear that the preservation of the sum $b_1 + b_2$ leads to the preservation of the sum of all the tuples. The site quality $Q_{\mathcal{S}}$ is implemented using the algorithm 2.

3.3 Site Extraction

The first step of our algorithm is to extract the sites of a document. The number of site in a document is bounded. We denote X the function that can extract all the sites of a document and m the number of sites in a document.

Definition 7 (The site extraction X). *The function $X : \{1, \dots, m\} \times \mathcal{D} \to \mathcal{S}$ extracts the i^{th} site $i \in \{1, \dots, m\}$ from the document $d \in \mathcal{D}$. We have to choose $\mathcal{S}$, $Q_{\mathcal{S}}$ and X so that given two documents d and d':*

$$\Big(\forall i \in \{1, \dots, m\} \quad Qs(X(i, d), X(i, d')) = 1\Big) \implies \Big(Q_{\mathcal{D}}(d, d') = 1\Big) \qquad (1)$$

Application to geographical documents. The X function is implemented in our geographical data watermarking scheme using the algorithm 3. This algorithm requires the use of the Delaunay mesh M of the point cloud V. The mesh is computed once in complexity $O(m. \log(m))$. As the number of faces incident to a vertex can be majored by a constant, the complexity to extract all the sites from a geographical document is $O(m. \log m)$.

In that particular case, the total number of sites is equal to the number of vertices in the document ($m = |V|$). The local quality function $Q_{\mathcal{S}}$ checks, for each vertex of the graph, if the Delaunay triangulation is modified or if the vertex has moved more than the precision δ'. Preserving each site quality will preserve each vertex position within the precision δ' and the Delaunay Thus, the document quality is preserved and (1) is satisfied.

Application to Databases. For the database watermarking scheme, the site extraction algorithm consists in extracting a couple of tuple. For each tuple, we keep the identifier and the modifiable bit value of the tuple. Hence, a site is represented by $((i_1, b_1), (i_2, b_2))$ where $i_1 > i_2$. (i_1, b_1) represents the identifier and the modifiable bit of the first tuple and (i_2, b_2) are the identifier and the modifiable bit of the second tuple.

Algorithm 3. The algorithm of the function X for the geographical watermarking scheme

Data: $G = (V, E)$: a graph
Data: M : the Delaunay mesh associated to the point cloud V
Data: i : the index of the site to construct
Result: s : the i^{th} site in the document
begin
  $c \leftarrow$ The i^{th} vertex in V ;
  $faces \leftarrow$ faces of M incidents to c (ordered counterclockwise) ;
  $j \leftarrow 0$;
  foreach $f \in faces$ **do**
    $n_j \leftarrow$ counterclockwise neighbour of c in the face f ;
    $m_j \leftarrow$ mirror vertex of c by the face f ;
    $j = j + 1$;
  $E_c \leftarrow$ subset of E between the vertices $c, n_0, \ldots, n_N$;
  $s \leftarrow (c, (n_1, \ldots, n_N), (m_1, \ldots, m_N), E_c)$;
  return s ;
end

In this algorithm, we extract $m = \frac{n(n-1)}{2}$ sites, where n represents the number of tuples in the database. Thus, the number of sites extracted behaves quadratically with the number of tuples. That means the complexity of the watermarking algorithm is $O(n^2)$. Such a complexity might be a problem for large databases. Hopefully, the annex presents a solution to reduce the complexity to $O(n)$. This solution doesn't modify the general idea of the scheme.

3.4 Site Selection

The idea is to introduce an artificial statistical bias in a given sites subset. The selection of this subset is based on a secret key and a code computed for each site with the function C. This function must be chosen such that a modification of the site preserving its quality must not modify its encoding. In addition, there must be enough different encodings in a document.

The function P_p partitions the sites set in p disjoint parts using each site code and the secret key. The next step of the watermarking scheme is to modify the sites belonging to a given a part to satisfy a given property and to modify the sites belonging to another partition to not satisfy this property.

Definition 8 (The coding function C). *$C : \mathcal{S} \to \mathbb{N}$ associates a number to a site, such that : $Q_{\mathcal{S}}(s, s') = 1 \implies C(s) = C(s')$ for any sites $s, s' \in \mathcal{S}^2$. Furthermore, the number of sites having the same code in a document must be bounded.*

Definition 9 (The site partitioning function P_p). *Using a secret key $k \in \mathcal{K}$ and a chosen partition size $p \in \mathbb{N}$, we define $P_p : \mathcal{S} \times \mathcal{K} \to \{0, 1, \ldots, p-1\}$ that associates a part number to a site. This function associates a value $v =$*

$H_p(k, C(s))$ to a site $s \in \mathcal{S}$ and a key $k \in \mathcal{K}$. The hash function $H_p : \mathcal{K} \times \mathbb{N} \to \{0, 1, \ldots, p-1\}$ has to be defined such that the images of two different inputs must not be correlated.

Given the definition of H_p, clearly P_p respects the following conditions:

- For a given key k, if two sites s and s' have the same code, whatever the number of parts is, both sites belongs to the same partition.
- If two sites have different codes, they belong to independent parts.
- Without the key knowledge, it is impossible to tell to which part a site belongs to.
- For two different keys, the parts containing a site are independent. So, if someone knows the part associated to a site for a given key, he has no information concerning the part associated to the same site for another key.

Application to geographical documents. In our geographical data watermarking scheme, we base the C function only on local and topological properties of the site. First, for a site $(c, (n_1, \ldots, n_N), (m_1, \ldots, m_N), E_c)$, we associate N matrices $\mathbf{M}_{k \in \{1,\ldots,N\}}$ of size $N \times N$ such that:

$$\mathbf{M}_k = \{a_{i,j}\}_{i,j \in \{1,\ldots,N\}} \text{ and } a_{i,j} = \begin{cases} 1 & \text{if } (j = N) \wedge \big((n_{i+k}, c) \in E_c\big) \\ 1 & \text{if } (j \neq N) \wedge \big((n_{i+k}, n_{i+k+j}) \in E_c\big) \\ 0 & \text{otherwise} \end{cases}$$

The C function computes $\mathbf{M}' = \max_{1 \leq k \leq N}(\mathbf{M}_k)$. Then, it returns a serialized version of the matrix $\mathbf{M}'$. The maximum matrix is used in order to make the encoding algorithm independent from the choice of the first neighbor of the site central vertex. Furthermore, using the maximum matrix, the encoding algorithm is robust against rotation. The site code is based on the matrix definition. This definition is based upon the site edges and the Delaunay triangulation. The site quality definition requires that both sites having the same quality must have the same edges and local Delaunay triangulation. Thus, two sites having the same quality, have the same code.

Application to Databases. The site is composed of two tuples (i_1, b_1) and (i_2, b_2), with $i_1 > i_2$. The coding algorithm we propose is to simply return an hashing value of (i_1, i_2). Given the site definition, two sites have the same quality only if their identifiers are the same. And, if the identifiers are the same, the coding of both sites are the same. Thus, for two sites $(s, s') \in \mathcal{S}$ with the same quality $Q_{\mathcal{S}}(s) = Q_{\mathcal{S}}(s')$, the coding of these sites are identical $C(s) = C(s')$.

3.5 Statistical Bias Introduction

Using the partitioning algorithm, the sites are distributed among many groups. The sites of the first and second parts are modified in order to respectively satisfy the properties Φ_0 and Φ_1. In this section we present a way to take advantage of

the randomness of some of these parts. First, we must find two site properties Φ_0 and Φ_1 such that the probability that a site satisfies Φ_0 or Φ_1 corresponds to two Bernoulli trials of respective parameters μ_0 and μ_1. Furthermore, a site must not satisfy at the same time both properties Φ_0 and Φ_1.

In order to operate on these two parts, we define for $x = 0$ and 1 two functions T_x and M_x. The function T_x is used to test if the site satisfies the property Φ_x. The function M_x try to enforce a site to satisfy Φ_x. The site modifications must always preserve the site quality. Thus, these functions may fail to enforce certain sites to satisfy Φ_x. Function M_x allows us to introduce a statistical bias ε in the data.

Definition 10 (the test functions T_x). *For $x = 0, 1$, the function $T_x : \mathcal{S} \times \mathcal{K} \to \{0, 1\}$ must be independent from the site quality. It must return 1 if the site satisfies Φ_x, otherwise 0. We pick up with uniform distribution over $\mathcal{S}$, for any key $k \in \mathcal{K}$, $\Pr(T_x(S, k) = 1) = \mu_x$.*

Definition 11 (The modification function M_x). *We denote by $M_x : \mathcal{S} \times \mathcal{K} \to \mathcal{S}$, a function that tries to enforce a site to satisfy Φ_x. M_x must be defined such that, given a random variable S uniformly distributed over the sites of a non-watermarked document and for two different keys $(k, k') \in \mathcal{K}^2$:*

$$\Pr(T_x(M_x(S, k), k') = 1) = \mu_x$$
$$\textit{But } \Pr(T_x(M_x(S, k), k) = 1) > \mu_x + \varepsilon$$

In addition, M_x must preserve the quality of the site. In other words, for any site $s \in \mathcal{S}$, this function have to satisfy $Q_{\mathcal{S}}(s, M_x(s)) = 1$.

The idea to define a property, a test function and a modification function is taken from the article [6]. We added the use of a secret key in the functions definitions in order to prevent an attacker to take advantage of the knowledge of which sites satisfy Φ_0 or Φ_1. Indeed, if many sites of the documents have the same encoding, they belong to the same partition, whatever the key is. When many sites have the same encoding, an attacker may wash the document using the modification functions in order to enforce each sites group having the same encoding to check Φ_0 or Φ_1 with the probabilities μ_0 and μ_1. This fault is corrected when we use a secret key for the properties definitions.

Application to geographical documents. In the case of geographical data watermarking, the properties Φ_0 and Φ_1 are inspired from the paper [6]; but, we give here a self-contained presentation of these functions. The properties use the discrete distance between the site central vertex and a reference vertex. The reference vertex is computed using an hash code of the key and the site encoding.

The function T_0 is defined by the algorithm 4. This algorithm computes the parity of the discrete distance between the central vertex of the site c and the reference vertex. As the reference vertex is computed using an hash function, we can make the assumption that Φ_0 follows a Bernoulli distribution of parameter $\mu_0 = 0.5$.

Algorithm 4. The T_0 function algorithm for the geographical data watermarking scheme

Data: $s = (c, (n_1, \dots, n_N), (m_1, \dots, m_N), E_c)$: a site
Data: $k \in \mathcal{K}$: a key
Result: True if s satisfies Φ, False otherwise
begin
  $x \leftarrow \text{hash}(k, C(s))$;
  $y \leftarrow \text{hash}(\text{hash}(k), C(s))$;
  $dist \leftarrow$ distance between (x, y) and c;
  return *True if* $\lfloor \frac{dist}{\delta'} \rfloor$ *is odd, False otherwise* ;
end

Algorithm 5. The M_0 function algorithm for the geographical data watermarking scheme

Data: $s = (c, (n_1, \dots, n_N), (m_1, \dots, m_N), E_c)$: a site
Data: $k \in \mathcal{K}$: a key
Result: s': a site
begin
  if $T(s) = 0$ **then return** s ;
  $x \leftarrow \text{hash}(k, C(s))$;
  $y \leftarrow \text{hash}(\text{hash}(k), C(s))$;
  $c' \leftarrow c$ moved toward (x, y) by δ' ;
  $s' \leftarrow (c', n_1, \dots, n_N, m_1, \dots, m_N, E_c)$;
  return s' *if* $Q_{\mathcal{S}}(s, s') = 1, s$ *otherwise* ;
end

The property Φ_1 is satisfied if and only if the property Φ_0 is not satisfied. Consequently the function T_1 is the opposite of T_0 and the probability that a site satisfies Φ_1 is $\mu_1 = 0.5$.

The algorithm 5 implements the function M_0. Given a site s, if it checks Φ_0, s is returned unchanged. Otherwise, the algorithm returns a site s' similar to s except for the site central vertex that is moved towards the reference vertex. The M_1 function algorithm is similar except that in case s has to be modified, its central vertex is moved away from the reference vertex. In both cases, M_1 and M_1 return the original site instead of the modified one if the quality would be altered.

Application to databases. Here, the function T_0 returns 1 if and only if, given a site $s = ((i_1, b_1), (i_2, b_2))$, $b_1 = 0$ and $b_2 = 1$. The function M_0 enforces the bits b_1 and b_2 to be respectively equal to 0 and 1. On the opposite, T_1 returns 1 if the $((i_1, b_1), (i_2, b_2))$ satisfies $b_1 = 1$ and $b_2 = 0$. M_1 sets the modifiable bits of the site to $b_1 = 1$ and $b_2 = 0$.

The function M_0 and M_1 return the modified site if its quality is equivalent to the quality of the original one. We assume the value of the modifiable bit of

tuple follows a Bernouilli distribution of parameter 0.5. Thus, the probability for a site to satisfy Φ_0 or Φ_1 is 0.25.

In this case, as the coding function is injective (no collision), an attacker will have no information about the parts choosed to enforce Φ_0 or Φ_1. Consequently, there is no need for a key to hide the properties.

3.6 Site Replacement

The last step of the watermarking is to replace the sites modified by the functions M_0 and M_1 in the document. The replacement function takes a rank i, a document and a site and returns the document with the i^{th} site replaced by the new site. The site replacement function must respect the quality of the document. It means that if someone replace a site with a site of similar quality, the replacement function will not change the document quality. Due to performance constraints, the replacement is done in-place in the document d.

Note that we don't really need to identify the sites with their ranks but simply be able to enumerate them; the use of the rank simplifies the following definition

Definition 12 (The site replacement function R). *We introduce the function $R : \{1, \dots, m\} \times \mathcal{D} \times \mathcal{S} \to \mathcal{D}$. For any rank i, any document d and any site s, $R(i, d, s)$ returns a document similar to d except for its i^{th} site being replaced by s. For any site $s \in \mathcal{S}$, any document $d \in \mathcal{D}$ and any value $i \in \{1, \dots, m\}$, R respects that :*

$$Q_{\mathcal{S}}\big(X(i,d), s'\big) = 1 \quad \Longrightarrow \quad Q_{\mathcal{S}}\big(X(i,d), X(i, R(i,d,s'))\big) = 1 \tag{2}$$

Application to geographical documents. For the geographical data watermarking scheme, the site replacement function R is implemented using the algorithm 6. This algorithm modifies the i^{th} vertex document coordinates by the new site central vertex coordinates. Using this algorithm, and given the definition of $Q_{\mathcal{S}}$: The relation (3) holds and the algorithm respects the relation (2).

$$Q_{\mathcal{S}}\big(X(i,d), s\big) = 1 \quad \Longrightarrow \quad X\big(i, R(i,d,s)\big) = s \tag{3}$$

Application to databases. Here, the replacement function R of a given site $((i_1, b_1), (i_2, b_2))$ consists in setting the modifiable bits of the databases tuples identified by i_1 and i_2 respectively to the value of the bits b_1 and b_2.

3.7 The Watermark Application Algorithm

The full watermarking algorithm is obtained by application of the functions X, C, R, M_0, M_1 and R. It first builds the sites partition using the secret key. Then the sites belonging to the two first parts are enforced to respectively check Φ_0 and Φ_1. The algorithm is generic and independent from the kind of document to watermark. Only the sub-algorithms X, C, M_0, M_1 and R depend upon the document type and have to be defined for a new type of document.

Algorithm 6. The R function algorithm for geographical documents

Data: $i \in \{1, \ldots, m\}$: the index of the site to modify
Data: $d \in \mathcal{D}$: the document to be modified
Data: $s = (c, (n_1, \ldots, n_N), (m_1, \ldots, m_N), E_c)$: the new value of the site
Result: $d' \in \mathcal{D}$: the modified document
begin
  Assert $Q_{\mathcal{S}}(X(i, d), s) = 1$;
  $p \leftarrow$ the i^{th} vertex of the document ;
  Modify d such that the coordinates of p becomes the coordinates of c;
  return d
end

Algorithm 7. The watermark application algorithm

Data: $d \in \mathcal{D}$: the document to be watermarked, contains m sites
Data: $k \in \mathcal{K}$: the key
Result: $w \in \mathcal{D}$: the watermarked document
begin
  $w \leftarrow \text{copy}(d)$;
  foreach $i \in \{1, \ldots, m\}$ **do**
    $s \leftarrow X(i, w)$;
    $j \leftarrow P_p(s, k)$;
    if $j = 0$ **then** $w \leftarrow R(i, w, M_0(s, k))$;
    else if $j = 1$ **then** $w \leftarrow R(i, w, M_1(s, k))$;
  return w ;
end

3.8 The Watermark Detection Algorithm

The watermark detection algorithm has also a generic definition. It first selects and partition the sites using the same method as for the watermarking algorithm. Then it uses a Chernoff bound in order to detect a bias in the distribution of the property among the parts.

We assume that the probabilities that a site satisfies the property Φ_0 and Φ_1 follow a Bernoulli law with respective parameter μ_0 and μ_1 and the sites are independent Bernoulli random variables. Let E be a set of n sites, S_n be the random variable of the number of sites which satisfies the property Φ_x, $x = 0$ or 1. For any $0 \leq n_p \leq n$, we apply the following Chernoff bound which provides an upper bound of the probability that S_n is greater or equal to n_p: $\Pr(|n\mu_x - S_n| \geq c\sqrt{n} \leq 2\ e^{-2c^2}$, where $c = \frac{|n\ \mu_x - n_p|}{\sqrt{n}}$. The Test function computes this Chernoff bound:

$$Test(E, \mu_x, T_x, k) = 2\ e^{-2\ \frac{(n\mu_x - n_p)^2}{n}}$$

Using the functions X, C and $Test$, we define the generic watermark detection algorithm 8. This algorithm computes the probability there is no watermark

Algorithm 8. The watermark detection algorithm

Data: $d \in \mathcal{D}$: the document to check for watermark, d contains m sites
Data: $k \in \mathcal{K}$: the secret key
Result: $True$: if the document seems watermarked with key k, $False$ otherwise
begin
  $part_0 \leftarrow part_1 \leftarrow \emptyset$;
  foreach $i \in \{1, ..., m\}$ **do**
    $s \leftarrow X(i, d)$;
    $j \leftarrow P_p(s, k)$;
    if $j = 0$ **then** add s in $part_0$;
    else if $j = 1$ **then** add s in $part_1$;
  return $Test(part_0, \mu_0, T_0, k) * Test(part_1, \mu_1, T_1, k) < \lambda$
end

inserted in a given document. The value of $\lambda \in [0, 1]$ represents the watermark detection threshold. This threshold must be well choosen. If its value is too high, the chances to detect a watermark in a non watermarked document increases. On the contrary, if its value is too low, the chances for not detecting a watermark in a watermarked document increases.

The exact choice for this threshold value depends on the properties of the given corpus of document to watermark and on the confidence degree we want to obtain for the scheme.

4 Watermark Resilience

In this section we present the watermark resilience to various transformations. We consider here mainly natural transformations implied by an ordinary usage of the documents. Thanks to the abstract scheme we are able to reveal the specific part that depends on an application from the generic part relevant to any application. This allows a better study of the resiliency level.

The specific part requires some knowledge about the kind of document. Indeed, each kind of document has its own set of transformations. Each of these transformations induces a sites set modification. It is mandatory to quantify the sites set modifications induced by a given transformation in order to quantify the resiliency level. For example, a cropping of a digital map is viewed as the deletion of the sites out of the cropped region and the modification of the site located at the boundary. The quantification of the alteration can be done by experiments over one or several documents. If a transformation implies too much degradation of the sites set, it may be interesting to reconsider the site definition.

In addition, we can investigate how the sites set transformations influence a possible subsequent watermark detection. At this step, we may conduct the study without direct reference to the document content, but only in terms of sites manipulation. Sites may be added to or removed from the document or simply modified. The table 1 illustrates the way those sites set alterations influence the Chernoff bound. For these experiments, we set the sites set size to $m = 10000$,

Table 1. Exemple of influence of the sites set modification to the detection algorithm ($m = 10000$, $p = 10$, $\mu_0 = 0.5$ and $\mu_1 = 0.5$. The success ratio of M_0 and M_1 is 80%.)

Sites set modification	Probability that the document is not watermarked
No modification	$< 10^{-156.3}$
Removing 1000 sites	$< 10^{-140.7}$
Removing 5000 sites	$< 10^{-78.2}$
Removing 9000 sites	$< 10^{-15.6}$
Modifying 1000 sites	$< 10^{-126.6}$
Modifying 5000 sites	$< 10^{-39.1}$
Modifying 7000 sites	$< 10^{-14.1}$
Modifying 9000 sites	$< 10^{-1.6}$
Adding 1000 sites	$< 10^{-142.1}$
Adding 10000 sites	$< 10^{-78.2}$
Adding 100000 sites	$< 10^{-14.2}$
Adding 200000 sites	$< 10^{-7.4}$

the number of parts to $p = 10$, and $\mu_0 = \mu_1 = 0.5$. Furthermore, we make two assumptions. First, the sites are equally distributed among the 10 parts; second, the percentage of sites successfully modified by M_0 and M_1 is 80%. Those values might represent the scheme applied to a geographical document containing 10 000 vertices. This corresponds to the roads of few square kilometers of a urban area.

We have considered in this part the resiliency against natural transformations. But an attacker can define his own transformations. It seems difficult (impossible?) to guess which attacks an attacker may attempt. An attack can contain random steps and be concentrated in very specific parts of the document. Nevertheless, we can circumscribe its power. Indeed, an acceptable transformation should preserve the document quality and we could precise which kind of transformations does not damage it. This restriction will allow to model the attack in terms of sites set modifications.

5 Conclusion and Further Work

In this paper, we introduce a generic watermarking scheme. This scheme aims to watermark a document while preserving its quality. In order to preserve the whole document quality, we split the document in small parts : the sites. Then, we must define the site quality so that if every sites quality is preserved, the whole document quality is preserved. The site quality helps to define the data-type specific functions used in the watermarking process. Of course, the scheme is only applicable if one can express the document quality in term of site quality.

We introduced two applications of the general scheme: a geographical data watermarking scheme that preserves the Delaunay mesh and a database watermarking scheme preserving a sum query. As both these schemes are based upon the general watermarking scheme they are blind, they guaranty the document

quality and have a low complexity. Furthermore, we can express the scheme robustness in term of sites set modifications. By example, we show how the sites set modifications impact the detection algorithm.

It is essential to note that the scheme generalization presents an unified view of various watermarking techniques operating on different data types. First, this approach leads to a better understanding and enhancement of existing watermarking schemes; then, it allows to develop new ones that inherit the good resilience properties described in the previous section. In addition, it becomes possible to build formal attacks and counter-measures on the watermarking scheme independently from the data specificity and from the algorithm implementation.

Furthermore, some studies may be conducted using only the data type abstracted scheme. For example, we are currently working on a watermark-detection protocol based upon the abstracted scheme presented in this paper. This protocol involves a third party who solely knows the secret key and only needs some encoding of the document to prove the document is watermarked or not, without knowing the whole document. It is interesting to note that this detection algorithm will be directly applicable to the geographical watermarking scheme and the database watermarking scheme.

References

1. Cox, I., Miller, M., Bloom, J.: Digital Watermarking. Morgan Kaufmann, San Francisco (2001)
2. Bender, W., Gruhl, D., Morimoto, N., Lu, A.: Techniques for data hiding. IBM Systems Journal 35, 313–336 (1996)
3. Niu, X., Shao, C., Wand, X.: A survey of digital vector map watermarking. ICIC International (2006)
4. Ohbuchi, R., Ueda, H., Endoh, S.: Watermarking 2d vector maps in the mesh-spectral domain. In: Shape Modeling International (2003)
5. Lafaye, J., Beguec, J., Gross-Amblard, D., Ruas, A.: Invisible graffiti on your buildings: Blind & squarring-proof watermarking of geographical databases. In: Papadias, D., Zhang, D., Kollios, G. (eds.) SSTD 2007. LNCS, vol. 4605, pp. 312–329. Springer, Heidelberg (2007)
6. Bazin, C., Le Bars, J.M., Madelaine, J.: A fast, blind and robust method for geographical data watermarking. In: ASIACCS 2007 (2007)
7. Agrawal, R., Kiernan, J.: Watermarking relational databases. In: 28th International Conference on Very Large Databases (VLDB), vol. 2. IEEE, Los Alamitos (2002)
8. Open Geospatial Consortium, `http://www.opengeospatial.org`
9. Georges, P.L., Borouchaki, H.: Triangulation de Delaunay et maillage. Hermes (1997)
10. Yvinec, M.: 2d triangulations. In: CGAL User and Reference Manual. 3.3 (2007)
11. Gross-Amblard, D.: Query-preserving watermarking of relational databases and xml documents. In: PODS 2003, San Diego, CA (2003)
12. Constantion, C., Gross-Amblard, D., Guerrouani, M.: Watermill: an optimized fingerprinting system for highly constrained data. In: ACM multimedia and security workshop (2005)
13. Dakowicz, M., Gold, C.: Structuring kinetic maps. In: Progress in Spatial Data Handling, University of Glamorgan, pp. 477–493 (2006)

A Annexes

A.1 Decreasing Database Watermarking Scheme Site Number

We consider the document is a set of n tuples where each tuple has an identifier. Let q be a parameter constant, we distribute the tuples among $\frac{n}{q}$ parts according to k bits of the identifier hash whose length is at least $\log_2 n$ and where k satisfies $\frac{n}{q} = \lceil 2^k \rceil$. We may assume that each part has a size close to q. By definition a site is a pair of elements belonging to the same part. Hence the numbers of sites is around $\binom{q}{2} \frac{n}{q} = n\,(q-1)$ which is linear on the number of tuples.

The Elliptic Curve Discrete Logarithm Problem: State of the Art

Alfred Menezes

Department of Combinatorics & Optimization
University of Waterloo
ajmeneze@uwaterloo.ca

Abstract. Elliptic curve cryptography was invented by Neal Koblitz and Victor Miller in 1985. However, for many years its security was viewed with suspicion by cryptographers. These fears were not entirely meritless, especially in light of the subsequent (and ongoing) development of innovative algorithms for solving special instances of the elliptic curve discrete logarithm problem (ECDLP) and the discrete logarithm problem in the divisor class groups of other families of algebraic curves. Surprisingly, cryptographers seem to have cast aside their fears after the invention of pairing-based cryptography in 2000. I will overview progress made on algorithms for solving the ECDLP and related problems, and will examine the long and curious struggle for ECC acceptance.

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, p. 218, 2008.

An Application of the Boneh and Shacham Group Signature Scheme to Biometric Authentication*

Julien Bringer[1], Hervé Chabanne[1], David Pointcheval[2], and Sébastien Zimmer[2]

[1] Sagem Sécurité
[2] École Normale Supérieure

Abstract. We introduce a new way for generating strong keys from biometric data. Contrary to popular belief, this leads us to biometric keys which are easy to obtain and renew. Our solution is based on two-factor authentication: a low-cost card and a biometric trait are involved.

Following the Boneh and Shacham group signature construction, we introduce a new biometric-based remote authentication scheme. Surprisingly, for ordinary uses no interactions with a biometric database are needed in this scheme. As a side effect of our proposal, privacy of users is easily obtained while it can possibly be removed, for instance under legal warrant.

Keywords: Biometric Data, Privacy, Group Signature.

1 Introduction

Authentication is a central problem in cryptography and many cryptographic tools have been created to established authenticated channels. Since these cryptographic tools are often difficult to attack, an adversary may prefer concentrate her efforts on a weaker part of the authentication process. This is the goal of social engineering attacks which focus on the user mistakes to discover his password. There is no cryptographic solution against this type of attacks, one has to deploy external security protections. However, a way to strengthen the security level of authentication procedures is to combine several authentication means in such a way that, to impersonate an honest user, the adversary is compelled to break *all* the authentication factors.

Traditionally, three possible human authentication factors are distinguished (even if a forth one has already been introduced [5]):

- what I *know* (as a password),
- what I *have* (as a card),
- who I *am* (as a biometric trait).

* Work partially supported by french ANR RNRT project BACH.

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 219–230, 2008.

All the combinations can be used, however in this paper we focus on a two-factor authentication using a low-cost card ("what I have"), such as a plastic card, and a biometric trait ("who I am").

Biometric authentication, the "who I am" part of the tryptic, has less to do with cryptographic techniques. This is mainly due to the fact that biometric data involved during the authentication must be considered as public data if a high security level is required. Indeed, they are easy to obtain by an adversary: for example a fingerprint can be recovered from any object that have just been touched and an image of an iris scan can be taken with a good camera. This rises two main problems.

Firstly, since an adversary can recover biometric data, no secrecy is used during the biometric authentication procedure. Therefore, what prevents the adversary to use some (public) biometric templates to impersonate an honest user? Biometrics are valuable for authentication only if one can check that the biometric template used during authentication comes directly from a real living person and not from a fake. Both technical and cryptographic solutions exist to guarantee integrity of biometric data (authenticated channel, supervision of the biometric acquisition by a trusted human, acquisition of several biometrics at the same time, tamper proof resistance of the sensor, ...). The assumption that the biometric template comes directly from a real person is called the *liveness assumption*. The liveness assumption while mandatory for biometric systems is ensured by technical means beyond the scope of this paper and not described here.

Secondly, since the (public) biometric data is unequivocally linked to a person, if some information about the biometric data is leaked during the protocol, then the identity of the user involved in the authentication could be revealed. To protect the privacy of the users, authentication should stay anonymous and the biometric data involved should stay private. Our work thus focuses on the integration of biometric data into cryptographic protocols to respect the privacy of their participants.

1.1 Our Solution

In this paper we propose a two-factor biometric authentication protocol. As no processing power is needed on the side of the user except from the sensor, one can imagine to simply write the reference biometric template on a plastic card. The contribution of this paper is two-fold: first, we propose a simple way to generate strong biometric secret keys, and second, we integrate it in a new authentication protocol which guarantees client anonymity and some other nice features described below.

The protocol presented in this paper is the consequence of a basic observation: the biometric measures are prone to a large amount errors which cannot reasonably be predicted. In other words, the global error between two measures has a high entropy, therefore, even knowing the distribution of the biometrics, an adversary cannot guess the exact value of a biometric template with a good precision.

Traditionally, the errors are seen as an obstacle but, with this remark in mind, one can try to take benefits of these errors to introduce a secret in the biometric part of the authentication protocol, secret which does not exist without it. If the reference biometric template is acquired privately, then its exact value cannot be guessed by an adversary. We propose to hash this template to generate a secret key. More precisely, the reference biometric template is stored in the plastic card and it is compared with a fresh biometric template by the sensor. The matching is made on the client side and if this new template comes from the same biometric trait as the reference template, then the reference template is hashed to regenerate the secret key. The latter is used to authenticate the user to the server.

Another simple solution would be to hide the reference biometric template and any random secret key on a card. However, it requires additional infrastructure to bind the template and the key together (e.g. a PKI). Indeed, without any link between these elements, an adversary can read the key in the card, replace the reference biometric template by one of her own and then she is able to authenticate to the server with the same secret key. In other words, the security relies only on the card and not also on the biometric data. Whereas, our solution makes profit of the properties of the biometrics which link unequivocally the secret key and the biometric data, we guarantee a secure two factor authentication.

Besides, to complete the authentication protocol, we propose to use the Boneh and Shacham group signature scheme [2]. This scheme has several nice features that can be used in our context. First of all, it allows to guarantee the anonymity of the client towards the server. This is the main property that one want to preserve. Moreover, it allows to revoke compromised keys by generating a Revocation List: compromised keys are revoked and added to the list, so that the server can check online if an adversary tries to use one of the compromised keys to authenticate. Since the only secret is the error and not the biometric data, to generate a new secret key one only has to acquire a new reference biometric template from the same biometric trait. Finally, if the entity which generates the keys, the Card Issuer, is allowed to securely store for every user the reference biometric template, then under legal warrant it can reveal to whom some secret key belongs.

1.2 Related Works

In order to integrate biometrics into cryptographic protocols, it is often proposed to replace traditional matching algorithms by error-correction techniques [3,4,6,7,8,9,10,11,12,17,18,19] and to use Secure Sketches [9] instead of traditional template generators. The main problem with these techniques comes from the use of decoding algorithms instead of classical matching algorithms. This doing, performances are degraded. Moreover, for some kind of biometrics, as for instance for the worldwide deployed fingerprints' minutiae, theoretical solutions have recently been proposed [14] but effective coding solutions – which achieve performances comparable with existing matching based solutions – are still to be found.

1.3 Organization of This Paper

In Section 2, we explain the principles of our solution. In Section 3, we give statistical evidence on biometric data to justify our work. In Section 4, we introduce a new protocol for secure remote biometric authentication with our solution. Finally, we conclude in Section 5.

2 Our Procedure in a Nutshell

Let B denotes a biometric trait. We write:

- $b, b', \ldots \leftarrow B$ for different acquisitions of the same biometric trait B,
- $b \sim b'$ indicates that a matching algorithm will consider b and b' as belonging to the same biometric source B.

During enrollment phase: A biometric trait is acquired: $b \leftarrow B$. This particular acquisition is treated as confidential (see justification in Section 3.2). A copy of b is kept by his owner on a plastic card. Let x denote $x = H(b)$, a cryptographic hash of b.

The main idea here is that whereas the trait B is considered as public, it is not the case of one acquisition b.

During verification phase: A user comes with his card containing the biometric reference b. The same biometric trait is acquired: $b' \leftarrow B$. If $b \sim b'$, a remote cryptographic proof of knowledge of x is made in order to validate the verification.

It means we first run a biometric matching for a local authentication and thereafter we can authorize the use of $x = H(b)$ for a remote authentication.

3 Some Elements on Volatility of Biometric Acquisitions

3.1 An Example

We give an example based on the iris biometrics to illustrate the variations occurring between two acquisitions of the same biometric trait. Iris is often considered as the easiest biometric technology to integrate into cryptographic protocols as it encodes biometric trait as binary vectors and as the matching is made by computing a simple Hamming distance (see below).

What we say for iris is also true for other kind of biometric data. But often in these cases, as for instance for the minutiae matching of fingerprints, we do not even know how to handle them efficiently as binary vectors.

We report some results on a public iris database. This is called the Iris Challenge Evaluation (ICE) database and is used to evaluate matching algorithms [13,15]. It contains 2953 images from 244 different eyes. For each picture, we compute a 256 bytes information vector I together with a 256 bytes mask vector M. The mask vector indicates for each bit whether or not some information is

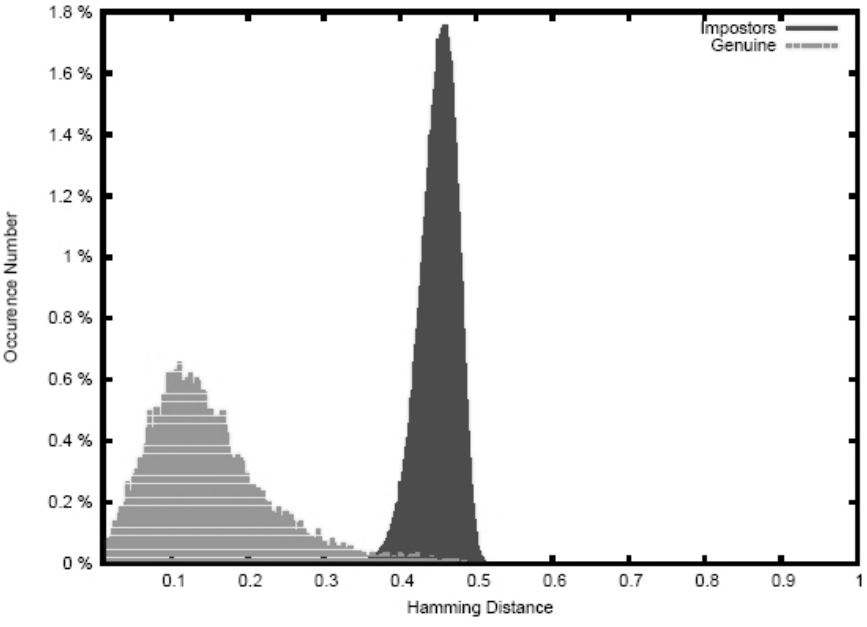


Fig. 1. Inter-eyes and intra-eye distributions

available in I (due to eyelid, eyelashes, reflections, some bits may be missing, ...). The matching of two iris b_1 and b_2 represented as I_1, I_2 with masks M_1, M_2 is done by computing their relative Hamming distance, i.e. the Hamming distance computed over the portion of the information vectors not erased by masks:

$$\mu(b_1, b_2) = \frac{||(I_1 \oplus I_2) \cap M_1 \cap M_2||}{||M_1 \cap M_2||} \tag{1}$$

This leads to the following distributions of matching scores (cf. Fig. 1) where we observe a large number of errors to handle. For instance, if we accept to wrongly reject at most 5 % of the users, we then have to deal with at least 29 % of errors, i.e. up to $\mu(b_1, b_2) = 0.29$. An additional difficulty comes from the number of bits where no information is available which varies from 512 to 1977.

3.2 How Can Volatility Help us to Secure x?

Let $b_i \leftarrow B$, $i = 1, \ldots, l$ be different acquisitions of the same biometric trait. As it is indicated in the introduction, we consider that an adversary may have access to some of these b_i.

Below we explain how different can be two biometric captures of the same biometric trait.

Suppose we are dealing with binary vectors of length n and that any two matching data $b \sim b'$ have more than ϵn differences, then to retrieve b from the

knowledge of b' there is at least $\binom{n}{\epsilon n}$ possibilities which correspond to switch the value of ϵn coordinates of b' (we assume for the moment errors to be uniform and independent random bits). More precisely an adversary has to search among the vectors of the Hamming spheres $S(b', r)$ of center b' and radius $r \geq r_0$, starting with $r_0 = \epsilon n$.

For instance, for the ICE iris database introduced in the previous section 3.1, we have $n = 2048$ and the distance, computed as in (1) only on the non-erased positions, varies from 44 to 658, amongst the 26874 possible comparisons with two non-equal matching data. This means that without taking masks into consideration, the " closest " b_i differs from b in 44 bits, i.e. $\epsilon \approx 2.15\%$. Thereafter an adversary must switch at least 44 coordinates of b_i to recover b, this leads to $\binom{2048}{44} \approx 2^{302}$ possibilities. Moreover, the masks are also different between two biometric measures, so he will have to correct their differences. Let M_b, M_{b_i} the masks of b and b_i, even if he knows M_b, he will need to choose a value in $\{0, 1\}$ for each erased position of b_i which was not erased for b, i.e. in $\overline{M_{b_i}} - (\overline{M_b} \cap \overline{M_{b_i}})$. The overall number of possibilities is

$$\binom{||M_b \cap M_{b_i}||}{\mu(b, b_i) \times ||M_b \cap M_{b_i}||} \times 2^{||\overline{M_{b_i}} - (\overline{M_b} \cap \overline{M_{b_i}})||}.$$

In the previous example it leads to about 2^{539} possibilities as the number of common non-erased positions is 1056 and the number of differences between the masks is 280. For all the database, the minimum is around 2^{500}. So, in practice even for errors not "uniformly" random[1], the number of possibilities might stay very large.

In practice an adversary may try to reduce the complexity of recovering b by collecting several different b_i. In general, it would remain hard to recover b whereas for situations where variability is not sufficient we can embed additional random bits under the erased coordinates of b (at least 512 positions for the ICE database). This allows to rely on a random data while it stays transparent for the biometric matching (cf. eq. (1)).

4 An Application to Secure Remote Biometric Authentication

4.1 Introduction

In our system for biometric-based authentication schemes, we consider four types of components.

- Human user $\mathcal{H}$, who uses his biometric data to authenticate himself to a service provider.
- Sensor client $\mathcal{S}$, which extracts human user's biometric trait using some biometric sensor and communicates with the service provider.

[1] Which seems more realistic however the entropy of errors between two biometric captures is very hard to estimate.

- Service provider $\mathcal{P}$, who deals with human user's authentication request, granting access or not.
- Card Issuer $\mathcal{I}$, who holds two master secrets: γ which is needed to derive keys in the scheme (see below and next Section 4.2) and λ which is the private key only used in case of legal warrant, with Λ the corresponding public key (see Section 4.3).

Remark 1. *In our system, we make a separation between everyday interactions where user $\mathcal{H}$ deals with his service provider $\mathcal{P}$ and exceptional procedures. For daily authentications, our proposal provides privacy for $\mathcal{H}$. However, in case of legal warrant, Card Issuer $\mathcal{I}$ has the capability – by using his cryptographic keys – to retrieve who is authenticating himself (cf. Section 4.3).*

The authentication link we want to establish is the following:

Human user $\leftrightarrow$ his biometric trait
$\leftrightarrow$ his biometric key (cf. Section 2): x
$\leftrightarrow$ his private key in the system (cf. below): (x, A)

Hereafter, the x's we have introduced before serves as part of a private key for a group signature [2]. A group signature scheme enables a member of a group to anonymously sign a message on behalf of the group (here, the users who has subscribed to $\mathcal{P}$). A is derived from x by the Card Issuer $\mathcal{I}$ under his master secret γ. The pair (x, A) forms the private key of $\mathcal{H}$ in this system. Details are given in the next section. At this point, we only need to know that (x, A) verifies an algebraic relation:

$$A^{x+\gamma} = g_1 \tag{2}$$

where g_1 is a public parameter. The cryptographic part of the authentication consists in convincing $\mathcal{P}$ that $\mathcal{H}$ holds a private key verifying such a relation.

Keeping notations from previous sections, we have:

Enrollment phase at Card Issuer's facilities: A biometric trait is acquired for user $\mathcal{H}$: $b \leftarrow B$. The Card Issuer $\mathcal{I}$ computes A with the help of γ. A plastic card $\mathcal{C}$ containing (b, A) is issued by $\mathcal{I}$ for $\mathcal{H}$. The Card Issuer keeps b in a database $\mathcal{DB}$ of his own.

Remark 2. *Following a standard trick of the biometric world, it is also possible to envisage that $\mathcal{DB}$ is used during enrollment phase to ensure that none user registers twice.*

During verification phase:

1. A sensor $\mathcal{S}$ takes "fresh" biometric trait b' of a user $\mathcal{H}$. $\mathcal{S}$ verifies the liveness of b'.
2. A match of b' against the b stored in the plastic card $\mathcal{C}$ is performed.
3. In case of success, a proof of knowledge of the private key (x, A) is made by signing a challenge sent by the service provider $\mathcal{P}$ following the Boneh and Shacham scheme [2] which is described in the next Section.

4.2 The Boneh-Shacham Group Signature Scheme

System parameters

- $\mathbb{G}_1$ is a multiplicative cyclic group of prime order p.
- $\mathbb{G}_2$ is a multiplicative group whose order is some power of p.
- ψ is a homomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$.
- g_2 is an order-p element of $\mathbb{G}_2$ and g_1 is a generator of $\mathbb{G}_1$ such that $\psi(g_2) = g_1$.
- $\mathbb{G}_T$ is a multiplicative cyclic group of prime order p.
- $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear non-degenerate map.
- H is a hash function from $\{0,1\}^*$ to $\mathbb{Z}_p$ and H_0 is another hash function mapping $\{0,1\}^*$ to $\mathbb{G}_2^2$.

Some other elements

- group public key: $gpk = (g_1, g_2, w)$ where $w = g_2^\gamma$ for some secret $\gamma \in \mathbb{Z}_p$.
- private key: a pair (x, A), where $A \in \mathbb{G}_1$ and $x \in \mathbb{Z}_p$ verifying (2), i.e. $e(A, wg_2^x) = e(g_1, g_2)$. Given x, A must be computed by the owner of γ.

We are now ready to recall the principles of the signature from [2]. We here consider that the verifier sends a challenge M to be signed by the prover.

Signature of M. The prover obtains generators

$$(\hat{u}, \hat{v}) = H_0(gpk, M, r) \in \mathbb{G}_2^2 \tag{3}$$

with a random $r \in \mathbb{Z}_p$. He computes their images $u = \psi(\hat{u}), v = \psi(\hat{v})$. He then selects a random $\alpha \in \mathbb{Z}_p$ and computes:

- $T_1 = u^\alpha$, $T_2 = Av^\alpha$
- $\delta = x\alpha$

From random r_α, r_x, r_δ in $\mathbb{Z}_p$, he computes helper values:

- $R_1 = u^{r_\alpha}$, $R_2 = e(T_2, g_2)^{r_x}.e(v, w)^{-r_\alpha}.e(v, g_2)^{-r_\delta}$, $R_3 = T_1^{r_x}.u^{-r_\delta}$

Let $c = H(gpk, M, r, T_1, T_2, R_1, R_2, R_3) \in \mathbb{Z}_p$. The prover computes:

- $s_\alpha = r_\alpha + c\alpha$, $s_x = r_x + cx$, $s_\delta = r_\delta + c\delta$.

He sends to the verifier the signature of M: $\sigma = (r, c, T_1, T_2, s_\alpha, s_x, s_\delta)$.

Verification. The verifier recovers $(\hat{u}, \hat{v})$ from (3) and their images u, v in $\mathbb{G}_1$. He computes the helper data as:

$$\tilde{R}_1 = u^{s_\alpha}/T_1^c$$
$$\tilde{R}_2 = e(T_2, g_2)^{s_x}.e(v, w)^{-s_\alpha}.e(v, g_2)^{-s_\delta}.(e(T_2, w)/e(g_1, g_2))^c$$
$$\tilde{R}_3 = T_1^{s_x}.u^{-s_\delta}$$

He checks whether $c = H(gpk, M, r, T_1, T_2, \tilde{R}_1, \tilde{R}_2, \tilde{R}_3)$ and accepts the signature accordingly.

4.3 Implementation Issues

Sensors $\mathcal{S}$ and liveness detection. The sensor has to acquire biometric traits and to verify that these traits are coming from "living persons". This liveness link must be maintained throughout the authentication. For achieving this, we propose that only trusted sensors are allowed to communicate with the service provider $\mathcal{P}$. This is a quite usual assumption[2] where we assume that the system follows the protocol honestly. In practice, we proceed as follows:

1. $\mathcal{P}$ sends the challenge M to sensor $\mathcal{S}$.
2. $\mathcal{S}$ gets the "fresh" biometric trait b' and reads (b, A) from the card.
3. $\mathcal{S}$ checks whether $b' \sim b$, and in this case computes $x = H(b)$.
4. $\mathcal{S}$ computes the signature σ of M under (x, A).
5. The sensor $\mathcal{S}$ encrypts b' – with a semantically secure encryption scheme – under the key Λ of the Card Issuer $\mathcal{I}$. It then signs this encrypted value together with σ to obtain a signature Σ and sends both σ and Σ to $\mathcal{P}$.
6. $\mathcal{P}$ verifies the signature Σ. This doing, $\mathcal{P}$ insures itself that the signature σ and the encryption of b' can be trusted, as computed by a trusted sensor. The latter is important as the value could be requested under legal warrant. Finally $\mathcal{P}$ checks the signature σ of M to accept the authentication or not.

Remark 3. *To enforce b secrecy, one can think to use Match On Card (MOC) [16]. With this technology, new traits $b' \leftarrow B$ are acquired by a sensor but the matching is made inside the card $\mathcal{C}$ which has to decide whether $b \sim b'$. This way, the confidentiality of b also relies on inherent protections of smartcards against physical threats. Moreover, b does not go out of the card. In this case, $\mathcal{C}$ performs directly the signature of challenges on behalf of the group as previously described in Section 4.2.*

Privacy. Privacy for the signature scheme of the Boneh and Shacham relies on the following problem:
Decision Linear Problem in $\mathbb{G}$: Given $u, v, h, u^{a_1}, v^{a_2}, h^{a_3} \in \mathbb{G}$ as input, output yes if $a_1 + a_2 = a_3$ and no otherwise.

It is proved in [2] that, in the Random Oracle model and assuming the Decision Linear Problem holds in the group $\mathbb{G}_2$, a group member can tell whether he generated a particular signature σ but if he didn't, he learns nothing else about the origin of σ. This property is known as selfless-anonymity.

Hence, the biometric authentication becomes anonymous against the service provider $\mathcal{P}$ and any external eavesdropper.

Revocation. The scheme of Boneh and Shacham has another interesting property: a Revocation List (RL) constituted with the second parts – the A's – of the private keys of revoked users can be established. A verifier in possession of this list (RL) can then determine whether he is interacting with a revoked prover or not.

[2] It could require to deal with protected sensors – e.g. via tamper proof hardware.

In our system, the Revocation List (RL) is held by the Service Provider $\mathcal{P}$. To check whether a user belongs to this list (RL), $\mathcal{P}$ verifies that A is encoded in (T_1, T_2) which happens when $e(T_2/A, \hat{u}) = e(T_1, \hat{v})$.

More precisely, when a user $\mathcal{H}$ has a problem with his plastic card – for instance, if he looses it – he has to go to the Card Issuer $\mathcal{I}$ to declare the event. $\mathcal{I}$ then sends the corresponding A to $\mathcal{P}$ for being added to the Revocation List (RL). Enrolled data can then easily be renewed. The same holds if part of the system, e.g. $\mathcal{P}$, detects a malicious behavior: he can alert $\mathcal{I}$ in order to add a key into RL.

Legal Warrant and database $\mathcal{DB}$ of the Card Issuer. Under legal warrant, the service provider $\mathcal{P}$ can forward the encryption of b' to the Card Issuer $\mathcal{I}$ who can decrypt it to check to who this biometric trait is corresponding in his database $\mathcal{DB}$. The database $\mathcal{DB}$ can also be used to check the coherence of b' with the underlying biometric key x coming with σ.

Note that for ordinary uses, no information has to be sent to $\mathcal{I}$: $\mathcal{P}$ decides by itself if the authentication is a success or not. The database $\mathcal{DB}$ is in fact optional and can be avoided if the property above is not required.

Remark 4. *1. As explained in [2], the scheme of Boneh and Shacham also achieves traceability, i.e. an adversary cannot forge a signature σ without being able to trace him back to one of the users in the coalition which has served to generate σ. This means a user's key is strongly binded to the enrolled biometric data. See [2] for further details.*

2. *Boneh, Boyen and Shacham propose in [1] an extension of the previous protocol which is zero-knowledge. This extension does not share the same properties, in particular the revocation capability of the scheme we used. This is why we do not choose this extension. Nevertheless, note that in [1], the subject of the simultaneous revocation of many keys – for instance, at the end of a cryptoperiod – is studied. This mechanism can also be employed here.*

5 Conclusion

Taking to our advantage the very nature of biometrics, i.e. their volatility and their need to interact with trusted sensors, we present a simple but effective way of considering biometric data for remote authentication. Our procedure does not depend on the kind of underlying biometric information as we can take back traditional matching algorithms.

We combine our idea with a group signature scheme due to Boneh and Shacham [2] to obtain an effective protocol for remote biometric authentication with anonymity features.

Acknowledgment

The authors wish to thank Julien Stern for helpful comments on the scheme.

References

1. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M.K. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004)
2. Boneh, D., Shacham, H.: Group signatures with verifier-local revocation. In: Atluri, V., Pfitzmann, B., McDaniel, P.D. (eds.) ACM Conference on Computer and Communications Security, pp. 168–177. ACM, New York (2004)
3. Boyen, X.: Reusable cryptographic fuzzy extractors. In: Atluri, V., Pfitzmann, B., McDaniel, P.D. (eds.) CCS 2004: Proceedings of the 11th ACM conference on Computer and communications security, pp. 82–91. ACM Press, New York (2004)
4. Boyen, X., Dodis, Y., Katz, J., Ostrovsky, R., Smith, A.: Secure remote authentication using biometric data. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 147–163. Springer, Heidelberg (2005)
5. Brainard, J.G., Juels, A., Rivest, R.L., Szydlo, M., Yung, M.: Fourth-factor authentication: somebody you know. In: ACM Conference on Computer and Communications Security, pp. 168–178 (2006)
6. Bringer, J., Chabanne, H., Cohen, G., Kindarji, B., Zémor, G.: Optimal iris fuzzy sketches. In: IEEE First International Conference on Biometrics: Theory, Applications and Systems, BTAS 2007 (2007)
7. Crescenzo, G.D., Graveman, R., Ge, R., Arce, G.: Approximate message authentication and biometric entity authentication. In: Patrick, A.S., Yung, M. (eds.) FC 2005. LNCS, vol. 3570, pp. 240–254. Springer, Heidelberg (2005)
8. Dodis, Y., Katz, J., Reyzin, L., Smith, A.: Robust fuzzy extractors and authenticated key agreement from close secrets. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 232–250. Springer, Heidelberg (2006)
9. Dodis, Y., Reyzin, L., Smith, A.: Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 523–540. Springer, Heidelberg (2004)
10. Juels, A., Sudan, M.: A fuzzy vault scheme. Design Codes and Cryptography 38(2), 237–257 (2006)
11. Juels, A., Wattenberg, M.: A fuzzy commitment scheme. In: ACM Conference on Computer and Communications Security, pp. 28–36 (1999)
12. Linnartz, J.M.G., Tuyls, P.: New shielding functions to enhance privacy and prevent misuse of biometric templates. In: Kittler, J., Nixon, M.S. (eds.) AVBPA 2003. LNCS, vol. 2688, pp. 393–402. Springer, Heidelberg (2003)
13. Liu, X., Bowyer, K.W., Flynn, P.J.: Iris Recognition and Verification Experiments with Improved Segmentation Method. In: Fourth IEEE Workshop on Automatic Identification Advanced Technologies (AutoID), Buffalo, New York, October 17-18 (2005)
14. Nandakumar, K., Jain, A.K., Pankanti, S.: Fingerprint-based fuzzy vault: Implementation and performance. IEEE Transactions on Information Forensics and Security (2008)
15. National Institute of Standards and Technology (NIST). Iris Challenge Evaluation (2005), `http://iris.nist.gov/ICE`
16. National Institute of Standards and Technology (NIST). The minutiae interoperability exchange test, `http://fingerprint.nist.gov/minex/`

17. Tuyls, P., Akkermans, A.H.M., Kevenaar, T.A.M., Jan Schrijen, G., Bazen, A.M., Veldhuis, R.N.J.: Practical biometric authentication with template protection. In: Kanade, T., Jain, A.K., Ratha, N.K. (eds.) AVBPA 2005. LNCS, vol. 3546, pp. 436–446. Springer, Heidelberg (2005)
18. Tuyls, P., Goseling, J.: Capacity and examples of template-protecting biometric authentication systems. In: ECCV Workshop BioAW, pp. 158–170 (2004)
19. Tuyls, P., Verbitskiy, E., Goseling, J., Denteneer, D.: Privacy protecting biometric authentication systems: an overview. In: EUSIPCO 2004 (2004)

Analysis of a Biometric Authentication Protocol for Signature Creation Application

Anongporn Salaiwarakul and Mark D. Ryan

School of Computer Science,
University of Birmingham, UK
{A.Salaiwarakul,M.D.Ryan}@cs.bham.ac.uk

Abstract. This paper presents an analysis of biometric authentication for signature creation application. We extend the established protocol in order to verify the two properties: secrecy and safety. We have analysed the protocol using applied pi calculus and ProVerif. The verification of the secrecy property shows that the protocol holds the biometric data securely while the verification of the safety property shows that an intruder could not deceive the application to allow her to sign any document using a legitimate user's signature.

1 Introduction

Biometric user authentication is a way to authenticate the user by using his biometric data: fingerprint, face recognition, or iris, for example. Biometric data cannot be considered a secret in the way that private keys or passwords can. In contrast with private keys, biometric data is given to possibly hostile hosts to which a user wishes to authenticate. In contrast with passwords, biometric data cannot be changed, and a user cannot conveniently choose different biometric data to present to different hosts in the way that one might use a different password for a webmail account as for a bank account. Moreover, in contrast with keys and passwords, biometric data such as user's facial characteristics and fingerprints are in the public domain, and can be captured without the user's consent or knowledge.

For this reason, protocols for biometric authentication should rely on proof of freshness of biometric data and cannot rely on its secrecy. Nevertheless, these protocols should protect its secrecy; we take the view that biometric data should be kept private as a matter of good practice. In this respect, it is rather like credit card numbers, which are not really private, since we voluntarily cite them on the phone and by unencrypted email, and allow restaurateurs and other retailers to handle the cards bearing them in our absence. Nevertheless, it seems sensible not to allow such data to be spread around without restriction. The same idea applies to biometric data. Even if user's biometric data (BD) could be captured by agents having access to smooth surfaces the user touches, or agents to whom the user authenticates, it should not be unnecessarily made easy for malicious agents to acquire it.

Processes involved in a biometric authentication could be classified as two steps: enrolment and verification. In the enrolment process, the user's registered biometric code (BC) is either stored in a system or on a smart card which is kept by the user. In

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 231–245, 2008.

the verification process, user presents his biometric data (BD) to the system so that the biometric data will be compared with the stored biometric code. User verification can either be carried out within the smart card, a process called on-card matching, or in the system outside the card, known as off-card matching.

The on-card matching algorithm protects the user's stored biometric code. The biometric code is not necessarily transferred to the outside environment if using this type of matching. Even though the biometric data is not considered to be secret, the protocol should not reveal it without the user's agreement.

When the biometric data is used for biometric authentication, it should not only be protected from disclosure to an attacker, but also its origin should be guaranteed; this prevents an attacker presents the previously captured biometric data to the system in order to authenticate himself as the authorised user.

One of the applications that can use biometric authentication as part of the application is signature creation application. The application is used for signing a document, for example by using user's key which is stored in smart card, in order to proof the originator of the document. The application using biometric authentication protocol not only challenges with the classical problem, inappropriate disclosure of biometric data to an intruder but also the specific problem to the application, signature creation application, such as an intruder deceives the application to sign her document using an legitimate user's signature.

Our paper demonstrates a protocol that uses an on-card matching mechanism to protect the stored biometric code, and an encryption and cryptographic checksum mechanism to protect the presented biometric data. We also extend the protocol to complete the signature creation so that the safety property can be analysed. We analyze the protocol and verify the intended properties of the protocol.

2 Description of Protected Transmission of Biometric User Authentication Data for an On-card Matching Protocol

This protocol is presented by Waldmann, Scheuerman and Eckert [1]. The protocol prevents the user's biometric data from escaping from a biometric reader and protects the data packet using a cryptographic mechanism.

A signature creation application that stores the users biometric code on a smart card is used here to illustrate this protocol. This application enables the user to sign a document using his private key. The user's private key is stored on the smart card. It will be released if the user is successfully verified by using his biometric data.

The physical setup of the system is shown in Fig.1. The system consists of a PC and a terminal case. The PC contains a service application such as the signature creation application. Inside the terminal case are the security module card (SMC), tamper resistant terminal, and user card containing the user's credentials. In order to prevent fraud and interruption from an intruder, the fingerprint reader (including biometric feature extraction), secured viewer and smartcard interaction module (SIM) are embedded in the tamper resistant terminal.

Let us describe the biometric authentication process that takes place when the user wishes to sign a document using his signature. The user, Bob, uses his PC to open the

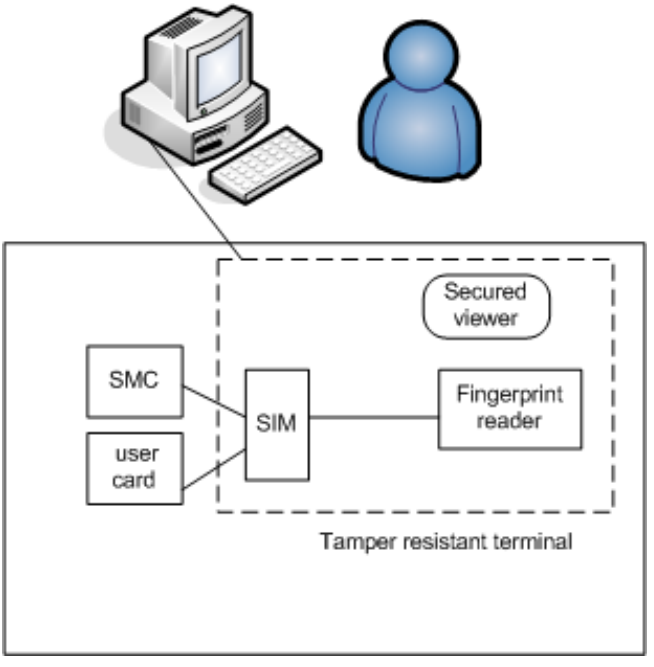


Fig. 1. The physical setup of how components are connected

signature creation application and he is shown a document via the secured viewer. If he agrees to sign it, he will present his biometric data (in this example, his fingerprint) to the sensor. The security protocol is performed via SIM in order to validate the user (detailed description in the next section). If the user verification is successful, the user card will release Bobs signature to sign the document. The signing process is performed inside the tamper resistant terminal.

Fig.2 illustrates the processes involved in the security protocol. The three components of the system that perform the security functions are the SMC, the SIM and the user card.

The SMC is responsible for generating the session keys for encryption and decryption of biometric data. It is a plug-in card to give flexibility to the manufacturer of the service system. For example, the certificate of the service system can be changed easily, if necessary. The user card holds the user's biometric code and other user credentials such as the user's signature if the service system is used for signature creation. The SMC and the user card cannot communicate directly and are outside the tamper resistant terminal so the SIM is responsible for the security protocol between the SMC and the user card.

Let us briefly describe how the protocol proceeds. The legitimate user, Bob, holds his user card, which stores his biometric code and private key. Before user authentication, the SMC and the user card perform mutual authentication, e.g. by using the Needham Schroeder Lowe protocol; if this succeeds, they will calculate the session keys $SK.CG$ and $SK.CC$, and the initial value of the send sequence counter (SSC).

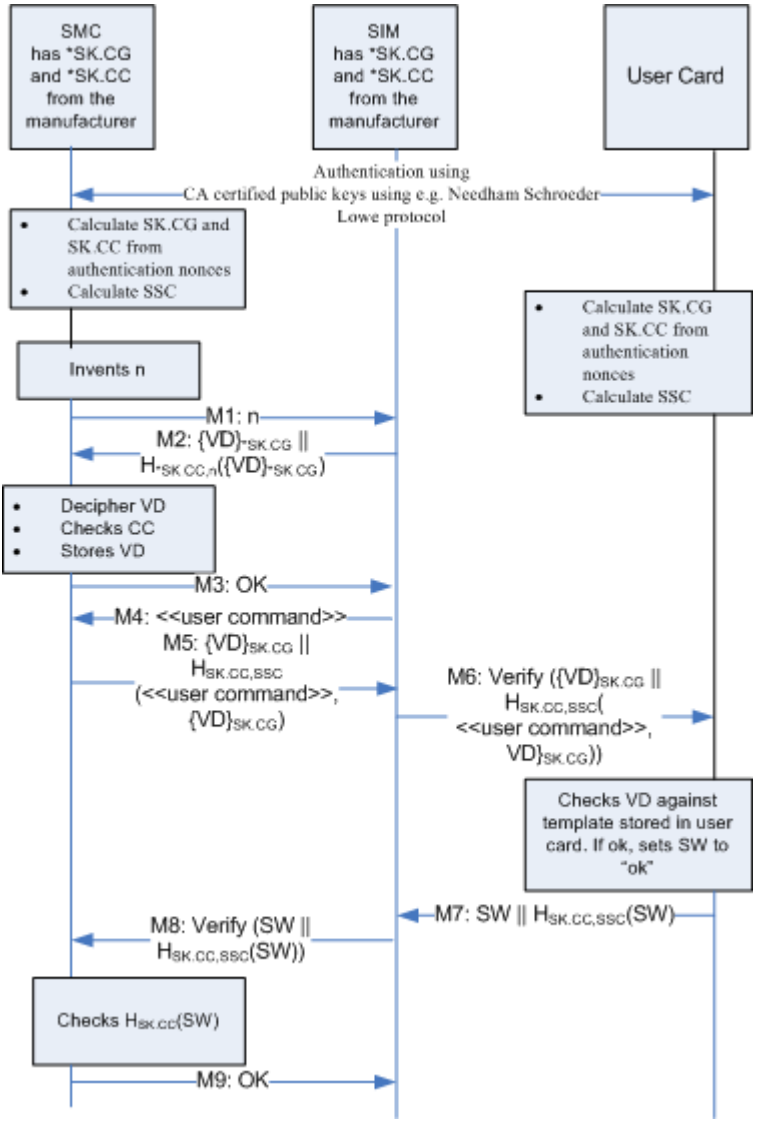


Fig. 2. The message sequence chart of [1]

Apart from the new generated session keys, the SMC holds static keys, *SK.CC and *SK.CG, which are generated by the manufacturer. These keys are also installed in the SIM.

The CC which is included in the notation denotes the cryptographic checksum for ensuring data integrity while the CG represents the cryptogram which is used for data encryption. Consider the following example that represents the message M, which is encrypted using key *SK.CG and then hashed using *SK.CC as key.

$$\{\mathrm{M}\}_{*SK.CG} \,||\mathrm{H}_{*SK.CC}(\{\mathrm{M}\}_{*SK.CG})$$

The receiver of the above message could check the integrity of the received message by performing the hash function of the first argument and then comparing the result with the second argument. Moreover, the receiver could get the content of the message by performing message decryption using static key *SK.CG. The same idea applies to the message that uses the session key for encryption and hash function.

In order to sign a document using his electronic signature, Bob is shown the document via the secured viewer. The secured viewer is proposed in consideration of preventing an attacker that could interfere with the signal of the PC monitor. It is installed in the tamper resistant terminal so that an intruder could not interfere. If he agrees to sign it, he presents his fingerprint to the biometric reader that is situated in the tamper resistant terminal. To prevent replay of the presented biometric data, the SMC invents a fresh random nonce and sends it to the SIM to verify that the received message is fresh.

Before sending Bob's biometric data to the SMC, the SIM encrypts it with $^{*}SK.CG$ and also carries out the cryptographic checksum of encrypted user's biometric data using *SK.CC and the nonce.

After the SMC receives the message, it verifies its authenticity and validity. If this check is successful, it will send a reply "OK" back to the SIM. The SIM then sends a sign command to user card.

The SMC calculates the cryptogram of the biometric data, and the cryptographic checksum of the cryptogram along with the user command by using the session keys (*SK.CG* and *SK.CC*) and sends this packet to the SIM. On receipt, the SIM forwards this data package to the user card. The user card deciphers the package, checks the correctness, and verifies the received biometric data against the stored biometric code. Then the user card sends the result of the verification as well as the cryptographic checksum of the result back to the SMC via the SIM. The SMC verifies the correctness of the data it receives and the result of the user's verification. A positive result leads to the agreement of the signing process by the user card, the detail of which is beyond the scope of this protocol.

3 Completing the Protocol: Creating the Signature

It is necessary to extend the protocol, as described in the previous section, in order to completely verify the protocol and its properties. Here, we give some observations on the protocol and explain how the protocol should be completed in signature creation.

One of the purposes of the protocol is to enable the user to sign the document using the user's stored private key stored on the smart card. To verify the protocol and guarantee correctness, the protocol has to be extended. After the user biometric authentication succeeds (more precisely after the message M9 has finished), in order to sign a document using the user's key, the SIM sends a hash value for the document to the SMC. The hash value of the document is encrypted with one of the static keys, **SK.CG*. The SMC deciphers it and forwards the hash of the document, which is encrypted by the session key (shared by the SMC and the user card) *SK.CG* to the user card. The user card signs the hash value of the document and sends it back to the SIM. The document

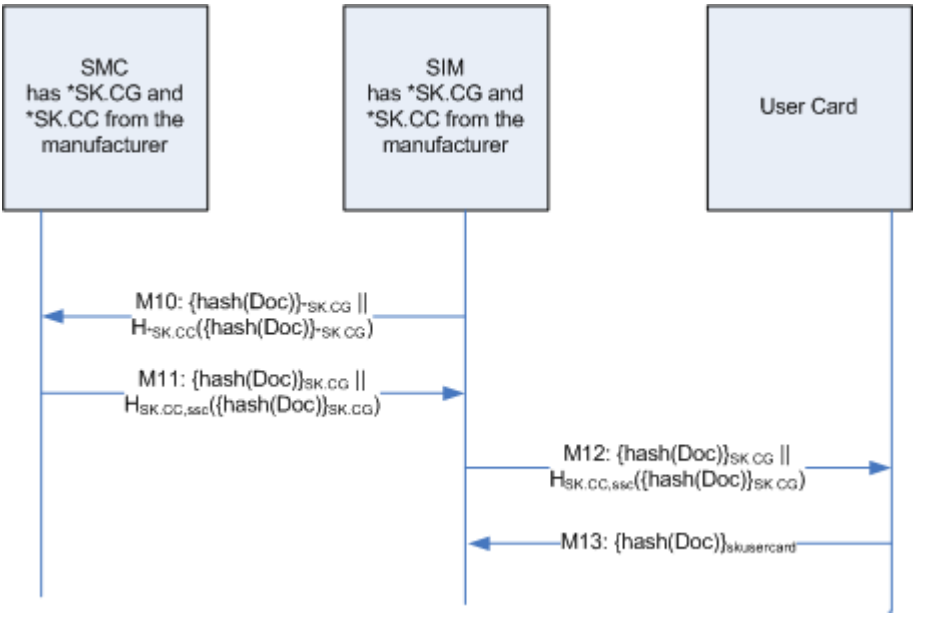


Fig. 3. The message sequence chart for signature creation

is signed only if the user is satisfied with the document he views from the terminal (via the secured viewer in the tamper resistant terminal). In accordance with signing a document, the rest of the protocol should be completed as shown in Fig.3.

4 Capabilities of the Attacker

A Dolev-Yao style attacker can generate, mix, and replay messages transmitted in channels [2], even in cabling communication.

Biometric authentication uses a biometric reader in order to retrieve the user's biometric data. It is connected to the system via a USB cable. In addition, if a smart card is used to store the user's biometric code, a smart card reader is also connected to the system.

Although a smart card is a tamper resistant device in which the stored value cannot be modified without using the appropriate protocol, an attacker can still listen to the communication signal between smart card and reader. There is a prototype model that can be used as an example to describe this concept [4]. The smart card itself does not have a display; it needs another device then, i.e. a smart card reader, to show any value to the user. Communication between the user and the smart card must take place via the reader. If it is modified by a corrupted merchant, information flow between the smart card and the card reader can be intercepted. So if the smart card is used for storing the

biometric code for user verification, the attacker can listen to the messages and capture this data easily.

5 ProVerif Model

ProVerif is a protocol verifier developed by Bruno Blanchet [5], that is able to take as input a variant of the applied pi calculus [6]. This tool has been used to prove the security properties of various protocols [7,8,9,10]. It can be used to prove secrecy, authenticity and strong secrecy properties of cryptographic protocols. It can handle an unbounded number of sessions of the protocol and an unbounded message space. The grammar of processes accepted by ProVerif is described briefly below.

In order to verify properties of a protocol, query commands may be executed. The query 'attacker: m' is satisfied if an attacker may obtain the message m by observing the messages on public channels and by applying functions to them. The query $ev : f(x_1, \ldots, x_n) \Rightarrow ev : f'(y_1, \ldots, y_m)$ is satisfied if the event $f'(y_1, \ldots, y_m)$ must have been executed before any occurrence of the event $f(x_1, \ldots, x_n)$.

An advantage of using ProVerif as a verifier is that it models an attacker which is compliant with the Dolev-Yao model automatically. We do not need to explicitly model the attacker.

P, Q, R	processes
0	null process
P |Q	parallel composition
new n; P	name restriction
new x; P	variable restriction
if M = N then P else Q	conditional
event x; P	event launch
let x = M in P	replace the variable x with the term M in process P
in(M,x); P	message input
out(M,x); P	message output

5.1 Signature and Equational Theory

In our model, ProVerif uses the following signatures and equations for calculating and solving messages. The function *getkey* retrieves the public key of the particular identity from the public key table which is stored in the server. In addition, *getkey* is coded as a private function to prevent components, other than those involving the system, using this function. The symmetric encryption and decryption functions are utilising using *senc* and *sdec* respectively while the asymmetric encryption and decryption functions are done using *enc* and *dec*. In order to resolve an encrypted message, ProVerif uses the decryption equation to decrypt a message using a recognized key. The signed messages are extracted using the *checksign* equation. In our ProVerif model, some messages are hashed using such hash functions as h, g, or f. These hash functions implement two arguments; one is a key while the other one is a message content.

```
   (*Signature*)
private fun getkey/1. (*key retrieval*)
fun sk/1.             (*session key*)
fun senc/2.           (*symmetric encryption*)
fun sdec/2.           (*symmetric decryption*)
fun enc/2.            (*encryption*)
fun dec/2.            (*decryption*)
fun sign/2.           (*signature *)
fun checksign/2.      (*recovering signature*)
fun pk/1.             (*public key*)
fun host/1.           (*host function*)
fun h/2.              (*hash function*)
fun g/2.              (*hash function *)
fun f/2.              (*hash function*)
fun hashDoc/1.        (*hash function for a document*)
   (*Equation*)
equation getkey(host(x)) = x.
equation sdec(senc(x,K),K) = x.
equation dec(enc(x,pk(y)),y) = x.
equation checksign(sign(x,y),pk(y)) = x.
```

5.2 SMC Process

This process represents the operations and message transmission associated with the SMC. First, the SMC performs mutual authentication with the user card. It is not stated how this is done in [1]; we have used the Needham Schroeder Lowe protocol. If successful, it will calculate session keys (*SK.CG* and *SK.CC*) and SSC from the authentication nonces.

The user's biometric data package is received and deciphered. Next, it encrypts and calculates the cryptographic checksum of the biometric data, and sends it to the SIM. If the user's authentication is successful, it will receive the verification result back from the user card, send the reply back to the SIM, and wait for the hash of the document to be sent back. After receiving the hash of the document, it will verify the validity of the document, encrypt, and calculate the cryptographic checksum of the hash of the document with the session keys *SK.CG* and *SK.CC* respectively.

```
let SMC =
(* Authentication using Needham Schroeder
   Lowe Protocol *)
  in(c,hostX);
  let hostSMC = host(pkSMC) in
  out(c,(hostSMC,hostX));
  in(c,ms);
  let(pkX,=hostX) = checksign(ms,pkS) in
  new Na;
  out(c,enc((Na,hostSMC),pkX));
```

```
  in(c,m);
  let(=Na,NX2,=hostX) = dec(m,skSMC) in
  out(c,enc(NX2,pkX));
  let SKCG = h(Na,NX2) in
  let SKCC = g(Na,NX2) in
  let SSC = f(Na,NX2) in
(*  After the authentication succeeds*)
  new n;
  out(c,n);
  in(c,(m1,m2));
  if h((sSKCC,n),m1) = m2 then
  (
    let BDreceived = sdec(m1,sSKCG) in
    out(c,OK);
    in(c,m13);
    let BDsenc = senc(BDreceived,SKCG) in
    out(c,(BDsenc,h((SKCC,SSC),(m13,BDsenc))));
    in(c,(m8,m9));
    if h((SKCC,SSC),m8) = m9 then
      if m8 = success then
        out(c,OK);
    in(c,(m16,m17));
    if h(sSKCC,m16) = m17 then
    (
       let M1 = sdec(m16,sSKCG) in
       let M1senc = senc(M1,SKCG) in
       out(c,(M1senc,h((SKCC,SSC),M1senc)))
     )

   ).
```

5.3 SIM Process

In the real-life model, the user is presented with the document that he will sign using his key on the secured viewer. If he agrees to sign it, then he places his biometric data on the biometric reader which is installed in the SIM. Therefore, for ease of understanding, in the ProVerif model, the document that the user wants to sign is created within the SIM. The SIM receives a fresh random nonce and then sends the user's biometric data encrypted with **SK.CG*, along with the cryptographic checksum created using **SK.CC*, and the nonce, to the SMC. When the SIM receives the signal from the SMC that the user's biometric data is correct, it sends the user's command authorizing the signature as a reply.

The SIM carries out the security protocol between the SMC and the user card by receiving and forwarding messages between those two components. After the user's authentication succeeds, the SIM generates the hash value of the document, encrypts

it, and calculates its cryptographic checksum. It then sends these data to the SMC. The SMC is waiting to receive the document which is to be signed by the user card.

```
let SIM =
(* communiation messages start *)
   in(c,nx);
   in(userChBD,BD);
   in(userChText,userText);
   let BDsenc = senc(BD,sSKCG) in
   out(c,(BDsenc,h((sSKCC,nx),BDsenc)));
   in(c,m20);
   if m20 = OK then
   (
     out(c,userCommand);
     in(c,(m4,m5));
     out(c,(m4,m5));
     in(c,(m6,m7));
     out(c,(m6,m7));
     in(c,okm);
     if okm = OK then
     (
     let digest = senc(hashDoc(userText),sSKCG) in
     out(c,(digest,h(sSKCC,digest)));
     in(c,(m13,m14));
     out(c,(m13,m14));
     in(c,m15)
     )
   ).
```

5.4 UserCard Process

First, the user card executes the mutual authentication with the SMC. Then, it calculates the session keys *SK.CG* and *SK.CC*, and *SSC*. Next, the user card awaits a package of the user's biometric data. It verifies the validity and authenticity of the received message. It decrypts the package and verifies the received biometric data against the stored biometric code. If they match, the verification result is set to be successful. In our ProVerif model, they are always set to match so that we can verify the protocol until the end (the signing process of the document) without blocking through failure in biometric verification. The verification result is sent out along with the checksum of the result which is computed using *SK.CC*. Then, it acquires the hash of the document and signs it using the user's signature, which is stored in the user card.

```
let UserCard =
(* Authentication using Needham Schroeder
   Lowe Protocol *)
  in(c,m);
  let(NY,hostY) = dec(m,skUserCard) in
```

```
  let hostUserCard = host(pk(skUserCard)) in
  out(c,(hostUserCard,hostY));
  in(c,ms);
  let(pkY,=hostY) = checksign(ms,pkS) in
  new Nb;
  out(c,enc((NY,Nb,hostUserCard),pkY));
  in(c,m3);
  if Nb = dec(m3,skUserCard) then
    let skcg = h(NY,Nb) in
    let skcc = g(NY,Nb) in
    let ssc = f(NY,Nb) in
(* The authentication succeeds,
   message communication starts *)
  in(c,(m10,m11));
  if  h((skcc,ssc),(userCommand,m10)) = m11 then
  (
      let BDsdec = sdec(m10,skcg) in
      if BDsdec = BD then
        let SW = success in
        let m12 = h((skcc,ssc),SW) in
        out(c,(SW,m12));
        in(c,(m18,m19));
        if h((skcc,ssc),m18) = m19  then
        (
          let M2 = sdec(m18,skcg) in
          out(c,sign(M2,skUserCard))
         )
   ).
```

5.5 U Process

To demonstrate user interaction in the protocol, we model the process U. The user receives a document and checks it. If he is satisfied with the contents, he will place his finger on the reader.

```
let U =
  in(TextCh,t);
  if t = Text  then
   out(userChBD,BD);
   out(userChText,t).
```

5.6 S Process

In order to authenticate identities using the Needham Schroeder Lowe protocol, the server is modelled using process S, which is used for providing a public key to the identity. The server process receives the request from the identity a that it wants to

communicate with identity b. The server process retrieves the public key of the identity b from the server's public key table. It then signs the package of the public key and the identity b using its private key and outputs to the channel. The receiver of this package ensures that the public key it received comes from the genuine server by checking the signature. The public key will be used later in the receiver process in order to perform the Needham Schroeder Lowe authentication which needs the public keys for decryption.

```
let S =
  in(c,m);
  let(a,b) = m in
  let sb = getkey(b) in
  out(c,sign((sb,b),skS)).
```

5.7 Main Process

In the main process, the static keys **SK.CG* and **SK.CC*, and the private keys of the SMC, the SIM and the user card are created. Private channels for the user's document and biometric data are set up. The public keys of each of the components are distributed on the public channels. There are many *SMC*, *SIM*, *user card*, and *U* processes in the system. The U processes represent Alice and Bob who input the documents and the biometric data.

```
process
new userChBD;
new userChText;
new AliceTextCh;
new BobTextCh;
new BobBD;
new AliceBD;
new sSKCG;
new sSKCC;
new skSMC;
let pkSMC = pk(skSMC) in
out(c,pkSMC);
new skBobUserCard;
new skAliceUserCard;
let pkBobUserCard = pk(skBobUserCard) in
let pkAliceUserCard = pk(skAliceUserCard) in
out(c,pkBobUserCard);
out(c,pkAliceUserCard);
new skS; let pkS = pk(skS) in
out(c,pkS);
!out(AliceTextCh,AliceText);
!out(BobTextCh,BobText);
```

```
((!S) | (!SMC) | !SIM |

(let TextCh = AliceTextCh in
 let Text = AliceText in
 let BD = AliceBD in !U) |

(let TextCh = BobTextCh in
 let Text = BobText in
 let BD = BobBD in !U) |

(let skUserCard = skAliceUserCard in
 let BD = AliceBD in !UserCard) |

(let skUserCard = skBobUserCard in
 let BD = BobBD in !UserCard))
```

6 Analysis of the Protocol

A signature application protocol is used as an example of using biometric authentication in order to verify the user who uses the smart card to sign a document that he is the correct user.

An intruder could interfere between the smart card and smart card reader to try to listen to the communication and capture user's biometric data [4]. Moreover, an intruder could play with messages to lead a legitimate user to sign her messages.

6.1 Secrecy of the Biometric Data

This property is used to verify that the protocol does not reveal the user's biometric data without permission. Even though we consider the biometric data to be public, it is good practice to keep it private so that no one else except the sender and the receiver knows the content of messages. The protocol should not allow the data presented by user to be announced to others. Analysis of this property verifies whether an attacker can intercept the biometric data when it is sent from one component to another. In our model, the biometric data is represented as BobBD, the biometric data of legitimate user, Bob. The ProVerif implementation is:

```
query attacker : BobBD
```

ProVerif responds to the *query* command by using a Dolev-Yao style attacker to attempt to compose or decompose messages and establish whether an attacker can reach the biometric data (BobBD).

6.2 Safety

This property is used to verify that the document is signed only if the user's authentication is successful and that only the legitimate user signs the agreed document. We

analyze this by checking whether an attacker can sign someone else's documents using the signature of the legitimate user. From our assumption, we check whether an intruder, Alice, can intercept messages to make the legitimate user, Bob, sign her document. The ProVerif implementation is:

```
query attacker : sign(AliceText,skBobUserCard)
```

ProVerif analyzes this *query* command by checking whether an attacker can sign AliceText (which is not the document that is shown to the legitimate user, Bob) using Bob's signature. We assume that the user's signature is the same as the private key of the user card that the user holds.

7 Conclusion

We have analyzed two properties of the protocol: *secrecy* and *safety*.

Although we consider the biometric data to be public, we still need to verify that the protocol which uses this resource does not reveal it without the user's consent. The data should not be revealed to anyone who is neither the sender nor the intended receiver. The positive result of the verification illustrates that the presented biometric data remains private within the protocol and an attacker cannot acquire it.

The positive result of the safety property shows that the protocol guarantees that even if the presented biometric data is captured from the previous submitted data packet, it cannot lead the user card to sign a document that the user is not willing to sign.

Since the biometric data can be captured (as explain in section 1), the hardware is required to be capable of ensuring that the biometric data has come from the user's live presentation, not (for example) a fake rubber finger.

The properties of the biometric authentication protocol should be proposed and stated clearly when creating a biometric authentication protocol. In future work, we will consider which properties are desirable of a biometric authentication protocol.

References

1. Waldmann, U., Scheuermann, D., Eckert, C.: Protected Transmission of Biometric User Authentication Data for Oncard-Matching. In: ACM Symposium on Applied Computing, pp. 425–430 (2004)
2. Dolev, D., Yao, A.C.: On the Security of Public Key Protocols. In: Proceedings of 22nd IEEE Symposium on Foundations of Computer Science, pp. 350–357 (1981)
3. Gobioff, H., Smith, S., Tygar, J.D., Yee, B.: Smart Cards in Hostile Environments. In: 2nd USENIX Workshop on Electronic Commerce (1996)
4. Bond, M.: Chip and Pin (EMV) Point-of-Sale Terminal Interceptor. availability = internet (2007), http://www.cl.cam.ac.uk/~mkb23/interceptor/
5. Blanchet, B.: ProVerif: Automatic Cryptographic Protocol Verifier User Manual (2005)
6. Abadi, M., Fournet, C.: Mobile Values, New Names, and Secure Communication. In: POPL 2001 (2001)
7. Abadi, M., Blanchet, B., Fournet, C.: Just Fast Keying in the Pi Calculus. ACM Transactions on Information and System Security (TISSEC) 10(3), 1–59 (2007)

8. Abadi, M., Blanchet, B.: Computer-Assisted Verification of a Protocol for Certified Email. SAS 2003 58(1-2), 3–27 (2005); Special issue SAS 2003
9. Kremer, S., Ryan, M.: Analysis of an Electronic Voting Protocol in the Applied Pi Calculus. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 186–200. Springer, Heidelberg (2005)
10. Delaune, S., Kremer, S., Ryan, M.: Coercion-resistance and Receipt-freeness in Electronic Voting. In: 19th Computer Security Foundations Workshop. IEEE Computer Society Press, Los Alamitos (2006)
11. Prabhakar, S., Paankanti, S., Jain, A.K.: Biometric Recognition: Security and Privacy Concerns. IEEE Security & Privacy, 33–42 (2003)
12. Chen, L., Person, S., Proundler, G., Chen, D., Blanceheff, B.: How can you trust a computing platform? In: Proceedings of Information Security Solutions Europe Conference (ISSE 2000) (2000)
13. Davida, G.I., Frankel, Y., Matt, B.J.: On Enabling Secure Application Through Off-line Biometric Identification. In: IEEE Symposium on Security and Privacy, vol. 1008, pp. 148–157
14. Matsumoto, T., Matsumoto, H., Yamada, K., Hoshino, S.: Impact of Artificial Gummy Fingers on Fingerprint Systems. In: Proceedings of SPIE. Optical Security and Counterfeit Deterrence Techniques IV, vol. 4677 (2002)

Efficient Secure Labeling Method under Dynamic XML Data Streams

Dong Chan An, So Mi Park, and Seog Park

Department of Computer Science & Engineering, Sogang University,
C.P.O. Box 1142, Seoul Korea 100-611
{channy,thathal,spark}@sogang.ac.kr

Abstract. In this paper, we propose an improved method of an efficient secure access control labeling under dynamic XML data streams environment. The proposed method enables an efficient secure real-time processing of a query in a mobile terminal and supports insertion of new nodes at arbitrary positions in the XML tree without re-labeling and without conflicting. Although some researches have been done to maintain the document order in updating, the main drawbacks in the most of these works are if the deletion and/or insertion occur regularly, then expensive re-computing of affected labels is needed. Therefore we focus on how to design an efficient secure labeling scheme for XML trees which are frequently updated under dynamic XML data streams.

Keywords: Access Control, Labeling Method, Query Processing, Dynamic XML, Data streams.

1 Introduction

In recent, XML has become an active research area and a popular standard for representation and exchanging data over the Internet. Query language like XPath [2] and XQuery [5] are developed by W3C group for XML data. The efficient secure processing of XPath or XQuery is an important research topic. Since the logical structure of an XML document is a tree, establishing a relationship between nodes such as parent-child relationship or ancestor-descendant relationship is essential for processing the structural part of the queries. For this purpose many proposals have been made such as structural indexing and nodes labeling. Relatively little works have been done to enforce access controls particularly for XML data in the case of query access control. Moreover, the access control within traditional environments has been a system-centric method for environments including finite, persistent, and static data. However, more recently, access control policies have become increasingly needed in continuous data streams [1], and existing access control models and mechanisms cannot be adequately adopted on data streams [7]. In particular, the need for an efficient secure access control method of dynamic XML data streams in ubiquitous environment has become very important.

The main contributions of this paper are summarized as follows. First, we propose an improved method that supports the inserting of new nodes at arbitrary position in

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 246–260, 2008.

the XML tree without re-labeling. Second, our proposal supports the representation of ancestor-descendent relationships between any two given nodes by looking at their unique label. Third, given the unique label of the node, one can easily and securely determine its accessible roles and its ancestor/descendant relationship.

The rest of this paper is organized as follows. Section 2 presents related works in the area of XML access control, query processing, and labeling scheme. Section 3 introduces the algorithm and method of the improved access control labeling. Section 4 shows the experimental results from our implementation and shows the query processing efficiency and security of our framework. Our conclusions are contained in Section 5.

2 Background

2.1 XML Access Control

The existing XML access control enforcement mechanism [3, 4, 8, 9] is a view-based enforcement mechanism. The semantics of access control to user is a particular view of the documents determined by the relevant access control policies. It provides a useful algorithm for computing the view using tree labeling. However, aside from its high cost and maintenance requirement, this algorithm is also not scalable for a large number of users.

To overcome the view-based problems, M. Murata et al. [16] proposed the filtering method to filter out queries that do not satisfy access control policies. B. Luo et al. [15] took extra steps to rewrite queries in combination with related access control policies before passing these revised queries to the underlying XML query system for processing. However, the shared Nondeterministic Finite Automata (NFA) of access control rules is made by a user (or a user's role). Thus, the shared NFA involves many unnecessary access control rules from the user's query point of view, which further result in time-consuming decisions during which the user's query should have already been accepted, denied, or rewritten.

An authorization defined by a security manager is called explicit and an authorization defined by the system, on the basis of an explicit authorization, is called implicit in the hierarchical data model. An implicit authorization is used with an appropriate 'propagation policy' to benefit the advantage of storage. With an assumption that the optimized propagation policy varies under each of the different environments, 'most-specific-precedence' is generally used. On the other hand, 'denial-takes-precedence' is used to resolve a 'conflict' problem that could be derived from propagation policy by such implicit authorization. Since positive authorization and negative authorization are also used together, 'open policy, which authorizes a node that does not have explicit authorization, and 'closed policy', which rejects access, are used. The policies 'denial-takes-precedence' and 'closed policy' are generally used to ensure tighter data security [16].

2.2 XML Query Processing

XML data has a hierarchical structure and the required capacity might be very huge. The high practicality of mobile terminals and computing power is necessary for the

feasibility of ubiquitous computing. The efficiency of memory, energy, and processing time is also especially needed. A method that can takes XML data into appropriate fragmentation so as to process it in pieces is consequently needed for the small memory of a mobile terminal to manage massive XML data [8, 20]. When XML data streams are generated under a sensor network, the data is structurally fragmented and transmitted and processed in XML piece streams, the efficiency of memory and the processing time of mobile terminals can be reconsidered. Moreover, when certain data are updated in an XML data streams, not the whole XML data but only the changed fragment needs to be transmitted, taking advantage of a reduced transmission cost.

The Hole-Filler Model [4, 7] has been proposed as a method that fragments XML data structurally. XFrag [6] and XFPro [12] proposed an XML fragmentation processing method adopting the Hole-Filler Model. Nonetheless, this method has problems of late processing time and waste of memory space due to additional information for the Hole-Filler Model. The XFPro method has improved processing time by improving the pipeline, but does not solve some of the Hole-Filler Model

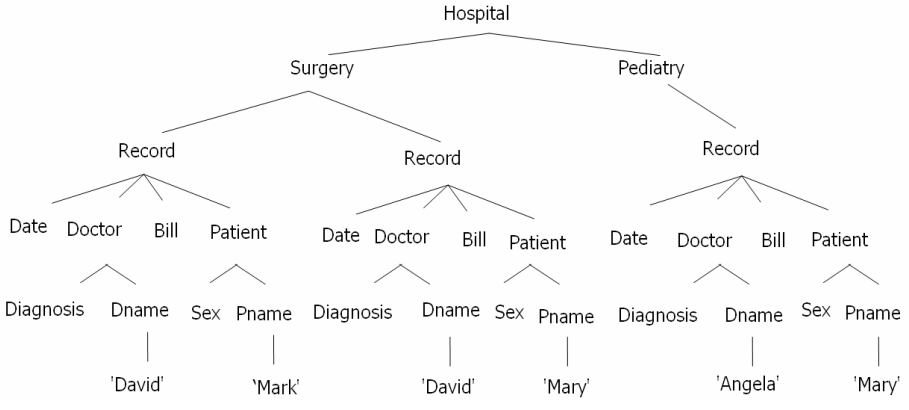


Fig. 1. XML Medical Record

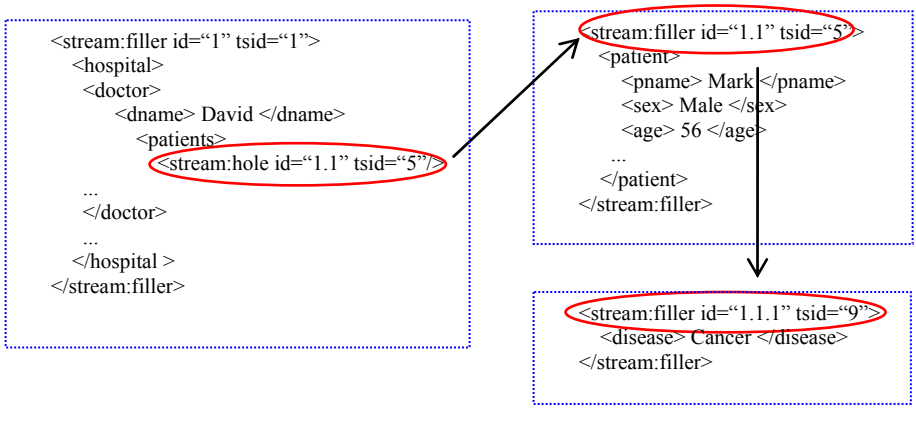


Fig. 2. Fragmented XML Document

issues. A medical records XML document [11] is shown in Fig. 1, and a fragmented XML document by the Hole-Filler Model [6] is shown in Fig. 2.

2.3 XML Labeling

For the query processing of a dynamic XML document, a labeling method, which is easily applied to insert and delete elements, is needed. Some existing labeling methods lack the document updating capability and search whole XML document trees again to re-calculate the overall label of the node, thus bringing costs higher [16].

A new labeling method has shown up as a consequence of the appearance of the dynamic XML document. This method is typical of the prime number labeling scheme [18, 19] applied to information which rarely affects other labels. This method assigns a label for each node, a prime number, to represent the ancestor-descendant relation and is designed not to affect the label of other nodes when updating the document. However, since it searches a considerable range of the XML tree again and re-records updated order information during the updating process, it presents a higher updating cost.

3 Efficient Secure Access Control

The proposed environment of an efficient and secure access control is shown in Fig. 3. It depicts medical records that need accurate real-time query answers by checking the authority and the role of valid users via access control policy when a query is requested.

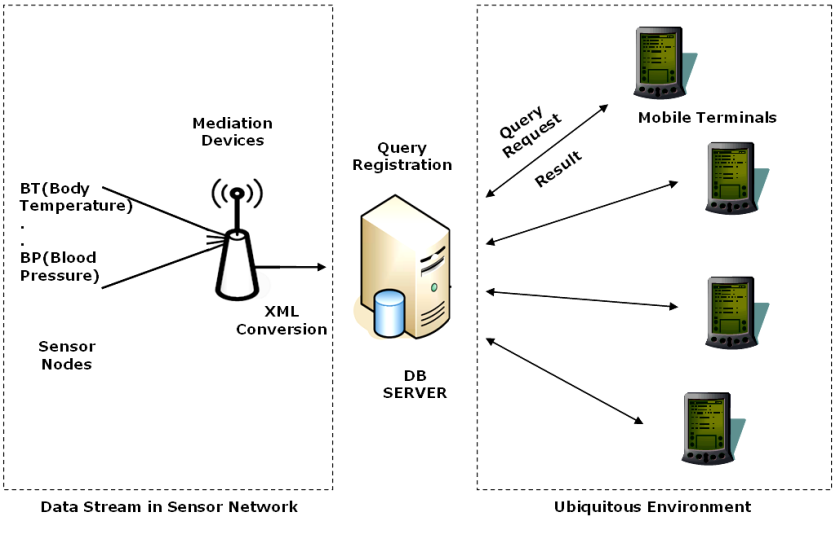


Fig. 3. Query Processing under XML Data Streams

3.1 Improved Role-Based Prime Number Labeling

An improved role-based prime number labeling scheme is explained under a certain environment as Fig. 3. First of all, considering the characteristics of the proposed environment, the fragmentation of the XML document in Fig. 1 is shown in Fig. 4. Problems such as low processing time and waste of memory space needed due to additional information for the Hole-Filler Model in existing XFrag [6] is minimized as shown in Fig. 4. This means that information such as tag structure is no longer needed because the order of XML documents no longer needs to be considered.

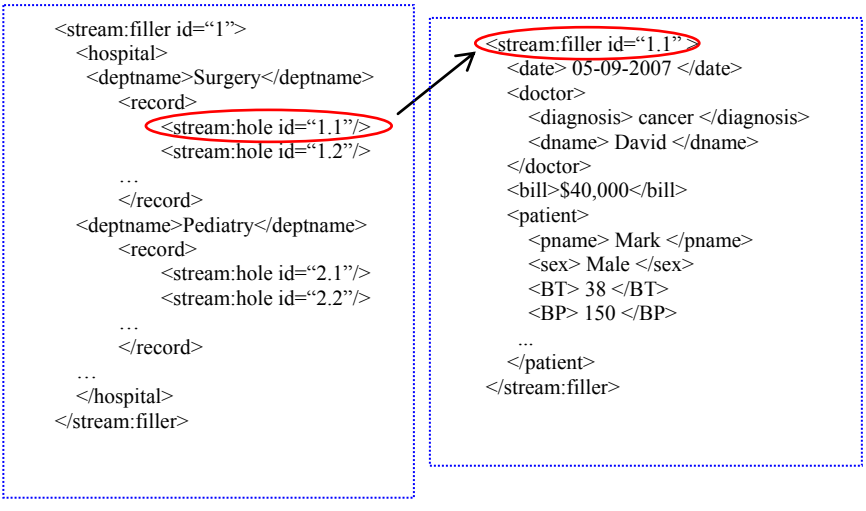


Fig. 4. Partial Fragmentation in XML Data Streams

Table 1. Role-Based Prime Number Table

Role	Role-Based Prime Number
Patient	2
Doctor	3
Researcher	5
Insurer	7

After a fragmenting procedure of XML data streams, proper role-based prime numbers are assigned to nodes of the medical records XML data streams of Fig. 1 as shown in Table 1. Since roles are limited in any organization, it is possible to represent roles with a prime number.

Improved Role-Based Prime Number Labeling Notation. The labels of all nodes are constructed by three significant components (*L1, L2, and L3*), which are unique.

1. *Ancestor label component (L1)* – The component that represents to the label of the parent node.
2. *Self label component (L2)* – The component that represents the self label of a node in the XML document.
3. *Prime number product component (L3)* – The component that represents the prime number product of accessible role for a node. A unique label is created by three components, which are concatenated by a "delimiter(.)".

Improved Role-Based Prime Number Labeling Scheme. The Labeling for an XML document is following.

L1.L2.L3

1. The root node is labeled with "null" because the root node does not have a sibling node and a parent node.
2. The first child of the root node is labeled with label (N_1),"*a1.L3*". The second child of the root node is labeled with label (N_2), "*b1.L3*". Because of the parent node's label (L1) is "null", L1 and L2 is concatenated. The third component *(L3)* is optional in this level. If the third component is all roles' accessible, the third component is able to omit.
3. The first child of the second level N_1 is labeled with label (NN_1), "*a1.a1.L3*". The second child of the second level N_2 is labeled with label (NN_2), "*b1.a1.L3*". Because of the parent node's label (*L2*) is inherited. The third component *(L3)* is labeled with prime number product for a node's accessible roles.
4. The first child of the third level NN_1 is labeled with label (NNN_1), "*a1a1.a1.L3*". The second child of the third level NN_2 is labeled with label (NNN_2). "*b1a1.a1.L3*". Because of the parent node's label (*L1 and L2*) is inherited.
5. The first child of the third level NNN_1 is labeled with label, "*a1a1a1.a1.L3*". The second child of the third level NNN_2 is labeled with label, "*b1a1a1.a1.L3*". Because of the parent node's label (*L1 and L2*) is inherited.
6. The third component (*L3),* prime number product of accessible roles for node is generated by following.

- Date (210) : product of 2,3,5,7 ← accessible roles' prime number
- Doctor (30) : product of 2,3,5
- Bill (14) : product of 2,7
- Diagnosis (30) : product of 2,3,5
- Dname (6) : product of 2,3
- Patient (210) : product of 2,3,5,7
- Sex (210) : product of 2,3,5,7
- Pname (42) : product of 2,3,7

The below Fig. 5 is the labeled XML tree by applying the above labeling scheme. The root node is labeled with an empty string. Then, the self label of the first child node

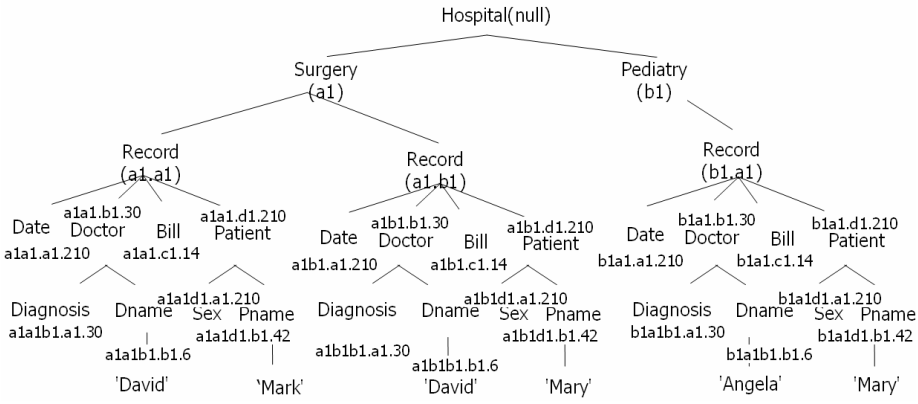


Fig. 5. Improved Role-Based Prime Number Labeling of Medical Records

is “a1”, the second child is “b1”, the third is “c1”, and the fourth is “d1”. In Fig. 5, the label of the node concatenates its parent’s label and own label. For the two nodes “a1.a1.L3” and “a1a1.a1.L3”, “a1.a1.L3” is an ancestor of “a1a1.a1.L3” if “a1.a1.L3” is a prefix of “a1a1.a1.L3”.

3.2 Query Processing Using Improved Role-Based Prime Number Labeling

The proposed access control system’s architecture is shown in Fig. 6. The query processing procedure in Fig. 6 can be considered in two steps. The role check is done

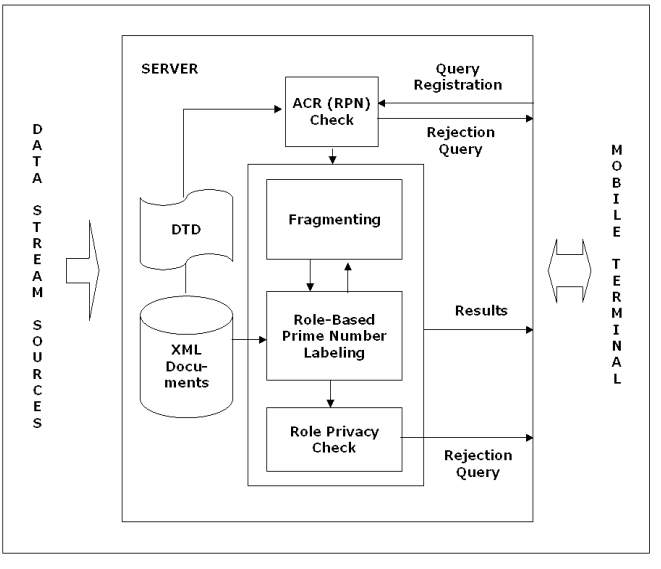


Fig. 6. Efficient Secure Access Control System

Table 2. Role Privacy Table

Department	Record	Role			
		Patient	Doctor	Insurer	Researcher
Sugery (a1)	a1.a1	Mark	David	ING	-
	a1.b1	Mary	David	AIG	-
	...	...	…	…	-
	…	...	...	…	-
Pediatry (b1)	b1.a1	Mary	Angela	AIG	-
	...	...	…	…	-
	…	…	...	…	-
...	...	…	...	…	-

in Step 1 using the 'Role-Based Prime Number Table' and final access authority is checked at Step 2 using the 'Role Privacy Table'. Once a query from a mobile terminal is registered, access authority is checked at Step 1 by checking the prime number of the query terminal node. That is, access to Step 1 is permitted when the remainder of the *L3* (prime number product of accessible roles) divided by the role of user becomes zero. Accessibility is finally checked at Step 2 referring to the 'Role Privacy Table' of Table 2. Moreover, as indicated in Section 2.1, query access is rejected by 'denial-takes-precedence'.

Example 1. (predicate + positive access control + positive access control)

```
(1) //record/patient[pname=Mark]
(2) "David" with role of doctor requests a query
```

- step1, terminal node Pname=Mark's label is verified : 'a1a1d1.b1.42'
- David's role = doctor : 3
- 42%3=0, access is permitted
- step2, only prefix label 'a1a1' and 'a1b1' are permitted for David by 'Role Privacy Table'
- finally, 'a1a1d1.b1.42' (*//record/patient[pname=Mark])* access permitted
- response to the query

Example 2. (predicate + positive access control + negative access control)

```
(1) //record/patient[pname=Mark]
(2) "Angela" with role of doctor requests a query
```

- step1, terminal node pname=Mark's is verified : 'a1a1d1.b1.42'
- Angela's role = doctor : 3
- 42%3=0, access is permitted

- step2, Only prefix label ‘b1a1’ is permitted for Angela by 'Role Privacy Table'
- [pname=Mark] is ‘a1a1d1.b1.42’, access rejected
- access to step1 permitted, access to step2 rejected.
- finally query access rejected

Example 3. (negative access control)

```
(1) //record/patient/pname
(2) one with role of researcher requests a query
```

- step1, terminal node pname’s label is verified : *.*.42
- researcher's role : 5
- 42%5≠0, access rejected
- finally, query rejected

As shown in Example 3, the main benefit of the proposed method is that it processes the rejection query quickly. Because step 1 verify target node label and query requester’s role.

3.3 Dynamic Role-Based Prime Number Labeling

In this section, updates on XML documents are described by commonly used tree operations, namely, *INSERT* an element, text, or attribute, and *DELETE* an element, text, or attribute. In fact, deletion can be realized more easily. The deletion of a node will not affect the ordering of the nodes in the XML tree. However, *INSERT* operation is slightly more complicated than *DELETE* operation.

Definition 1. (Insert Operation) *For any two existing adjacent self labeling, N_{left} and N_{right}, a new labeling N_{new} can always be allocated between them without modifying already allocated labeling by the following Insert operation, where len(N) is defined as the bit length.*

1. *Insert a node before the first sibling node, $N_{new} = N_{left}$ with the last bit “1” change to “01” ($\oplus$ means concatenation)*
2. *Insert a node after the last sibling node, $N_{new} = N_{right}$ with the last bit “1” + 1 (+ means summation, If the last bit is “9”, last bit extension “9→91”, “99→991”)*
3. *$len(N_{left}) \geq len(N_{right})$ then, $N_{new} = N_{left} \oplus 1$*
4. *$len(N_{left}) < len(N_{right})$ then, $N_{new} = N_{right}$ with the last bit “1” change to “01”*

Given two label strings “a01” and “a1”, “a01”$\prec$“a1” lexicographically because the comparison is from left to right, and the second bit of “a01” is “0”, while the second bit of “a1” is “1”. Another example, “a1”$\prec$“a10” because “a1” is a prefix of “a10”.

Definition 2. (Lexicographical order $\prec$) *Given two labels N_{left} and N_{right}, N_{left} is lexicographically equal to N_{right} iff they are exactly the same. N_{left} is said to be lexicographically smaller than N_{right} ($N_{left} \prec N_{right}$) iff*

1. N_{left} *is a prefix of* N_{right}, *or*
2. *"a01"* $\prec$ *"a1"* $\prec$ *"a10"* $\prec$ *"a2", or compare* N_{left} *and* N_{right} *by string.*

If the current string $N_{current}$ *of* N_{left} *and the current string* $N_{current}$ *of* N_{right} *satisfy condition (1), then* $N_{left} \prec N_{right}$ *and the comparison is discontinued* [13].

Algorithm 1. Insert Between (N_{left}, N_{right}**)**

Input: $N_{left} \prec N_{right}$

Output N_{new} such that $N_{left} \prec N_{new} \prec N_{right}$ lexicographically

Description:

1 : **if** $len(N_{left}) \geq len(N_{right})$ **then**
2 : $N_{new} = N_{left} \oplus 1$ // $\oplus$means concatenatin
3 : **else if** $len(N_{left}) < len(N_{right})$ **then**
4 : $N_{new} = N_{right}$ with the last bit "1" change to "01"
5 : **end if**
6 : **return** N_{new}

To insert a label string between "a1" and "b1", the size of "a1" and "b1"is equal, therefore we directly concatenate one more "1" after "a1" (see lines 1 and 2 in Algorithm 1). The inserted label string is "a11", and "a1" $\prec$"a11" $\prec$"b1" lexicographically. To insert a label string between "a1" and "a11", the size of "a1" is 2 which is smaller than the size 3 of "a11", therefore we change the last "1" of "a11" to "01", i.e. the inserted label string is "a101" (see lines 3 and 4 in Algorithm 1). Obviously, "a1" $\prec$ "a101" $\prec$ "a11" lexicographically. To insert a label string between "a11" and "b1", the size of "a11" is 3 which is larger than the size 2 of "b1", therefore we directly concatenate one more "a111" after "a11" (see lines 1 and 2 in Algorithm 1). The inserted label string is "a111", and "a11" $\prec$"a111" $\prec$"b1" lexicographically.

Our proposed dynamic role based-prime number labeling scheme guarantee that we don't have update costs in XML updating such as Fig. 7.

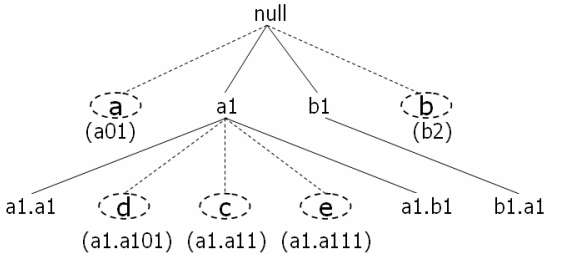


Fig. 7. Dynamic Update Labeling

3.4 Node Relationship

Using labels of the parent nodes as a part of creating labels for child nodes helps to determine the ancestor-descendant relationships and the sibling relationship between nodes. For instance, in Fig. 7, by knowing a node called “a1.a1”, we can understand that its parent node label is “a1” and all nodes beginning with “a1” are its siblings. All of its children nodes shall have “a1a1” attached at front.

Therefore, our labeling scheme will help to reduce the number of nodes that need to be accessed to carry out those tasks. These advantages make the tasks of doing retrieving, inserting, deleting or updating a lot easier.

4 Experiments

In this section, we compared our proposed labeling method in various ways. We used one PC with an Intel Pentium IV 2.66GHz CPU, with a main memory of 1GB using the MS Windows XP Professional OS. We have implemented a proposed labeling method in JAVA (JDK1.5.0) and used SAX from Sun Microsystems as the XML parser. For the database, we used an XMark datasets to generate XML documents [17].

4.1 Label Size

We analyzed the label size of GRP (GRoup based Prefix labeling) scheme and LSDX (Labeling Scheme for Dynamic XML data) scheme, which supports dynamic XML data and we used XMark to generate the first ten XML documents, same as that used by [14]. We generated labels from those documents using our proposed dynamic labeling scheme and compared with those two schemes in term of total length of labels. We discovered that our dynamic labeling scheme can be average 3 times shorter compared to GRP scheme and more than average 2 times shorter compared to LSDX scheme [10]. Detailed figures are presented in the Fig. 8.

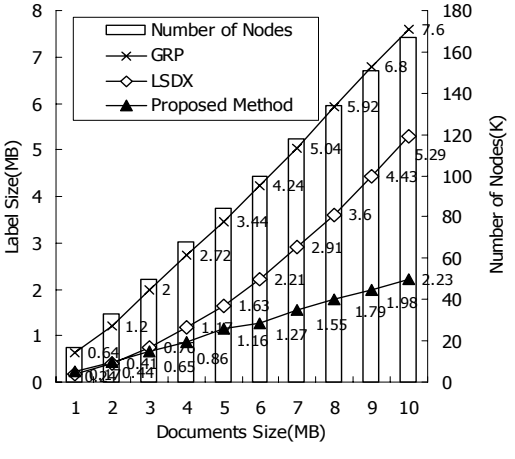


Fig. 8. Label Size

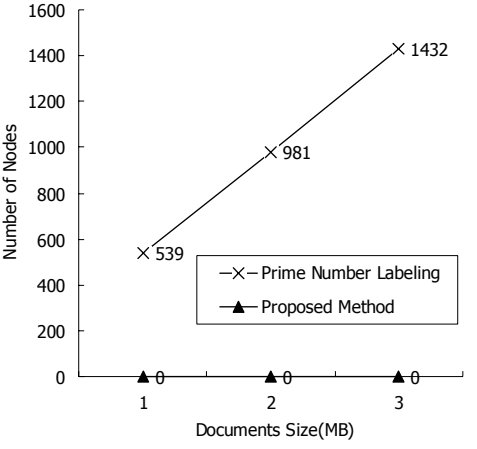


Fig. 9. Re-label Size

For prime number labeling [18] are required to re-calculate is counted in Fig 9. But our proposed labeling need not re-label the existing nodes in updates.

4.2 Rejection Query

A rejection query is a user query that has to be rejected at all cases. Thirty intended rejection queries that suited each type of query were made up, and access control policy and actual number of detection of rejection queries was compared to this. The result is shown in Fig. 10. The experiment was conducted in three cases: "/" axis, "//" axis, and a case that has a predicate in a terminal node. The result demonstrates that the intended 30 rejection queries were detected 100%.

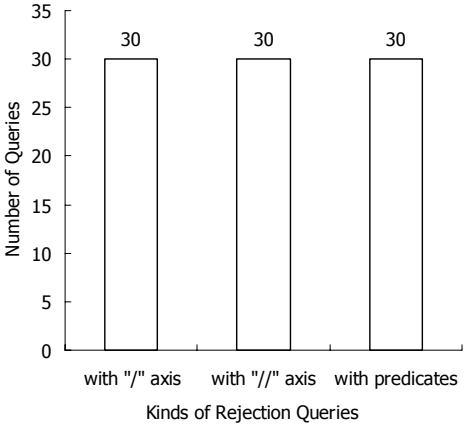


Fig. 10. Detection of Rejection Query

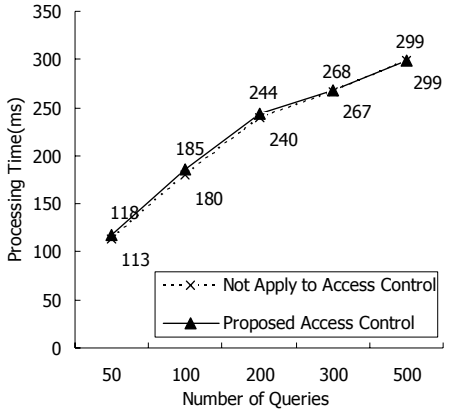


Fig. 11. Query Processing Time

4.3 Query Processing Time

Average query processing time was compared in two cases: one applied the access control method proposed in this paper and the other did not. Average processing time was measured according to random samples of XPath query numbers (50, 100, 200, 300, and 500). Processing time is represented by an average time so that error of measurement can be minimized. Proposed access control time was not included in the processing time in the proposed method because it is reconstructed when an update such as insertion or deletion of medical records XML documents is made. Referring to role-based prime number which is generated before query processing, and query processing time including the pure procedure of authority checking was measured. Nonetheless, the fact that referring to access control information does not affect the system performance was discovered. Fig. 11 shows the results.

5 Conclusions

We presented the improved role-based prime number labeling scheme for an efficient secure access control in dynamic XML data streams. And we pointed out the limitations of existing access control and labeling schemes for XML data assuming that documents are frequently updated. We proposed the improved dynamic role-based prime number labeling method where labels are encoded ancestor-descendant relationships and sibling relationship between nodes but need not to be regenerated when the document is updated. Our improved labeling scheme supports an infinite number of updates and guarantees the arbitrary nodes insertion at arbitrary position of the XML tree without label collisions.

In this paper, Medical records XML documents and the proposed environment have the characteristic of infinite additions in width rather than in depth because of the increment of patients. In this manner, the proposed method was able to fully manage the size of documents that increase to infinity and can minimize the maintenance cost caused by dynamic changes. In terms of security, system load is

minimized and a perfect access control is implemented by application of two-step access point check.

Acknowledgements. This work was supported by the Korea Science and Engineering Foundation(KOSEF) grant funded by the Korea government(MOST) (No. R01-2006-000-10609-0).

References

1. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and Issues in Data Stream Systems. In: PODS 2002 (2002)
2. Berglund, A., Boag, S., Chamberlin, D., Fernandez, M.F., Kay, M., Robie, J., Simon, J.: XML path language (XPath) 2.0. W3C working draft 16. Technical Report WD-xpath20-20020816. World Wide Web Consortium (2002)
3. Bertino, E., Castano, S., Ferrari, E., Mesiti, M.: Specifying and Enforcing Access Control Policies for XML Document Sources. WWW Journal (2000)
4. Bertino, E., Ferrari, E.: Secure and Selective Dissemination of XML Documents. TISSEC 5(3) (2002)
5. Boag, S., Chamberlin, D., Fernandez, M.F., Florescu, D., Robie, J., Simon, J.: XQuery 1.0: An XML Query Language. W3C working draft 16. Technical Report WD-xquery-20020816. World Wide Web Consortium (2002)
6. Bose, S., Fegaras, L.: XFrag: A Query Processing Framework for Fragmented XML Data. Web and Databases (2005)
7. Carminati, B., Ferrari, E., Tan, K.L.: Specifying Access Control Policies on Data Streams Outsourced Data. In: Li Lee, M., Tan, K.-L., Wuwongse, V. (eds.) DASFAA 2006. LNCS, vol. 3882. Springer, Heidelberg (2006)
8. Damiani, E., Vimercati, S., et al.: Securing XML Document. In: Zaniolo, C., Grust, T., Scholl, M.H., Lockemann, P.C. (eds.) EDBT 2000. LNCS, vol. 1777. Springer, Heidelberg (2000)
9. Damiani, E., Vimercati, S., et al.: Access Control System for XML Documents. ACM Trans. Information and System Sec. 5(2) (2002)
10. Duong, M., Zhang, Y.: LSDX: A new Labeling Scheme for Dynamically Updating XML Data. In: Australasian Database Conference (2005)
11. Fan, W., Fundulaki, I., Geerts, F., Jia, X., Kementsietsidis, A.: A View Based Security Framework for XML. In: AHM (2006)
12. Huo, H., Wang, G., Hui, X., Boning, R.Z., Xiao, C.: Efficient Query Processing for Streamed XML Fragments. In: Li Lee, M., Tan, K.-L., Wuwongse, V. (eds.) DASFAA 2006. LNCS, vol. 3882, pp. 468–482. Springer, Heidelberg (2006)
13. Li, C., Ling, T.W., Hu, M.: Efficient Processing of Updates in Dynamic XML Data. In: ICDE (2006)
14. Lu, J., Wang Ling, T.: Labeling and Querying Dynamic XML Trees, APWeb (2004)
15. Luo, B., Lee, D.W., Lee, W.C., Liu, P.: Qfilter: Fine-grained Run-Time XML Access Control via NFA-based Query Rewriting. In: CIKM 2004 (2004)
16. Murata, M., Tozawa, A., Kudo, M.: XML Access Control Using Static Analysis. In: ACM CCS, New York (2003)
17. Schmidt, A., Waas, F., Kersten, M., Carey, M.J., Manolescu, I., Busse, R.: XMark: a benchmark for XML data management. In: VLDB (2002)

18. Wu, X., Li, M., Hsu, L.: A Prime Number Labeling Scheme for Dynamic Ordered XML Trees. In: ICDE (2004)
19. Yoshikawa, M., Amagasa, T.: XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases. ACM Transactions on Internet Technology 1(1) (2001)
20. http://www.w3.org/TR/xml-fragment

Bitstream Encryption and Authentication Using AES-GCM in Dynamically Reconfigurable Systems

Yohei Hori[1], Akashi Satoh[1], Hirofumi Sakane[1], and Kenji Toda[1]

National Institute of Advanced Industrial Science and Technology (AIST)
1-1-1 Umezono, Tsukuba-shi, Ibaraki 305-8568, Japan

Abstract. A secure and dependable dynamic partial reconfiguration (DPR) system based on the AES-GCM cipher is developed, where the reconfigurable IP cores are protected by encrypting and authenticating their bitstreams with AES-GCM. In DPR systems, bitstream authentication is essential for avoiding fatal damage caused by inadvertent bitstreams. Although encryption-only systems can prevent bitstream cloning and reverse engineering, they cannot prevent erroneous or malicious bitstreams from being accepted as valid. If a bitstream error is detected after the system has already been partly configured, the system must be reconfigured with an errorless bitstream or at worst rebooted since the DPR changes the hardware architecture itself and the system cannot recover itself to the initial state by asserting a reset signal. In this regard, our system can recover from configuration errors without rebooting. To the authors' best knowledge, this is the first DPR system featuring both bitstream protection and error recovery mechanisms. Additionally, we clarify the relationship between the computation time and the bitstream block size, and derive the optimal internal memory size necessary to achieve the highest throughput. Furthermore, we implemented an AES-GCM-based DPR system targeting the Virtex-5 device on an off-the-shelf board, and demonstrated that all functions of bitstream decryption, verification, configuration, and error recovery work correctly. This paper clarifies the throughput, the hardware utilization, and the optimal memory configuration of said DPR system.

1 Introduction

Some recent Field-Programmable Gate Arrays (FPGAs) provide the ability of *dynamic partial reconfiguration (DPR)*, where a portion of the circuit is replaced with another module while the rest of the circuit remains fully operational. By using DPR, the functionality of the system is reactively altered by replacing hardware modules according to, for example, user requests, performance requirements, or environmental changes. To date, various applications of DPR have been reported: content distribution security [1], low-power crypto-modules [2], video processing [3], automotive systems [4], fault-tolerant systems [5] and software-defined radio [6] among others. It is expected that in the near future, it will be more popular for mobile terminals and consumer electronics to download hardware modules from the Internet in accordance with the intended use.

In DPR systems where intellectual property (IP) cores are downloaded from networks, encrypting the hardware configuration data (= bitstream) is a requisite for protecting the IP cores against illegal cloning and reverse engineering. Several FPGA families have

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 261–278, 2008.

embedded decryptors and can be configured using encrypted bitstreams. However, such embedded decryptors are available only for the entire configuration and not for DPR. In addition to bitstream encryption, bitstream authentication is significant for protecting DPR systems [7]. Encryption-only systems are not sufficiently secure as they cannot prevent erroneous or malicious bitstreams from being used for configuration. Since DPR changes the hardware architecture of the circuits, unauthorized bitstreams can cause fatal, unrecoverable damage to the system. In this regard, a mechanism of error recovery is essential for the practical use of DPR systems. If a bitstream error is detected after the bitstream has already been partly configured, the system must be reconfigured with an errorless bitstream. Note that the system cannot be recovered by asserting a reset signal since the hardware architecture itself has changed.

Based on the above considerations, we developed a DPR system which is capable of protecting bitstreams using AES-GCM (Advanced Encryption Standard [8]-Galois/Counter Mode [9, 10]) and recovering from configuration errors. To the authors' best knowledge, a DPR system featuring all mechanisms of bitstream encryption, bitstream verification and error recovery has not yet been developed, although several systems without recovery mechanism have been reported so far [11, 12, 13].

AES-GCM is one of the latest authenticated encryption (AE) ciphers which can guarantee both the confidentiality and the authenticity of message, and therefore AE could be effectively applied to DPR systems. Indeed, data encryption and authentication can be achieved with two separate algorithms, but if the area and speed performance of the two algorithms are not balanced, the overall performance is determined by the worse-performing algorithm. Therefore, AE is expected to enable more area-efficient and high-speed DPR implementations. Since other AE algorithms are not parallelizable or pipelinable, and thus not necessarily suitable for hardware implementation [14], the use of AES-GCM is currently the best solution for protecting bitstreams.

The configuration of a downloaded IP core starts after its bitstream is successfully verified. Bitstreams of large IP cores are split into several blocks, and verification is performed for each block. If the bitstream verification of a particular block fails after some other blocks have already been configured, the configuration process is abandoned, and reconfiguration starts with an initialization bitstream. In this configuration method, the size of the split bitstream significantly influences both the speed and the area performance. Since the decrypted bitstream must not flow out of the device and is thus stored to the internal memory, the size of the split bitstream determines the required memory resources. Although it is often thought that the speed performance can be improved by increasing the size of the available memory, our study revealed that the overall throughput can be maximized by using optimally sized internal memory.

This paper describes the architecture, memory configuration, implementation results, and performance evaluation of an AES-GCM-based DPR system featuring an error recovery mechanism. The system is implemented targeting Virtex-5 on an off-the-shelf board, and we demonstrate that its mechanisms of bitstream encryption, verification and error recovery work successfully. The rest of this paper is organized as follows. Section 2 introduces past studies on DPR security. Section 3 explains the process of partial reconfiguration of a Xilinx FPGA. Section 4 briefly explains the cryptographic algorithms related to our implementation. Section 5 describes the architecture of our

DPR system and explains the functions implemented in it. Section 6 determines the optimal memory configuration of the DPR system and describes the experimental results, implementation results, and evaluation of the systems. Finally, Section 7 summarizes this paper and presents future work.

2 Related Work

Xilinx Virtex series devices support configuration through encrypted bitstreams by utilizing built-in bitstream decryptors. Virtex-II and Virtex-II Pro support the Triple Data Encryption Standard (Triple-DES) [15] with a 56-bit key, while Virtex-4 and Virtex-5 support AES with a 256-bit key. The key is stored to the dedicated volatile memory inside the FPGA. Therefore, the storage must always be supplied with power through an external battery. Unfortunately, the functionality of configuration through encrypted bitstreams is not available when using DPR, and if the device is configured using the built-in bitstream decryptor, the DPR function is disabled. Therefore, in DPR systems, partial bitstreams must be decrypted by utilizing user logic.

Bossuet et al. proposed a secure configuration method for DPR systems [11]. Their system allows the use of arbitrary cryptographic algorithms since the bitstream decryptor itself is implemented as a reconfigurable module. However, although their method uses bitstream encryption, it does not consider the authenticity of the bitstreams.

Zeineddini and Gaj developed a DPR system which uses separate encryption and authentication algorithms for bitstream protection [12], where AES was used for bitstream encryption and SHA-1 for authentication. AES and SHA-1 were implemented as C programs and run on two types of embedded microprocessors: PowerPC and MicroBlaze. The total processing times needed for the authentication, decryption, and configuration of a 14-KB bitstream on PowerPC and MicroBlaze were approximately 400 ms and 2.3 sec, respectively. Such performances, however, would be insufficient for practical DPR systems.

Parelkar used AE to protect FPGA bitstreams [13], and implemented various AE algorithms: Offset CodeBook (OCB) [16], Counter with CBC-MAC (CCM) [17] and EAX [18] modes of operation with AES. In order to compare the performance of the AE method with separate encryption and authentication methods, SHA-1 and SHA-512 were also implemented using AES-ECB (Electronic CodeBook).

3 Partial Reconfiguration of FPGAs

This section briefly describes the architecture of Xilinx FPGAs and the features of partial reconfiguration with Xilinx devices. Detailed information about Xilinx FPGAs can be found in [19,20]. For more detailed information about Xilinx partial reconfiguration, see [21].

3.1 Xilinx FPGA

Xilinx FPGAs consist of *Configurable Logic Blocks (CLBs)*, which compute various logic, and an interconnection area which connects the CLBs. CLBs are composed of

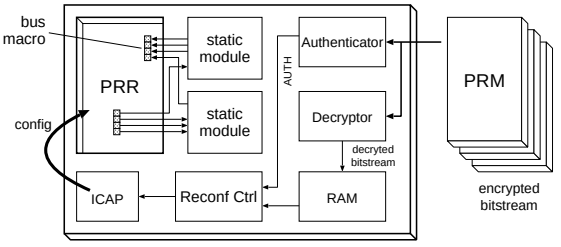


Fig. 1. Structure of a partially reconfigurable circuit in a Xilinx FPGA

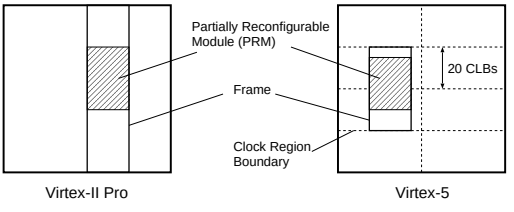


Fig. 2. Frame of Xilinx FPGAs

several reconfigurable units called *slices*, and slices in turn contain several *look-up tables (LUTs)*, which are the smallest reconfigurable logic units. In Virtex-5, each CLB contains two slices, and each slice contains four 6-input LUTs. In Virtex-4 and earlier Virtex series devices, each CLB contains four slices, and each slice contains two 4-input LUTs. While the LUTs can be used as memory, Xilinx FPGAs also contain dedicated memory blocks referred to as BlockRAMs or BRAMs.

3.2 Partial Reconfiguration Overview

In Xilinx FPGAs, modules which can be dynamically replaced are called *Partially Reconfigurable Modules (PRMs)*, and the areas where PRMs are placed are called *Partially Reconfigurable Regions (PRRs)*. PRMs are rectangular and can be of arbitrary size. Figure 1 shows an example structure of the partially reconfigurable design.

The smallest unit of a bitstream which can be accessed is called a *frame*. In Virtex-5 devices, a frame designates a 1312-bit piece of configuration information corresponding to the height of 20 CLBs. A bitstream of PRMs is a collection of frames. In Virtex-II Pro and earlier Virtex devices, the height of the frame is the same as the height of the device. Figure 2 illustrates the frames of Virtex-II Pro and Virtex-5.

3.3 Bus Macro

All signals between the PRMs and the fixed modules must pass through *bus macros* in order to lock the wiring. In Virtex-5 devices, the bus macros are 4-bit-wide pre-routed macros composed of four 6-input Lookup Tables (LUTs). The bus macros must be placed inside the PRMs. Furthermore, the bus macros of older device families are

8-bit-wide pre-routed macros composed of sixteen 4-input LUTs, which are placed on the PRM boundary.

3.4 Internal Configuration Access Port

Virtex-II and newer Virtex series devices support *self DPR* through the *Internal Configuration Access Port* (*ICAP*). ICAPs practically work in the same manner as the SelectMAP configuration interface. Since user logic can access the configuration memory through ICAPs, the partial reconfiguration of FPGAs can be controlled by internal user logic. In Virtex-5 devices, the data width of the ICAP can be set to 8, 16 or 32 bits.

4 Cryptographic Algorithm

4.1 Advanced Encryption Standard

AES is a symmetric key block cipher algorithm standardized by the U.S. National Institute of Standard and Technologies (NIST) [8]. AES replaces the previous Data Encryption Standard (DES) [22], whose 56-bit key is currently considered too short and not sufficiently secure. The block length of AES is 128 bits, and the key length can be set to 128, 196, or 256 bits.

4.2 Galois/Counter Mode of Operation

The GCM [9] is one of the latest modes of operation standardized by NIST [10]. Figure 3 shows an example of GCM operation mode.

In order to generate a message authentication code (MAC), which is also called a *security tag*, GCM uses universal hashing based on product-sum operation in the finite field $GF(2^w)$. The product-sum operation in $GF(2^w)$ enables faster and more compact hardware implementation compared to integer computation. The encryption and the decryption scheme of GCM is based on the CTR mode of operation [23], which can be highly parallelized and pipelined. Therefore, GCM is suitable for hardware implementation, entailing a wide variety of performance advantages such as compactness to high

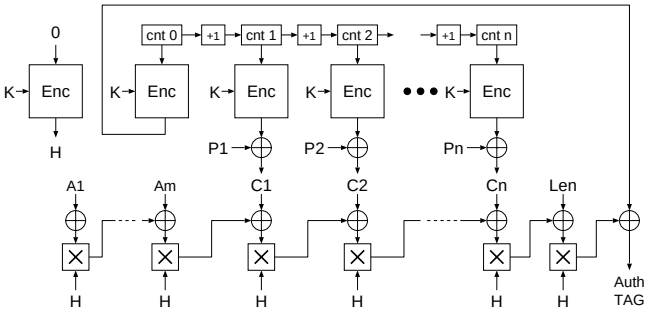


Fig. 3. Example operation of the Galois/Counter Mode (GCM)

speed [24, 25]. Other AE algorithms are not necessarily suitable for hardware implementation as they are impossible to parallelize or pipeline [14].

AES-GCM is one of the GCM applications which uses AES as the encryption core. Since AES is also based on the product-sum operation in $GF(2^w)$, either compact or high-speed hardware implementation is possible. Therefore, the use of AES-GCM can meet various performance requirements and is the best solution for protecting FPGA bitstreams in DPR systems.

5 AES-GCM-Based DPR Systems

This section describes the architecture of our DPR system, which uses AES-GCM for bitstream encryption/decryption and verification and is capable of recovering from configuration errors. Figure 4 shows a block diagram of said system. The length of the AES key and the initial vector (IV) are set to 128 bits and 96 bits, respectively, and the AES key is embedded into the system.

5.1 Configuration Flow Overview

Encrypted bitstreams from PRMs are transferred from the host computer via RS232 and are stored to the external 36x256K-bit SSRAM. The configuration of the PRM starts when a configuration command is sent from the host computer. The downloaded bitstreams are decrypted by the AES-GCM module, and their authenticity is verified simultaneously. Since the plain bitstreams must not leak out to the device, the decrypted bitstreams must be stored to the internal memory (Block RAM). Furthermore, since the size of the internal memory is relatively small, large bitstreams are split into several blocks, and decryption and verification is performed to each bitstream block. To distinguish the divided bitstream block from the AES 128-bit data block, we define the former as *Bitstream Block* (*BSB*). In the system, the memory size is set to $128x2^k$ bits, and is at most 128x8192 (1 Mb) due to device resource limitations. After the integrity of the bitstream has been verified, the decrypted bitstream is read from the internal memory and transferred to the ICAP to configure the PRM.

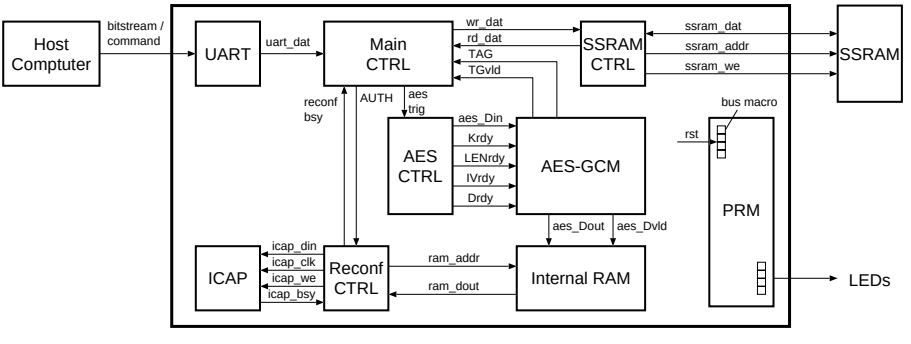


Fig. 4. Overview of the system using AES-GCM

Note that AES-GCM requires initial processing such as key scheduling and IV setup for each BSB. Therefore, the computation effort for the same bitstream increases with the number of BSBs. The smaller the internal memory is, the more compact the system will be; however, computation effort will increase. Conversely, if the memory size is large, computation effort will decrease, although the system will require more hardware resources. Furthermore, since additional data such as a security tag, IV, and data length, are attached to each BSB, the size of the downloaded data increases with the number of BSBs. The trade-off between internal memory size, downloaded data size and computation effort is clarified in Section 5.3 and Section 5.4.

The consideration is that simply dividing a bitstream into several BSBs will be vulnerable against *removal* or *insertion* of a BSB. Though AES-GCM can detect tampering with the BSB, it does not care the number or order of the successive BSBs. For example, even if one of the successive BSBs is removed, AES-GCM cannot detect the disappearance of the BSB and thus the system would be incompletely configured. In addition, if a malicious BSB with its correct security tag is inserted to the series of the BSBs, AES-GCM will recognize the inserted BSB as legitimate and thus the malicious BSB will be configured in the device, causing system malfunction, data leakage and so on. Therefore, some protection scheme to prevent BSB removal and insertion is necessary for DPR systems. The protection scheme against these problems is discussed in section 5.7.

5.2 Data Structure

In order to decrypt a PRM bitstream with AES-GCM, information about the security tag, data length, and IV need to be appended to the head of the bitstream. Large bitstreams are divided into several BSBs, and each BSB contains such header information. In addition, the first BSB contains information about the total bitstream length. Figure 5

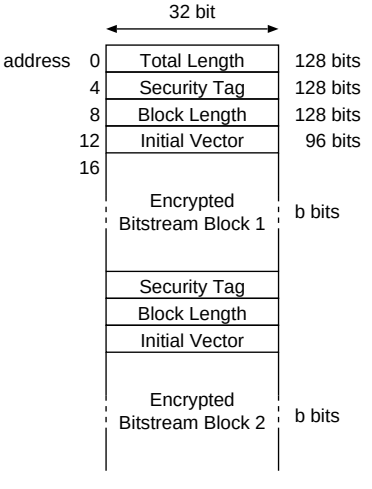


Fig. 5. General structure of bitstreams stored to SSRAM

shows the structure of the downloaded bitstream together with the header information, which is loaded from SSRAM and set to the registers in the AES-GCM module when the PRM configuration begins.

5.3 Bitstream Decryption and Verification

In the AES-GCM module, the major component (the S-box) is implemented using composite field. The initial setup of AES-GCM takes 59 cycles, and the first BSB takes 19 additional cycles for setting up the total length of the entire bitstream. A 128-bit data block is decrypted in 13 clock cycles, including SSRAM access time, and the decrypted data are stored to the internal memory. The last block of BSB requires 10 clock cycles in addition to the usual 13 for the purpose of calculating the security tag. The security tag is calculated using $GHAS\,H$ function defined below, where A is the additional authentication data, C is the ciphertext and H is the hash subkey.

$$X_i = \begin{cases} 0 & i = 0 \\ (X_{i-1} \oplus A_i) \cdot H & i = 1, \ldots, m-1 \\ (X_{m-1} \oplus (A_m^* \| 0^{128-v})) \cdot H & i = m \\ (X_{i-1} \oplus C_{i-m}) \cdot H & i = m+1, \ldots, m+n-1 \\ (X_{m+n-1} \oplus (C_n^* \| 0^{128-u})) \cdot H & i = m+n \\ (X_{m+n} \oplus (\mathrm{len}(A) \| \mathrm{len}(C))) \cdot H & i = m+n+1 \end{cases} \tag{1}$$

The final value X_{m+n+1} becomes the security tag. In $GHAS\,H$ function, the 128 x 128-bit multiplication over Galois Field (GF) is achieved using 128 x 16-bit GF multiplier eight times for saving the hardware resources. Fig.6 shows the GF multiplier implemented in the AES-GCM module. The partial products of the 128 x 16-bit multiplier are summed up into the 128-bit register Z. The calculation of Z finishes in 8 clock cycles.

An example timing chart of the AES-GCM module including the initial setup is shown in Figure 7. Suppose that the size of the entire bitstream is S bits, and that it is split into n BSBs. Let the size of the k th BSB be b_k bits, and $b_1, b_2, \ldots, b_{n-1}$ be BSBs of the same size b. Then, the entire size S is expressed as follows:

$$S = \sum_{k=1}^{n} b_k = \sum_{k=1}^{n-1} b + b_n = (n-1) \cdot b + b_n. \tag{2}$$

As Figure 7 illustrates, the required number of clock cycles T_{aes} for the decryption and verification of the entire bitstream is

$$\begin{aligned} T_{aes} &= 19 + (n-1) \cdot \left(59 + 13 \cdot \frac{b}{128} + 10 + 2\right) + \left(59 + 13 \cdot \frac{b_n}{128} + 10 + 2\right) \\ &= 19 + \frac{13\,(n-1)\,b + 13\,b_n}{128} + 71\,n \\ &= \frac{13\,S}{128} + 71\,n + 19 \qquad (\because S = (n-1)\,b + b_n)\,. \end{aligned} \tag{3}$$

As the above equation indicates, the computation effort for AES-GCM increases with the number of BSBs n.

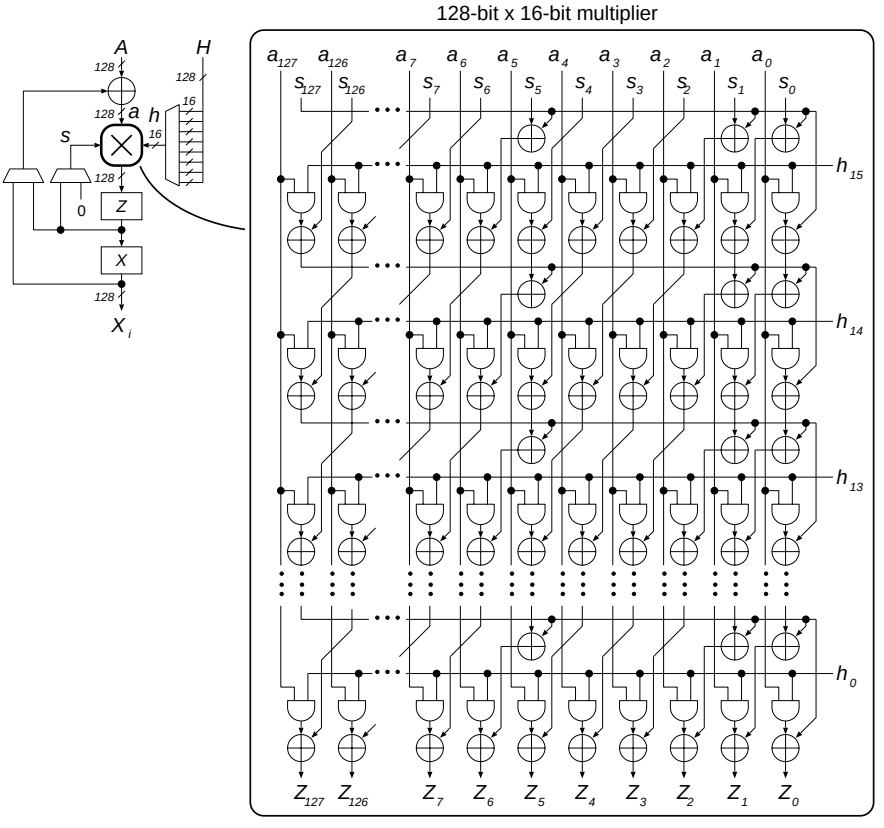


Fig. 6. The architecture of the Galois Field multiplier

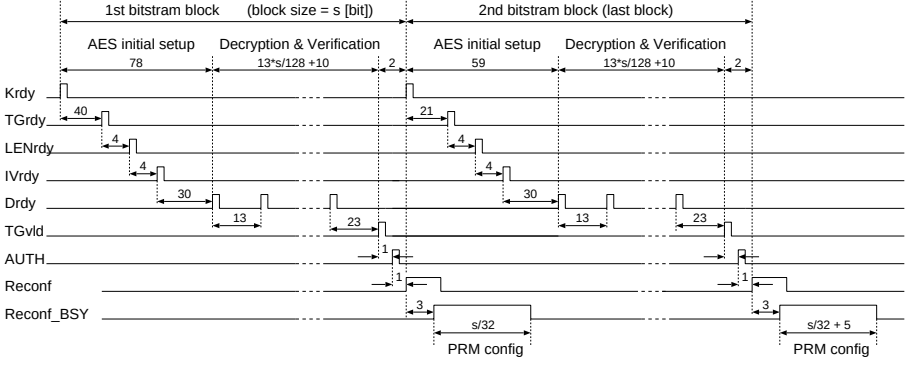


Fig. 7. Timing chart of decryption, verification, and reconfiguration

5.4 PRM Configuration

Unlike other DPR systems, our system does not use an embedded processor to control the partial reconfiguration. The input data and control signals from the ICAP are directly connected to and controlled by the user logic. Thus, our system is free from the delay of processor buses. In the system, the width of the ICAP data port is set to 32 bits. When the frequency of the input data to the ICAP is f [MHz], the throughput of the reconfiguration process P_{icap} is

$$P_{icap} = 32f \text{ [Mbps]}. \tag{4}$$

In Virtex-5, the maximum frequency of the ICAP is limited to 100 MHz, thus the ideal throughput of the reconfiguration process is 3,200 Mbps.

Figure 7 also shows the timing of the configuration of the PRM bitstream. When the size of the BSB is b bits, the configuration of the BSB finishes in $b/32$ cycles. The last BSB takes 5 additional cycles to flush the buffer in the device. Therefore, the required number of computation cycles for the PRM configuration T_{reconf} is

$$T_{reconf} = (n-1) \cdot \frac{b}{32} + \left(\frac{b_n}{32} + 5\right) = \frac{S}{32} + 5 \qquad (\because S = (n-1)\, b + b_n). \tag{5}$$

5.5 Error Recovery

In the system, the first several bytes of the SSRAM are reserved for the *initialization PRM*, which is used for recovering the system from DPR errors. The use of the initialization PRM enables the system to return to the start-up state without rebooting the entire system. Thus, processes executed in other modules can retain their data even when DPR errors occur. The bitstream of the initialization PRM is encrypted and processed in the same way as that of other PRMs. If the bitstream size is S bits, the computation time for decryption, verification, and configuration is derived from equations (3) and (5).

When bitstream verification fails with AES-GCM, the current process is abandoned and configuration of the initialization PRM is started. Note that the unauthorized BSB is still in the internal memory and it will be overwritten by the initial PRM. Therefore, the unauthorized bitstream will be safely erased and will not be configured in the system. If the verification of the initialization PRM fails due to, for example, bitstream tampering or memory bus damage, the system discontinues the configuration process and prompts the user to reboot the system.

5.6 Overall Computation Time

The decryption, verification, and configuration of the BSBs is processed in a course-grained pipeline, as shown in Figure 7. The configuration of all BSBs except the last BSB overlaps with the decryption process. Therefore, the total computation time T, including bitstream encryption, verification, and configuration, is

$$\begin{aligned} T &= \left(\frac{13}{128}S + 71\,n + 19\right) + \left(\frac{b_n}{32} + 5\right) \\ &= \frac{13}{128}S + 71\,n + \frac{b_n}{32} + 24. \end{aligned} \tag{6}$$

If the bitstream encryption, verification and configuration cannot be processed in a pipeline, the total number of computation cycles T' is

$$\begin{aligned} T' &= T_{aes} + T_{reconf} \\ &= \left(\frac{13}{128}S + 71\,n + 19\right) + \left(\frac{S}{32} + 5\right) \\ &= \frac{17}{128}S + 71\,n + 24. \end{aligned} \tag{7}$$

Considering that $S \geq b_n$, the improvement of the computation time due to the use of a course-grained pipeline architecture is

$$T' - T = \frac{S - b_n}{32} \ (\geq 0). \tag{8}$$

5.7 Countermeasure against BSB Removal and Insertion

As mentioned in section 5.1, dividing the bitstream into several BSBs is vulnerable against attacks of BSB removal and insertion. One scheme to protect such attacks is to use sequential numbers as the initial vector (IV) for calculating security tag. In this protection scheme, each BSB has Block Number (BN) that denotes the position of the BSB in the bitstream. The initial BN is unique to each PRM. The BN of the first BSB is used as IV and simultaneously stored to the internal register or memory. The stored BN is incremented and used as IV every time a BSB is loaded. If the loaded BSB has different BN from the stored value, the configuration is immediately terminated and the recovery process is started.

The computation time slightly increases when BN is used for the bitstream protection, because reading BN from SSRAM takes several clock cycles. Suppose that the length of BN is l_{BN}. The clock cycles required to read BN are $\lceil l_{BN}/32 \rceil$, as the width of the SSRAM is 32 bits. In this case, the total computation time with pipeline processing (T_{BN}) is

$$\begin{aligned} T_{BN} &= \left(\frac{13}{128}S + \left(71 + \left\lceil \frac{l_{BN}}{32} \right\rceil\right) n + 19\right) + \left(\frac{b_n}{32} + 5\right) \\ &= \frac{13}{128}S + \left(71 + \left\lceil \frac{l_{BN}}{32} \right\rceil\right) n + \frac{b_n}{32} + 24. \end{aligned} \tag{9}$$

The increased time due to the use of BN is

$$T_{BN} - T = \left\lceil \frac{l_{BN}}{32} \right\rceil n. \tag{10}$$

The equation (10) indicates that the additional BN will have more effect on computation time as the number of the BSBs n increases. As is given in Section 6.2, the size of n is typically 4 to 16. Thus, the time increase caused by using BN is quite small compared to the total computation time.

This study is the first step toward developing a secure practical DPR system and its main purpose is to demonstrate the feasibility of the recovery mechanism of the AES-GCM-based DPR system, so the additional protection logic with BN is currently not implemented. Implementing the additional protection logic is left as future work.

6 Implementation

This section describes the implementation results of the abovementioned AES-GCM-based DPR system (hereinafter PR-AES-GCM). PR-AES-GCM is implemented targeting Virtex-5 (XC5VLX50T-FFG1136) on an ML505 board [26]. The systems are designed using Xilinx Early Access Partial Reconfiguration (EA PR) flow [27] and are implemented with ISE 9.1.02i_PR10 and PlanAhead 9.2.7 [28].

6.1 PRM Implementation

In order to test whether all mechanisms of bitstream encryption, verification, and error recovery work properly, we implemented two simple function blocks, a 28-bit up-counter, and a 28-bit down-counter as PRMs. In addition, two bus macros were placed in the PRR for the input and output signals, respectively. The most significant 4 bits of the counter were the outputted from the PRM and connected to LEDs on the board. The PRR contained 80 slices, 640 LUTs, and 320 registers. The size of the bitstream for this area became about 12 KB (= 96 K bits), which could change slightly depending on the complexity of the implemented functions. The sizes of the up-counter and down-counter PRMs were 87,200 and 85,856 bits, respectively.

6.2 Internal Memory

In order to determine the required size of the internal memory, equation (6) should be transformed to express the relationship between T and b. For estimation purposes, we suppose that the size of the last BSB b_n is b bits. In this case, equation (6) is rewritten as follows:

$$\begin{aligned} T &= \left(\frac{13}{128}S + 71\,n + 19\right) + \left(\frac{b_n}{32} + 5\right) \\ &= \frac{13}{128}S + \frac{71\,S}{b} + \frac{b}{32} + 24. \quad (\because S = n \cdot b) \end{aligned} \tag{11}$$

Figure 8 illustrates the variation of the total computation time T in accordance with the BSB size b under the conditions $S = 2^{16}$, 2^{17}, 2^{18}, 2^{19} and 2^{20}. For comparison, equation (7) is transformed as follows, and its graph is also shown in Figure 8.

$$\begin{aligned} T' &= T_{aes} + T_{reconf} \\ &= \left(\frac{13}{128}S + 71\,n + 19\right) + \left(\frac{S}{32} + 5\right) \\ &= \frac{17}{128}S + \frac{71\,S}{b} + 24. \quad (\because S = n \cdot b) \end{aligned} \tag{12}$$

As Figure 8 clearly shows, the course-grained pipeline architecture is effective for shortening the overall processing time. The computation cycles in non-pipelined circuits decrease monotonically, while those in pipelined circuits have minimal values, as indicated by the arrows in Figure 8. When the entire size S is 2^{16}, 2^{17}, 2^{18}, 2^{19} or 2^{20}, the respective BSB sizes b which minimize T are 2^{14}, 2^{14}, 2^{15}, 2^{15} and 2^{16}. The most

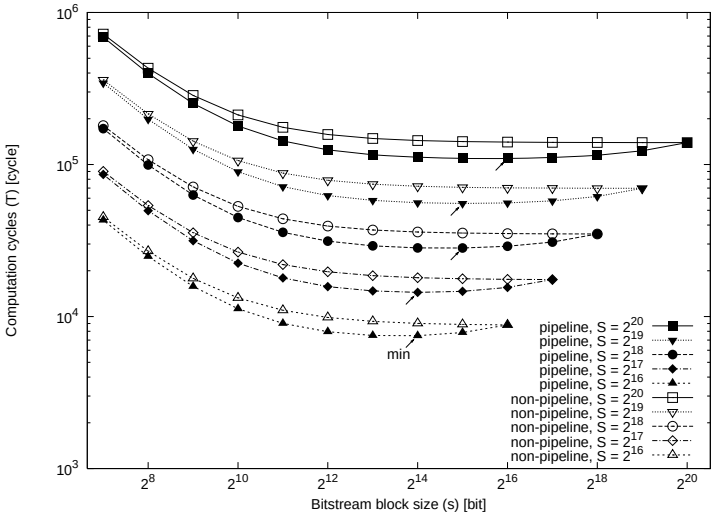


Fig. 8. Relationship between the BSB size b and the total number of computation cycles T and T'

time-efficient DPR systems were realized by setting the size of the internal memory to b as derived here. Equation (11) is useful for balancing the computation time and circuit size under the required speed and area performance.

Incidentally, the system with the BN-based protection shows completely the same results as ones given above, that is, the respective optimal sizes b are 2^{14}, 2^{14}, 2^{15}, 2^{15} and 2^{16} for the same S values.

After deriving the relationship between T and b, we determined the most time-efficient memory configuration for the PRMs introduced in Section 6.1. The size S should be set to a slightly larger value than the prepared PRMs in order to accommodate other PRMs with different sizes. Therefore, S is set to 2^{17}, which is the minimal 2^w meeting the requirement $2^w > 87200$. As Figure 8 illustrates, the optimal BSB size b under the condition $S = 2^{17}$ is 2^{14}. Therefore, the internal memory configuration is set to 128×128 ($= 2^{14}$) bits.

6.3 Hardware Resource Utilization

Table 1 shows the hardware utilization of PR-AES-GCM implemented on a Virtex-5. The "Overall" item shows the total amount of hardware resources used by all modules except PRM. Table 1 also describes the hardware utilization of each module as a standalone implementation.

The hardware architecture of Virtex-5 is vastly different from that of earlier devices such as Virtex-II Pro and Virtex-4. Each slice in Virtex-5 contains four 6-input LUTs, whereas that of earlier devices contains two 4-input LUTs. Thus, the number of slices is smaller in the Virtex-5 implementation. In order to give a fair comparison with other studies, we also implemented the above system on a Virtex-II Pro (XC2VP30-FF896). The hardware utilization of PR-AES-GCM on Virtex-II Pro is given in Table 2.

Table 1. Hardware utilization of the static module of PR-AES-GCM on Virtex-5 (XC5VLX50T)

Module	Register	(%)	LUT	(%)	Slice	(%)	BRAM	(%)
Overall	2,876	10.0%	5,965	20.7%	1,958	27.2%	5	8.3%
AES-GCM	1,382	4.8%	3,691	12.8%	1,615	22.4%	0	0.0%
MAIN_CTRL	463	1.6%	643	2.2%	360	5.0%	0	0.0%
AES_CTRL	164	0.6%	277	1.0%	192	2.7%	0	0.0%
SSRAM_CTRL	103	0.4%	174	0.6%	97	1.3%	1	1.7%
RECONF_CTRL	68	0.2%	142	0.5%	76	1.1%	0	0.0%
RAM_CTRL	143	0.5%	156	0.5%	161	2.2%	0	0.0%
CONFIG_RAM	0	0.0%	0	0.0%	0	0.0%	4	6.7%

Table 2. Hardware utilization of the static module of PR-AES-GCM on Virtex-II Pro (XC2VP30)

Module	Register	(%)	LUT	(%)	Slice	(%)	BRAM	(%)
Overall	2,900	10.6%	8,080	29.5%	4,900	35.8%	4	2.9%
AES-GCM	1,387	5.1%	5,566	20.3%	3,233	23.6%	0	0.0%
MAIN_CTRL	463	1.7%	1,133	4.1%	713	5.2%	0	0.0%
AES_CTRL	173	0.6%	316	1.2%	166	1.2%	0	0.0%
SSRAM_CTRL	103	0.4%	218	0.8%	132	1.0%	0	0.0%
RECONF_CTRL	59	0.2%	153	0.6%	94	0.7%	0	0.0%
RAM_CTRL	143	0.5%	168	0.6%	97	0.7%	0	0.0%
CONFIG_RAM	0	0.0%	0	0.0%	0	0.0%	4	2.9%

Here we consider the hardware utilization of the additional protection logic using BN. The logic needs registers or memory to store BN and comparators to verify if the BSB has correct BN. In addition, an adder is required to increment the BN stored in the register. To estimate the required resources for the protection logic, we implemented it on Virtex-5 and Virtex-II Pro under the condition that the size of BN is 128 bits. As a result, the logic utilizes 129 registers, 173 LUTs and 45 slices on Virtex-5, and 129 registers, 194 LUTs and 99 slices on Virtex-II Pro. These utilizations are all less than 1% of the entire resources. Therefore, the additional circuit will have little effect on the resource utilization of the whole system.

6.4 DPR Experiments

In order to experimentally demonstrate that all functions of bitstream encryption, verification, and configuration as well as the error recovery mechanism operate correctly, we configured the PRMs on the developed DPR system. Figure 9 shows the structure of the bitstreams in the DPR experiment. The PRM with the up-counter (hereinafter PRM0) is placed at address 0 as the initialization bitstream, and the PRM with the down-counter (hereinafter PRM1) is placed at address 8192. Configuration with an erroneous bitstream was emulated by inverting the first byte of the bitstream of PRM1 and using the bitstream thus obtained for PRM2.

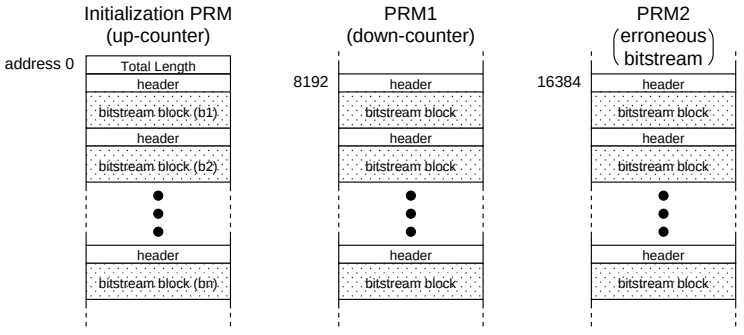


Fig. 9. Bitstream structure in SSRAM in the DPR experiment

The experimental procedure is outlined below.

1. The system is booted. Note that the most significant 4 bits of the counter in the PRM0 are connected to LEDs on the board.
2. The configuration command is sent from the host computer with the SSRAM address "8192" to configure PRM1.
3. The bitstream at address 8192 is loaded from SSRAM, decrypted, verified, and configured.
4. The configuration command is sent from the host computer with the SSRAM address "16384" to configure PRM2.
5. The bitstream at address 16384 is loaded from SSRAM, decrypted, verified, and configured.

When the system was booted, the LEDs indicated that the up-counter was implemented in PRM0, and after PRM1 was configured, the LEDs indicated that the down-counter was implemented in PRM1. This result shows that the decryption and verification with AES-GCM worked correctly and that DPR was performed successfully.

After PRM2 was configured, the LEDs indicated that the up-counter was implemented in PRM0. Note that PRM2 is an erroneous bitstream generated based on the output of PRM1, which is equipped with the down-counter. This result shows that the configuration of PRM2 failed and the system was reconfigured with PRM0, which is equipped with the up-counter. Therefore, the error recovery mechanism was demonstrated to operate correctly.

6.5 Performance Evaluation

The clock frequency of PR-AES-GCM is 100 MHz. In order to enable comparison with [12], the computation time required to configure a 14,112-byte (112,896-bit) PRM is described in Table 3. Decryption, verification, and configuration with PR-AES-GCM can be implemented in a pipeline, and the respective computation time is derived from equation (6).

In PowerPC and MicroBlaze systems, authentication, decryption, and reconfiguration are performed sequentially, and therefore the overall processing time is simply the

Table 3. Comparison of the performances of different secure PR systems (14,112 bytes PRM)

System	Device	Slice	Verification	Decryption	Configuration	Overall	Ratio
PR-AES-GCM	XC5VLX50T	4,900*	119.110 μs		35.3 μs	123.72 μs	1
			947.8 Mbps		3195 Mbps	913 Mbps	
PowerPC [12]	XC2VP30	1,334**	139 ms	208 ms	56 ms	403 ms	3257
			812 kbps	543 kbps	2016 kbps	280 kbps	
MicroBlaze [12]	XC2VP30	1,706**	776 ms	1472 ms	32 ms	2280 ms	18429
			145 kbps	77 kbps	3528 kbps	50 kbps	
AES-OCB [13]	XC4VLX60	2,964	601 Mbps		-	-	
AES-CCM [13]	XC4VLX60	2,799	255 Mbps		-	-	
AES-EAX [13]	XC4VLX60	2,993	287 Mbps		-	-	

* The slice utilization of Virtex-II Pro is shown for the purpose of fair comparison.
** Includes only the reconfiguration controllers.

sum of the processing times of each step. Table 3 also gives the throughput of other AE algorithms as reported in [13].

6.6 Analysis of the Results

The results of the experiment in Section 6.4 indicate that all functions of bitstream decryption, verification, configuration, and error recovery work properly. Thus, the system described above is the first operational DPR system featuring both bitstream protection and error recovery mechanisms.

As shown in Table 3, PR-AES-GCM achieved the highest overall throughput of over 900 Mbps with only about 1/3 slice utilization. Note that PR-AES-GCM includes error recovery logic, an SSRAM controller, etc. Additionally, the AES-GCM module achieved a throughput of about 950 Mbps, which is faster than those of other AE methods of OCB, CCM, and EAX. It is remarkable that such high throughput is achieved with such small size of the internal memory as determined by equation (11). The performance of the system is often thought to improve as the memory size increases. However, in course-grained DPR architectures, equation (11) reveals that optimally sized internal memory can maximize the throughput of the entire system. The device can accommodate at most 128×2^{13} bits of memory, while our system uses only 128×2^7 bits. Therefore, sufficient memory resources are available for various user logic.

Furthermore, PowerPC and MicroBlaze DPR systems require an overall computation time between several hundred milliseconds and several seconds, which is unacceptable for practical DPR systems. Therefore, authentication, decryption, and reconfiguration should be processed using dedicated hardware in order to realize practical DPR systems. Compared to software AE systems, our approach attained extremely high performance, where PR-AES-GCM achieved a 3257 times higher throughput than the PowerPC system and an 18429 times higher throughput than the MicroBlaze system.

7 Conclusions

We developed a secure and dependable dynamic partial reconfiguration (DPR) system featuring AES-GCM authentication and error recovery mechanisms. Furthermore, it

was experimentally demonstrated that the functions of bitstream decryption, verification, configuration, and error recovery operate correctly. To the authors' best knowledge, this is the first operational DPR system featuring both bitstream protection and error recovery mechanisms.

Through the implementation of the above system on a Virtex-5 (XC5VLX50T), AES-GCM achieved a throughput of about 950 Mbps, and the entire system achieved a throughput of more than 910 Mbps, which is sufficient for practical DPR use, and utilized only 1/3 of the slices. This performance is higher than that of other modes of operation such as OCB, CCM, and EAX.

Remarkably, it was found that using optimally sized internal memory entails the highest throughput in the DPR system. Although it is often thought that the performance of the system improves as the memory increases, our study revealed that optimizing the size of the internal memory depending on the size of the entire bitstream provides the shortest processing times. Thus, our system was able to achieve the highest throughput with the least amount of memory resources.

The future work of this study is to implement further security mechanisms to prevent attacks such as the bitstream block removal and insertion. This paper showed that the protection scheme using block numbers as the initial vector would be implemented with hardly sacrificing the computation time and hardware resources. Another future work is to develop various application systems, such as content distribution and multi-algorithm cryptoprocessors, based on the DPR system described above.

References

1. Hori, Y., Yokoyama, H., Sakane, H., Toda, K.: A secure content delivery system based on a partially reconfigurable FPGA. IEICE Trans. Inf.&Syst. E91-D(5), 1398–1407 (2008)
2. Hori, Y., Sakane, H., Toda, K.: A study of the effectiveness of dynamic partial reconfiguration for size and power reduction. In: IEICE Tech. Rep. RECONF2007-56, pp. 31–36 (January 2008) (in Japanese)
3. Claus, C., Zeppenfeld, J., Muller, F., Stechele, W.: Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system. In: DATE 2007, pp. 498–503 (2007)
4. Becker, J., Hubner, M., Hettich, G., Constapel, R., Eisenmann, J., Luka, J.: Dynamic and partial FPGA exploitation. Proc. IEEE 95(2), 438–452 (2007)
5. Emmert, J., Stroud, C., Skaggs, B., Abramovici, M.: Dynamic fault tolerance in FPGAs via partial reconfiguration. In: FCCM 2000, pp. 165–174 (2000)
6. Delahaye, J.P., Gogniat, G., Roland, C., Bomel, P.: Software radio and dynamic reconfiguration on a DSP/FPGA platform. J. Frequenz 58(5-6), 152–159 (2004)
7. Drimer, S.: Authentication of FPGA bitstreams: Why and how. In: Diniz, P.C., Marques, E., Bertels, K., Fernandes, M.M., Cardoso, J.M.P. (eds.) ARCS 2007. LNCS, vol. 4419, pp. 73–84. Springer, Heidelberg (2007)
8. National Institute of Standards and Technology: Announcing the advanced encryption standard (AES). FIPS PUB 197 (November 2001)
9. McGrew, D.A., Viega, J.: The Galois/counter mode of operation (GCM) (May 2005), `http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html`
10. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. National Institute of Standards and Technology. SP 800-38D edn. (November 2007)

11. Bossuet, L., Gogniat, G.: Dynamically configurable security for SRAM FPGA bitstreams. Int. J. Embedded Systems 2(1/2), 73–85 (2006)
12. Zeineddini, A.S., Gaj, K.: Secure partial reconfiguration of FPGAs. In: ICFPT 2005, pp. 155–162 (2005)
13. Parelkar, M.M.: Authenticated encryption in hardware. Master's thesis, George Mason University (2005)
14. McGrew, D.A., Viega, J.: The security and performance of the Galois/counter mode (GCM) of operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer, Heidelberg (2004)
15. National Institute of Standards and Technology: Recommendation for the triple data encryption algorithm (TDEA) block cipher (May 2004)
16. Rogaway, P., Bellare, M., John, B.: OCB: A block-cipher mode of operation for efficient authenticated encryption. ACM Trans. Information and System Security 6(3), 365–403 (2003)
17. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM). RFC3610 (September 2003)
18. Bellare, M., Rogaway, P., Wagner, D.: A conventional authenticated-encryption mode (2003), `http://www-08.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/eax/eax-spec.pdf`
19. Xilinx, Inc.: Virtex-5 User Guide (2007)
20. Xilinx, I.: Virtex-4 User Guide (2007)
21. Lysaght, P., Blodget, B., Mason, J., Young, J., Bridgford, B.: Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs. In: FPL 2006, pp. 12–17 (2006)
22. U.S. Department of Commerce/National Institute of Standards and Technology: Data Encryption Standard (DES). FIPS PUB 46-3 edn. (1999)
23. Dworkin, M.: Recommendation for Block Cipher Modes of Operation. National Institute of Standards and Technology. SP 800-38A edn. (December 2001)
24. Satoh, A.: High-speed parallel hardware architecture for Galois counter mode. In: ISCAS 2007, pp. 1863–1866 (2007)
25. Satoh, A., Sugawara, T., Aoki, T.: High-speed pipelined hardware architecture for Galois counter mode. In: Garay, J.A., Lenstra, A.K., Mambo, M., Peralta, R. (eds.) ISC 2007. LNCS, vol. 4779, pp. 118–129. Springer, Heidelberg (2007)
26. Xilinx, Inc.: ML505/ML506 Evaluation Platform. UG347(v2.4) edn. (October 2007)
27. Xilinx, Inc.: Early Access Partial Reconfiguration User Guide For ISE 8.1.01i (2006)
28. Jackson, B.: Partial Reconfiguration Design with PlanAhead 9.2. Xilinx, Inc. (August 2007)

The Long-Short-Key Primitive and Its Applications to Key Security

Matthew Cary[1], Matthias Jacob[2], Mariusz H. Jakubowski[3], and Ramarathnam Venkatesan[3]

[1] Google
[2] Nokia
[3] Microsoft Research

Abstract. On today's open computing platforms, attackers can often extract sensitive data from a program's stack, heap, or files. To address this problem, we designed and implemented a new primitive that helps provide better security for ciphers that use keys stored in easily accessible locations. Given a particular symmetric key, our approach generates two functions for encryption and decryption: The *short-key* function uses the original key, while the functionally equivalent *long-key* version works with an arbitrarily long key derived from the short key. On common PC architectures, such a long key normally does not fit in stack frames or cache blocks, forcing an attacker to search memory space. Even if extracted from memory, the long key is neither easily compressible nor useful in recovering the short key. Using a pseudorandom generator and additional novel software-protection techniques, we show how to implement this construction securely for AES. Potential applications include white-box ciphers, DRM schemes, software smartcards, and challenge-response authentication, as well as any scenario where a key of controllable length is useful to enforce desired security properties.

1 Introduction

On today's computing systems, security often relies on public-key cryptography and symmetric ciphers. Protection of cryptographic keys against eavesdroppers is a crucial issue. While the problem of keeping secret keys secret during key exchange is well understood in principle, hiding data in memory or on disk is difficult on open platforms. In particular, symmetric keys are often used for performance reasons, but support only "private" operations that must be secured against malicious observation and tampering. For example, viruses and bots can scan a user's hard drive or launch side-channel attacks on cryptographic functions [25, 30]. Widely available tools like debuggers and system monitors facilitate both manual and automated extraction of keys from unsecured storage.

In practice, software has used various key-protection approaches to create roadblocks against access to sensitive data at runtime. For example, white-boxing [8, 9] transforms keys into code that performs cryptographic operations without using key material explicitly. Obfuscation and tamper-resistance techniques [1, 2, 6, 10, 11, 21, 28] similarly help to prevent hackers from easily finding

K. Matsuura and E. Fujisaki (Eds.): IWSEC 2008, LNCS 5312, pp. 279–297, 2008.

and extracting keys. Unfortunately, such methods provide limited solutions that work only under restrictions and offer few security guarantees or analyses.

In this paper, we approach the problem from a different angle and implement a secure cryptographic system based on our *long-short-key primitive (LSK)*. LSK is a universal tool to address concerns with easy key extraction and distribution. Like white-boxing, our method generates code for cryptographic operations with a given key. However, instead of obfuscation, we transform the key into an arbitrarily long data stream that is used to implement encryption and decryption. This transform is one-way, so that the original short key cannot be efficiently reconstructed from the long version. Moreover, the long key cannot be easily compressed, forcing a lower bound on the size of any hack derived from the long key. The idea of LSK is related to theoretical approaches to bounded storage and retrieval [5, 14]; however, earlier work has not applied such techniques to the problem of keys exposed during cryptographic operations.

More specifically, our approach accepts a symmetric key as input and generates two corresponding functions for cryptographic operations using that key:

- The *short-key* function uses the original unprotected cipher key for efficient encryption and decryption. The secret key may be either incorporated into the function code or passed in as a parameter, so that a single short-key function can suffice for all keys. Typically compact and efficient, this function's implementation should run only in a secure environment, such as a protected server or tamper-resistant smartcard.
- The *long-key* function is operationally equivalent to the short-key version, but uses an arbitrarily large data block derived from the short key. The size of the long key is a security parameter. The owner of the short key generates the long key from the short key and an optional secret. Given the long key, efficient recovery of the short key is not possible. The entire long key is required for encryption and decryption of data in general, though a particular input text may use only a portion of the long key.

The main goal of our approach is to provide a guaranteed security property, namely minimum effort needed to analyze a long-key function, as well as to extract, store and reuse a long key. This has various applications, such as protection of long-lived symmetric keys in DRM systems and secure smartcard simulation in program code where key exchange is not possible. Usage scenarios extend beyond encryption; for example, a challenge-response authentication protocol can depend on a server's short key to verify a client's possession of a long key (e.g., via periodic requests for random blocks of that key).

Our method does not fully address all possible security issues with key protection. For example, an adversary can still copy a long key, as well as simply call a long-key function to perform encryption and decryption. However, such situations are beyond the scope of this paper. As in bounded-data models [5, 14], we assume that an adversary has restricted storage capabilities, or at least limited bandwidth for data retrieval. We analyze security mainly in this context.

Depending on usage scenario, obfuscation of the long key may be useful to complicate the task of extracting key bits and calling the long-key function. For

example, one may build a platform-specific long-key implementation bound to a particular application via additional mechanisms. This paper also presents some novel obfuscation techniques to increase the attacker's workload, but our basic method does not depend on obfuscation for its main security property.

Our approach focuses on symmetric block ciphers such as DES and AES, but is applicable to stream ciphers and public-key systems as well. More generally, the duality of long and short keys is of independent interest in various cryptographic scenarios, including protocols and authentication.

The following is a summary of our contributions in this paper:

- We define a new cryptographic construction, the *long-short-key primitive*, and propose its usage as an alternative to other key-protection methods in some scenarios.
- Using a cryptographic pseudorandom generator and a standard block cipher, we show that an LSK scheme can be implemented in practice with provable security (equivalent to breaking the generator and cipher).
- We present several applications, including white-boxing, DRM, software smartcards, and challenge-response authentication.
- We explain some LSK-specific obfuscation techniques and describe an AES-based implementation of the LSK primitive with reasonable performance penalty.

1.1 The Case for Long Keys

The long-short-key primitive can significantly improve security in various scenarios, as described later. Below we summarize several applications:

Block-Cipher Security. Since the short key may change on every block-cipher operation, our construction significantly improves security of encryption. For example, it becomes difficult to carry out the types of ciphertext-correlation attacks possible in ECB mode. In addition, the long key can improve security of white-box ciphers, which aim to protect keys by hard-coding them into programs.

Digital Rights Management. On modern computing systems, reverse engineers and hackers can typically inspect contents of memory and disk. Widely accessible tools, such as debuggers and system monitors, allow sophisticated users to access runtime state and files, including sensitive data like cryptographic keys and authentication credentials. Even when hardware enforces data protection, security exploits and side-channel attacks [29] may still render secret bits open to observation. Such transparency creates problems for various popular applications that need to protect cryptographic keys. For example, DRM systems decrypt content on PCs, enabling attackers to extract symmetric keys from memory. Since such keys are quite short (e.g., 128 or 256 bits), they can be easily shared across different devices and distributed via pirate channels. In general, symmetric-key encryption usually relies on well known, standardized ciphers that allow easy reuse of keys lifted from memory. A leaked key may even be typed in by a PC or device user. Using LSK, however, a long key can be megabytes or possibly gigabytes in size, preventing such attacks.

Software Smartcards. In many cases, it is desirable to implement software smartcards that perform the same functionality as their hardware counterparts. While hardware can make key extraction very difficult via physical tamper-resistance, this is not possible in open software. A smartcard key extracted from software enables hardware forgery of the smartcard. Using LSK, however, it is possible to have a software long key from which it is difficult to extract the short smartcard key. The adversary is forced to extract the code that carries out the encryption or other key-based operations. Moreover, the long key may not be practical to include or use on a smartcard, preventing easy hardware forgery.

Remote Timing Attacks. Side-channel attacks [29], such as remote timing attacks, are a common threat on the Internet. By simply measuring the time required by a remote node to encrypt selected messages, it is possible to derive the full secret key. If the remote node encrypts under LSK, enough randomness is added to encryption timing, complicating implementation of remote timing attacks.

Challenge-Response Authentication. In a standard challenge-response protocol, the challenger picks a set of arbitrary variables presented to the responder. The responder is authenticated only upon presenting the correct answer to the challenger. Many challenge-response protocols use cryptographic techniques, with the responder needing to encrypt a nonce under a specific key. When the key leaks, an adversary can always answer the challenge. With LSK, however, key leakage is less likely, depending on key length and any additional key-hiding measures.

2 Security Model

For most considerations in this paper, we assume an open platform such as a PC. The user has full access, and can use tools like debuggers and disassemblers to reverse engineer code and inspect memory. When we discuss remote attacks, a weakened model applies: The adversary is able only to trigger encryptions remotely and measure the time it takes to return results.

As a building block, we use a presumably secure cipher (e.g., AES). The adversary has access to the public code of the cipher, but not to the secret key. However, when attacking the cipher, the adversary is able to carry out side-channel attacks such as fault injection and timing analysis.

In addition, we assume that storage size is bounded, and an adversary can copy only a limited amount of data at a time (similar to [5, 14]). Storage size and bandwidth may vary, depending on the attack scenario – e.g., a virus transmitting data remotely versus a local attacker using a debugger to retrieve data.

3 The Long-Short-Key Primitive

The LSK primitive stipulates a short key k and a long key l. Both keys can be used to encrypt plaintext x such that ciphertext $c = E'_k(x) = E'_l(x)$. The

long key l can be derived from the short key k. However, given the long key l, efficient retrieval of the short key k is not possible. In reality, the long key can be hundreds of megabytes, whereas the short key may contain only 128 or 256 bits, as in AES.

This paper focuses on LSK-based encryption schemes built with a block cipher. The encryption and decryption operations can be implemented in various ways, each of which has its own advantages and disadvantages. In every scheme, the encryption function requires an initialization vector IV:

- *Sequential key-block encryption* uses a sequence of long-key blocks to encrypt plaintext x. The sequence is predefined, and depends only on the IV and the length of the plaintext.
- *Counter-based key-block encryption* uses a random sequence of long-key blocks as encryption keys for consecutive plaintext blocks, but without dependencies on ciphertext.
- *Selective key-block encryption* selects a random block of the long key and passes this to the cipher, retrieving the next random key block based on the resulting ciphertext.

3.1 Long-Key Construction

One secure implementation mechanism for LSK involves using a cryptographic generator to derive the long key, with the short key serving as a seed; we then treat blocks of the long key as separate symmetric keys. More specifically, we construct the N-bit long key l from a short key k by using a cryptographic pseudorandom number generator or stream cipher $R(k, N)$. R accepts k as a seed and generates N pseudorandom bits. For efficiency, R can be a random-access stream cipher, such as a block cipher in counter mode, which enables generation of an arbitrary long-key block without first computing any other long-key parts [20]. Encryption and decryption use blocks of the long key-stream $l = R(k, N)$ as individual cipher keys. Since these operations use the original standard cipher $E()$ as a building block, cipher security is preserved.

After Alice computes the long key l from the short key k, she sets up encryption by handing a portion of l to Bob. During the encryption, l serves as the 'code book' for Alice and Bob. When Alice encrypts a plaintext message m, she computes $c = E_{k'}(m)$, where k' is the j-th sub-key of the long key (for some index j). She then sends (c, j) to Bob. Bob looks up the j-th sub-key of the long key l and decrypts c. Alice is much more powerful than Bob in this setting, since she can derive any long-key portion from her short key k, whereas Bob is tied to the long-key fragment he received. He is able to encrypt as well, but only within his set of sub-key blocks. When Alice decides to black-out Bob, she simply switches to a different range of sub-keys in l. The following paragraphs will show how important this seemingly simple technique can be in implementations.

Operation Modes For LSK. In our construction so far, Alice is solely responsible for security, since she can pick which parts of the long key l to use

for encryption. In the worst case, she could degrade the whole long-key security to the cipher's security – i.e., when she picks only the first long-key block every time she encrypts a plaintext message. To eliminate this risk, we present three different mechanisms that enhance the security of the long-key encryption scheme.

Sequential key-block encryption. One scheme is to use consecutive long-key blocks as keys. In the first ciphertext message, Alice sends to Bob the pair (c, i), where c is the ciphertext and i indicates the i-th long-key block used as the encryption key; in all consecutive messages, she sends only c. Bob automatically uses blocks $i + 1, i + 2, ...$ for decryption. Alternately, Alice can simply encrypt each plaintext with long-key blocks $0, 1, 2, ...$

Counter-based key-block encryption. In the counter-based scheme, Alice encrypts sequences of numbers $i, i + 1, i + 2, ...$, using the cipher E_{IV} under some initialization vector IV. When she sets up the communication, she sends IV to Bob. When she encrypts the first data block, she uses key-block with index $E_k(i) \mod N$ from the long key; for the next data block, she uses $E_k(i + 1) \mod N$, and so on. Bob also computes the same sequence of block indices and decrypts the messages accordingly. For a simpler scheme, i can begin at 0, and the first long-key block can be used as the initial value IV.

As compared to sequential key-block encryption, the advantage of the counter-based method is less locality during encryption. To decrypt a ciphertext, an adversary needs to obtain arbitrary parts of the long key l. In addition, this method has a random-access property for decryption; i.e., any block can be decrypted independently of other blocks.

Selective key-block encryption. Similar to CBC mode in block ciphers, the next key block to use from the long key can also be decided based on the most recent ciphertext and key. Alice starts encryption using some block k_j of the long key (e.g., the first block or a block determined by an initial value). She then computes the first ciphertext block c_0 based on key block k_j and first plaintext block p_0. The next key-block index is then $c_0 \mod N$. In general, for $i > 0$, the $i-$th key-block index is $c_{i-1} \mod N$, which encrypts the $i-$th plaintext block p_i.

Selective key-block encryption has a significant advantage over the counter-based method, since the former uses a different key block combination for different plaintexts. Therefore, this mode randomizes the key block accesses best and requires an adversary to obtain all long-key bits in general.

Depending on the random distribution used in an implementation, key blocks in both counter-based and selective key-block encryption are potentially scattered across the entire long key. In addition, different random subsets of the long key may be used for operations with each specific short key. With the selective key-block encryption scheme, the long-key subsets also depend on the input text. This provides better protection against an adversary who has partial knowledge of the long key. Also, this defends against side-channel attacks, such as cache-timing analysis. However, these methods could be slow because the random memory accesses may cause many cache misses.

Sequential key-block encryption simply uses consecutive long-key blocks to process consecutive text blocks, wrapping around whenever text size exceeds long-key size. This is a simple strategy that does not offer the capability to use random subsets of the long key. The apparent trade-off that we need to analyze more carefully is the security through randomization vs. the performance penalty due to random memory accesses.

4 Applications

4.1 Block-Cipher Security

A typical problem with block ciphers is susceptibility to correlation attacks in ECB mode. If used in CBC mode, ciphers must be synchronized. This often causes problems in unreliable communication channels, such as UDP-based transmissions. LSK improves upon ECB without this drawback of CBC, because an adversary without knowledge of the long key cannot correlate identical ciphertext blocks, even if Alice uses a simple operational mode and sends the long-key-block index on every transmission.

In addition, LSK improves the security of white-box block-cipher implementations. Often it is possible to carry out fault-injection attacks (e.g., [22]) to extract the key from the white-box cipher. With support from the long key, this attack becomes difficult, because the cipher key may change on every block encryption.

4.2 Digital Rights Management

A main goal in DRM is to protect a secret key k of a common publicly known cipher $E_k(x)$ on an open platform (such as Microsoft Windows). After k leaks, an attacker can decrypt any ciphertext c using $x = D_k(c)$. To determine k, an attacker can use disassemblers, debuggers and other reverse-engineering tools that help investigate memory during program execution. When k is located within a single stack frame or memory location, identifying and extracting k may pose few difficulties. Similarly, side-channel attacks like cache-timing analysis and fault injection can isolate k, which may comprise only a few bytes.

With a long key, an attacker's tasks become more difficult. First, the key k does not fit into a single stack frame or memory location. Secondly, side-channel attacks are significantly more difficult when an attacker has more key bits to retrieve. Also, common white-box techniques are more effective when applied to a cipher that uses a large amount of key material.

To ensure security, it is also important that the construction of the long key be transparent and derivable from common cryptographic primitives. Our present work does not generate a new cipher, but uses existing symmetric encryption to create a new cipher construction that has better security properties for implementation on open platforms.

4.3 Software Smartcards

Smartcards are a ubiquitous technique for authenticating users. It would be desirable to have a software pendant for hardware-based smartcards on PC platforms – not only to avoid proliferation of hardware readers, but also to streamline smartcard updates. One problem, however, is to protect smartcard-specific secrets in software. When the user types in a PIN code, smartcard software typically uses this to compute a ciphertext validated against a stored ciphertext. When smartcard-specific secrets leak, an adversary can forge PIN codes and compromise the system.

On an open platform, LSK hides the short smartcard secret inside the long key. Smartcard-simulator software runs in selective key-block mode and contains only the long key. When the user types a PIN, the software encrypts the PIN via the LSK construction, and compares this to a stored ciphertext. There is no need to secure a separate secret on the open platform, while hardware tamper-resistance protects the short key in the smartcard.

4.4 Remote Timing Attacks

In remote timing attacks (e.g., [4]), an attacker exploits timing properties of an encryption algorithm (e.g., RSA-CRT) to extract secret keys. Typically, these attacks can be countered by blinding the input data. This additional randomness, which is unknown to the adversary, forces the encryption algorithm to process a different plaintext message. In many encryption schemes, such as RSA, blinding can be inverted after encryption by exploiting algorithmic properties. For the adversary, the key becomes hard to extract, while the performance penalty is small. However, blinding does not work on all encryption schemes, because it may be hard to revert the blinding operation without handing out the secret randomness to every receiver. For remote attacks to be successful, the adversary must measure the timing of a certain number of encryptions under the same key. An LSK-based scheme fixes this problem, because LSK uses a different cipher key for every encryption. Given that the long key can comprise a few hundred megabytes, this will stretch the duration of the attack significantly, if not render the attack practically impossible.

4.5 Challenge-Response Authentication

In cryptographic challenge-response authentication protocols, the key must not leak. If it does, an adversary can easily answer any arbitrary cryptographic challenge from the verifying party. Using an approach based on LSK, the challenger can simply hand the long key to the responder and later request parts of the long key, combining this with encryption of nonces to provide better security. For an adversary, it is virtually impossible to answer challenges without knowing the entire long key, which is difficult to extract because of its size and any additional protection measures.

5 Implementation

An important part of this work is implementation of LSK-based schemes in a secure manner. In some contexts, a problem stems from an adversary's ability to copy the long key onto media like DVDs or over the network, transferring the key to another machine. Therefore, the long key should also be hidden in software code. Extracting portions of the key may be feasible, but obtaining the entire key should be difficult.

In practice, choosing an appropriate length for the long key will depend on the application. If the goal is mainly to prevent humans from memorizing or writing down keys, even several hundred bytes or a few kilobytes of long key may suffice. If we wish to make electronic key distribution unwieldy, several hundred megabytes or more may be the minimum if the Internet or CD/DVD media are involved. Advances in networking and storage capabilities may affect the required long-key size as well. In general, the protection designer should carefully consider aspects such as hardware and software capabilities, as well as security requirements in particular scenarios.

We have implemented a construction we call *ExAES*, which is an LSK-based scheme with AES as a cipher. In ExAES, we expand the secret AES key into a long key using a pseudorandom number generator. We then replicate the AES code, embedding blocks of the long key in each replica. Finally, we merge the different AES cipher implementations to hinder attacks that attempt to extract portions of the key.

5.1 Iterated Obfuscation

As a novel technique, we use *iterated obfuscation* to establish a general framework for the implementation of our obfuscation techniques. This methodology involves the iterated application and recombination of various obfuscating transformations (or *primitives*) over code, with the output of each successive transformation serving as input to the next. Via this strategy, even simple and easy to implement primitives can be cascaded to yield effective obfuscation.

As an example, the technique of oblivious hashing (OH) [7] can serve as an obfuscation primitive. A single OH transformation injects code to hash a program's runtime state (i.e., data and control flow), thus ascertaining execution integrity of the original code. Applying OH again to the transformed program protects both the original program and the first OH round. In general, each new OH round verifies the integrity of both the original program and all previous OH rounds.

To increase security further, arbitrary other primitives can be combined and iterated with the OH rounds. For ease of design and implementation, such primitives can be quite simple – e.g., conversion of variable references to pointer references, and even source-to-source translation among different code dialects or languages. Via iteration, the interaction of simple primitives can lead to emergent code behavior and achieve the effect of far more complex obfuscation operators.

This is related to both iterative complex systems (e.g., cellular automata) and iterated rounds in cryptographic constructions.

5.2 Generic Block-Cipher Obfuscation

In this section, we define an obfuscation operation $O'(\cdot)$ intended to compose two different cipher keys. This operation transforms a concatenated block cipher $E_{k_0} \cdot E_{k_1}$ into an obfuscated sequence $O'(E_{k_0} \cdot E_{k_1})$ in which it is difficult to isolate E_{k_0} and E_{k_1}. Both ciphers are obfuscated individually on a per-round basis, but have the same security as the original. We assume that the per-cipher obfuscation is weak, so that key bits can be extracted in constant time from every round.

Interleaving Rounds. To combine two cipher instances, we interleave their rounds. As in shuffling two decks of cards, we pick cipher rounds randomly from each cipher, with the long key serving as a source of randomness. Formally, given $E_{k_0} = R^1_{k_0} \cdot R^2_{k_0} \cdot \ldots \cdot R^n_{k_0} = \cup_{i=1}^{n} R^i_{k_0}$ and $E_{k_1} = R^1_{k_1} \cdot R^2_{k_1} \cdot \ldots \cdot R^n_{k_1} = \cup_{i=1}^{n} R^i_{k_1}$, we construct $E_{k_1} \odot E_{k_2} = \cup_{i=1}^{2n} R^{\rho(i)}_{k_{\pi(i)}}$, where $\rho(i)$ is a permutation and $\pi(i) \in \{0, 1\}$ randomly selects k_0 or k_1.

Weakly obfuscated, the rounds themselves remain in the same order as in the original ciphers (i.e., $\rho(i) < \rho(i + k)$ when $\pi(i) = \pi(i + k)$). The computed ciphertexts may be different than in the original cipher, depending on the interleaving. However, security remains unchanged, since the same round functions are used. In the appendix, we describe methods to merge and obfuscate the rounds themselves.

5.3 Performance

A main trade-off in our white-box is the number of rounds for the long key. This number is significantly larger than the number of rounds in a normal cipher. To gauge the effect, we measured the number of cycles consumed for every encrypted byte with different numbers of rounds and different obfuscation levels.

Figure 1 shows the results. The additional performance overhead becomes superlinear relatively early when the obfuscation level becomes larger. However, these additional operations are necessary to disguise round boundaries.

We also compare our white-box (WB) to AES and RSA (from Microsoft's Crypto API) in terms of performance. We chose a key size of 512 bits in all cases.

Figure 2 shows the performance comparison with AES and RSA in cycles per encrypted byte. The compact white-box implementation with the short key and without any additional obfuscation is only slightly slower than AES. However, even with obfuscation turned on, our white-box is still faster than RSA. (Involving symmetric and asymmetric encryption, this comparison is only for illustration.)

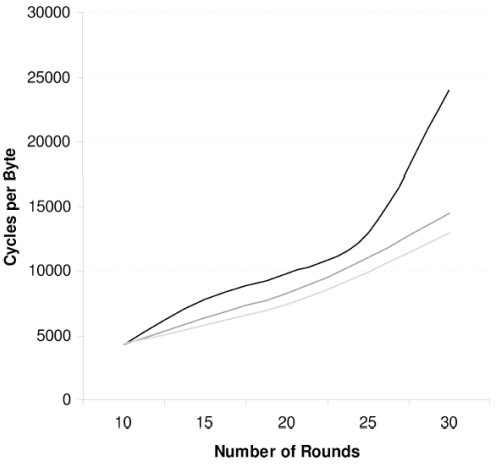


Fig. 1. White-box performance when increasing the total number of rounds and the obfuscation level

Algorithm	Cycles per Byte
AES	200
RSA	2000
WB Compact	202
WB	1500

Fig. 2. Performance comparison between WB, AES, and RSA

5.4 Security Assessment

Our obfuscation algorithm does not depend on precise types and quantities of primitives. Due to space constraints, we have omitted such specifics in our description. However, when analyzing the security of our algorithm, we assume the adversary has complete knowledge of all obfuscation primitives used; we do not assume any security-by-obscurity in choice of the primitives. The adversary does not know the random sequence which determines the application of the primitives, as such knowledge would be tantamount to knowing the private key.

The powerful side-channel and fault-injection attacks [22] mentioned in §6 show that round boundaries in an obfuscated cipher must be difficult to detect. Below we provide an example of an attack on ExAES, given knowledge of round boundaries. The problem is that each round is composed of several distinct types of operations. For example, the AES and ExAES ciphers used here alternate variable XOR operations with permutation and field operations implemented via table lookups. The operations must be applied sequentially; thus, if the distribution of hardware instructions reveals this alternation, the round boundaries could be discovered and the cipher broken.

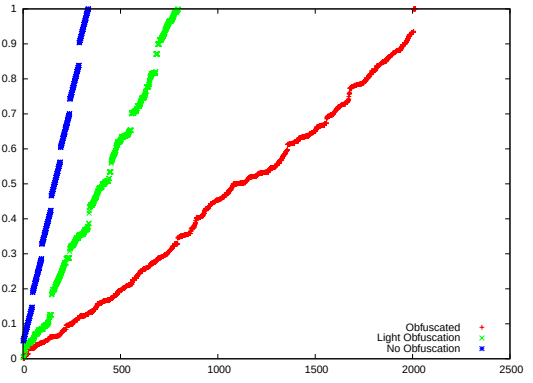


Fig. 3. Effect of obfuscation on round boundaries. Note the visible segmentation in the leftmost curve, which clearly indicates round boundaries. As more obfuscation is applied, the segmentation becomes less apparent, thus hiding the boundaries.

As a concrete example, consider the Mix operation in AES. The multiplication in the AES field is implemented by a lookup table indexed by data bytes whose entries are XORed together to produce the result. To implement this in hardware, XOR instructions with both operands variable are used, in contrast to XOR instructions with static constants that are used to mix in key data before applying the S-Box. These variable XOR instructions may be discovered statically and their cumulative total plotted against the number of instructions to produce a cumulative distribution curve.

This has been done in Figure 3 with various levels of obfuscation applied. The figure plots the cumulative distribution of variable XOR instructions by total number of instructions with various levels of obfuscation applied. Plots with higher levels of obfuscation are flatter, as the total number of instructions is higher. The key point is to see how the round structure is clearly visible in the non-obfuscated version (leftmost curve), and that structure disappears as the obfuscation level is increased (middle and rightmost curves).

Attack Against a Single Round. To illustrate the importance of hiding round boundaries, we show how a single ExAES round may be attacked even if no information is known about the byte permutation or the S-Boxes. We note that an attack against a round of AES is even simpler, since the key is involved only in the XOR performed before the S-Box.

The crux is that the Mix operation mixes only four bytes at a time. Hence, by comparing the action of the round on inputs that vary at single bytes, the permutation can be discovered up to the order within each column. After the Mix operation, unknown dummy operations will XOR with each column to produce the output, 32 bits in total. The Mix operation itself is known. By guessing the 32 bits of dummy XOR after the Mix operation, then guessing which of the 24 possible permutations were used on the input identified as forming the column, we arrive at a guess of the state of those bytes immediately after the

S-Box operation. Varying each input byte to reconstruct the S-Box and confirm the guess completes the attack against that column. Repeating this for the remaining columns reveals the rest of the round key.

This attack takes approximately 2^{40} work, which is very feasible on a modern processor. The attack can recover the several thousand bytes of key that are used in the case of independent random S-Boxes, in addition to the permutation and final layer of dummy operations. Further improvements to this attack only underscore the need for an obfuscator to hide fully the round boundaries of the cipher.

6 Related Work

Software obfuscation is a well known problem that has never seen a comprehensive solution. [10] is a systematic taxonomy of obfuscation techniques in the context of Java bytecode. Many of these apply to our problem, and are incorporated into our system. In particular, we use data obfuscation by aggregation and ordering, as well as control-flow obfuscation; see [10] for definitions. Some of the obfuscation techniques in [10] are used with complex control flow, data structures or procedure networks, and do not apply to the less general problem of white-boxing ciphers that we consider here. The extension of [11] describes *opaque predicates*, which are used for obfuscated control flow. We use a limited form of these in our current version and will incorporate more complex constructions in future versions.

Other approaches to secure computation include secure circuit evaluation [16, 18, 23, 32, 33]. In this model, several parties hold separate inputs to a common function, and want to compute the function without revealing any information about their inputs other than what is contained in the function output. The multiparty-communication nature of this problem produces solutions inappropriate for our white-box model. In particular, protocols for secure circuit evaluation rely on random choices that must change with each run of the protocol. In a white-box context, an adversary is able to reset any state, including random counters within the attacked program.

The specific obfuscation problem of creating a white-box cipher (DES) was addressed in [8, 9]. This was later attacked using fault-injection techniques [22, 27]. The idea of [8, 9] first hard-codes the DES round keys into the round computation. The DES encoding is then seen as a series of functions – some affine, while others entirely non-linear and implemented with lookup tables (the S-Boxes). Random invertible functions are chosen, paired with their inverses, and inserted into the series of DES-related functions. The encoding operation is then reassociated to combine the random functions with the DES steps, expanding the affine functions into lookup tables as appropriate. The goal is to hide the key operations with the random invertible functions.

The construction in [9] was attacked in [22] by injecting faults before the last round to probe for key bits. The essential point is that once the boundaries can be discovered, the round function can be taken apart in spite of the obfuscating

random functions applied to it. Their construction differs from ours in being a very specific algorithm—table composition and re-association—applied at a high-level to the cipher. As described below, our method is applied across all levels of white-box program generation, from the high-level representation of the operation sequence to the the low-level access of individual bytes.

Much work has been done in areas of distributed execution and security of mobile code [17, 24]. This often focuses on running untrusted programs on trusted machines, which is opposite to our problem of running a trusted program on an untrusted machine. A related area is software tamper-resistance [26, 31], where the problem is to guarantee that distributed code is executed unmodified on untrusted machines. In white-box applications, it is immaterial whether or not the white-box code is modified, provided that the output is correct. The only aim is to compute while keeping a secret hidden. Many tamper-resistance solutions implicitly rely on keeping obfuscated secrets, such as locations of checksumming that occurs in code. Software white-boxing addresses this secret-hiding explicitly and does not attempt wider security goals.

Recently, side-channel attacks have been shown to be effective against even software implementations of ciphers [3, 4, 29]. In cache-timing attacks, an adversary detects the number of cache misses during encryption operations and reconstructs the key bits based on this information. In particular, ciphers like AES that access memory locations based on the values of the key are prone to this attack. Our white-box protects against these attacks because of the long key and the additional amount of randomness in the cipher algorithm.

The extended limitations on obfuscation presented in [19] suggest that obfuscation is more difficult in the presence of auxiliary dependent information. An example of this in the context of our system would be an adversary who has access to two white-box versions of the cipher with a particular key. While we believe our system should be practically secure against this attack, a detailed analysis remains to be performed.

Finally, there is a large body of work on cryptographic algorithms for bounded storage [5, 12, 13, 14, 15]. Typically, the security framework is similar to our LSK platform, with the exception of remote attacks in bounded-data models (e.g., a virus sending secret data over a network with limited bandwidth).

7 Conclusion

We proposed a new construction, the long-short-key primitive, that hides keys and improves implementation security of block ciphers. Our method creates a compact, efficient implementation of encryption, along with white-boxed long-key decryption code that may be arbitrarily large, as controlled by the user. Unless a cryptographic pseudorandom generator is broken, any hack of the white-box must have a provably large size to recover the full long key. In essence, we provide a private encryption function that uses a short key, along with a corresponding public decryption function that requires a long key. In addition, different encrypted content requires different sections of the long key for decryption,

so that breaking the obfuscation enough to decrypt one ciphertext does not necessarily allow decryption of others. The key sections required for decryption may be revealed only as decryption proceeds.

Our method has a number of practical applications, including DRM key management and software-based smartcard simulators designed to hide a short key present on tamper-resistant hardware smartcards. Our main provable security metric, namely the minimum size of any white-box hack, is of independent theoretical interest as well. While our system provides a symmetric cipher, the novel asymmetric paradigm of short "private" keys and arbitrarily long "public" keys may find other applications that currently rely on true asymmetric cryptography.

References

1. Aucsmith, D.: Tamper resistant software: An implementation. In: Anderson, R. (ed.) IH 1996. LNCS, vol. 1174, pp. 317–333. Springer, Heidelberg (1996)
2. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001)
3. Bernstein, D.J.: Cache-timing attacks on AES, `http://cr.yp.to/papers.html#cachetiming`
4. Boneh, D., Brumley, D.: Remote timing attacks are practical. In: USENIX Security Symposium (2003)
5. Cash, D., Ding, Y.Z., Dodis, Y., Lee, W., Lipton, R., Walfish, S.: Intrusion-resilient key exchange in the bounded retrieval model. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 479–498. Springer, Heidelberg (2007)
6. Chang, H., Atallah, M.J.: Protecting software code by guards. In: Digital Rights Management Workshop, pp. 160–175 (2001)
7. Chen, Y., Venkatesan, R., Cary, M., Pang, R., Sinha, S., Jakubowski, M.H.: Oblivious hashing: Silent verification of code execution. In: Proceedings of the 2002 Information Hiding Workshop (October 2002)
8. Chow, S., Eisen, P., Johnson, H., van Oorschot, P.: White-box cryptography and an AES implementation. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595. Springer, Heidelberg (2003)
9. Chow, S., Eisen, P., Johnson, H., van Oorschot, P.: A white-box DES implementation for DRM applications. In: Feigenbaum, J. (ed.) DRM 2002. LNCS, vol. 2696, pp. 1–15. Springer, Heidelberg (2003)
10. Collberg, C., Thomborson, C., Low, D.: A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Science, The University of Auckland, New Zealand (July 1997)
11. Collberg, C., Thomborson, C., Low, D.: Manufacturing cheap, resilient, and stealthy opaque constructs. In: Principles of Programming Languages, POPL 1998, pp. 184–196 (1998)
12. Di Crescenzo, G., Lipton, R.J., Walfish, S.: Perfectly secure password protocols in the bounded retrieval model. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 225–244. Springer, Heidelberg (2006)
13. Dagon, D., Lee, W., Lipton, R.: Protecting secret data from insider attacks. In: Proceedings of Financial Cryptography (2005)

14. Dziembowski, S.: Intrusion-resilience via the bounded-storage model. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 207–224. Springer, Heidelberg (2006)
15. Dziembowski, S.: On forward-secure storage. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 251–270. Springer, Heidelberg (2006)
16. Feige, U., Killian, J., Naor, M.: A minimal model for secure computation (extended abstract). In: STOC 1994: Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing, pp. 554–563. ACM Press, New York (1994)
17. Gao, D., Reiter, M.K., Song, D.X.: On gray-box program tracking for anomaly detection. In: USENIX Security Symposium, pp. 103–118 (2004)
18. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: STOC 1987: Proceedings of the Nineteenth Annual ACM Conference on Theory of Computing, pp. 218–229 (1987)
19. Goldwasser, S., Kalai, Y.T.: On the impossibility of obfuscation with auxiliary input. In: FOCS 2005: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science (2005)
20. Dj Golić., J.: Stream cipher encryption of random access files. Information Processing Letters 69(3), 145–148 (1999)
21. Horne, B., Matheson, L.R., Sheehan, C., Tarjan, R.E.: Dynamic self-checking techniques for improved tamper resistance. In: Digital Rights Management Workshop, pp. 141–159 (2001)
22. Jacob, M., Boneh, D., Felten, E.: Attacking an obfuscated cipher by injecting faults. In: ACM CCS-9 Workshop (DRM) (2002)
23. Kilian, J.: A general completeness theorem for two party games. In: STOC 1991: Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing, pp. 553–560 (1991)
24. Kiriansky, V., Bruening, D., Amarasinghe, S.P.: Secure execution via program shepherding. In: USENIX Security Symposium, pp. 191–206 (2002)
25. Kocher, P.: Timing attacks on implementation of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109. Springer, Heidelberg (1996)
26. Lie, D., Thekkath, C.A., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J.C., Horowitz, M.: Architectural support for copy and tamper resistant software. In: ASPLOS, pp. 168–177 (2000)
27. Link, H., Neumann, W.: Clarifying obfuscation: Improving the security of white-box encoding. Cryptology ePrint Archive Report 2004/025 (2004)
28. Lynn, B., Prabhakaran, M., Sahai, A.: Positive results and techniques for obfuscation. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 20–39. Springer, Heidelberg (2004)
29. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: The case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)
30. Shamir, A., van Someren, N.: Playing hide and seek with stored keys. In: Financial Cryptography (1998)
31. van Oorschot, P.C., Somayaji, A., Wurster, G.: Hardware-assisted circumvention of self-hashing software tamper resistance. IEEE Transactions on Dependable and Secure Computing 2(2), 82–92 (2005)
32. Yao, A.C.: Protocols for secure computations. In: FOCS 1982: Proceedings of the Twenty-third IEEE Symposium on Foundations of Computer Science, pp. 160–164 (1982)

33. Yao, A.C.: How to generate and exchange secrets. In: FOCS 1986: Proceedings of the Twenty-seventh IEEE Symposium on Foundations of Computer Science, pp. 162–167 (1986)

A Merging Rounds

When block-cipher rounds are simply shuffled, a brute-force method could successfully rearrange rounds in the original ciphers. In addition, various other key-extraction attacks are possible if round boundaries are easily discernible. In this section, we present a list of operations that preserve semantics but enhance security by merging operations of different rounds.

Our approach obfuscates an operation sequence by applying a series of obfuscation transformations. Most of these are peephole; i.e., they are applied locally to small groups of instructions, usually pairs. The transformations are organized below into several classes. When variously composed and iterated, the combined transformations will increase the complexity of the obfuscation more than the sum of each individual transformation. As attacks against our obfuscation develop, new transformations will be become apparent and can easily be integrated into our system.

We combine these transformations randomly to obfuscate code. Some tuning is required to maximize the avalanche effect of the obfuscations while minimizing the increase of program size. For example, dummy permutations, which translate into a large number of instructions, are inserted sparsely into rounds between commuting and combining transformations, which have a smaller impact on code size. This heuristically maximizes the amount of diffusion from each permutation while minimizing the total number of permutations performed.

Space constraints do not permit us to describe all transformations in detail. We will elaborate on several illustrative techniques and leave the remainder general. An advantage of our obfuscation method is that the exact transformations used are not critical; we believe most reasonable simple ones can be used iteratively to produce effective obfuscation.

Our machine model assumes a simple processor operating over a random-access array. We view a program is a sequence of operations over a logical array of bits. The operations are only those needed to implement the cryptographic functions of interest, such as bitwise Boolean operations, permutation, copying, addition, multiplication, and some simple control flow. We do not expect the operations to be universal in the sense of ability to describe any computation. This operation set was chosen as the smallest sufficient to compute AES efficiently, as well as introduce execution that cannot be analyzed in a purely static way. In particular, while control flow is not strictly necessary to produce an AES-like program, this is useful in obfuscation when combined with opaque predicates [11].

Simple Transformations. Our first class of obfuscations is simple transformations that do not cause internal changes to any operation. For example, a random permutation of a subset of the working data can be applied, followed by

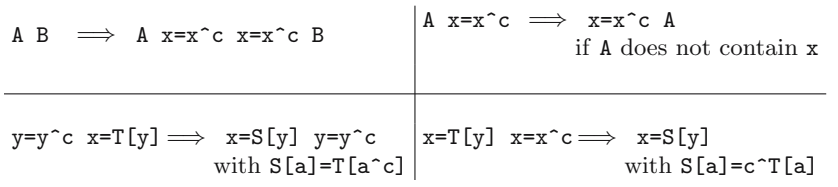


Fig. 4. Some simple obfuscation transformations. In the above, ^ denotes XOR, lower-case letters are working data locations, upper-case letters signify generic operations, and braces are used to denote table lookup.

a permutation that moves the working data back to its original location. Similarly, two XOR operations with the same random constant can be performed on a subset of the working data. Both these operations may seem nonsensical, but become quite powerful when combined with other operations below. We also note that if the input and output sets of an adjacent pair of operations do not intersect, they may be commuted. That is, the sequence of two operations `[ A B ]` may be replaced by the sequence `[ B A ]`.

Morphing Transformations. Techniques in this class of obfuscations commute two operations in ways that cause the operations, but not their number, to change. For example, an operation and a permutation may be commuted by modifying the operation to apply the permutation to its inputs and outputs. The permutation or its inverse is applied, depending on whether the operation precedes the permutation or vice-versa. This makes the simple transformation above useful. Several more complicated ones are shown in Figure 4.

Lookup-Table Obfuscations. These transformations apply to lookup tables. The above transformations can take a single lookup table from an unobfuscated version of a program and create several obfuscated versions, each permuted and changed. While this increases the obfuscation, it also leads to a large amount of static data. Furthermore, because most of the tables are byte-oriented, they are vulnerable to exhaustive analysis of all 2^{16} possible modifications.

We use two techniques to address these problems – *gradual lookup-table correction* and *dummy lookup tables*. In the former, the lookup tables are stored with errors that are corrected and changed as the program executes. The latter allows for dummy operation sequences, described below, to use actual lookup tables for computation interleaved with the error correction. This obscures which values in the lookup tables are correct and which are erroneous.

B Low-Level Obfuscation Techniques

Our low-level obfuscations hide patterns of data access by operations independently from the semantics of the operations. After these obfuscations, the code generated for a byte-array read or write is more complicated than a simple lookup into the array.

An example of this technique is byte-array indirection. Logical bytes are stored discontinuously throughout the lookup tables, which are changed over the course of the program's execution. A set of indirection indices is used to look up the location of each byte. These indices are spread discontinuously throughout the lookup-table data. As their locations are known statically at generation time, moving indices will effect a permutation of data elements indirectly.

Author Index