Lecture Notes in Computer Science 1977
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Bimal Roy Eiji Okamoto (Eds.)

Progress in Cryptology – INDOCRYPT 2000

First International Conference in Cryptology in India
Calcutta, India, December 10-13, 2000
Proceedings

Springer

Volume Editors

Bimal Roy
Indian Statistical Institute
Calcutta, India
E-mail: bimal@isical.ac.in

Eiji Okamoto
University of Wisconsin
Department of Computer Science
Milwaukee, Wisconsin, USA
E-mail: okamoto@cs.uwm.edu

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Progress in cryptology : proceedings / INDOCRYPT 2000, First
International Conference in Cryptology in India, Calcutta, India,
December 10 - 13, 2000. Bimal Roy ; Eiji Okamoto (ed.). - Berlin ;
Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ;
Paris ; Singapore ; Tokyo : Springer, 2000
(Lecture notes in computer science ; Vol. 1977)
ISBN 3-540-41452-5

CR Subject Classification (1998): E.3, G.2.1, D.4.6, K.6.5, F.2.1-2, C.2, J.1

ISSN 0302-9743
ISBN 3-540-41452-5 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
Printed on acid-free paper SPIN 10781218 06/3142 5 4 3 2 1 0

Preface

The field of Cryptology witnessed a revolution in the late seventies. Since then it has been expanded into an important and exciting area of research. Over the last two decades, India neither participated actively nor did it contribute significantly towards the development in this field. However, recently a number of active research groups engaged in important research and developmental work have crystalized in different parts of India. As a result, their interaction with the international crypto community has become necessary. With this backdrop, it was proposed that a conference on cryptology – INDOCRYPT, be organized for the first time in India. The Indian Statistical Institute was instrumental in hosting this conference. INDOCRYPT has generated a large amount of enthusiasm amongst the Indians as well as the International crypto communities. An INDOCRYPT steering committee has been formed and the committee has plans to make INDOCRYPT an annual event.

For INDOCRYPT 2000, the program committee considered a total of 54 papers and out of these 25 were selected for presentation. The conference program also included two invited lectures by Prof. Adi Shamir and Prof. Eli Biham.

These proceedings include the revised versions of the 25 papers accepted by the program committee. These papers were selected from all the submissions based on originality, quality and relevance to the field of Cryptology. Revisions were not checked and the authors bear the full responsibility for the contents of the papers in these proceedings.

The selection of the papers was a very difficult and challenging task. I wish to thank all the Program Committee members who did an excellent job in reviewing the papers and providing valuable feedback to the authors. Each submission was reviewed by at least three (only a few by two) reviewers. The program committee was assisted by many colleagues who reviewed submissions in their areas of expertise. The list of external reviewers has been provided separately. My thanks go to them all.

My sincere thanks goes to Springer-Verlag, in particular to Mr. Alfred Hofmann, for the inclusion of the seminar proceedings in their prestigious series Lecture Notes in Computer Science. I am also indebted to Prof. Jacques Stern, Prof. Jennifer Seberry, and Prof. Cunsheng Ding for giving their valuable advise and suggestions towards making the publication of the proceedings of INDOCRYPT 2000 possible.

I gratefully acknowledge financial support from diffferent organizations towards making INDOCRYPT 2000 a success. The contributors were AgniRoth (California, USA), Tata Conusltancy Service (Calcutta, India), CMC Limited (New Delhi, India), Cognizant Technology Solutions (Calcutta, India), Gemplus (Bangalore, India), Ministry of Information Technology (Govt. of India), and IDRBT (Hyderabad, India). I once again thank them all.

In organizing the scientific program and putting together these proceedings I have been assisted by many people. In particular I would like to thank Subhamoy Maitra, Sarbani Palit, Arindom De, Kishan Chand Gupta, and Sandeepan Chowdhury.

Finally I wish to thank all the authors who submitted papers, making this conference possible, and the authors of successful papers for updating their papers in a timely fashion, making the production of these proceedings possible.

December 2000 Bimal Roy

Organizing Committee

Aditya Bagchi	Indian Statistical Institute, India
V. P. Gulati	IDRBT, India
Rajeev L. Karandikar	Indian Statistical Institute, India
Subhamoy Maitra	Indian Statistical Institute, India
Mandar Mitra	Indian Statistical Institute, India
Sarbani Palit	Indian Statistical Institute, India
Bimal Roy	Indian Statistical Institute, India
M. Vidyasagar	Tata Consultancy Services, India
K. S. Vijayan	Indian Statistical Institute, India

List of External Reviewers

Aditya Bagchi	Indian Statistical Institute, India
S S Bedi	SAG,India
A K Bhateja	SAG, India
Carlo Blundo	Universita di Salerno, Italy.
Johan Borst	Katholieke Universiteit Leuven, Belgium
Antoon Bosselaers	Katholieke Universiteit Leuven, Belgium
Dr Chris Charnes	University of Melbourne, Australia
Suresh Chari	IBM, T. J. Watson Lab, USA
Patrik Ekdahl	Lund University,Lund, Sweden
Shai Halevi	IBM, T. J. Watson Lab, USA
Fredrik Jnsson	Lund University,Lund, Sweden
Mike Just	Entrust Technologies, Canada
Meena Kumari	SAG, India
Subhamoy Maitra	Indian Statistical Institute, India
Nasir D. Memon	Polytechnic University, New York, USA.
Serge Mister	Entrust Technologies, Canada
Mandar Mitra	Indian Statistical Institute, India
Anish Ch. Mukherjee	Indian Statistical Institute, India
Pinakpani Pal	Indian Statistical Institute, India
Sarbani Palit	Indian Statistical Institute, India
Matthew Parker	University of Bergen, Norway
Enes Pasalic	Lund University,Lund, Sweden
Rajesh Pillai	SAG, India
David Pointcheval	ENS, France
Havard Raddum	University of Bergen, Norway
Pankaj Rohatgi	IBM, T. J. Watson Lab, USA
Reihaneh Safavi-Naini	University of Wollongong, Australia
Yuriy Tarannikov	Moscow State University, Russia
Serge Vaudenay	EPFL, France
Frederik Vercauteren	Katholieke Universiteit Leuven, Belgium
Robert Zuccherato	Entrust Technologies, Canada

Table of Contents

Stream Ciphers and Boolean Functions

The Correlation of a Boolean Function with Its Variables 1
Dingyi Pei and Wenliang Qin

On Choice of Connection-Polynomials for LFSR-Based Stream Ciphers . . . 9
Jambunathan K

On Resilient Boolean Functions with Maximal Possible Nonlinearity 19
Yuriy V. Tarannikov

Cryptanalysis I : Stream Ciphers

Decimation Attack of Stream Ciphers 31
Eric Filiol

Cryptanalysis of the A5/1 GSM Stream Cipher 43
Eli Biham and Orr Dunkelman

Cryptanalysis II : Block Ciphers

On Bias Estimation in Linear Cryptanalysis 52
Ali Aydın Selçuk

On the Incomparability of Entropy and Marginal Guesswork in Brute-Force Attacks 67
John O. Pliam

Improved Impossible Differentials on Twofish 80
Eli Biham and Vladimir Furman

Electronic Cash & Multiparty Computation

An Online, Transferable E-Cash Payment System 93
R. Sai Anand and C.E. Veni Madhavan

Anonymity Control in Multi-bank E-Cash System 104
Ik Rae Jeong and Dong Hoon Lee

Efficient Asynchronous Secure Multiparty Distributed Computation 117
K. Srinathan and C. Pandu Rangan

Tolerating Generalized Mobile Adversaries in Secure Multiparty Computation 130
K. Srinathan and C. Pandu Rangan

Digital Signatures

Codes Identifying Bad Signatures in Batches 143
Jarosław Pastuszak, Josef Pieprzyk and Jennifer Seberry

Distributed Signcryption 155
Yi Mu and Vijay Varadharajan

Fail-Stop Signature for Long Messages 165
Rei Safavi-Naini, Willy Susilo and Huaxiong Wang

Elliptic Curves

Power Analysis Breaks Elliptic Curve Cryptosystems even Secure against the Timing Attack 178
Katsuyuki Okeya and Kouichi Sakurai

Efficient Construction of Cryptographically Strong Elliptic Curves 191
Johannes Buchmann and Harald Baier

Fast Arithmetic

High-Speed Software Multiplication in F_{2^m} 203
Julio López and Ricardo Dahab

On Efficient Normal Basis Multiplication 213
A. Reyhani-Masoleh and M. A. Hasan

Cryptographic Protocols

Symmetrically Private Information Retrieval 225
Sanjeev Kumar Mishra and Palash Sarkar

Two-Pass Authenticated Key Agreement Protocol with Key Confirmation . 237
Boyeon Song and Kwangjo Kim

Anonymous Traceability Schemes with Unconditional Security 250
Reihaneh Safavi-Naini and Yejing Wang

Block Ciphers & Public Key Cryptography

New Block Cipher DONUT Using Pairwise Perfect Decorrelation 262
Dong Hyeon Cheon, Sang Jin Lee, Jong In Lim and Sung Jae Lee

Generating RSA Keys on a Handheld Using an Untrusted Server 271
Dan Boneh, Nagendra Modadugu and Michael Kim

A Generalized Takagi-Cryptosystem with a modulus of the form $p^r q^s$ 283
Seongan Lim, Seungjoo Kim, Ikkwon Yie and Hongsub Lee

Author Index 295

The Correlation of a Boolean Function with Its Variables [⋆]

Dingyi Pei and Wenliang Qin

State Key Laboratory of Information Security,
Graduate School of Chinese Academy of Science

Abstract. The correlation of a Boolean function with its variables is closely related to the correlation attack on stream cipher. The Walsh transformation is the main tool to study the correlation of Boolean functions. The Walsh transformation of a Boolean function with r variables has 2^r coefficients. Let k denote the number of non–zero coefficients of the Walsh Transformations. The paper studies the functions with $1 \leq k \leq 8$. It is proved that the functions with $k = 1$ are the linear functions only, there are no functions with $k = 2, 3, 5, 6, 7$, and finally we construct all functions with $k = 4$ or 8.

keywords: Boolean function, correlation, Walsh transformation, stream cipher

1 Introduction

The Boolean functions are widely used in communication and cryptography. It is important to choice Boolean functions with desired properlies in practice. This paper studies the correlation of a Boolean function with its variables which is a property closely related to the correlation attack on stream cipher [1–4].

Let F_2 be the field with two elements, and $f(x) = f(x_0, \cdots, x_{r-1})$ be a Boolean function from F_2^r to F_2. There is an one – to – one correspondence between the elements $(x_0, x_1, \cdots, x_{r-1})$ of F_2^r and the integers $0 \leq x < 2^r$, defined by $x = x_0 + 2x_1 + \cdots + 2^{r-1}x_{r-1}$. Let $i = i_0 + 2i_1 + \cdots + 2^{r-1}i_{r-1}$ ($i_j = 0$ or 1) be another integer and put $i \cdot x = i_0 x_0 + \cdot + i_{r-1}x_{r-1}$. The Walsh transformation of the function $f(x)$ is defined by

$$a(i) = \sum_{x=0}^{2^r-1} (-1)^{f(x)+i\cdot x}, \quad 0 \leq i < 2^r \tag{1}$$

($f(x) + i \cdot x$ is understood as a read number), which plays an improtant rule in studying of Boolean functions. It is easy to know that $a(i)$ is the difference between the number of x for which $f(x) = i \cdot x$ and the number of x for which $f(x) \neq i \cdot x$. The more large the absolute value of $a(i)$, the more strong the correlation of $f(x)$ with $i \cdot x$. Consider the correlation attack on stream cipher,

[⋆] Supported by NNSF under contract No. 19931010 we wish to find Boolean functions, for which the value $\max_i |a(i)|$ achieves its minimum.

It is well known that

$$\sum_{i=0}^{2^r-1} a(i)^2 = 4^r,$$

hence

$$\max_i |a(i)| \geq 2^{r/2}.$$

When the equality holds, the function f is called bent function, which is not balanced. We hope to find balanced Boolean functions with the value $\max_i |a(i)|$ as small as possible.

Let k denote the number of non–zero coefficients $a(i)$ $(0 \leq i < 2^r)$. The main result of this paper is to determine all Boolean functions with $k \leq 8$. It is possible to generalize the method of this paper for more larger k.

Theorem 1 *Let $\{a(i) \mid 1 \leq i < 2^r\}$ be the Walsh transformation of the Boolean function $f: \ F_2^r \longrightarrow F_2$ and $k = \#\{a(i) \mid a(i) \neq 0, 0 \leq i < 2^r\}$. Then*

(1) *There is no Boolean function with $k = 2, 3, 5, 6, 7$.*
(2) *All functions $f(x)$ with $k = 1$ are linear functions $f(x) = c_0x_0 + \cdots + c_{r-1}x_{r-1} + c_r$ $(c_i \in F_2)$.*
(3) *All functions $f(x)$ with $k = 4$ can be constructed by the following way, Let*

$$V_4 = \{(\alpha_0, \alpha_1, \alpha_2, \alpha_3) \in F_2^4 \mid \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 = 0\}$$

be the subspace of F_2^4. For each $0 \leq j < r$, take $(i_0(j), i_1(j), i_2(j), i_3(j)) \in V_4$ such that

$$i_l = i_l(0) + 2i_l(1) + \cdots + 2^{r-1}i_l(r-1), \ l = 0, 1, 2, 3$$

are 4 different integers. Define $f(x)$ by

$$(-1)^{f(x)} = \pm\frac{1}{2}\{(-1)^{i_0\cdot x} + (-1)^{i_1\cdot x} + (-1)^{i_2\cdot x} - (-1)^{i_3\cdot x}\},$$

then the Walsh transformation of $f(x)$ has four non–zero coefficients

$$(a(i_0), a(i_1), a(i_2), a(i_3)) = \pm(2^{r-1}, 2^{r-1}, 2^{r-1}, -2^{r-1}).$$

(4) *All functions $f(x)$ with $k = 8$ can be constructed by the following way. Put*

$$\begin{aligned} e_0 &= (1, 1, 1, 1, 1, 1, 1, 1), \\ e_1 &= (0, 0, 0, 0, 1, 1, 1, 1), \\ e_2 &= (0, 0, 1, 1, 0, 0, 1, 1), \\ e_3 &= (0, 1, 0, 1, 0, 1, 0, 1). \end{aligned}$$

Let V_8 be the subspace of solutions for the equation system

$$e_0 \cdot x = e_1 \cdot x = e_2 \cdot x = e_3 \cdot x = 0.$$

For each $0 \le j < 8$, take $(i_0(j), i_1(j), \cdots, i_7(j)) \in V_8$ such that $i_l = \sum_{j=0}^{r-1} i_l(j) \cdot 2^j \quad (0 \le l < 8)$ are 8 different integers. Define $f_1(x)$ and $f_2(x)$ by

$$\begin{aligned}(-1)^{f_1(x)} = \pm \frac{1}{4}\Big((-1)^{i_0 \cdot x} + (-1)^{i_1 \cdot x} + (-1)^{i_2 \cdot x} + (-1)^{i_3 \cdot x} + (-1)^{i_4 \cdot x} \\ + (-1)^{i_5 \cdot x} + (-1)^{i_6 \cdot x} - 3(-1)^{i_7 \cdot x}\Big) \\ (-1)^{f_2(x)} = \pm \frac{1}{4}\Big((-1)^{i_0 \cdot x} + (-1)^{i_1 \cdot x} + (-1)^{i_2 \cdot x} + (-1)^{i_3 \cdot x} - (-1)^{i_4 \cdot x} \\ - (-1)^{i_5 \cdot x} - (-1)^{i_6 \cdot x} + 3(-1)^{i_7 \cdot x}\Big)\end{aligned}$$

Then the Walsh transformation of $f_1(x)$ has eight non–zero coefficients

$$\begin{aligned}\big(a(i_0), a(i_1), \cdots, a(i_7)\big) = \pm\, (2^{r-2}, 2^{r-2}, 2^{r-2}, \\ 2^{r-2}, 2^{r-2}, 2^{r-2}, 2^{r-2}, -3 \cdot 2^{r-2}),\end{aligned}$$

the Walsh transformation of $f_2(x)$ has eight non–zero coefficients

$$\begin{aligned}\big(a(i_0), a(i_1), \cdots, a(i_7)\big) = \pm\, (2^{r-2}, 2^{r-2}, 2^{r-2}, \\ 2^{r-2}, -2^{r-2}, -2^{r-2}, -2^{r-2}, 3 \cdot 2^{r-2}),\end{aligned}$$

2 Some Lemmas

Lemma 1 *Let $\{a(i)\}$ be the Walsh transformation of $f(x)$, then*

$$\sum_{i=0}^{2^r-1} a(i)(-1)^{i \cdot j} = 2^r(-1)^{f(j)}, \tag{2}$$

and

$$\sum_{i=0}^{2^r-1} a(i)^2 = 4^r. \tag{3}$$

Let $l = (l_0, l_1, \cdots, l_{u-1}) \subset (0, 1, \cdots, r-1)$ $(u \le r)$ and $(l_u, \cdots, l_{r-1})$ be the complement of l in $(0, 1, \cdots r-1)$. Write x_i also as $x(i)$. Fixing $x(l_t) = y(l_t)$ $(u \le t < r)$ in $x = \big(x(0), \cdots, x(r-1)\big)$, $f(x)$ becomes a Boolean function of u variables $x(l_t)$ $(0 \le t < u)$ with Walsh transformation

$$\begin{aligned}\sum_{x(l_0), \cdots, x(l_{u-1})} (-1)^{f\left(\sum_{t=0}^{u-1} x(l_t) 2^{l_t} + \sum_{t=u}^{r-1} y(l_t) 2^{l_t}\right) + \sum_{t=0}^{u-1} x(l_t) \cdot i(l_t)} \\ = 2^{u-r} \sum_{i(l_u), \cdots, i(l_{r-1})} a(i)(-1)^{\sum_{t=u}^{r-1} y(l_t) i(l_t)}.\end{aligned}$$

Denote above summation of the right side by $S_l(i(l_0),\cdots,i(l_{u-1});$ $y(l_u),\cdots,y(l_{r-1}))$, and $S_l(i(l_0),\cdots,i(l_{u-1});0,\cdots,0)$ is also written as $S_l(i(l_0),\cdots,i(l_{u-1}))$. We have by **Lemma 1** that

$$\sum_{i(l_0),\cdots,i(l_{u-1})} S_l\big(i(l_0),\cdots,i(l_{u-1});y(l_u),\cdots,y(l_{r-1})\big)(-1)^{\sum\limits_{t=0}^{u-1} y(l_t)i(l_t)} = 2^r(-1)^{f(y)} \tag{4}$$

and

$$\sum_{i(l_0),\cdots,i(l_{u-1})} S_l^2(i(l_0),\cdots,i(l_{u-1});y(l_u),\cdots,y(l_{r-1})) = 4^r. \tag{5}$$

Note that (2)–(5) are the equalities satisfied by $\{a(i)\}$.

Lemma 2 *The diophantine equation*

$$x_1^2+x_2^2+x_3^2+x_4^2=4^r$$

has the solutions $(\pm 2^r,0,0,0)$ *and* $(\pm 2^{r-1},\pm 2^{r-1},\pm 2^{r-1},\pm 2^{r-1})$, *and these are the all of its solutions.*

Proof. Suppose (a_1,a_2,a_3,a_4) is a solution, $2^t\|\gcd(a_1,a_2,a_3,a_4)$ and $a_i=2^ty_i$. At least one of y_i $(1\le i\le 4)$ is odd, and $y_1^2+y_2^2+y_3^2+y_4^2=4^{r-t}$, hence we have $t\le r$. If $t=r$, then one of y_i is ±1 and the others are zero, so $(a_1,a_2,a_3,a_4)=(\pm 2^r,0,0,0)$. If $t=r-1$, then $y_i=\pm1$ $(1\le i\le 4)$ and $(a_1,a_2,a_3,a_4)=(\pm 2^{r-1},\pm 2^{r-1},\pm 2^{r-1},\pm 2^{r-1})$. If $t\le r-2$, then

$$y_1^2+y_2^2+y_3^2+y_4^2\equiv 0 \pmod 8.$$

It is impossible since $y^2\equiv 1 \pmod 8$ if y is odd and $y^2\equiv 0$ or $4 \pmod 8$ if y is even.

Lemma 3 *Suppose* $k\ge 5$ *and* $l=(i,j)\subset\{0,1,\cdots,r-1\}$. *If one of* $S_l(u,v)(u,v=0,1)$ *is a non–zero* $a(i)$, *then other three of them could not be a sum of two or three non–zero* $a(i)$.

Proof. Assume $a(i_t)\ne 0$ $(0\le t<k)$. We have by (3)

$$\sum_{t=0}^{k-1} a(i_t)^2=4^r. \tag{6}$$

Without loss of generality we may assume that $l=(0,1)$ and $S_{(0,1)}(0,0)=a(i_0)$. Since

$$S_{(0,1)}^2(0,0)+S_{(0,1)}^2(0,1)+S_{(0,1)}^2(1,0)+S_{(0,1)}^2(1,1)=4^r$$

by (5) and $S_{(0,1)}^2(0,0)=a^2(i_0)<4^r$, therefore

$$S_{(0,1)}^2(0,0)=S_{(0,1)}^2(0,1)=S_{(0,1)}^2(1,0)=S_{(0,1)}^2(1,1)=4^{r-1}$$

(**Lemma 2**). Suppose

$$S^2_{(0,1)}(0,1) = a(i_1) + a(i_2) + \cdots + a(i_s),$$

we should prove that $s \neq 2$ or 3. Similarly, we can prove the same conclusion for $S_{(0,1)}(1,0)$ and $S_{(0,1)}(1,1)$.

Assume that $s = 2$ first. Since $i_1 \neq i_2$, there exists $2 \leq j < r$ such that $i_1(j) \neq i_2(j)$. We can assume $j = 2$ and $i_1(2) = 0$, $i_2(2) = 1$.

	0	1	2
i_0	0	0	
i_1	0	1	0
i_2	0	1	1

We know that $S^2_{(0,1)}(0,1) = \big(a(i_1) + a(i_2)\big)^2 = 4^{r-1}$. Similarly, by (5) and **Lemma 2**, we have $S^2_{(0,1)}(0,1;1,0,\cdots,0) = \big(a(i_1) - a(i_2)\big)^2 = 4^{r-1}$. It follows that $a(i_1)a(i_2) = 0$. This is impossible.

Assume that $s = 3$ next. Since i_1, i_2, i_3 are different to each other, we may assume $i_1(2) = 0$, $i_2(2) = i_3(2) = 1$, and $i_2(3) = 0$, $i_3(3) = 1$. Suppose $i_1(3) = 0$ (similarly to prove for the case of $i_1(3) = 1$).

	0	1	2	3
i_0	0	0		
i_1	0	1	0	0
i_2	0	1	1	0
i_3	0	1	1	1

We can show by the way as above that

$$\begin{aligned} S^2_{(0,1)}(0,1) &= \big(a(i_1) + a(i_2) + a(i_3)\big)^2 = 4^{r-1}, \\ S^2_{(0,1;1,0,\cdots,0)}(0,1) &= \big(a(i_1) - a(i_2) - a(i_3)\big)^2 = 4^{r-1}, \\ S^2_{(0,1;0,1,0,\cdots,0)}(0,1) &= \big(a(i_1) + a(i_2) - a(i_3)\big)^2 = 4^{r-1}. \end{aligned}$$

It follows that $a(i_1)^2 = a(i_2)^2 = a(i_3)^2 = 4^{r-1}$, and we already have $S^2_{(0,1)}(0,0) = a(i_0)^2 = 4^{r-1}$, this is contrary to (6).

3 Case of $1 \leq k \leq 4$

Taking $f(x) + 1$ instead of $f(x)$ if it is necessary, we can assume $f(0) = 0$.

Suppose $k = 1$. There exists an integer $0 \leq i_0 < 2^r$ such that $a(i_0) = \pm 2^r$ and $a(i) = 0$ when $i \neq i_0$. Hence we have $(-1)^{f(j)} = (-1)^{i_0 \cdot j}$ by (2). It follows that $f(j) = i \cdot j$ is a linear function.

It is easy to see by **Lemma 2** and (3) that $k \neq 2, 3$.

Suppose $k = 4$, $a(i_0), a(i_1), a(i_2), a(i_3)$ are non–zero, and the other $a(i) = 0$. Similarly we know $a(i_t) = \pm 2^{r-1}$ $(0 \le t \le 3)$ by **Lemma 2** and (3), Taking $j = 0$ in (2) we get

$$a(i_0) + a(i_1) + a(i_2) + a(i_3) = 2^r.$$

Therefore $\big(a(i_0), a(i_1), a(i_2), a(i_3)\big) = (2^{r-1}, 2^{r-1}, 2^{r-1}, -2^{r-1})$ (ignore the order). Put

$$i_l = \sum_{s=0}^{r-1} i_l(s) 2^s, \quad l = 0, 1, 2, 3,$$

we can show that

$$\big(i_0(s), i_1(s), i_2(s), i_3(s)\big) \in V_4, \quad 0 \le s < r. \tag{7}$$

In fact, if for some s (take $s = 0$) it is the case: $i_0(0) = 0$, $i_1(0) = i_2(0) = i_3(0) = 1$, then $S_{(0)}(0) = \pm 2^{r-1}$, $S_{(0)}(1) = 2^{r-1}$ or $3 \cdot 2^{r-1}$. It is contrary to (5).

Conversely, suppose the condition (7) holds. For any $0 \le j < 2^r$, $(i_0 \cdot j, i_1 \cdot j, i_2 \cdot j, i_3 \cdot j)$ belongs to V_4, therefore

$$(-1)^{i_0 \cdot j} + (-1)^{i_1 \cdot j} + (-1)^{i_2 \cdot j} - (-1)^{i_3 \cdot j} = \pm 2,$$

the conclusion (3) of the **Theorem** is true.

4 Case of $5 \le k \le 8$

Suppose $a(i_t)$ $(0 \le t < k)$ are non–zero and $i_t(0)$ $(0 \le t < k)$ are not all the same.

(i) If there is exactly one 0 among $i_t(0)$ $(0 \le t < k)$ (If there is exactly one 1, the case can be discussed by the same way. We will not consider the symmetrical case obtained by alternating 0 and 1 in the following). Assume $i_0(0) = 0$, $i_t(0) = 1$ $(0 < t < k)$. Then $0 < S_{(0)}(0)^2 = a^2(i_0) < 4^r$, it contradicts to $S^2_{(0)}(0) + S^2_{(0)}(1) = 4^r$ (**Lemma 2**).

(ii) If there are exactly three 0 among $i_t(0)$ $(0 \le t < k)$. Assume $i_0(0) = i_1(0) = i_2(0) = 0$, and $i_0(1) = 0$, $i_1(1) = i_2(1) = 1$. Then $S_{(0,1)}(0,0) = a(i_0)$, $S_{(0,1)}(0,1) = a(i_1) + a(i_2)$, it is impossible by **Lemma 3**.

So far we have proved $k \ne 5$. Suppose $6 \le k \le 8$ in the following.

(iii) If there are exactly two 0 among $i_t(0)$ $(0 \le t < k)$. Assume $i_0(0) = i_1(0) = 0$ and $i_0(1) = 0, i_1(1) = 1$. Using (i), (ii) already proved above and **Lemma 3**(take $(i,j) = (0,1)$), we need only to consider the case that there is only one 0 among $i_t(1)$ $(2 \le t < k)$. Assume $i_2(1) = 0, i_t(1) = 1$ $(3 \le t < k)$. Then $S_{(0,1)}(0,0) = a(i_0)$, $S_{(0,1)}(0,1) = a(i_1)$, $S_{(1,0)}(1,0) = a(i_2)$, $S_{(0,1)}(1,1) = a(i_3) + a(i_4) + \cdot + a(i_{k-1})$. Hence we have proved $k \ne 6$ (**Lemma 3**). Suppose $k = 7$ or 8, we have

$$a(i_0)^2 = a(i_1)^2 = a(i_2)^2 = \Big(a(i_3) + \sum_{t=4}^{k-1} a(i_t)\Big)^2 = 4^{r-1}, \tag{8}$$

We may assume that $i_t(2)$ $(3 \le t < k)$ are not all the same. If there is only one 0 among $i_t(2)$ $(3 \le t < k)$. Assume $i_3(2) = 0$, then $S_{(0,1)}(1,1;1,0,\cdots,0) = a(i_3) - \sum\limits_{t=4}^{k-1} a(i_t)$ and

$$\Big(a(i_3) - \sum_{t=4}^{k-1} a(i_t)\Big)^2 = 4^{r-1}$$

by (5) and **Lemma 2**. It follows $a(i_3)^2 = 4^{r-1}$ by (8) and $\sum\limits_{t=0}^{3} a(i_t)^2 = 4^r$, which contradicts to (3). If there are exactly two 0 among $i_t(2)$ $(3 \le t < k)$, assume $i_3(2) = i_4(2) = 0$. If $k = 7$, we need only to consider the case that $i_t(2) = 0$ $(0 \le t < 3)$. Since $i_5 \ne i_6$, we assume $i_5(3) = 0$, $i_6(3) = 1$. Then $S_{(1,2)}(1,1) = a(i_5) + a(i_6)$, $S_{(1,2)}(1,2;0,1,0,0,0) = a(i_5) - a(i_6)$. Hence

$$\big(a(i_5) + a(i_6)\big)^2 = 0 \text{ or } 4^r,$$
$$\big(a(i_5) - a(i_6)\big)^2 = 0 \text{ or } 4^r.$$

It follows that $a(i_5)^2 = a(i_6)^2 = 4^{r-1}$ and this fact together with (8) contradicts to (3). Hence we have proved $k \ne 7$. If $k = 8$, we need only to consider the case that there is only one 1 among $i_t(2)$ $(0 \le t < 3)$. taking $(i,j) = (1,2)$ when $i_0(2) = 1$ or $i_2(2) = 1$, or $(i,j) = (0,2)$ when $i_1(2) = 1$, we can prove it is also impossible by **Lemma 3**.

Now we assume $k = 8$. Summerizing what have proved above, for any $0 \le j < r$, $i_t(j)$ $(0 \le t < 8)$ are all 0, or all 1,or half of them are 0. The last case must appear since i_t $(0 \le t < 8)$ are different to each other. We may assume $i_t(0) = 0$ $(0 \le t < 4)$ and $i_t(0) = 1$ $(4 \le t \le 7)$. Furthermore we assume $i_t(1)$ $(0 \le t < 3)$ are not all the same. It is imposible that there is only one 0 (or 1) among $i_t(1)$ $(0 \le t < 3)$ (**Lemma 3**). Therefore we can assume $i_0(1) = i_1(1) = 0$, $i_2(1) = i_3(1) = 1$, $i_4(1) = i_5(1) = 0$, $i_6(1) + i_7(1) = 1$, and $i_0(2) = i_2(2) = i_4(2) = i_6(2) = 0$, $i_1(2) = i_3(2) = i_5(2) = i_7(2) = 1$. Taking $l = (0,1)$, $\big(y(2),\cdots,y(7)\big) = (0,0,\cdots,0)$ and $\big(y(2),\cdots,y(7)\big) = (1,0,\cdots,0)$ respectively in (5), we obtain

$$\big(a(i_0) + a(i_1)\big)^2 + \big(a(i_2) + a(i_3)\big)^2 + \big(a(i_4) + a(i_5)\big)^2 + \big(a(i_6) + a(i_7)\big)^2 = 4^r,$$
$$\big(a(i_0) - a(i_1)\big)^2 + \big(a(i_2) - a(i_3)\big)^2 + \big(a(i_4) - a(i_5)\big)^2 + \big(a(i_6) - a(i_7)\big)^2 = 4^r.$$

When a and b are non–zero integers, $(a+b)^2$ and $(a-b)^2$ could not be 4^{r-1} (or 0) simultaneously, hence we have by **Lemma 2**

$$\begin{cases} \big(a(i_0) + a(i_1)\big)^2 = \big(a(i_2) + a(i_3)\big)^2 = \big(a(i_4) + a(i_5)\big)^2 = \big(a(i_6) + a(i_7)\big)^2 = 4^{r-1}, \\ \big(a(i_0) - a(i_1)\big)^2 = \big(a(i_2) - a(i_3)\big)^2 = \big(a(i_4) - a(i_5)\big)^2 = 0, \big(a(i_6) - a(i_7)\big)^2 = 4^r, \end{cases}$$

or

$$\begin{cases} \big(a(i_0) + a(i_1)\big)^2 = \big(a(i_2) + a(i_3)\big)^2 = \big(a(i_4) + a(i_5)\big)^2 = 0, \big(a(i_6) + a(i_7)\big)^2 = 4^r. \\ \big(a(i_0) - a(i_1)\big)^2 = \big(a(i_2) - a(i_3)\big)^2 = \big(a(i_4) - a(i_5)\big)^2 = \big(a(i_6) - a(i_7)\big)^2 = 4^{r-1}. \end{cases}$$

Consider the first equation system. It follows $a(i_0) = a(i_1) = \pm 2^{r-2}$, $a(i_2) = a(i_3) = \pm 2^{r-2}$, $a(i_4) = a(i_5) = \pm 2^{r-2}$, $\big(a(i_6), a(i_7)\big) = \pm(3 \cdot 2^{r-2}, -2^{r-2})$. Taking $j = 0$ in (2) we get

$$a(i_0) + a(i_2) + a(i_4) \pm 2^{r-2} = 2^{r-1}.$$

Therefore we obtain two solutions

$$\begin{gathered} 2^{r-2}, 2^{r-2}, 2^{r-2}, 2^{r-2}, 2^{r-2}, 2^{r-2}, 2^{r-2}, -3 \cdot 2^{r-2} \\ 2^{r-2}, 2^{r-2}, 2^{r-2}, 2^{r-2}, -2^{r-2}, -2^{r-2}, -2^{r-2}, 3 \cdot 2^{r-2}. \end{gathered}$$

The second equation system has the same two solutions.

It is easy to check that

$$\begin{aligned} \frac{1}{4}\big((-1)^{i_0 \cdot x} + (-1)^{i_1 \cdot x} + (-1)^{i_2 \cdot x} + (-1)^{i_3 \cdot x} + (-1)^{i_4 \cdot x} + (-1)^{i_5 \cdot x} \\ + (-1)^{i_6 \cdot x} - 3 \cdot (-1)^{i_7 \cdot x}\big) = \pm 1 \end{aligned}$$

and

$$\begin{aligned} \frac{1}{4}\big((-1)^{i_0 \cdot x} + (-1)^{i_1 \cdot x} + (-1)^{i_2 \cdot x} + (-1)^{i_3 \cdot x} - (-1)^{i_4 \cdot x} - (-1)^{i_5 \cdot x} \\ - (-1)^{i_6 \cdot x} + 3 \cdot (-1)^{i_7 \cdot x}\big) = \pm 1 \end{aligned}$$

if $\big(i_0(j), i_1(j), \cdots, i_7(j)\big) \in V_8$ $(0 \le j < r)$. The **Theorem** is proved completely now.

References

1. V.Chepyzhov and B.Smeets, On a fast correlation attack on certain stream cipher, Advance in Cryptology–Eurocrypt'91 (**LNCS 547**) (1991), 176–185.
2. J.Golic, On the security of shift register based keystream generators, R.Anderson, editor, Fast Software Encryption, Cambridge Security Workshop, Springer–Verlag (**LNCS 809**) (1994), 90–100.
3. J.Golic and M.Mihaljevic, A generalized correlation attack on a class of stream cipher based on the Levenshtein distance, J. of Cryptology 3 (1991), 201–212.
4. R.A.Rueppel, Stream cipher, G.J.Simmons, editor, Contemporary Cryptology: The Science of Information Integrity (1992), 65–134.

On Choice of Connection-Polynomials for LFSR-Based Stream Ciphers

Jambunathan K

Indian Statistical Institute,
203, Barrackpore Trunk Road,
Calcutta 700 035.
India.

Abstract. Here I suggest a design criterion for the choice of connection-polynomials in LFSR-based stream-cipher systems. I give estimates of orders of magnitude of the sparse-multiples of primitive-polynomials. I show that even for reasonable degrees (degrees of the order of 100) of primitive connection-polynomials the degrees of their sparse-multiples are "considerably higher".

1 Introduction

A binary linear-feedback shift-register (LFSR, in short) is a system which generates a pseudo-random bit-sequence using a binary recurrence-relation of the form

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k}$$

where $c_k = 1$ and each c_i other than c_k belong to $\{0,1\}$.The length of the LFSR corresponds to the order k of the linear-recurrence-relation used. The number of taps of the LFSR is the number t of non-zero bits in $\{c_1, c_2, \ldots, c_k\}$.

Once the shift-register is initialised by assigning values to a_0, $a_1, \ldots, a_{k-1}$ i.e., once the seed of the LFSR is set, the successive bits of the sequence are emitted using the chosen recurrence relation.

The above LFSR is closely related to the following polynomial over GF(2)

$$c(X) = c_0 + c_1 X + c_2 X^2 + \ldots + c_k X^k$$

with c_0=1. This polynomial is called the connection-polynomial of the LFSR. If X in c(X) is interpreted as an operator that shifts left the argument sequence, it can be inferred that the connection polynomial define the fundamental recurrence over the LFSR generated sequence **a**. Similarly it can be seen that any multiple of the connection-polynomial correspondingly define a linear-recurrence-relation which holds on the LFSR-generated sequence.

The connection-polynomials are in general chosen as primitive-polynomials over GF(2) in order to generate a key-stream of maximum periodicity for the given length of the LFSR.

LFSRs are popularly employed in stream-cipher systems to genearte a key-stream sequence which is bitwise xored with message sequence to produce an encrypted message. In practical implementations, the key-stream is usually generated by combining the outputs of more than one LFSRs using a non-linear boolean combining function.This arrangement significantly increases the robustness of the system against possible attacks.

LFSR systems with their connection-polynomials very sparse are particularly very vulnerable to various known attacks. The underlying principles of these attacks are easily extendable to the situation where the feedback polynomial has many terms but is a factor of a low density polynomial of moderate degree. For example, the primitive-polynomial $1 + x^2 + x^4 + x^5 + x^6 + x^{10} + x^{11} + x^{12} + x^{13} + x^{14} + x^{17} + x^{18} + x^{21} + x^{24} + x^{26} + x^{27} + x^{30} + x^{31} + x^{32} + x^{33} + x^{35} + x^{36} + x^{37} + x^{39} + x^{40} + x^{42} + x^{43} + x^{44} + x^{45} + x^{47} + x^{48} + x^{52} + x^{53} + x^{56} + x^{58} + x^{60} + x^{61} + x^{62} + x^{63} + x^{64} + x^{65} + x^{66} + x^{69} + x^{70} + x^{71} + x^{72} + x^{74} + x^{75} + x^{77} + x^{78} + x^{81} + x^{82} + x^{84} + x^{87} + x^{90} + x^{92} + x^{93}$ though sufficiently dense does not qualify as Connection-polynomial of a LFSR. This is because this polynomial divides the moderate-degree 4-nomial $x^{285} + x^{279} + x^{12} + 1$. Meier and Staffelbach [1] state that "it appears to be very difficult to decide whether a given polynomial has this (*above mentioned*) property". We address this issue and suggest a design criterion for the choice of connection polynomials for LFSR-based stream-cipher systems.

2 Some Results on Sparse-Multiples of Polynomials

2.1 On Trinomial Multiples

In this section we treat trinomial-multiples of polynomials and their associated properties.

Theorem 1. *If f(x) is a primitive polynomial of degree d > 0 and if $x^s + x^t + 1$ is the least degree trinomial-multiple of it then $s \leq \frac{(2^d+2)}{3}$.*

Proof. Let f(x) be a primitive polynomial of degree d > 0 and let $x^s + x^t + 1$ be the least-degree trinomial-multiple of it. Also let e be the exponent to which f(x) belongs. Now consider the following set of polynomials

$$S_1 = \{x, x^2, x^3, \ldots, x^s\}$$
$$S_2 = \{x^s + x, x^s + x^2, x^s + x^3, \ldots, x^s + x^{s-1}\}$$
$$S_3 = \{x^t + x, x^t + x^2, x^t + x^3, \ldots, x^t + x^{s-1}\}$$

Now we make the following claims:
1) The set S_1 contains elements that are distinct (mod f(x)).

If this were not true we would have $x^i \equiv x^j$ (mod f(x)) for some 1 $\leq$ i, j $\leq$ s and i $\neq$ j. Without loss of generality assume that i > j. Now since we are given that f(x) divides a trinomial with non-zero constant term, we can infer that f(x) is prime to x. So cancelling out common x-power terms in the above congruence we would have $x^{(i-j)} \equiv 1$ (mod f(x)).This implies that e divides (i-j). But since (i-j) < s, we can infer that e < s.

Now let s' and t' be the least non-negative residues (mod e) of s and t respectively. Since $x^s + x^t + 1 \equiv 0$ (mod f(x)) we must have $x^{s'} + x^{t'} + 1 \equiv 0$ (mod f(x)). Since f(x) is prime to x, neither s' nor t' can be zero.Without loss of genearlity assume that s' > t' so that the degree of the trinomial $x^{s'} + x^{t'} + 1$ is s'. Since $x^{s'} + x^{t'} + 1$ is a trinomial-multiple of f(x) we should have s' ≥ s. But we inferred that s > e in the last para. So putting these inequalities together we get s' > e. But this cannot be true. Hence our initial assumption must be wrong and the set S_1 should indeed contain elements distinct (mod f(x)).

2) The sets S_2 and S_3 also contain elements that are distinct (mod f(x)).
The proof of this is very similar to that given for claim (1) above.

3) No two elements belonging to sets S_1 and S_2 are congruent (mod f(x)).
If this were not true we would have $x^i \equiv x^s + x^j$ (mod f(x)), 1 ≤ i ≤ s, 1 ≤ j < s. Since f(x) is prime to x, s ≠ i and i ≠ j. Also s ≠ j. (i.e., s, i, j are all different). In this case as before, we could cancel out the common x-power terms in the above congruence and end up with a trinomial-multiple of f(x) whose degree is less than s. But this would be a contradiction.

4) No two elements belonging to sets S_1 and S_3 are congruent (mod f(x)).
The proof is similar to that given for claim (3) above.

5) No two elements of sets S_2 and S_3 are congruent (mod f(x)).
If this were not true we would have $x^s + x^i \equiv x^t + x^j$ (mod f(x)) for some 1 ≤ i, j ≤ s-1. Noticing that $x^s + x^t \equiv 1$ (mod f(x)), we would have $1 + x^i + x^j \equiv 0$ (mod f(x)). Furthermore i cannot be equal to j. Thus, as before we have ended up with a trinomial multiple of f(x) the degree of which is less than s. This cannot be true.

The claims (1) to (5) proved above, in effect, say that the sets S_1, S_2 and S_3 contain (3s-2) elements distinct (mod f(x)). This is possible only if (3s-2) ≤ 2^d. i.e., s ≤ $\frac{(2^d+2)}{3}$. □

Theorem 2. *An irreducible polynomial belonging to exponent e divides a trinomial iff (* $x^e + 1$, $(x+1)^e + 1$*) is non-trivial.*

Proof. Let f(x) be an irreducible polynomial of degree d belonging to an exponent e. Let α be a root of it. Now consider the polynomials

$$x^e + 1 = \prod_{i=0}^{e-1} (x + \alpha^i) \tag{1}$$

$$(x+1)^e + 1 = \prod_{i=0}^{e-1} (x + \alpha^i + 1) \tag{2}$$

If polynomials (1) and (2) have a non-trivial gcd then they have a common root implying that $\alpha^m = \alpha^n + 1$ for some non-negative m, n < e. This suggests that α is a root of the polynomial $x^m + x^n + 1$. Since f(x) is the minimal polynomial of α this in turn suggests that f(x) divides the trinomial $x^m + x^n + 1$.

Conversely, if f(x) divides some trinomial $x^m + x^n + 1$ then it also divides the trinomial $x^{m'} + x^{n'} + 1$ where m' and n' are the least positive residues (mod e) of m and n respectively. Therefore, $\alpha^{m'} + \alpha^{n'} + 1 = 0$. This suggests that polynomials (1) and (2) have a common root and hence a non-trivial gcd. □

For the sake of illustration, consider the polynomial $x^4 + x^3 + x^2 + x + 1$ which is irreducible over GF(2). This polynomial belongs to exponent 5 and does not divide any trinomial.

Theorem 3. *If f(x) is a primitive-polynomial of degree d and if $x^m + x^n + 1$ is a trinomial divisible by f(x) then m and n belong to the same-length cyclotomic-coset (mod$(2^d - 1)$).*

Proof. Assume that

$$x^m + x^n + 1 \equiv 0 \ (mod\, f(x)) \tag{3}$$

and let l_m and l_n be the length of the cyclotomic-cosets ($\mathrm{mod}(2^d - 1)$) to which m and n belong. So we have,

$$\begin{aligned} 2^{l_m} m &\equiv m \ \ (mod(2^d - 1)) \\ 2^{l_n} \ n &\equiv n \ \ (mod(2^d - 1)) \end{aligned}$$

Raising both sides of the congruence (3) to the power 2^{l_m} we have,

$$\begin{aligned} x^{m2^{l_m}} + x^{n2^{l_m}} + 1 &\equiv 0 \ \ (mod\, f(x)) \\ x^m + x^{n2^{l_m}} + 1 &\equiv 0 \ \ (mod\, f(x)) \end{aligned} \tag{4}$$

Adding congruences (3) and (4) and after rearranging terms we get

$$x^{(2^{l_m}-1)n} \equiv 1 \ (mod\, f(x))$$

Therefore

$$(2^{l_m} - 1)n \equiv 0 \ (mod(2^d - 1))$$

which implies that l_n divides l_m. By similar reasoning, it follows that l_m divides l_n. Therefore $l_m = l_n$ and the theorem follows. □

For the sake of illustration, consider the case d=6. The set of all cyclotomic-cosets ($mod(2^6-1)$) are,

$$\begin{aligned}
C_0 &= \{0\}\\
C_1 &= \{1, 2, 4, 8, 16, 32\}\\
C_3 &= \{3, 6, 12, 24, 48, 33\}\\
C_5 &= \{5, 10, 20, 40, 17, 34\}\\
C_7 &= \{7, 14, 28, 56, 49, 35\}\\
C_9 &= \{9, 18, 36\}\\
C_{11} &= \{11, 22, 44, 25, 50, 37\}\\
C_{13} &= \{13, 26, 52, 41, 19, 38\}\\
C_{15} &= \{15, 30, 60, 57, 51, 39\}\\
C_{21} &= \{21, 42\}\\
C_{23} &= \{23, 46, 29, 58, 53, 43\}\\
C_{27} &= \{27, 54, 45\}\\
C_{31} &= \{31, 62, 61, 59, 55, 47\}
\end{aligned}$$

The polynomial $x^6+x^4+x^3+x+1$ is primitive. The set of all trinomials of degree less than (2^6-1) that it divides are

$$\begin{array}{ccc}
x^8+x^7+1 & x^{13}+x^3+1 & x^{16}+x^{14}+1\\
x^{23}+x^{11}+1 & x^{26}+x^6+1 & x^{27}+x^9+1\\
x^{30}+x^5+1 & x^{32}+x^{28}+1 & x^{34}+x^{15}+1\\
x^{35}+x^4+1 & x^{38}+x^{33}+1 & x^{39}+x^{17}+1\\
x^{41}+x^{24}+1 & x^{42}+x^{21}+1 & x^{43}+x^{37}+1\\
x^{44}+x^{29}+1 & x^{45}+x^{36}+1 & x^{46}+x^{22}+1\\
x^{48}+x^{19}+1 & x^{49}+x^2+1 & x^{51}+x^{40}+1\\
x^{52}+x^{12}+1 & x^{53}+x^{50}+1 & x^{54}+x^{18}+1\\
x^{56}+x^1+1 & x^{57}+x^{20}+1 & x^{58}+x^{25}+1\\
x^{59}+x^{31}+1 & x^{60}+x^{10}+1 & x^{61}+x^{47}+1\\
 & x^{62}+x^{55}+1 &
\end{array}$$

Note that the powers of x occurring in the same trinomial-multiple belong to the same-length cyclotomic-coset ($mod(2^6-1)$).

2.2 On 4-nomial Multiples

In this section we give a upper bound on the degree of the minimum-degree 4-nomial-multiple of a Polynomial.

Theorem 4. *All primitive-polynomials of degree* $d \geq 3$ *divide some 4-nomial of degree* $\leq \lfloor \frac{1+\sqrt{1+4.2^{d+1}}}{2} \rfloor$.

Proof. Let f(x) be a primitive-polynomial of degree d. Let $f_0 = \lfloor \frac{1+\sqrt{1+4.2^{d+1}}}{2} \rfloor$ Consider the set of all binomials of the form $x^i + x^j$, where 0 ≤ i, j ≤ f, for some f and i ≠ j. There are $\frac{(f+1)f}{2}$ of them. For the choice of f=f_0, the number of these binomials exceed 2^d. Since there are only 2^d different congruence classes (mod f(x)), by the pigeon-hole principle atleast two of these binomials should be congruent (mod f(x)). Thus there are two different un-ordered pairs r_1, s_1 and r_2, s_2 such that

$$x^{r_1} + x^{s_1} \equiv x^{r_2} + x^{s_2} \, (mod \, f(x))$$

For d ≥ 3, if r_1 were equal to r_2, then

$$x^{s_1} \equiv x^{s_2} \, (mod \, f(x))$$

which implies that $s_1 \equiv s_2 (mod (2^d - 1))$. Since $0 \leq s_1$, $s_2 \leq f_0$ and $f_0 < (2^d - 1)$ it follows that $s_1 = s_2$ contradicting the fact that r_1, s_1 and r_2, s_2 are different un-ordered pairs. Thus the above congruence give rise to 4-nomial divisible by f(x) and of degree atmost f_0. □

2.3 On Degrees of Sparse-Multiples of Primitive-Polynomials

In this section we study the nature of upper and lower bounds on the degrees of sparse-multiples of primitive-polynomials.

Firstly we show that there are relatively fewer number of primitive-polynomials of reasonable degree that divide a lower-weight polynomial of lesser degree. This result shows that any randomly chosen primitive-polynomial of reasonable degree qualifies as a connection polynomial of a LFSR with high probability.

Subsequently we comment on how small the degrees of sparse-multiples of certain primitive-polynomials could be.

Lemma 1. *For all integers d,* $\phi(2^d - 1) > (1.548^d - 1)$

Proof. Consider the factorization of $2^d - 1$ in to primes. Let

$$2^d - 1 = \prod_{i=1}^{r} p_i^{\alpha_i}$$

where each p_i is a prime and each $\alpha_i \geq 1$. Then

$$\phi(2^d - 1) = (2^d - 1) \prod_{i=1}^{r} (1 - \frac{1}{p_i}) \tag{5}$$

Since $2^d - 1$ is odd, each $p_i \geq 3$ and $(1 - \frac{1}{p_i}) \geq (2/3)$. Therefore,

$$\prod_{i=1}^{r}(1 - \frac{1}{p_i}) \geq (2/3)^r \tag{6}$$

Also, $2^d - 1 \geq \prod_{i=1}^{r} p_i \geq 3^r$. So $2^d > 3^r$ and $r < (log_3 2)d$. Therefore,

$$(2/3)^r > (2/3)^{(log_3 2)d} \tag{7}$$

Equation (5) together with inequalities (6) and (7) give that

$$\phi(2^d - 1) > (2^d - 1)(2/3)^{(log_3 2)d}$$

i.e.,

$$\phi(2^d - 1) > (1.548^d - 0.774^d) > (1.548^d - 1)$$

This proves the lemma. □

Theorem 5. *For a given* $t \geq 2$ *and* $s \geq 1$*, if* d *is such that* $1.548^d - 1 \geq (t-1)\binom{d^s+1}{t}$ *then there exists atleast one primitive-polynomial of degree d which does not divide any t-nomial of degree* $\leq d^s$*.*

Proof. Given t $\geq$ 2 and s $\geq$ 1, let d be chosen such that $1.548^d - 1 \geq (t-1)\binom{d^s+1}{t}$. Let us assume that for this choice of d, all primitive-polynomials of degree d divide some t-nomial of degree $\leq d^s$. If $\pi_{pri}(x, d)$ denotes the product of all primitive-polynomials of degree d and $\pi_{t-nomial}(x, d^s)$ denotes the product of all t-nomials prime to x and of degree $\leq d^s$, then

$$\pi_{pri}(x, d) \text{ divides } \pi_{t-nomial}(x, d^s)$$

and

$$\text{the degree of } \pi_{pri}(x, d) \leq \text{the degree of } \pi_{t-nomial}(x, d^s).$$

The degree of $\pi_{pri}(x, d)$ is $\phi(2^d - 1)$

The degree of $\pi_{t-nomial}(x, d^s)$ $\quad \sum_{\alpha=t}^{d^s} \alpha \binom{\alpha-1}{t-2}$

$$\sum_{\alpha=0}^{d^s} \alpha \binom{\alpha-1}{t-2}$$

$$(t-1)\sum_{\alpha=0}^{d^s}[\binom{\alpha+1}{t} - \binom{\alpha}{t}]$$

$$(t-1)\binom{d^s+1}{t}$$

Therefore,

$$\phi(2^d - 1) \leq (t-1)\binom{d^s+1}{t} \tag{8}$$

By lemma (1) we have,

$$\phi(2^d - 1) > 1.548^d - 1 \tag{9}$$

Inequalities (8) and (9) together give

$$1.548^d - 1 < (t-1)\binom{d^s+1}{t} \tag{10}$$

This inequality contradicts the choice of d and hence our initial assumption must be wrong. This establishes the theorem. □

Note here that the above theorem can be easily extended to the case where d is such that $\phi(2^d-1) > (t-1)\binom{d^s+1}{t}$.

Theorem 6. *Given $t \geq 2$ and $s \geq 1$, if d is such that $\phi(2^d-1) > (t-1)\binom{d^s+1}{t}$ then the probability that a randomly chosen primitive-polynomial of degree d does not divide any t-nomial of degree $\leq d^s$ is $\geq 1 - \frac{(t-1)\binom{d^s+1}{t}}{\phi(2^d-1)}$.*

Proof. Given t $\geq$ 2 and s $\geq$ 1, let d be chosen as above. For this choice of d, let $n(d,t,s)$ denote the number of primitive-polynomials $f(x)$ of degree d that divide some t-nomial of degree $\leq d^s$. Then the product of all such polynomials $f(x)$ divides $\pi_{t-nomial}(x, d^s)$. Considering the degrees of the respective products, it follows that,

$$n(d,t,s)\, d \leq (t-1)\binom{d^s+1}{t}$$

i.e.,

$$n(d,t,s) \leq \frac{(t-1)\binom{d^s+1}{t}}{d}$$

Since number of primitive-polynomials of degree d is $\frac{\phi(2^d-1)}{d}$, number of primitive-polynomials of degree d that do not divide any t-nomial of degree $\leq d^s$ is $\geq \frac{\phi(2^d-1)}{d} - \frac{(t-1)\binom{d^s+1}{t}}{d}$ If we denote by $p(d,t,s)$ the probability that a randomly chosen primitive-polynomial of degree d does not divide any t-nomial of degree $\leq d^s$, then

$$p(d,t,s) \;\geq\; 1 - \frac{(t-1)\binom{d^s+1}{t}}{\phi(2^d-1)} \tag{11}$$

This proves the theorem. □

Note here that the above lower bound for $p(d,t,s)$ is arbitrarily close to 1 for sufficiently large values of d (irrespective of the choices of s and t).

For the sake of completion refer to Table (1). The table gives the least degree d $\leq$ 128 such that $1 - \frac{(t-1)\binom{d^s+1}{t}}{\phi(2^d-1)} \geq 0.9$ for various s, t pairs.

For example the probability that a randomly chosen primitive polynomial of degree 78 does not divide any trinomial of degree $\leq 78^4$ (approx. 2^{25}) is more than 0.9.

Table 1. The least degree d ≤ 128 for which $1 - \frac{(t-1)\binom{d^s+1}{t}}{\phi(2^d-1)} \geq 0.9$

s \ t	3	4	5	6	7	8	9	10
2	33	45	57	71	85	97	111	125
3	55	77	99	121	-	-	-	-
4	78	109	-	-	-	-	-	-
5	103	-	-	-	-	-	-	-

Theorem 7. *There exists primitive-polynomials of degree d which divide a trinomial of degree 3d and a 4-nomial of degree < 6d.*

Proof. Let f(x) be a primitive trinomial of odd degree d [2]. Let α be a root of it. So that,

$$f(\alpha) = 0 \tag{12}$$

Since d is odd $(3, 2^d - 1) = 1$. Let $\beta = \alpha^{1/3 \; mod(2^d-1)}$. Note that β is primitive and that

$$\beta^3 = \alpha \tag{13}$$

Consider the minimal polynomial g(x) of β. g(x) is primitive and it's degree is d. Equations (12) and (13) give

$$f(\beta^3) = 0 \tag{14}$$

Since g(x) is the minimal polynomial of β, we have that

$$f(x^3) \equiv \; 0(mod\, g(x)) \tag{15}$$

Thus we have produced a primitive-polynomial g(x) of degree d which divides a trinomial $f(x^3)$ of degree 3d. Now construct a 4-nomial from $f(x^3)$ as follows. For the sake of simplicity call the polynomial $f(x^3)$ as F(x). Let F(x)= $x^s + x^t + 1$ where (s= 3d) > t. We have from equation (15),

$$F(x) \equiv 0 \; (mod\, g(x)) \tag{16}$$

So,

$$F(x^2) \equiv 0 \; (mod\, g(x)) \tag{17}$$

Also,

$$x^s F(x) \equiv 0 \; (mod\, g(x)) \tag{18}$$

[2] There possibly exists good number of such primitive trinomials

Equations (17) and (18) yield

$$x^s F(x) + F(x^2) \equiv 0 \ (mod\, g(x))$$

i.e.,

$$x^{s+t} + x^s + x^{2t} + 1 \equiv 0 \ (mod\, g(x))$$

Note that $x^{s+t} + x^s + x^{2t} + 1$ is a 4-nomial of degree < 6d. Thus we have shown that a primitive-polynomial g(x) of degree d as chosen above divides a 4-nomial of degree < 6d. □

For example, if we choose $x^{17} + x^3 + 1$ as the primitive trinomial f(x) in the above given construction we can see that the primitive polynomial $x^{17} + x^{15} + x^{11} + x^{10} + x^9 + x^8 + x^4 + x^2 + 1$ divides the 4-nomial $x^{60} + x^{51} + x^{18} + 1$.

It is worth noticing here that the construction used in the above theorem is quite general. We could have as well started with a primitive 5-nomial or 7-nomial and used any other smaller n th root as appropriate, instead of cubic root (of the primitive element α) and derived corresponding results.

3 Conclusions

Here we have showed that the degrees of sparse-multiples of a primitive-polynomial of reasonable degree, in general are sufficiently high. This conclusively establishes that sparse-multiples variant of various LFSR attacks are in general infeasible requiring very long ciphertexts.

References

1. W.Meier and O.Staffelbach, Fast Correlation Attacks on Certain Stream-Ciphers, Journal of cryptology(1989) 1:159-176 10

On Resilient Boolean Functions with Maximal Possible Nonlinearity

Yuriy V. Tarannikov

Mech. & Math. Department Moscow State University
119899 Moscow, Russia
yutaran@mech.math.msu.su, taran@vertex.inria.msu.ru

Abstract. It is proved that the maximal possible nonlinearity of n-variable m-resilient Boolean function is $2^{n-1}-2^{m+1}$ for $\frac{2n-7}{3} \le m \le n-2$. This value can be achieved only for optimized functions (i. e. functions with an algebraic degree $n-m-1$). For $\frac{2n-7}{3} \le m \le n-\log_2 \frac{n+2}{3} - 2$ it is suggested a method to construct an n-variable m-resilient function with maximal possible nonlinearity $2^{n-1}-2^{m+1}$ such that each variable presents in ANF of this function in some term of maximal possible length $n-m-1$.

Keywords: *stream cipher, Boolean function, nonlinear combining function, correlation-immunity, resiliency, nonlinearity, algebraic degree, Siegenthaler's Inequality, hardware implementation, pseudorandom generator.*

1 Introduction

One of the most general types of stream cipher systems is several Linear Feedback Shift Registers (LFSRs) combined by nonlinear Boolean function. This function must satisfy certain criteria to resist different attacks (in particular, correlation attacks suggested by Siegenthaler [14] and different types of linear attacks). Besides this function must have sufficiently simple scheme implementation in hardware So, the following factors are considered as important properties of Boolean functions for using in stream cipher applications.

1. *Balancedness.* A Boolean function must output zeroes and ones with the same probabilities. 2. Good *correlation-immunity* (of order m). The output of Boolean function must be statistically independent of combination of any m its inputs. A balanced correlation-immune of order m Boolean function is called m-*resilient.* 3. Good *nonlinearity.* The Boolean function must be at the sufficiently large distance from any affine function. 4. High *algebraic degree.* The degree of Algebraic Normal Form (ANF) of Boolean function must be sufficiently large. 5. High *algebraic degree of each individual variable.* Each variable of Boolean function must appear in ANF of this function in some term of sufficiently large length. 6. Simple *implementation in hardware.* The Boolean function must have sufficiently simple scheme implementation.

There are a lot of papers where only one of these criteria is studied. It was found that the nonlinearity of a Boolean function does not exceed $2^{n-1} - 2^{\frac{n}{2}-1}$ [9]. The consideration of pairs of these criteria gave some trade-offs between them. So, the Boolean function with maximal possible nonlinearity can not be balanced. Another result is Siegenthaler's Inequality: [13] if the function f is a correlation-immune function of order m then $\deg(f) \leq n - m$, moreover, if f is m-resilient, $m \leq n-2$, then $\deg(f) \leq n-m-1$. Siegenthaler and other authors pointed out that if the Boolean function is *affine* or depends *linearly* on a big number of variables then this function has a simple implementation. But such function can not be considered as good for cryptographic applications because of another criteria, in particular, algebraic degrees of linear variables are 1.

The variety of criteria and complicated trade-offs between them caused the next approach: to fix one or two parameters and try to optimize others. The most general model is when researchers fix the parameters n (number of variables) and m (order of correlation-immunity) and try to optimize some other criptographically important parameters. Here we can call the works [12,2,5,4,6,7,8]. In [15,11,16] it was proved (independently) that the nonlinearity of n-variable m-resilient Boolean function does not exceed $2^{n-1} - 2^{m+1}$.

The present paper is based on our preprint [15], it continues the investigations in this direction and gives new results. In Section 2 we give preliminary concepts, notions and some simple lemmas. In Section 3 we give recently established new trade-off between resiliency and nonlinearity, namely, that the nonlinearity of n-variable m-resilient Boolean function does not exceed $2^{n-1}-2^{m+1}$. Moreover, it appears that this bound can be achieved only if Siegenthaler's Inequality is achieved too. In Section 4 we discuss a concept of a linear variable and introduce a new important concept of a pair of *quasilinear* variables which works in the following sections. We discuss the connection of linear and quasilinear dependence with resiliency and nonlinearity of the function and give a representation form for the function with a pair of quasilinear variables. In Section 5 we present our main construction method. This method allows to construct recursively the functions with good cryptographic properties using the functions with good cryptographic properties and smaller number of variables. By means of this method for $\frac{2n-7}{3} \leq m \leq n-2$ we construct an m-resilient Boolean function of n variables with nonlinearity $2^{n-1} - 2^{m+1}$, i. e. the function that achieves the upper bound for the nonlinearity given in Section 3. The combination of this construction with upper bound gives the exact result: the maximal possible nonlinearity of n-variable m-resilient Boolean function is $2^{n-1} - 2^{m+1}$ for $\frac{2n-7}{3} \leq m \leq n-2$. This result was known only for $m = n-2$ (trivial), $m = n-3$ [8] and some small values of n. In Section 6 we strengthen the previous construction and show that for $\frac{2n-7}{3} \leq m \leq n - \log_2 \frac{n+2}{3} - 2$ it is possible to construct an n-variable m-resilient function with maximal possible nonlinearity $2^{n-1} - 2^{m+1}$ such that each variable presents in ANF of this function in some term of maximal possible length $n-m-1$ (i. e. each individual variable achieves Siegenthaler's Inequality). Note that in [15] we also discuss how to implement in hardware the functions constructed in previous sections. We suggest a concrete hardware scheme for n-

variable, m-resilient function, $n \equiv 2 \pmod 3$, $m = \frac{2n-7}{3}$, that achives a maximal possible nonlinearity and a maximal possible algebraic degree for each variable simultaneously. It is given a scheme of hardware implementation for such function. It is remarkably that this scheme has a circuit complexity **linear** on n. It contains $2n-4$ gates EXOR and $\frac{2n-1}{3}$ gates AND. This scheme has a strongly regular cascade structure and can be used efficiently in practical design. In this paper the section on implementation is omitted because of lack of space.

2 Preliminary Concepts and Notions

We consider V^n, the vector space of n tuples of elements from $GF(2)$. A *Boolean function* is a function from V^n to $GF(2)$. The *weight* $wt(f)$ of a function f on V^n is the number of vectors $\widetilde{\sigma}$ on V^n such that $f(\widetilde{\sigma}) = 1$. A function f is said to be *balanced* if $wt(f) = wt(f \oplus 1)$. Obviously, if a function f on V^n is balanced then $wt(f) = 2^{n-1}$. A *subfunction* of the Boolean function f is a function f' obtained by substituting some constants for some variables in f. If we substitute in the function f the constants $\sigma_{i_1}, \ldots, \sigma_{i_s}$ for the variables $x_{i_1}, \ldots, x_{i_s}$ respectively then the obtained subfunction is denoted by $f^{\sigma_{i_1},\ldots,\sigma_{i_s}}_{x_{i_1},\ldots,x_{i_s}}$. If a variable x_i is not substituted by a constant then x_i is called a *free* variable for f'.

It is well known that a function f on V^n can be uniquely represented by a polynomial on $GF(2)$ whose degree in each variable is at most 1. Namely, $f(x_1, \ldots, x_n) = \bigoplus\limits_{(a_1,\ldots,a_n)\in V^n} g(a_1, \ldots, a_n) x_1^{a_1} \ldots x_n^{a_n}$, where g is also a function on V^n. This polynomial representation of f is called the *algebraic normal form* (briefly, ANF) of the function and each $x_1^{a_1} \ldots x_n^{a_n}$ is called a *term* in ANF of f. The *algebraic degree* of f, denoted by $\deg(f)$, is defined as the number of variables in the longest term of f. The *algebraic degree of variable* x_i in f, denoted by $\deg(f, x_i)$, is the number of variables in the longest term of f that contains x_i. If $\deg(f, x_i) = 0$ then the variable x_i is called *fictitious* for the function f. If $\deg(f, x_i) = 1$, we say that f depends on x_i *linearly.* If $\deg(f, x_i) \geq 2$, we say that f depends on x_i *nonlinearly.* A term of length 1 is called a *linear* term. If $\deg(f) \leq 1$ then f is called an *affine* function.

The *Hamming distance* $d(\widetilde{\sigma}_1, \widetilde{\sigma}_2)$ between two vectors $\widetilde{\sigma}_1$ and $\widetilde{\sigma}_2$ is the number of components where vectors $\widetilde{\sigma}_1$ and $\widetilde{\sigma}_2$ differ. For two Boolean functions f_1 and f_2 on V^n, we define the distance between f_1 and f_2 by $d(f_1, f_2) = \#\{\widetilde{\sigma} \in V^n | f_1(\widetilde{\sigma}) \neq f_2(\widetilde{\sigma})\}$. The minimum distance between f and the set of all affine functions is called the *nonlinearity* of f and denoted by $nl(f)$.

A Boolean function f on V^n is said to be *correlation-immune of order* m, with $1 \leq m \leq n$, if the output of f and any m input variables are statistically independent. This concept was introduced by Siegenthaler [13]. In equivalent non-probabilistic formulation the Boolean function f is called correlation-immune of order m if $wt(f') = wt(f)/2^m$ for any its subfunction f' of $n-m$ variables. A balanced mth order correlation immune function is called an *m-resilient* function. In other words the Boolean function f is called m-resilient if $wt(f') = 2^{n-m-1}$ for any its subfunction f' of $n-m$ variables. From this point of view we can consider formally any balanced Boolean function as 0-resilient (this convention is accepted in [1,7,8]) and an arbitrary Boolean function as (-1)-resilient. The concept of an m-resilient function was introduced in [3].

Siegenthaler's Inequality [13] states that if the function f is a correlation-immune function of order m then $\deg(f) \le n-m$. Moreover, if f is m-resilient, $m \le n-2$, then $\deg(f) \le n-m-1$. An m-resilient Boolean function f is called *optimized* if $\deg(f) = n-m-1$ $(m \le n-2)$.

The next two lemmas are well-known.

Lemma 2.1 *Let $f(x_1,\ldots,x_n)$ be a Boolean function on V^n. Then $\deg(f) = n$ iff $wt(f)$ is odd.*

Lemma 2.2 *Let $f(x_1,\ldots,x_n)$ be a Boolean function represented in the form*

$$f(x_1,\ldots,x_n) = \bigoplus_{(\sigma_1,\ldots,\sigma_l)} (x_1 \oplus \sigma_1)\ldots(x_l \oplus \sigma_l) f(\sigma_1 \oplus 1,\ldots,\sigma_l \oplus 1, x_{l+1},\ldots,x_n).$$

Suppose that all 2^l subfunctions $f(\sigma_1 \oplus 1,\ldots,\sigma_l \oplus 1, x_{l+1},\ldots,x_n)$ are m-resilient. Then the function f is m-resilient too.

The Lemma 2.2 was proved in a lot of papers including (for $l=1$) the pioneering paper of Siegenthaler (Theorem 2 in [13]). General case follows immediately from the case $l=1$.

3 Upper Bound for the Nonlinearity of Resilient Functions

Let n and m be integers, $-1 \le m \le n$. Denote by $nlmax(n,m)$ the maximal possible nonlinearity of m-resilient Boolean function on V^n. It is well-known that the nonlinearity of a Boolean function does not exceed $2^{n-1} - 2^{\frac{n}{2}-1}$ [9]. Thus,

$$nlmax(n,-1) \le 2^{n-1} - 2^{\frac{n}{2}-1}, \tag{1}$$

This value can be achieved only for even n.

In [15,11,16] it was proved (independently) that $nlmax(n,m) \le 2^{n-1} - 2^{m+1}$. Here we give this result without the proof.

Theorem 3.1 *Let $f(x_1,\ldots,x_n)$ be an m-resilient Boolean function, $m \le n-2$. Then*

$$nl(f) \le 2^{n-1} - 2^{m+1}. \tag{2}$$

Corollary 3.1 *$nlmax(n,m) \le 2^{n-1} - 2^{m+1}$ for $m \le n-2$.*

If $m \le \frac{n}{2} - 2$ the inequality (2) does not give us any new information because of well-known inequality (1). But in the following sections we show that the inequality (2) is achieved for wide spectrum of large m.

The next theorem is proved in [15].

Theorem 3.2 *Let $f(x_1,\ldots,x_n)$ be an m-resilient nonoptimized Boolean function, $m \le n-3$. Then $nl(f) \le 2^{n-1} - 2^{m+2}$.*

Corollary 3.2 *The inequality (2) can be achieved only for optimized functions.*

Thus, the inequality (2) can be achieved only if Siegenthaler's Inequality is achieved too.

4 On Linear and Quasilinear Variables

Recall that a variable x_i is called *linear* for a function $f = f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)$ if $\deg(f, x_i) = 1$. Also we say that a function f depends on a variable x_i *linearly*. If a variable x_i is linear for a function f we can represent f in the form

$$f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) = g(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \oplus x_i.$$

Another equivalent definition of a linear variable is that a variable x_i is linear for a function f if $f(\widetilde{\delta}_1) \neq f(\widetilde{\delta}_2)$ for any two vectors $\widetilde{\delta}_1$ and $\widetilde{\delta}_2$ that differ only in ith component. By analogy with the last definition we give a new definition for a pair of quasilinear variables.

Definition 4.1 *We say that a Boolean function $f = f(x_1, \ldots, x_n)$ depends on a pair of its variables (x_i, x_j)* quasilinearly *if $f(\widetilde{\delta}_1) \neq f(\widetilde{\delta}_2)$ for any two vectors $\widetilde{\delta}_1$ and $\widetilde{\delta}_2$ of length n that differ only in ith and jth components. A pair (x_i, x_j) in this case is called a* pair of quasilinear variables *in f.*

Lemma 4.1 *Let $f(x_1, \ldots, x_n)$ be a Boolean function. Then (x_i, x_j), $i < j$, is a pair of quasilinear variables in f iff f can be represented in the form*

$$f(x_1, \ldots, x_n) = g(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n, x_i \oplus x_j) \oplus x_i. \quad (3)$$

Proof. If f is represented in the form (3) then, obviously, a pair (x_i, x_j) is quasilinear in f. Suppose that a pair (x_i, x_j) is quasilinear in f. The function f can be written in the form $f = g_1 x_i x_j \oplus g_2 x_i \oplus g_3 x_j \oplus g_4 = h(x_i, x_j)$ where g_k are functions of the remaining variables. We have $h(0,0) \neq h(1,1)$ and $h(0,1) \neq h(1,0)$. So $g_1 \oplus g_2 \oplus g_3 \neq 0$ and $g_3 \neq g_2$. Thus $g_1 \oplus g_2 \oplus g_3 = 1$ and $g_2 \oplus g_3 = 1$, so $g_1 = 0$. Also $g_2 = g_3 \oplus 1$. Therefore $f = (g_3 \oplus 1)x_i \oplus g_3 x_j \oplus g_4 = (g_3(x_i \oplus x_j) \oplus g_4) \oplus x_i$, as desired. □

Lemma 4.2 *Let $f(x_1, \ldots, x_n)$ be a Boolean function. If f depends on some variable x_i linearly then f is balanced.*

Proof. Combine all 2^n vectors of the function f into pairs so that any pair $(\widetilde{\sigma}_1, \widetilde{\sigma}_2)$ contains vectors $\widetilde{\sigma}_1$ and $\widetilde{\sigma}_2$ that differ in ith component and coincide in all other components. Then $f(\widetilde{\sigma}_1) \neq f(\widetilde{\sigma}_2)$. So, $wt(f) = 2^{n-1}$ and f is balanced. □

Corollary 4.1 *Let $f(x_1, \ldots, x_n)$ be a Boolean function. If f depends on some variables x_{i_1}, x_{i_2}, …, x_{i_s} linearly then f is $(s-1)$-resilient.*

Note that the Corollary 4.1 agrees with our assumption that a balanced function is 0-resilient, and an arbitrary Boolean function is (-1)-resilient. (In the last case $s = 0$.)

Lemma 4.3 *Let $f(x_1, \ldots, x_n)$ be a Boolean function. If f depends on some pair of variables (x_i, x_j) quasilinearly then f is balanced.*

Proof. Combine all 2^n vectors of the function g into pairs so that any pair $(\widetilde{\sigma}_1, \widetilde{\sigma}_2)$ contains vectors $\widetilde{\sigma}_1$ and $\widetilde{\sigma}_2$ that differ in ith and jth components and coincide in all other components. Then $f(\widetilde{\sigma}_1) \neq f(\widetilde{\sigma}_2)$. So, the function f is balanced. □

The next two lemmas can be proved easily.

Lemma 4.4 *Let $f(x_1,\ldots,x_n,x_{n+1}) = g(x_1,\ldots,x_n) \oplus cx_{n+1}$ where $c \in \{0,1\}$. Then $nl(f) = 2nl(g)$.*

Lemma 4.5 *Let $f(x_1,\ldots,x_n)$ be a Boolean function on V^n and f depends on some pair of variables (x_i,x_j) quasilinearly. Then $nl(f) = 2nl(g)$ where g is a function used in the representation of f in the form (3) in Lemma 4.1.*

5 A Method of Constructing

Theorem 3.1 shows that the nonlinearity of m-resilient Boolean function on V^n can not exceed $2^{n-1} - 2^{m+1}$. Earlier in papers [12,2,6,7] the authors developed methods for the constructing of m-resilient Boolean functions of n variables with high nonlinearity, and, in particular, the nonlinearity $2^{n-1} - 2^{m+1}$ in these four papers can be achieved for $m + 3 \geq 2^{n-m-2}$. The methods suggested in these papers are quite different but in the part of spectrum given by the inequality $m + 3 \geq 2^{n-m-2}$ these methods give really the same construction. Combination of these results with our upper bound (2) from Theorem 3.1 proves that $nlmax(n,m) = 2^{n-1} - 2^{m+1}$ for $m + 3 \geq 2^{n-m-2}$. In this section we prove a stronger result, namely, we prove that $nlmax(n,m) = 2^{n-1} - 2^{m+1}$ for $\frac{2n-7}{3} \leq m \leq n-2$.

Lemma 5.1 *Let n be a positive integer. Let $f_1(x_1,\ldots,x_n)$ and $f_2(y_1,\ldots, y_n)$ be m-resilient Boolean functions on V^n such that $nl(f_1) \geq N_0$, $nl(f_2) \geq N_0$. Moreover, there exist two variables x_i and x_j such that f_1 depends on the variables x_i and x_j linearly, and f_2 depends on a pair of the variables (x_i,x_j) quasilinearly. Then the function*

$$f_1'(x_1,\ldots,x_n,x_{n+1}) = (x_{n+1}\oplus 1)f_1(x_1,\ldots,x_n) \oplus x_{n+1}f_2(x_1,\ldots,x_n) \qquad (4)$$

is an m-resilient Boolean function on V^{n+1} with nonlinearity $nl(f_1') \geq 2^{n-1}+N_0$, and the function

$$\begin{aligned} f_2'(x_1,\ldots,x_n,x_{n+1},x_{n+2}) = (x_{n+1}\oplus x_{n+2}\oplus 1)f_1(x_1,\ldots,x_n)\oplus \\ (x_{n+1}\oplus x_{n+2})f_2(x_1,\ldots,x_n)\oplus x_{n+1} \end{aligned} \qquad (5)$$

is an $(m+1)$-resilient Boolean function on V^{n+2} with nonlinearity $nl(f_2') \geq 2^n + 2N_0$. Moreover, f_2' depends on a pair of the variables (x_{n+1},x_{n+2}) quasilinearly.

Proof. At first, consider the equation (4). Both subfunctions $(f_1')^0_{x_{n+1}} = f_1(x_1,\ldots,x_n)$ and $(f_1')^1_{x_{n+1}} = f_2(x_1,\ldots,x_n)$ are m-resilient, hence by Lemma 2.2 f_1' is m-resilient too. Let $l = \bigoplus_{i=1}^{n+1} c_ix_i \oplus c_0$ be an arbitrary affine function on V^{n+1}. Then $d(f_1',l) = d(f_1,l^0_{x_{n+1}}) + d(f_2,l^1_{x_{n+1}}) = wt(f_1\oplus l^0_{x_{n+1}}) + wt(f_2\oplus l^1_{x_{n+1}})$. We state that at least one of two functions $f_1\oplus l^0_{x_{n+1}}$ and $f_2\oplus l^1_{x_{n+1}}$ is balanced. Indeed, if $c_i = 0$ or $c_j = 0$ then the function $f_1\oplus l^0_{x_{n+1}}$ depends on x_i or x_j linearly, hence, by Lemma 4.2 the function $f_1\oplus l^0_{x_{n+1}}$ is balanced. In the remained case $c_i = 1$ and $c_j = 1$ it is easy to see from the representation (3) that the function $f_2\oplus l^1_{x_{n+1}}$ depends on a pair of the variables (x_i,x_j) quasilinearly, therefore by Lemma 4.3 the function $f_2 \oplus l^1_{x_{n+1}}$ is balanced. Thus, $d(f_1', l) \geq 2^{n-1} + N_0$. An affine function l was chosen arbitrary, therefore, $nl(f_1') \geq 2^{n-1} + N_0$.

Next, consider the equation (5). By conctruction (5) and representation (3) we see that f_2' depends on a pair of the variables (x_{n+1}, x_{n+2}) quasilinearly. Now we want to prove that the function f_2' is $(m+1)$-resilient. Substitute arbitrary $m+1$ variables by constants generating the subfunction $\hat{f}$. If both variables x_{n+1} and x_{n+2} are free in $\hat{f}$ then $\hat{f}$ depends on a pair (x_{n+1}, x_{n+2}) quasilinearly, therefore by Lemma 4.3 the function $\hat{f}$ is balanced. If at least one of two variables x_{n+1} and x_{n+2} was substituted by constant then we substituted by constants at most m of first n variables $x_1, \ldots, x_n$. But the functions $\hat{f}^{0}_{x_{n+1},}{}^{0}_{x_{n+2}} = f_1$, $\hat{f}^{0}_{x_{n+1},}{}^{1}_{x_{n+2}} = f_2$, $\hat{f}^{1}_{x_{n+1},}{}^{0}_{x_{n+2}} = f_2 \oplus 1$, $\hat{f}^{1}_{x_{n+1},}{}^{1}_{x_{n+2}} = f_1 \oplus 1$ are m-resilient, thus, by Lemma 2.2 the function $\hat{f}$ is balanced. A subfunction $\hat{f}$ was chosen arbitrary. So, the function f_2' is $(m+1)$-resilient.

Finally, we need to prove the lower bound for the nonlinearity of f_2'. Let $l = \bigoplus_{i=1}^{n+2} c_i x_i \oplus c_0$ be an arbitrary affine function on V^{n+2}. Then $d(f_2', l) = d(f_1, l^{0}_{x_{n+1},}{}^{0}_{x_{n+2}}) + d(f_2, l^{0}_{x_{n+1},}{}^{1}_{x_{n+2}}) + d(f_2 \oplus 1, l^{1}_{x_{n+1},}{}^{0}_{x_{n+2}}) + d(f_1 \oplus 1, l^{1}_{x_{n+1},}{}^{1}_{x_{n+2}}) = wt(f_1 \oplus l^{0}_{x_{n+1},}{}^{0}_{x_{n+2}}) + wt(f_2 \oplus l^{0}_{x_{n+1},}{}^{1}_{x_{n+2}}) + wt(f_2 \oplus l^{1}_{x_{n+1},}{}^{0}_{x_{n+2}} \oplus 1) + wt(f_1 \oplus l^{1}_{x_{n+1},}{}^{1}_{x_{n+2}} \oplus 1)$. By the same reason as it was given above at least one of two functions $f_1 \oplus l^{0}_{x_{n+1},}{}^{0}_{x_{n+2}}$ and $f_2 \oplus l^{0}_{x_{n+1},}{}^{1}_{x_{n+2}}$ is balanced, and at least one of two functions $f_2 \oplus l^{1}_{x_{n+1},}{}^{0}_{x_{n+2}} \oplus 1$ and $f_1 \oplus l^{1}_{x_{n+1},}{}^{1}_{x_{n+2}} \oplus 1$ is balanced. Thus, $d(f_2', l) \geq 2^n + 2N_0$. An affine function l was chosen arbitrary, therefore, $nl(f_2') \geq 2^n + 2N_0$. □

Lemma 5.2 *Suppose that there exist an m-resilient Boolean function $f_{n,1}$ on V^n, $nl(f_{n,1}) \geq N_0$, and $(m+1)$-resilient Boolean function $f_{n+1,2}$ on V^{n+1}, $nl(f_{n+1,2}) \geq 2N_0$, besides the function $f_{n+1,2}$ depends on some pair of its variables (x_i, x_j) quasilinearly. Then there exist an $(m+2)$-resilient Boolean function $f_{n+3,1}$ on V^{n+3}, $nl(f_{n+3,1}) \geq 2^{n+1} + 4N_0$, and $(m+3)$-resilient Boolean function $f_{n+4,2}$ on V^{n+4}, $nl(f_{n+4,2}) \geq 2^{n+2} + 8N_0$, besides the function $f_{n+4,2}$ depends on some pair of its variables quasilinearly.*

Proof. We can assume that $i < j$. Denote

$$f_1(x_1, \ldots, x_{n+2}) = f_{n,1}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{j-1}, x_{j+1}, \ldots, x_{n+2}) \oplus x_i \oplus x_j,$$

$$f_2(x_1, \ldots, x_{n+2}) = f_{n+1,2}(x_1, \ldots, x_{n+1}) \oplus x_{n+2}.$$

By Lemmas 4.2 and 4.4 the functions f_1 and f_2 are $(m+2)$-resilient functions on V^{n+2}, $nl(f_1) \geq 4N_0$, $nl(f_2) \geq 4N_0$. Moreover, f_1 depends on the variables x_i and x_j linearly, and f_2 depends on a pair of the variables (x_i, x_j) quasilinearly. Substituting f_1 and f_2 to (4) and (5) (we shift $n \to n+2$) we have

$$f_1'(x_1, \ldots, x_n, x_{n+3}) = (x_{n+3} \oplus 1) f_1(x_1, \ldots, x_{n+2}) \oplus x_{n+3} f_2(x_1, \ldots, x_{n+2})$$

and

$$f_2'(x_1, \ldots, x_n, x_{n+4}) = (x_{n+3} \oplus x_{n+4} \oplus 1) f_1(x_1, \ldots, x_{n+2}) \oplus$$

$$(x_{n+3} \oplus x_{n+4}) f_2(x_1, \dots, x_{n+2}) \oplus x_{n+3}.$$

By Lemma 5.1 we have constructed an $(m+2)$-resilient Boolean function $f_{n+3,1} = f'_1$ on V^{n+3}, $nl(f_{n+3,1}) \geq 2^{n+1} + 4N_0$, and an $(m+3)$-resilient Boolean function $f_{n+4,2} = f'_2$ on V^{n+4}, $nl(f_{n+4,2}) \geq 2^{n+2} + 8N_0$, besides the function $f_{n+4,2}$ depends on a pair of its variables (x_{n+3}, x_{n+4}) quasilinearly. □

Corollary 5.1 *Suppose that for $m \leq n-2$ there exist an m-resilient Boolean function $f_{n,1}$ on V^n, $nl(f_{n,1}) = 2^{n-1} - 2^{m+1}$, and $(m+1)$-resilient Boolean function $f_{n+1,2}$ on V^{n+1}, $nl(f_{n+1,2}) = 2^n - 2^{m+2}$, besides the function $f_{n+1,2}$ depends on some pair of its variables (x_i, x_j) quasilinearly. Then there exist an $(m+2)$-resilient Boolean function $f_{n+3,1}$ on V^{n+3}, $nl(f_{n+3,1}) = 2^{n+2} - 2^{m+3}$, and $(m+3)$-resilient Boolean function $f_{n+4,2}$ on V^{n+4}, $nl(f_{n+4,2}) = 2^{n+3} - 2^{m+4}$, besides the function $f_{n+4,2}$ depends on some pair of its variables quasilinearly.*

Proof. The hypothesis of Corollary 5.1 is the hypothesis of Lemma 5.2 for $N_0 = 2^{n-1} - 2^{m+1}$. By Lemma 5.2 we can construct the functions $f_{n+3,1}$ and f_{n+4} with required properties and nonlinearities $nl(f_{n+3,1}) \geq 2^{n+1} + 4N_0 = 2^{n+2} - 2^{m+3}$, $nl(f_{n+4,2}) \geq 2^{n+2} + 8N_0 = 2^{n+3} - 2^{m+4}$. By Theorem 3.1 the right parts of the last inequalities are also upper bounds. So, we have equalities $nl(f_{n+3,1}) = 2^{n+2} - 2^{m+3}$, $nl(f_{n+4,2}) = 2^{n+3} - 2^{m+4}$. □

Theorem 5.1 *$nlmax(n, m) = 2^{n-1} - 2^{m+1}$ for $\frac{2n-7}{3} \leq m \leq n-2$.*

Proof. If $m = n-2$ then by Siegenthaler's Inequality any $(m-2)$-resilient function on V^n is affine. So, $nlmax(n, n-2) = 0$. Next, take $f_{2,1} = x_1 x_2$, $f_{3,2} = x_1(x_2 \oplus x_3) \oplus x_2$. These functions satisfy to the hypothesis of Corollary 5.1 with $n = 2$, $m = -1$. By Corollary 5.1 we construct the functions $f_{5,1}$ and $f_{6,2}$ such that the function $f_{5,1}$ is an 1-resilient Boolean function on V^5, $nl(f_{5,1}) = 2^4 - 2^2$, the function $f_{6,2}$ is a 2-resilient Boolean function on V^6, $nl(f_{6,2}) = 2^5 - 2^3$, besides $f_{6,2}$ depends on a pair of the variables (x_5, x_6) quasilinearly. Substitute the functions $f_{5,1}$ and $f_{6,2}$ to the hypothesis of Corollary 5.1, and so on. By this way, for each integer k, $k \geq 3$, we construst an m-resilient Boolean function $f_{n,1}$ on V^n with nonlinearity $2^{n-1} - 2^{m+1}$ where $n = 3k-7$, $m = 2k-7$. Let $\frac{2n-7}{3} \leq m \leq n-3$. Put $f(x_1, \dots, x_n) = f_{3(n-m)-7,1}(x_1, \dots, x_{3(n-m)-7}) \bigoplus\limits_{i=3(n-m)-6}^{n} x_i$. By the hypothesis of Theorem 5.1 we have $3(n-m)-7 \leq n$. The resiliency of the function f is $(2(n-m)-7) + (n-(3(n-m)-7)) = m$, the nonlinearity of the function f is $2^{n-(3(n-m)-7)} \left(2^{(3(n-m)-7)-1} - 2^{(2(n-m)-7)+1}\right) = 2^{n-1} - 2^{m+1}$. Thus, for $\frac{2n-7}{3} \leq m \leq n-2$ we have constructed an m-resilient Boolean function on V^n with nonlinearity $2^{n-1} - 2^{m+1}$. Taking into account the upper bound (2) from Theorem 3.1 we complete the proof. □

Note that a recent conjecture $nlmax(n, n-4) = 2^{n-1} - 2^{n-3}$ (for $n \geq 5$) in [8] is a special case of our Theorem 5.1.

6 Optimization of Siegenthaler's Inequality for Each Individual Variable

Some lack of the construction given in the proof of Theorem 5.1 is that for $\frac{2n-7}{3} < m$ the constructed function depends on some variables linearly. Note that the functions with the nonlinearity $2^{n-1} - 2^{m+1}$ constructed in [12,2,6,7] (for $m+3 \geq 2^{n-m-2}$) depends nonlinearly on all its variables only in some cases when $m+3 = 2^{n-m-2}$ or $m+2 = 2^{n-m-2}$. In general, those functions depends nonlinearly on $2^{n-m-2} + n - m - 4$ or $2^{n-m-2} + n - m - 3$ variables. In this section for $\frac{2n-7}{3} \leq m \leq n - \log_2 \frac{n+2}{3} - 2$ we suggest a method to construct an m-resilient Boolean function on V^n that achieves Siegenthaler's Inequality for each its individual variable (i. e. $\deg(f, x_i) = n - m - 1$ for all variables x_i). Simultaneously we give a more general way of constructing than it was done in previous section.

We say that a variable x_i is a *covering* for a function f if each other variable of f is contained together with x_i in some term of maximal length in ANF of f. We say that a quasilinear pair of variables (x_i, x_j) is a *covering* for a function f if each other variable of f is contained together with x_i in some term of maximal length in ANF of f (and consequently together with x_j in some term of maximal length in ANF of f).

Lemma 6.1 *For integers k and n provided $k \geq 3$, $3k-7 \leq n < 3 \cdot 2^{k-2} - 2$, there exists a Boolean function $f^k_{n,1}$ on V^n satisfying to the next properties:*

(1 i) $f^k_{n,1}$ is $(n-k)$-resilient;

(1 ii) $nl(f^k_{n,1}) = 2^{n-1} - 2^{n-k+1}$;

(1 iii) $\deg(f^k_{n,1}, x_i) = k-1$ for each variable x_i;

(1 iv) $f^k_{n,1}$ has a covering variable.

For integers k and n provided $k \geq 3$, $3k-7 < n \leq 3 \cdot 2^{k-2} - 2$, there exists a Boolean function $f^k_{n,2}$ on V^n satisfied to the next properties:

(2 i) $f^k_{n,2}$ is an $(n-k)$-resilient;

(2 ii) $nl(f^k_{n,2}) = 2^{n-1} - 2^{n-k+1}$;

(2 iii) $\deg(f^k_{n,2}, x_i) = k-1$ for each variable x_i;

(2 iv) $f^k_{n,2}$ has a quasilinear pair of covering variables.

Proof. The proof is by induction on k. For $k = 3$ we can take $f^3_{2,1} = x_1x_2$, $f^3_{3,1} = f^3_{3,2} = x_1(x_2 \oplus x_3) \oplus x_2$, $f^3_{4,2} = (x_1 \oplus x_2)(x_3 \oplus x_4) \oplus x_1 \oplus x_3$. It is easy to check that these functions satisfy to all required conditions.

Suppose that the statement is valid for k. We want to prove it for $k+1$. We search the functions $f^{k+1}_{n,1}$ and $f^{k+1}_{n,2}$ in the form

$$\begin{gathered} f^{k+1}_{n,1} = (x_n \oplus 1)\left(f^k_{n_1}(x_1, \ldots, x_{n_1}) \bigoplus_{i=n_1+1}^{n-1} x_i\right) \\ \oplus x_n \left(\bigoplus_{i=1}^{n-1-n_2} x_i \oplus f^k_{n_2,2}(x_{n-n_2}, \ldots, x_{n-1})\right), \\ n_1 + n_2 \geq n-1, \quad n_1 \leq n-3, \quad n_2 \leq n-2, \end{gathered} \tag{6}$$

and

$$\begin{gathered} f^{k+1}_{n,2} = (x_{n-1} \oplus x_n \oplus 1)\left(f^k_{n_1}(x_1, \ldots, x_{n_1}) \bigoplus_{i=n_1+1}^{n-2} x_i\right) \\ \oplus (x_{n-1} \oplus x_n)\left(\bigoplus_{i=1}^{n-2-n_2} x_i \oplus f^k_{n_2,2}(x_{n-n_2-1}, \ldots, x_{n-2})\right) \oplus x_{n-1}, \\ n_1 + n_2 \geq n-2, \quad n_1 \leq n-4, \quad n_2 \leq n-3, \end{gathered} \tag{7}$$

where $f^k_{n_1}(x_1,\dots,x_{n_1})$ is $f^k_{n_1,1}(x_1,\dots,x_{n_1})$ or $f^k_{n_1,2}(x_1,\dots,x_{n_1})$ (if $f^k_{n_1}=f^k_{n_1,2}$ then $n_2\neq n-2$ in (6) and $n_2\neq n-3$ in (7)). Besides we suppose that a covering variable in $f^k_{n_1}$ is x_1 (or a quasilinear pair of covering variables in $f^k_{n_1,2}$ is (x_1,x_2)), and we suppose that a quasilinear pair of covering variables in $f^k_{n_2,2}$ is (x_{n-2},x_{n-1}) in (6) or (x_{n-3},x_{n-2}) in (7).

The functions $f^{k+1}_{n,1}$ and $f^{k+1}_{n,2}$ satisfy to all required properties. Indeed:

(1 i) The resiliency of the function $f^k_{n_1}(x_1,\dots,x_{n_1})\bigoplus\limits_{i=n_1+1}^{n-1}x_i$ is $(n_1-k)+(n-1-n_1)=n-k-1$, the resiliency of the function $\bigoplus\limits_{i=1}^{n-1-n_2}x_i\oplus f^k_{n_2,2}(x_{n-n_2},\dots,x_{n-1})$ is $n-1-n_2+(n_2-k)=n-k-1$. So, by Lemma 5.1 the resiliency of the function $f^{k+1}_{n,1}$ is $n-(k+1)$.

(2 i) The resiliency of the function $f^k_{n_1}(x_1,\dots,x_{n_1})\bigoplus\limits_{i=n_1+1}^{n-2}x_i$ is $(n_1-k)+(n-2-n_1)=n-k-2$, the resiliency of the function $\bigoplus\limits_{i=1}^{n-2-n_2}x_i\oplus f^k_{n_2,2}(x_{n-n_2-1},\dots,x_{n-2})$ is $n-2-n_2+(n_2-k)=n-k-2$. So, by Lemma 5.1 the resiliency of the function $f^{k+1}_{n,1}$ is $n-(k+1)$.

(1 ii) The nonlinearity of the function $f^k_{n_1}(x_1,\dots,x_{n_1})\bigoplus\limits_{i=n_1+1}^{n-1}x_i$ is $(2^{n_1-1}-2^{n_1-k+1})2^{n-1-n_1}=2^{n-2}-2^{n-k}$, the nonlinearity of the function $\bigoplus\limits_{i=1}^{n-1-n_2}x_i\oplus f^k_{n_2,2}(x_{n-n_2},\dots,x_{n-1})$ is $2^{n-1-n_2}(2^{n_2-1}-2^{n_2-k+1})=2^{n-2}-2^{n-k}$. The function $f^k_{n_1}(x_1,\dots,x_{n_1})\bigoplus\limits_{i=n_1+1}^{n-1}x_i$ depends on variables x_{n-2} and x_{n-1} linearly whereas the function $\bigoplus\limits_{i=1}^{n-1-n_2}x_i\oplus f^k_{n_2,2}(x_{n-n_2},\dots,x_{n-1})$ depends on a pair of variables (x_{n-2},x_{n-1}) quasilinearly. So, by Lemma 5.1 the nonlinearity of the function $f^{k+1}_{n,1}$ is $2^{n-2}+(2^{n-2}-2^{n-k})=2^{n-1}-2^{n-(k+1)+1}$.

(2 ii) The nonlinearity of the function $f^k_{n_1}(x_1,\dots,x_{n_1})\bigoplus\limits_{i=n_1+1}^{n-2}x_i$ is $(2^{n_1-1}-2^{n_1-k+1})2^{n-2-n_1}=2^{n-3}-2^{n-k-1}$, the nonlinearity of the function $\bigoplus\limits_{i=1}^{n-2-n_2}x_i\oplus f^k_{n_2,2}(x_{n-n_2-1},\dots,x_{n-2})$ is equal to $2^{n-2-n_2}(2^{n_2-1}-2^{n_2-k+1})=2^{n-3}-2^{n-k-1}$. The function $f^k_{n_1}(x_1,\dots,x_{n_1})\bigoplus\limits_{i=n_1+1}^{n-2}x_i$ depends on variables x_{n-3} and x_{n-2} linearly whereas the function $\bigoplus\limits_{i=1}^{n-1-n_2}x_i\oplus f^k_{n_2,2}(x_{n-n_2},\dots,x_{n-1})$ depends on a pair of variables (x_{n-3},x_{n-2}) quasilinearly. So, by Lemma 5.1 the nonlinearity of the function $f^{k+1}_{n,2}$ is $2^{n-2}+2(2^{n-3}-2^{n-k-1})=2^{n-1}-2^{n-(k+1)+1}$.

(1 iii), (1 iv) Each variable from the set $\{x_2,x_3,\dots,x_{n_1}\}$ is contained together with x_1 in some term of length $k-1$ in ANF of the function $f^k_{n_1,1}(x_1,\dots,x_{n_1})$ if

$f^k_{n_1} = f^k_{n_1,1}$ or each variable from the set $\{x_3, x_4, \ldots, x_{n_1}\}$ is contained together with x_1 in some term of length $k-1$ (and also together with x_2 in some term of this length) in ANF of the function $f^k_{n_1,2}(x_1, \ldots, x_{n_1})$ if $f^k_{n_1} = f^k_{n_1,2}$. The function $\bigoplus\limits_{i=1}^{n-1-n_2} x_i \oplus f^k_{n_2,2}(x_{n-n_2}, \ldots, x_{n-1})$ depends on the variable x_1 linearly (and also on the variable x_2 if $f^k_{n_1} = f^k_{n_1,2}$). So, after the removing of the parentheses and the reducing of similar terms each variable from the set $\{x_1, x_2, x_3, \ldots, x_{n_1}\}$ will be contained together with x_n in some term of length k in ANF of the function $f^{k+1}_{n,1}$. Analogously, each variable from the set $\{x_{n-n_2}, \ldots, x_{n-3}\}$ is contained together with x_{n-2} in some term of length $k-1$ (and also together with x_{n-1} in some term of such length) in ANF of the function $f^k_{n_2,2}(x_{n-n_2}, \ldots, x_{n-1})$. The function $f^k_{n_1}(x_1, \ldots, x_{n_1}) \bigoplus\limits_{i=n_1+1}^{n-1} x_i$ depends on the variables x_{n-2} and x_{n-1} linearly. So, after the removing of the parentheses and the reducing of similar terms each variable from the set $\{x_{n-n_2}, \ldots, x_{n-1}\}$ will be contained together with x_n in some term of length k in ANF of the function $f^{k+1}_{n,1}$. By condition $n_1 + n_2 \leq n-1$, therefore the union of the sets $\{x_1, x_2, x_3, \ldots, x_{n_1}\}$ and $\{x_{n-n_2}, \ldots, x_{n-1}\}$ is the set $\{x_1, \ldots, x_{n-1}\}$. Thus, x_n is a covering variable in $f^k_{n,1}$. The proof of properties (2 iii) and (2 iv) is analogous.

Finally, we note that according to (6) we can construct the function $f^k_{n,1}$ if $n \geq n_1 + 3 \geq (3k-7) + 3 = 3(k+1) - 7$ and if $n \leq n_1 + n_2 + 1 \leq 2(3 \cdot 2^{k-2} - 2) + 1 \leq 3 \cdot 2^{(k+1)-2} - 3$, and according to (7) we can construct the function $f^k_{n,2}$ if $n \geq n_1 + 4 \geq (3k-7) + 4 = 3(k+1) - 4$ and if $n \leq n_1 + n_2 + 2 \leq 2(3 \cdot 2^{k-2} - 2) + 2 \leq 3 \cdot 2^{(k+1)-2} - 2$. So, the step of induction is proven. □

Theorem 6.1 *For integers m and n provided $\frac{2n-7}{3} \leq m \leq n - \log_2 \frac{n+2}{3} - 2$, there exists an m-resilient Boolean function on V^n with nonlinearity $2^{n-1} - 2^{m+1}$ that achieves Siegenthaler's Inequality for each individual variable.*

Proof. Straightforword corollary from Lemma 6.1. □

References

1. P. Camion, C. Carlet, P. Charpin, N. Sendrier, On correlation-immune functions, Advances in Cryptology: Crypto '91, Proceedings, Lecture Notes in Computer Science, V. 576, 1991, pp. 86–100. 22
2. Seongtaek Chee, Sangjin Lee, Daiki Lee and Soo Hak Sung, On the Correlation Immune Functions and their Nonlinearity, Advances in Cryptology - Asiacrypt '96, Lecture Notes in Computer Science, V. 1163, 1996, pp. 232–243. 20, 24, 27
3. B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich, R. Smolensky, The bit extraction problem or t-resilient functions, IEEE Symposium on Foundations of Computer Science, V. 26, 1985, pp. 396–407. 22
4. T. W. Cusick, On constructing balanced correlation immune functions, in Sequences and Their Applications, Proceedings of SETA '98, Springer Discrete Mathematics and Theoretical Computer Science, 1999, pp. 184-190. 20
5. E. Filiol, C. Fontaine, Highly Nonlinear Balanced Boolean Functions with a Good Correlation Immunity, Advanced in Cryptology, Eurocrypt '98, Helsinki, Finland, Lecture Notes in Computer Sciences, Vol. 1403, 1998, pp. 475–488. 20

6. S. Maitra, P. Sarkar, Highly nonlinear resilient functions optimizing Siegenthaler's Inequality, Crypto '99, Lecture Notes in Computer Science, Vol. 1666, 1999, pp. 198–215. 20, 24, 27
7. S. Maitra, P. Sarkar, Construction of nonlinear resilient Boolean functions, Indian Statistical Institute, Technical Report No. ASD/99/30, 19 pp. 20, 22, 24, 27
8. E. Pasalic, T. Johansson, Further results on the relation between nonlinearity and resiliency for Boolean functions, IMA Conference on Cryptography and Coding, Lecture Notes in Computer Science, Vol. 1746, 1999, pp. 35–44. 20, 20, 22, 26
9. O. S. Rothaus, On bent functions, Journal of Combinatorial Theory, Series A20, pp. 300–305. 20, 22
10. P. Sarkar, S. Maitra, Construction of nonlinear Boolean functions with important cryptographic properties, In Advanced in Cryptology: Eurocrypt 2000, Lecture Notes in Computer Science, V. 1807, 2000, pp. 485–506.
11. P. Sarkar, S. Maitra, Nonlinearity bounds and constructions of resilient Boolean functions, In Advanced in Cryptology: Crypto 2000, Proceedings, Lecture Notes in Computer Science, V. 1880, 2000, pp. 515–532. 20, 22
12. J. Seberry, X. Zhang, Y. Zheng, On Constructions and Nonlinearity of Correlation Immune Functions, Advances in Cryptology, Eurocrypt '93, Proceedings, Lecture Notes in Computer Science, V. 765, 1993, pp. 181–199. 20, 24, 27
13. T. Siegenthaler, Correlation-immunity of nonlinear combining functions for cryptographic applications, IEEE Transactions on Information theory, V. IT-30, No 5, 1984, p. 776–780. 20, 21, 22, 22
14. T. Siegenthaler, Decrypting a Class of Stream Ciphers Using Ciphertext Only, IEEE Transactions on Computer, V. C-34, No 1, Jan. 1985, pp. 81–85. 19
15. Yu. Tarannikov, On resilient Boolean functions with maximal possible nonlinearity, Cryptology ePrint archive (http://eprint.iacr.org/), Report 2000/005, March 2000, 18 pp. 20, 20, 20, 22, 22
16. Y. Zheng, X.-M. Zhang, Improving upper bound on nonlinearity of high order correlation immune functions, to appear in SAC 2000, Lecture Notes in Computer Science, Springer Verlag, 2000. 20, 22

Decimation Attack of Stream Ciphers[⋆]

Eric Filiol[⋆⋆]

INRIA, projet CODES, Domaine de Voluceau
78153 Le Chesnay Cedex, FRANCE
Eric.Filiol@inria.fr

Abstract. This paper presents a new attack called *Decimation Attack* of most stream ciphers. It exploits the property that multiple clocking (or equivalently d-th decimation) of a LFSR can simulate the behavior of many other LFSRs of possible shorter length. It yields then significant improvements of all the previous known correlation and fast correlation attacks. A new criterion on the length of the polynomial is then defined to resist to the decimation attack. Simulation results and complexity comparison are detailed for ciphertext only attacks.

Keywords: stream cipher, linear feedback shift register, correlation attack, fast correlation attack, sequence decimation, multiple clocking.

1 Introduction

Despite growing importance of block ciphers, stream ciphers remain a very important class of cipher systems mainly used by governmental world.

In a binary additive stream cipher, the ciphertext is obtained by bitwise addition of the plaintext to a pseudo-random sequence called the *running key*. This latter is produced by a pseudo-random generator whose initial state constitutes the secret key. Most real-life designs center around *Linear Feedback Shift-Register* (LFSR) combined by a nonlinear Boolean function. Different variant exist: clock-controlled systems, filter generators, multiplexed systems, memory combiners, decimated generators,... This paper will focus on the most common class of combination generators depicted in Figure 1.

The cryptanalyst's problem often deals with that of recovering the initial states of some LFSRs, assuming that the structure of the generator is known to him. State of the art in generic stream ciphers cryptanalysis can be summarized as follows: correlation and fast correlation attacks. Both exploit an existing correlation (of order k) between the running key σ and some linear combination of the k input variables $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$. A so-called *divide and conquer attack* is conducted which consists to try to recover the initial state of the k target LFSRs independently of the other unknown key bits. Such correlations always

[⋆] This work was supported by Conseil Régional de Bretagne grant A0C271

[⋆⋆] also Ecoles Militaires de Saint-Cyr Coëtquidan DGER/CRECSC/DSI 56381 Guer Cedex, FRANCE *efiliol@mailhost.esm-stcyr.terre.defense.gouv.fr*

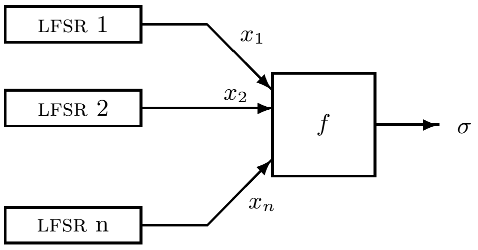

Fig. 1. Nonlinear combination generator

exist but functions offering a good cryptographic resistance generally offer a high correlation order k thus imposing on the cryptanalyst to consider simultaneously several LFSRs [7]. In this case it is obvious that the k distinct LFSRs can be seen as a single LFSR of length L; the length and the feedback polynomial P of this LFSR can be derived from the feedback polynomials of the constituent polynomials [17]. In the following we will generalize by speaking of the single target LFSR of length L. A single LFSR will then be a particular case for $k = 1$.

– In correlation attacks [18,19], the $2^L - 1$ possible initializations of the LFSR are exhaustively tried and each time its corresponding produced sequence x is compared to the captured running key σ. The initialization yielding the closest awaited statistical bias is supposed to be the correct one. This attack is limited by the length L of the target LFSR (by now $L \approx 50$) since it requires $2^L - 1$ trials and practical attacks can deal with only correlation order $k = 1$ (i.e. considering only one LFSR). Moreover the longer the LFSR is and the lower the correlation value, the longer the necessary running key sequence will be. This attack is nowadays no longer efficient except for weak schemes.
– Fast correlation attacks were introduced by Meier and Staffelbach [13] and avoid examining all possible initializations of the LFSR. The output of the target LFSR is considered to have passed through a noisy channel, most frequently modelled by the Binary (Memoryless) Symmetric Channel, BSC with some error probability $p < \frac{1}{2}$, where $\epsilon = \frac{1}{2} - p$ is usually very small. In this setting, an LFSR output sequence of fixed length N can be considered as a $[N, L]$ binary linear code. Each of its codewords can be uniquely related to a possible initialization. Then the cryptanalyst's problem becomes a decoding problem in the presence of a BSC with strong noise. Meier and Staffelbach attack uses iterative decoding process for low-density parity-check codes when feedback polynomial is of low-weight. Minor improvements have then been obtained [4,14,15]. Johansson and Jönsson recently presented new powerful ideas by considering convolutional codes [9] and turbo-codes [10]. Canteaut and Trabbia in [3] show that Gallager iterative decoding [8] with parity-check equations of weight 4 and 5 is usually more efficient than these previous attacks since it successfully decodes very high error probabilities with a relatively feasible time and memory complexity. Finally, Chepyzhov, Johansson

and Smeets [5] recently present a new, powerful fast correlation attack allowing significant reduction of memory requirements and complexity. The key idea is to use a code of dimension $K < L$ to improve decoding complexity.

All these attacks have the length L of the target LFSR as major limitation (particularly when the correlation order k is high). As soon as L increases too much and ϵ is too small, the memory and complexity requirements explode, making these attacks unfeasible. This paper presents how to possibly bypass this limitation. By considering d-fold clocking (or d-th decimation) of the target LFSR output sequence, we show how to use a simulated shorter LFSR thus improving the known attacks. This approach is particularly efficient when dealing with long LFSRs or with combining functions of high correlation order of real-life designs. With a relatively longer sequence we suppress the memory requirements and significantly reduce the complexity of the Canteaut/Trabbia attack [3]) and obtain a slightly better complexity than Chepyzhov, Johansson and Smeets attack. Moreover feasibility of this attack is determined only by the LFSR length and not the feedback polynomial, that however must be known (directly or after reconstruction [1]). We will only consider ciphertext only attack to make this attack more realistic.

This paper is organized as follows. Section 2 presents the theoretical tools we use in this attack. In Section 3, the Decimation Attack (DA) itself is described and simulation results for different cases are given. A new criterion is then defined allowing to choose LFSRs resisting this attack. Section 4 compares decimation attack with the two best known attacks of [3,5].

2 Decimation of LFSR Sequences

A linear feedback shift register of length L is characterized by L binary connection coefficients $(p_i)_{1\leq i\leq L}$. It associates to any L-bit initialization $(s_t)_{1\leq t\leq L}$ a sequence $(s_t)_{t>0}$ defined by the L-th order linear recurrence relation

$$s_{t+L} = \sum_{i=1}^{L} p_i s_{t+L-i}, \quad t \geq 0 \ .$$

The connection coefficients are usually represented by a univariate polynomial P over $\mathbf{F}_2$, called *the feedback polynomial*:

$$P(X) = 1 + \sum_{i=1}^{L} p_i X^i \ .$$

Most applications use a primitive feedback polynomial to ensure that the periods of all sequences produced by the LFSR are maximal.

Let us consider a sequence $\sigma = \sigma_1, \sigma_2, \ldots,$ produced by a LFSR of length L whose feedback polynomial is irreducible in $GF(q)[X]$. Suppose now that we operate a sampling on σ at intervals of d clock cycles (d-fold clocking) thus producing a subsequence $\rho = \rho_1, \rho_2, \ldots$. In other words, it is equivalent to the d-decimation of the original sequence σ. Thus we have $\rho_i = \sigma_{dj}$ for $j = 0, 1, 2, \ldots$ or in sequence notation $\rho = \sigma[d]$.

With d-fold clocking, the original LFSR will behave like a different LFSR which is called the *simulated LFSR* [16]. The interesting question is to determine its properties, especially relatively to the original LFSR. They are summarized in the following proposition.

Proposition 1 *[16, page 146] Let σ be the sequence produced by the original LFSR whose feedback polynomial $P(x)$ is irreducible in $GF(q)$ of degree L. Let α be a root of $P(x)$ and let T be the period of $P(x)$. Let ρ the sequence resulting from the d-th decimation of σ, i.e. $\rho = \sigma[d]$. Then the simulated LFSR, that is the LFSR directly producing ρ has the following properties:*

1. *The feedback polynomial $P^*(x)$ of the simulated LFSR is the minimum polynomial of α^d in $GF(q^L)$.*
2. *The period T^* of $P^*(x)$ is equal to $\frac{T}{\gcd(d,T)}$.*
3. *The degree L^* of $P^*(x)$ is equal to the multiplicative order of q in $\mathbf{Z}_{T^*}$.*

Moreover all d in C_k where $C_k = \{k, kq, kq^2, \ldots, \}$ mod T denotes the cyclotomic coset of k modulo T, result in the same simulated LFSR, except for different initial contents. Finally, every sequence producible by the simulated LFSR is equal to $\sigma[d]$ for some choice of the initial contents of the original LFSR.

In real case applications, $P(x)$ is primitive so $T = 2^L - 1$. Thus when T is not prime, by a careful choice of d such that $\gcd(2^L - 1, d) \neq 1$ one may expect to obtain a simulated LFSR shorter than the original one.

Example 1 *Let us consider the original LFSR with $P(x) = X^4 + X + 1$.*

– *$d \in C_3 = \{3, 6, 9, 12\}$, $T^* = 5$ and $L^* = 4$ since the multiplicative order of 2 in $\mathbf{Z}_5$ is 4.*
– *$d \in C_5 = \{5, 10\}$, $T^* = 3$ and $L^* = 2$ since the multiplicative order of 2 in $\mathbf{Z}_3$ is 2 (and $P^*(x) = X^2 + X + 1$).*

The feedback polynomial $P^*(x)$ of the simulated LFSR can be obtained by applying the Berlekamp-Massey LFSR synthesis algorithm [12] to the sequence ρ.

Finally, we recall that a combination generator is vulnerable to correlation attacks [19] if the output of the combining function statistically depends on one of its inputs. More generally, Siegenthaler [18] introduced the following criterion:

Definition 1 *A Boolean function is t-*th order correlation-immune *(denoted $CI(t)$) if the probability distribution of its output is unaltered when any t input variables are fixed.*

This property equivalently asserts that the output of f is statistically independent of any linear combination of t input variables. If f is $CI(t)$ then f if $CI(k)$ for any $k \leq t$ as well. Then the correlation-immunity order of a function f is the highest integer t such that f is $CI(t)$. Equivalently f is said $(t+1)$-th order correlated. Practically speaking, if f is $CI(t)$, then to exploit an (always) existing correlation, we have to consider at least $t + 1$ LFSRs at the same time [7].

3 The Decimation Attack

3.1 Description of the Algorithm

To deal with a more realistic attack, we will consider only ciphertext attack. So this will limit the drawback of ciphertext length increasing. Then the BSC with error probability p will model at the same time the Boolean function correlation $P[x_i = \sigma] = p^i_\sigma$ and the bit probability of plaintext $p_0 = P[m_t = 0]$. We take $p_0 = 0.65$ which corresponds to most real-life cases. Then we have $p = p_0 + p^i_\sigma - 2p_0p^i_\sigma$.

Let be a target LFSR of length L such that there exists $d|2^L - 1$ satisfying Proposition 1. In all these cases (see Section 3.3) the best d yields a simulated LFSR of length at most $\frac{L}{2}$.

With the decimated ciphertext sequence $\rho = \sigma[d]$ we try to recover the corresponding output sequence of the simulated LFSR which is the decimated sequence $x_i[d]$ of the (original) target LFSR output sequence x_i. It simply consists of a Siegenthaler attack [19] on this simulated LFSR (see Figure 1). It then needs only 2^{L^*} exhaustive searches.

Any kept L^*-bit candidate is used to generate L bits of the decimated sequence $x_i[d]$. Since each bit of sequence $x_i[d]$ is a bit of sequence x_i as well, it can be described by two different equations. One has L^* variables (the bit is seen as a bit of sequence $x_i[d]$). The other has L variables (the bit is seen as a bit of sequence x_i). Example 2 illustrates that.

The system of equations in L variables has obviously rank L^*. Then by taking L^* principal variables, we can express them, depending on the L^* other variables taken as parameters. An additional L^*-bit exhaustive search on these parameters allows to retrieve the correct L-bit initialization.

Note that in all our experiments, the correct L^*-bit initialization of the simulated LFSR was always detected within the very few first best estimator values (see farther), thus insuring detection and limiting cost of additional exhaustive search. However, to prevent a non such optimal detection, first step of L^*-bit exhaustive search is conducted on a few other shifted decimated sequences $\sigma_0[d], \sigma_1[d], \ldots, \sigma_k[d]$ where $\sigma_i[d]$ means that we decimate σ starting from index position i $(0 \leq i < d)$. Sorting the results always allowed to detect the correct L^*-bit initialization (we additionnally can test the very few kept L-bit initializations on the sequence σ). Good experimental values are $2 \leq k \leq L^*$.

We compute for each of the 2^{L^*} possible initializations the value of estimator

$$E = \sum_t (x^t_i[d] \oplus \sigma^t[d] \oplus 1)$$

For the correct L^*-bit initialization, E has Gaussian distribution with mean value $N(1-p)$ and variance $\sigma^2 = Np(1-p)$ ($\mathcal{H}_1$ hypothesis) whilst for all the wrong ones E has Gaussian distribution with mean value $\frac{N}{2}$ and variance $\sigma^2 = \frac{N}{4}$ ($\mathcal{H}_0$) where N is the minimum number of required ciphertext bits of $\sigma[d]$.

A decision threshold $\mathcal{T}$ will discriminate $\mathcal{H}_0$ from $\mathcal{H}_1$. If $|E| > \mathcal{T}$ then $\mathcal{H}_1$ is accepted and the initialization is kept otherwise $\mathcal{H}_0$ is chosen and the initialization is rejected. The minimum number N of required ciphertext bits of

$\sigma[d]$ depends on the number of wrong decisions that we accept. This number (as well as the threshold $\mathcal{T}$) is determined by the *false alarm probability* ($pfa = P[E > \mathcal{T}|\mathcal{H}_0]$) and the *non detection probability* ($pnd = P[E < \mathcal{T}|\mathcal{H}_1]$). If Φ denotes the normal distribution function

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{\infty}^{x} \exp(\frac{-t^2}{2})\, dt$$

then we have

$$pfa = 1 - \Phi\left(a = \frac{\mathcal{T} - \frac{N}{2}}{\sqrt{\frac{N}{4}}}\right) \qquad pnd = \Phi\left(b = \frac{\mathcal{T} - N(1-p)}{\sqrt{N(1-p)p}}\right)$$

Finally we obtain

$$N = \left(\frac{2b\sqrt{p(1-p)} - a}{1-2p)}\right)^2 \qquad \mathcal{T} = \frac{1}{2}\left(a\sqrt{N} + N\right) \tag{1}$$

In terms of ciphertext we then need $N \cdot d$ bits. Global complexity of the attack is then in $\mathcal{O}(N \cdot 2^{L^*+1})$ with no memory requirements.

3.2 Simulation Results

Now let us present simulation results of our attack.

Example 2 *CI(0) function.*
Let us consider a LFSR of length $L = 40$ *with feedback polynomial:*

$$P(x) = 1 + x^2 + x^3 + x^4 + x^5 + x^6 + x^8 + x^{11} + x^{12} + x^{15} + x^{17} + x^{19} + x^{22} + x^{24} + x^{25} + x^{26} + x^{27} + x^{28} + x^{29} + x^{35} + x^{40}$$

Consider a BSC with noise probability of $p = 0.49$ *(*$P_0 = 0.65$ *and* $P[x_t \neq y_t] = 0.47$*) modelling a CI(0) Boolean function and the plaintext noise at the same time. Since* $2^{40} - 1 = 3 \cdot 5^2 \cdot 11 \cdot 17 \cdot 31 \cdot 41 \cdot 61681$*, by choosing* $d = 1{,}048{,}577$ *as decimation factor for the LFSR output sequence* σ*, we obtain a simulated shorter LFSR of length* $L^* = 20$ *with feedback polynomial:*

$$P^*(x) = 1 + x + x^2 + x^3 + x^6 + x^7 + x^8 + x^{11} + x^{12} + x^{13} + x^{15} + x^{18} + x^{20}$$

With $pnd = 0.1$ *and* $pfa = 0.1$*, we then need by Equation 1,* $N = 13050$ *bits of decimated sequence* $\rho = \sigma[d]$ *that is to say* $N \cdot d = 2^{33}$ *bits of ciphertext. Complete computation time (*i.e. *recovering the correct inital state, with* $k = 2$*) required about 15 minutes on PII 400 Mhz with less than 1 Mo of memory.*

Note that the 20,971,541st bit (e.g.), when seen as belonging to sequence x *is described by the following (40 variables) equation (where* x_i $0 \leq i \leq 39$ *denotes ith unknown bit of the L-bit initialization):*

$$b_{20,971,541} = x_2 + x_4 + x_6 + x_7 + x_{14} + x_{16} + x_{17} + x_{19} + x_{20} + x_{22} + x_{23} + x_{25} + x_{26} + x_{28} + x_{29} + x_{30} + x_{33} + x_{36} + x_{37}$$

whilst, when seen as 21st bit of $x[d]$, we have the following (20 variables) equation (where y_i $0 \leq i \leq 19$ denotes ith unknown bit of the L^*-bit initialization):

$$b_{20,971,541} = y_0 + y_1 + y_2 + y_3 + y_6 + y_7 + y_8 + y_{11} + y_{12} + y_{13} + y_{15} + y_{18}$$

Example 3 *CI(1) function.*
Consider now two LFSRs P_1 of length $L_1 = 34$ and P_2 of length $L_2 = 39$ combined by a CI(1) Boolean function.

$$\begin{aligned} P_1(x) &= 1 + x^4 + x^5 + x^6 + x^7 + x^{12} + x^{13} + x^{14} + x^{15} + x^{16} + x^{18} + x^{20} + x^{21} \\ &\quad + x^{22} + x^{24} + x^{25} + x^{29} + x^{30} + x^{31} + x^{33} + x^{34} \\ P_2(x) &= 1 + x + x^2 + x^3 + x^4 + x^5 + x^{14} + x^{15} + x^{16} + x^{17} + x^{19} + x^{21} + x^{24} \\ &\quad + x^{21} + x^{24} + x^{27} + x^{28} + x^{29} + x^{31} + x^{32} + x^{36} + x^{37} + x^{39} \end{aligned}$$

We take the same BSC model as Example 2 but for a CI(1) Boolean function. This implies we must try all the possible initial states of at least two LFSRs at the same time thus requiring with previous known attacks to consider a single target LFSR of length 73. The best decimation factor in this case is $d = 131,073$ which divides $2^{34} - 1$. So we obtain these two new simulated LFSRs:

$$\begin{aligned} P_1^*(x) &= 1 + x + x^2 + x^5 + x^6 + x^7 + x^9 + x^{10} + x^{11} + x^{12} + x^{13} + x^{15} + x^{17} \\ P_2^*(x) &= 1 + x + x^4 + x^6 + x^7 + x^{14} + x^{16} + x^{17} + x^{18} + x^{20} + x^{22} + x^{24} + x^{28} \\ &\quad + x^{30} + x^{32} + x^{34} + x^{39} \end{aligned}$$

With $pnd = 0.1$ and $pfa = 0.1$, we then need by Equation 1, $N = 13050$ bits of decimated sequence $\rho = \sigma[d]$ that is to say $N \cdot d = 2^{30}$ bits of ciphertext. Complexity is in $\mathcal{O}(2^{56})$ with less than 1 Mo of memory. Partial experiment ($k = 8$) have been conducted on a PII 400 Mhz to verify that final pfa *was effectively close to zero.*

Note that rank of the system in L variables is 56, thus with 17 parameters. Then the additional exhaustive search is on 17 bits and is negligible compared to the first exhaustive search step.

3.3 Decimation Attack Resistance Criterion

By direct use of Proposition 1, we can define the following criterion for Decimation Attack resistance.

Proposition 2 *Let $L \in \mathbb{N}$. Any feedback polynomial of degree L will resist the decimation attack if and only if $\forall d < 2^L - 1$ such that $d | (2^L - 1)$, the multiplicative order of 2 in $\mathbb{Z}_{T^*}$ is equal to L where $T^* = \frac{2^L - 1}{\gcd(2^L - 1, d)}$*

We then obviously have

Corollary 1 *If $T = 2^L - 1$ is prime then any feedback polynomial of length L will resist the decimation attack.*

In fact, finite field theory allows to precise im a simpler way this criterion.

Theorem 1 *(Subfield Criterion) [11] Let $\mathbb{F}_q$ be a finite field with $q = p^n$ elements. Then every subfield of $\mathbb{F}_q$ has order p^m, where m is a positive divisor of n. Conversely, if m is a positive divisor of n, then there is exactly one subfield of $\mathbb{F}_q$ with p^m elements.*

Then we can easily state the resistance criterion as follows:

Proposition 3 *Any feedback polynomial of length L, such that L is prime, will resist the decimation attack.*

Proof. straightforward by direct application of Theorem 1 and Proposition 2.

Then, resistance is ensured as soon as the field $\mathbb{F}_{2^L}$ contains no subfield. It becomes then obvious that, when decimation attack is possible, there exists a value of L^* at most equal to $\frac{L}{2}$.

Note that this criterion is very new and must not be mistaken with that of **relative primality** of periods when considering the sum of outputs of LFSRs as described by the following theorem:

Theorem 2 *[11, p. 224] For each $i = 1, 2, \ldots, h$, let σ_i be an ultimately periodic sequence in $\mathbb{F}_q$ with least period r_i. If $r_1, r_2, \ldots, r_h$ are pairwise relatively prime, then the least period of the sum $\sigma_1 + \sigma_2 + \cdots + \sigma_h$ is equal to product $r_1 r_2 \ldots r_h$.*

When designing a stream cipher system, one must carefully check the degree of the chosen feedback polynomials (in connection with correlation order) to resist the Decimation Attack. The following table gives some values of L satisfying this new resistance criterion ("prime" relates to Corollary 1 and "order" relates to Proposition 2). A complete list of parameters d, T^* and L^* up to $L = 256$ has been computed and can be found in [6].

Comment	L
Prime	5 - 7 - 13 - 17 - 19 - 31 - 61 - 89 - 107 - 127
Order	11 - 23 - 29 - 37 - 41 - 43 - 47 - 53 - 59 - 67 - 71 - 73 83 - 97 - 101 - 109 - 113 - 131 - 139 - 149 - 151 - 157 - 163 167 - 173 - 178 - 179 - 181 - 191 - 193 - 197 - 199 211 - 223 - 227 - 229 - 233 - 239 - 241 - 251

4 Comparison with Previous Known Attacks

4.1 Canteaut and Trabbia Attack

Canteaut and Trabbia (CT) in [3] recently improved fast correlation attack on stream ciphers. They considered parity-check equations of weight $\delta = 4, 5$ with

Gallager iterative decoding. However, the main drawback of this approach, despite its relative efficiency, remains the huge amount of required memory both for preprocessing (generation of the parity-check equations) and decoding steps, specially for long LFSR [2]. This latter, though being very efficient best, still requires higher complexity than suitable for frequent key recovering.

We now give here comparison of our attack (DA) with CT attack. Suppose that by applying condition of Proposition 1 a reduction of ΔL bits on exhaustive search is possible. Then our attack has complexity $\mathcal{C}_{DA} = \mathcal{O}(N \cdot 2^{L-\Delta L+1})$ (where N is given by Equation 1) and a negligible amount of memory. Let us now compute the complexity gain from CT attack. In [3] the complexity is given by the following formula:

$$\mathcal{C}_{CT} = 5(\delta - 1)\left(\frac{K_\delta}{C_{\delta-2}(p)}\right)2^{\alpha_\delta(p)+\frac{L}{\delta-1}}$$

where δ is the weight of the parity-check equation ($3 \leq \delta \leq 5$) and $C_{\delta-2}(p)$ is the BSC capacity (with overall error probability $p_{\delta-2} = \frac{1}{2}(1-(1-2p)^{\delta-2})$) *i.e.* $C_{\delta-2}(p) = 1 + p_{\delta-2}\log(p_{\delta-2}) + (1-p_{\delta-2})\log(1-p_{\delta-2})$. The value $K_\delta \approx 1$ if $\delta \geq 4$ and $K_3 \approx 2$. Finally $\alpha_\delta(p) = \frac{1}{\delta-1}\log_2[(\delta-1)!\frac{K_\delta}{C_{\delta-2}(p)}]$.

The complexity gain of our attack is then

$$\frac{\mathcal{C}_{CT}}{\mathcal{C}_{DA}} = 5(\delta-1)\left(\frac{K_\delta}{N \cdot C_{\delta-2}(p)}\right)2^{\alpha_\delta(p)+\frac{L}{\delta-1}+\Delta L - L - 1}$$

For relatively short LFSR, the complexity gain is limited (provided δ is small) and the ciphertext sequence is longer for DA as illustated by Table 1. It is

Table 1. Comparison on Example 2

	Decimation Attack	CT Attack ($\delta = 5$)
Ciphertext bits	2^{33}	2^{20}
Complexity	2^{21}	2^{59}
Memory (bytes)	< 1024	2^{38}

easy to see that the gain increases with δ. Now using higher weight precisely constitutes the central point of CT attack. Moreover, the gain increases with the error probability p. These two facts reinforces the interest of our technique when dealing with longer LFSRs and higher error probability of real-life cases. Table 2 gives comparison for this case ("ppm" means preprocessing with memory and "ppwm" means preprocessing without memory). Concerning the memory requirement, CT attack needs $(\delta-1)\left(\frac{K_\delta}{C_{\delta-2}(p)}\right) + 2^{\alpha_\delta(p)+\frac{L}{\delta-1}+1}$ computer words of memory. This once again increases with δ and p.

Note that preprocessing step in CT attack must be done once and for all but for each different feedback polynomial of length L. On the contrary, Decimation Attack apply to all polynomials of degree L (for suitable L) and no preprocessing is required.

Table 2. Comparison on Example 3

	DA	CT ($\delta = 5$) (decoding step)	CT ($\delta = 5$) (ppm step)	CT ($\delta = 5$) (ppwm step)
Ciphertext bits	2^{30}	2^{29}	-	-
Complexity	2^{56}	2^{67}	2^{56}	2^{81}
Mem. (bytes)	< 1024	2^{39}	2^{57}	-

4.2 Chepyzhov, Johansson and Smeets Attack

In [5], V. Chepyzhov, T. Johansson and B. Smeets recently proposed a new powerful idea to improve fast correlation attacks (CJS attack). The key idea is to combine pairs of parity-check equations describing a $[N, L]$ code on a BSC of error probability p, in order to obtain a code of lower dimension $[n_2, K]$ $(K < L)$ over another BSC of parameter p_2. The price to pay is $n_2 > N$ and $p_2 = 2p(1-p) = \frac{1}{2} - 2\epsilon^2 > p$ where $\epsilon = \frac{1}{2} - p$. So a trade-off has to be found since lowering K increases n_2. In this part, only algorithm A1 of [5] will be considered since it has the best complexity and required sequence length than algorithm A2.

Consider a register of length L and a BSC of parameter $p = \frac{1}{2} - \epsilon$. For a given K the required length N of the observed sequence for algorithm A1 to succeed is

$$N \approx \frac{1}{2}\sqrt{k(\ln 2)}\epsilon^{-2}2^{\frac{L-K}{2}}.$$

Then the decoding step complexity has order $\mathcal{O}(2^k \cdot n_2)$ where n_2 has mathematical expectation given by $E(n_2) = \frac{N(N-1)}{2}2^{-(L-K)}$. Complexity of the precomputation is negligible. Storage requirement is at most $n_2(k + 2\log_2 N)$ bits (generator matrix G_2 of the new $[n_2, k]$ code). So by taking a small k, the 2^k factor is reduced but conversely it must be paid with the growth of n_2, and thus of the observed sequence length N.

Comparison with the Decimation attack is not easy. The great advantage of CJS is that it works even for register of prime length but conversely preprocessing is to be done for each different polynomial. In the other hand, Decimation attack seems to succeed more frequently for sequence length of quite the same order and requires no memory whilst memory requirements grows with n_2 (that is to say when K decreases) in CJS attack.

Thorough comparison need to be conducted and is at present on hand. DA attack as presented in this paper use only ciphertext and need to be considered in the known plaintext context to be compared to CJS attack. Moreover DA attack seems clearly less sensitive to BSC error probability p since it use a (optimal) Maximum Likelihood decoding. That implies that for higher p DA seems less complex and requires less ciphertext length, when considering same decoding error probability.

All these assumptions need to be confirmed. Unfortunately, data given in [5] were only for success probability set to $\frac{1}{2}$. As a first attempt in comparison, let us consider the example given in [5, Section 6] for $L = 40$. Data are presented in Table 3 for known plaintext attack in order to use the results given in [5]. Note that in all cases, CJS attack find only $K < L$ bits and need to recover

Table 3. Comparison of DA and CJS attacks

	Decimation Attack	CJS Attack
BSC error probability	0.45	0.45
Sequence length N	2^{29}	2^{23}
Success probability	0.9	$\frac{1}{2}$
Complexity	2^{30}	2^{25}

the remaining ones whilst DA recovers L bits. Complexity and probability of success of Table 3 have thus to be to be modified. Lowering K will in fact deplaces complexity on the $L - K$ bits recovering step and greatly increased failure probability ((particularly final probability of success should be far lower than $\frac{1}{2}$).

To conclude, as far as comparison between the two attacks is possible, respective complexity of the two attack seems to be close. Nevertheless, CJS attack is a very powerful, stimulating and challenging attack. Development of the potentiel of Decimation attack is at now under progress.

Acknowledgements

I would like to thank the anonymous reviewers for their useful comments, corrections and suggestions. I also would like to specially thank Anne Canteaut from Codes Project at INRIA. She particularly took time to help me in improving this paper. I am very grateful to her for it. It is always a great, stimulating pleasure to work with her.

References

1. A. Canteaut, E. Filiol, *Ciphertext Only Reconstruction of Stream Ciphers based on Combination Generators*, Fast Software Encryption 2000, New-York, to appear in Lecture Notes in Computer Science. 33
2. Anne Canteaut, Email Communication, March 2000. 39
3. A. Canteaut, M. Trabbia, *Improved Fast Correlation Attacks using Parity-check Equations of weight 4 and 5.* Advances in Cryptology - EUROCRYPT'00, LNCS 1807, pp 773–588, Springer Verlag 2000. 32, 33, 33, 38, 39
4. V. Chepyzhov, B. Smeets, *On a Fast Correlation Attack on Stream Ciphers*, Advances in Cryptology - EUROCRYPT'91, LNCS 547, Springer Verlag, 1991. 32
5. V. Chepyzhov, T. Johansson, B. Smeets, *A Simple Algorithm for Fast Correlation Attack on Stream Ciphers*, Fast Software Encryption 2000, New York, to appear in Lecture Notes in Computer Science, Springer Verlag. 33, 33, 40, 40, 40, 41, 41

6. http://www-rocq.inria.fr/codes/Eric.Filiol/index.html 38
7. E. Filiol, C. Fontaine, *Highly Nonlinear Balanced Boolean Functions with a Good Correlation-Immunity*, Advances in Cryptology - EUROCRYPT'98, LNCS 1403, Springer Verlag, 1998. 32, 34
8. R.G. Gallager, *Low-density parity-check codes*, IRE Trans. Inform. Theory, IT-8:21-28, 1962. 32
9. T. Johansson, F. Jönsson, *Improved Fast Correlation Attack on stream Ciphers via Convolutional Codes*, Advances in Cryptology - EUROCRYPT'99, LNCS 1592, pp 347–362, Springer Verlag, 1999 32
10. T. Johansson, F. Jönsson, *Fast Correlation Attack based on Turbo Codes Techniques*, Advances in Cryptology - CRYPTO'99, LNCS 1666, pp 181–197, Springer Verlag, 1999 32
11. R. Lidl, H. Niederreiter *Introduction to Finite Fields and their Applications*, Cambridge University Press, 1994. 38, 38
12. J.L. Massey *Shift-Register Synthesis and BCH Decoding*, IEEE Trans. on Info. Theory, Vol. IT-15, Jan. 1969. 34
13. W. Meier, O. Staffelbach, *Fast Correlation Attack on certain Stream Ciphers*, J. of Cryptology, pp 159–176, 1989. 32
14. M. Mihaljevic, J. Dj. Golic, *A Fast Iterative Algorithm for a Shift-Register Initial State Reconstruction given the Noisy Output Sequence*, Proc. Auscrypt'90, LNCS 453, Springer Verlag, 1990. 32
15. W. Penzhorn, *Correlation Attacks on Stream Ciphers: Computing low Weight Parity Checks based on Error-Correcting Codes*, FSE'96, LNCS 1039, Springer Verlag, 1996. 32
16. R.A. Rueppel, *Analysis and Design of Stream Ciphers*, Springer Verlag, 1986. 34, 34
17. E.S. Selmer, *Linear Recurrence Relation over Finite Fields*, Ph. D Thesis, University of Bergen, Norway, 1966. 32
18. T. Siegenthaler, *Correlation Immunity of Nonlinear Combining functions for Cryptographic Applications*, IEEE Transactions on Information Theory, Vol. 35 Nr 5, September 1984, pp 776–780. 32, 34
19. T. Siegenthaler, *Decrypting a Class of Stream Ciphers using Ciphertext Only*, IEEE Transactions on Computers, C-34, 1, pp 81–84, 1985. 32, 34, 35

Cryptanalysis of the A5/1 GSM Stream Cipher [*]

Eli Biham and Orr Dunkelman

Computer Science department,
Technion - Israel Institute of Technology,
Haifa 32000, Israel,
biham@cs.technion.ac.il
http://www.cs.technion.ac.il/~biham/
orrd@cs.technion.ac.il
http://vipe.technion.ac.il/~orrd/me/

Abstract. A5/1 is the stream cipher used in most European countries in order to ensure privacy of conversations on GSM mobile phones. In this paper we describe an attack on this cipher with total work complexity $2^{39.91}$ of A5/1 clockings, given $2^{20.8}$ known plaintext. This is the best known result with respect to the total work complexity.

1 Introduction

The GSM mobile phone system, which has more than 200 million users, uses encryption to protect the privacy of conversations. Two ciphers are used. A5/1 is used in most European countries, and A5/2 is used in most other countries. Both ciphers, combine a few (3 or 4) Linear Feedback Shift Registers (LFSR), whose clocks are controlled by some of their bits.

A5/1 and A5/2 were kept secret by the GSM companies for a long period of time. Only recently A5/1 and A5/2 were reverse-engineered and published by Briceno, Goldberg and Wagner at [4,5]. Afterwards A5/2 was also cryptanalyzed by Wagner [5].

Before A5/1 was reverse-engineered several researchers, and in particular Anderson, identified and published [1] the general structure of A5/1. Golic [7] attacked this structure with parameters which are very close to the actual parameters of A5/1 by solving a system of 2^{41} linear equations. This attack is likely to work on the real version of A5/1. The complexity of both attacks is equivalent to about 2^{47} A5/1 clockings.

A5/1 is a stream cipher with a 64-bit key. Encryption is also controlled by a 22-bit frame number which is publicly known. In practice, A5/1 is always used with only 54 bits of key, with the remaining 10 bits set to zero.

Biryukov and Shamir presented an improvement on the Golic Time-Memory tradeoff attack in [2]. The attack requires about 2 minutes of GSM conversation, and finds the key in a few seconds. However, their attack requires a large preprocessing phase equivalent to 2^{42} A5/1 clockings with memory complexity of

[*] This work was supported by the European Union fund IST-1999-12324 - NESSIE and by the Technion's Chais' Excellence Program.

four 73 GB hard-drives, or pre-processing of 2^{48} A5/1 clockings with memory complexity of two 73 GB hard-drives.

In this paper, we introduce a new technique for retrieving the internal state of the cipher. Once the internal state of the cipher is known it is easy to find the secret key by clocking the cipher backwards, using the technique presented in [2].

This paper is organized as follows: In Section 2 we describe the A5/1 algorithm. In Section 3 we briefly describe the previous analysis of this cipher. In Section 4 we describe the basic idea behind our attack. In Section 5 we show several improvements to the attack which requires $2^{20.8}$ known output stream, and only $2^{39.91}$ steps of analysis (the time unit is one A5/1 clocking). The attack finds the full internal state of the registers, from which the original key can be derived easily, as presented in [2].

2 A Description of A5/1

A5/1 combines 3 LFSRs as described in Figure 1. At each new step, 2 or 3 LFSRs are clocked, according to a clocking mechanism which we will describe later. The output is the parity of the outputs of the 3 LFSRs.

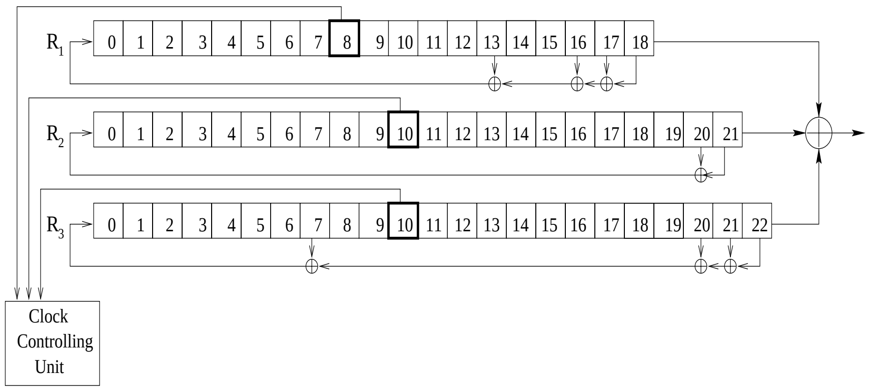

Fig. 1. The A5/1 Structure

We denote the LFSRs as R_1, R_2 and R_3. The lengths of R_1, R_2 and R_3 are 19, 22 and 23 bits, respectively. The output of each LFSR is the last bit (i.e., bits 18, 21 and 22, respectively). The registers are updated according to their primitive polynomials, which are summarized in Table 1. The clocking decision is based upon one bit of each register. The three bits are extracted (bit 8 of R_1, bit 10 of R_2 and bit 10 of R_3) and their majority is calculated. The two or three registers whose bit agrees with the majority are clocked.

We denote the bits $j_1, \ldots, j_l$ of register R_i by $R_i[j_1, \ldots, j_l]$.

Register Number	Length in bits	Primitive Polynomial	Clock-controling bit (LSB is 0)	Bits that are XORed
1	19	$x^{19}+x^5+x^2+x+1$	8	18,17,16,13
2	22	$x^{22}+x+1$	10	20,21
3	23	$x^{23}+x^{15}+x^2+x+1$	10	22,21,20,7

Table 1. Parameters of the A5/1 Registers

The initialization of the registers loads the bits of secret key Key, followed by the bits of the frame number $Frame$ and discarding 100 output bits, as follows:

1. Set all LFSRs to 0 ($R_1 = R_2 = R_3 = 0$).
2. For $i := 0$ to 63 do
   (a) $R_1[0] = R_1[0] \oplus Key[i]$
   (b) $R_2[0] = R_2[0] \oplus Key[i]$
   (c) $R_3[0] = R_3[0] \oplus Key[i]$
   (d) Clock all three registers (i.e., for $j > 0$ $R_i[j] \leftarrow R_i[j-1]$, and $R_i[0]$ is set to the result of the primitive polynomial on the previous value of R_i).
3. For $i := 0$ to 21 do
   (a) $R_1[0] = R_1[0] \oplus Frame[i]$
   (b) $R_2[0] = R_2[0] \oplus Frame[i]$
   (c) $R_3[0] = R_3[0] \oplus Frame[i]$
   (d) Clock all three registers.
4. For $i := 0$ to 99, clock the cipher by its regular majority clocking mechanism, and discard the output.

After the initialization, 228 bits of output stream are computed. 114 bits are used to encrypt data from the center to the mobile phone, and the other 114 bits are used to encrypt data from the mobile phone to the center.

3 Previous Work

Several papers analyzing variants of A5/1 have been published [7,1,2]. One of them [7] attacks an alleged version, which is very similar to the real A5/1. This attack takes on average $2^{40.16}$ workload and finds the internal state of the cipher. However in [7] the time unit is the time needed to solve a linear equations system, a unit which we do not use. We, on the other hand, use (like [2]) a workload unit of one A5/1 clocking. Golic [7] also presents a time-memory tradeoff, which was enhanced by Biryukov and Shamir in [2] after the first version of our paper was written.

Golic's first attack [3] is based on creating sets of linear equations and solving them. Although it seems that 64 linear equations are necessary, Golic observed that on average 63.32 equations are sufficient as only $2^{63.32}$ internal states are possible after clocking. This can be easily identified by additional equations which reject the 3/8 of the states that have no predecessors.

In the first step, the attacker guesses n clock controlling bits from each of the three registers. Since these bits are sufficient to know the clocking sequence of $4n/3$ bits on average, the attacker can receive $4n/3$ linear equations (on average) about the registers contents. In addition, the first output bit is known to be the parity of the most significant bits of the three registers, hence the attacker obtains another equation. Therefore, the attacker now has $3n + 4n/3 + 1$ linear equations. In the process of analysis we assumed the bits to be independent, thus, n cannot be bigger than the shortest distance between the clock controlling bit and the output bit.

For $n = 10$ the attacker can get on average $3n + 4n/3 + 1 = 44.33$ linear equations, thus he needs about 19 more equations. Golic observed that not all 2^{19} options need to be considered. Instead, the attacker builds a tree with all the valid options for the possible values for the three input bits to the majority clock-control function. Since in 3/4 of the cases two new bits are considered and in the remaining 1/4 of the cases, 3 bits are considered then each node contains $3/4 \cdot 4 + 1/4 \cdot 8 = 5$ options. However, when we walk along the path in the tree, we need to consider only the cases for which the output of the registers agrees with the output stream, which happens on average in 1/2 of the cases. Thus, we travel along the path on the tree which corresponds to the output stream. On this path each node has 2.5 valid options on average. Since the knowledge of $4/3m$ bits is sufficient to receive the linear equations about the m bits out of each register (due to the clocking mechanism), the tree of clocking options needs to be considered only until a depth of $4/3 \cdot 19/3 = 76/9 = 8.44$. Since each level has a branching factor of 2.5, the amount of time needed to search the tree is $2.5^{8.44} \approx 2^{11.16}$.

The total number of systems is $2^{30} \cdot 2^{11.16} = 2^{41.16}$ (there are 2^{30} possible guesses for the $n = 10$ bits of each register, and each guess requires $2^{11.16}$ additional equations). As we search exhaustively, on average only half of the options are needed until we encounter the right value, thus the complexity of this attack is equivalent to solving $2^{40.16}$ sets of linear equations.

In [3] Biryukov, Shamir and Wagner presented a time-memory tradeoff attack. In their attack a special pattern is chosen, and the attacker preprocesses many samples which generate this specific long pattern. Once the pattern occurs in the real output stream, the attacker finds the possible internal states, according to the rest of the output stream and the pre-processed table. The attack requires 2^{38} to 2^{48} pre-processing time, and has many trade-offs between the length of the conversation (and analysis), the pre-processing phase and the memory needed for the attack.

4 Our Basic Attack

The main idea behind the attack is to wait until an event which leaks a large amount of information about the key occurs and then to exploit it. Assume that for 10 consecutive rounds the third register (R_3) is not clocked. In such a case, we gain about 31 bits of information about the registers. Assume we know

$R_3[10]$ (the clock controlling bit) and $R_3[22]$ (the output bit), then we know for these 10 rounds that all the clock controlling bits in the other two registers are the complement of $R_3[10]$ (thus receiving 20 bits, a bit from each register each round). We also know that the output bit of the stream XOR $R_3[22]$ is equal to the output bits of R_1 XOR R_2 (each round). There are at least 11 rounds for which the same output bit is used from R_3, thus we receive another 11 linear equations about the registers. The equations are the parity of the couples $R_1[t] \oplus R_2[t+3]$ for $t = 8, \ldots, 18$.

Guessing 9 bits from R_1 ($R_1[18, 17, 16, 15, 14, 12, 11, 10, 9]$) and another one from R_2 ($R_2[0]$) uncovers all of R_1 and R_2. Note that we do not need to guess $R_1[13]$ as we know the parity bit $R_1[18] \oplus R_1[17] \oplus R_1[16] \oplus R_1[13]$, thus guessing $R_1[18], R_1[17], R_1[16]$ gives $R_1[13]$, in addition to the corresponding bits of R_2 ($R_2[21, 20, 19, 16]$). $R_2[11]$ can be easily computed since $R_1[8] \oplus R_2[11]$ is known (from the output stream), and $R_1[8]$ is known (since it was the first clock-controlling bit in register R_1).

The complexity so far is the cost of guessing twelve bits ($R_3[10]$, $R_3[22]$, $R_2[0]$, and 9 bits of R_1), and we recover all the bits of R_1 and R_2. We can continue to clock the states until R_3 moves, and gain another bit ($R_3[21]$).

By further guessing the bits $R_3[0], \ldots, R_3[9]$ we can continue clocking the third register and receive all the unknown bits of R_3 (total of 11 unknown bits). The workload of this stage is the cost of guessing 10 bits, multiplied by the cost of work needed to retrieve the unknown bits for each guess. Since each time the register R_3 advances we recover a new bit of R_3, we need to wait for 12 clockings of R_3 in order to retrieve the full register, and since the probability that the register R_3 advances is 3/4, it is expected that 16 clocks suffice to retrieve the whole register. Now we have a possibility for the full internal state and need to check for its correctness. The amortized cost for checking whether this is the right state is 2 clock cycles for each guess (all cases need to be clocked at least once, half of the cases should be clocked another time, 1/4 a third time, etc.). Thus, this stage requires $2^{10} \cdot 2^4 \cdot 2 = 2^{15}$ workload.

The total expected running time of this attack is $2^{12} \cdot 2^{15} = 2^{27}$, given the location where R_3 is static. This location is of course unknown, thus, we need to examine about 2^{20} possible starting locations (from many possible frames). Hence, the basic attack requires about 2^{47} running time and about 2^{20} bits of the output stream.

Note that this complexity is similar to the complexity of earlier attacks (solving 2^{41} linear equations sets in [7] can be done in about 2^{47} time).

5 Trick or Treat?

We now present several techniques for reducing both time complexity and memory complexity required for the attack.

The first technique is based on a technique presented in [3]. It is known that the polynomials of the registers of A5/1 are primitive, and thus the states (except for the all-zero state) form one huge cycle. Selecting any non-zero state and computing the next states iteratively would result in all (non-zero) states. Therefore, for each register we choose a non-zero state, and clock the register until we get the same state again. We store the states in a table in the order they are computed. We call this table the *next-state* table. In this table if entry i corresponds to state x, then the state consecutive to x is stored in entry $i+1$. During the generation of this table, we also generate a table of pointers, in which for each state x we keep its location in the next-state table.

Given the two tables (for each register), we can clock any state s by any number of clockings c in a fixed time unit. The method for doing so is to access entry s in the pointers table in order to find the location l of s in the next-state table, and then access entry $l+c$ in the next-state table. The cost of the first access is equivalent to two (original) A5/1 clockings (as we access two tables), but once the location in the next-state table is known, only one table needs to be accessed. Thus, when a state should be clocked again (or iteratively) the additional cost is equivalent to one A5/1 clocking.

The memory size needed for the tables is about 71.5 MB. For each state we need to store the next state and pointer to the next-state table. Since each of these fields is in the size of the register, the total size of an entry is bounded by $2 \cdot 23 = 46$ bits (or 6 bytes), and there are 2^{23} states in R_3, 2^{22} in R_2 and 2^{19} in R_1. Thus the tables' size is about $(2^{23}+2^{22}+2^{19}) \cdot 2 \cdot 23 = 2^{29.16}$ bits which are 71.5 MB.

For any possible value of the 5 most significant bits in each register and the next 5 clock-controlling bits, we calculate as many bits of the future output stream as possible. We build another table indexed by the first 5 bits of the output stream, and the 20 bits of R_1 and R_2 (5 MSB and 5 clock-controlling from each), which contains in each entry the possible values of the 10 bits from R_3 which generate this 5-bit output stream. Note that for some fraction of the cases 6 or 7 bits of output stream can be generated (given the 10 bits of each register). 8-bit or longer output stream cannot be generated due to lack of clock controlling bits, as each clocking requires at least two new clock controlling bits out of a total of 15. Thus, given the output stream we can reduce the number of possible values of the bits of R_3 which generate the output stream. In order to do so we separate the cases according to the length of the output stream, and if there is additional bits of the output stream, according to their value. Each entry also contains information on how many clocks each register needs to be clocked after the given known output stream has been generated by the 30 bits (10 from each register as above) of the state. The computation of this table requires about $2^{30} \cdot 6 \approx 2^{32.6}$ A5/1 clockings, which can be computed in advance.

The length of the predicted output stream is a function of the 15 clock controlling bits. In 15968 cases the output stream is in the length of 5 bits, in 14280 cases the stream's length is 6 bits, and in the remaining 2520 cases a 7-bit output stream is generated. In the case in which we know 20 bits of R_1 and R_2 and the output stream, we need to check all the corresponding cases with a 5-bit output stream, half of the cases with a 6-bit output stream and 1/4 of the cases with a 7-bit output stream. Thus, we need to consider on average $2^{4.53} \approx 23.2$ options for the 10 bits of R_3.

Note that the first time we access the table three bits are known ($R_3[22]$, $R_3[21]$, $R_3[10]$), so we have about $2^{1.53} \approx 2.9$ options for the rest of the $10-3=7$ bits. In order to use those known bits we need to sort the values in the table according to those 3 bits, and then retrieve the valid $2^{1.53}$ values directly in one table look-up. Sorting the table in that manner either increases the time of analysis or table size by a small factor.

For each remaining case we clock the registers as many times as needed according to the first table access (since the table contains the number of clockings for each register). Then, we approach the table again and get about $2^{4.53}$ options for 10 unknown bits of R_3.

We now lack 3 bits from R_3. Approaching the original table again would cost us too much (we would get $2^{4.53}$ options and need to check them all), thus we use a smaller table containing only 6 bits of each register. This table should suffice to give us the last 3 unknown bits. This table is expected to give us for each case 0.88 possible values on average. In this stage, we discard the wrong options by the regular method of clocking forward (while we can use the table to do this stage more efficiently, however the improvement of the complexity is negligible).

The total time complexity is calculated as follows: There are 2^{20} possible starting locations, and each has 2^{12} possible guesses for the bits of the first stage (9 from R_1, $R_2[0]$ and $R_3[10, 22]$). For each possibility we get about $2^{1.53}$ possible values for some of the unknown bits of R_3, and the clocking costs 2 clocking units (as this is the first approach to the next-state table). Then, for each possible value for the first 7 unknown bits from R_3, we get about $2^{4.53}$ possible values for the next 10 unknown bits of R_3. Each possibility needs to be clocked once, followed by accessing the smaller table. About 0.88 cases remain on average, and we need to clock each twice (on average) to check the guess. Thus, the time complexity of the attack is equivalent to $2^{20} \cdot 2^{12} \cdot 2^{1.53} \cdot 2 \cdot 2^{4.53} \cdot (1 + 1 + 2 \cdot 0.88) = 2^{40.97}$ A5/1 clockings.

We can improve this result by using more bits in the table. We build the table based on 12 bits from R_1 and R_2 (6 most significant bits, and 6 clock-controlling bits), and 10 bits from R_3. In that case 23328 options have a 5-bit output stream, 59808 a 6-bit output stream, 41496 a 7-bit output stream and 6440 a 8-bit output stream. This way given the 24 bits of register R_1 and R_2 and the 5 bits of the output-stream, there would be only 2^4 options on average for the 10 bits of R_3. The complexity in this case is $2^{20} \cdot 2^{12} \cdot 2^1 \cdot 2 \cdot 2^4 \cdot (1 + 1 + 2 \cdot 0.88) = 2^{39.91}$. The preprocessing time is about $2^{34} \times 8 = 2^{37}$ A5/1 clockings, and the table size is $2^{34} \cdot 2 = 2^{35}$ bytes, i.e., 32 GB of data. This needs to be multiplied by the overhead of keeping the table indexed both by 20 known bits (10 of R_1 and 10 of R_2) and by 23 bits (10 of R_1, 10 of R_2 and 3 of R_3), so in one memory access we get only the relevant possibilities, which is a factor 2, i.e., 64 GB.

We can, however, reduce the size of the tables by a factor of two, using the observation that if we switch the known bits of R_1 with those from R_2, the result for R_3 remains the same. This way we can reduce the size by almost a factor of 2.

6 Summary

We have shown a technique to cryptanalyze the A5/1 cipher. The attack is feasible with current technology, which suggests that this scheme should be replaced. The attack requires $2^{39.91}$ A5/1 clockings. The attack requires $2^{20} \cdot 228/(228 - 63) = 2^{20.8}$ bits of data, which are equivalent to about 2.36 minutes of conversation.

The retrieval of the key given the internal state can be easily done using the algorithm presented in [7].

We summarize our results and the previously known results in Table 2.

Attack	Pre-computation	Complexity of Analysis	Time Unit	Data Complexity (bits)	Memory Complexity
[7] - Basic Attack	0	$2^{40.16}$	Linear eq. set solving	64	0
[7] - TM Tradeoff	$2^{35.65}$	$2^{27.67}$	Linear eq. set solving	$2^{28.8}$	862 GB
[3] - Baised Birthday Attack	2^{48}	1 second	A5/1 Clocking	$2^{20.5}$	146 GB
[3] - Baised Birthday Attack	2^{42}	1 second	A5/1 Clocking	$2^{20.5}$	292 GB
[3] - Random Subgraph Attack	2^{48}	minutes	A5/1 Clocking	$2^{14.7}$	146 GB
Our Results	2^{38}	$2^{39.91}$	A5/1 Clocking	$2^{20.8}$	32 GB
Our Results	$2^{33.6}$	$2^{40.97}$	A5/1 Clocking	$2^{20.8}$	2 GB

Table 2. Attacks on A5/1 and their complexities

7 Acknowledgments

The authors would like to thank Adi Shamir and Alex Biryukov for fruitful conversations about the attack, Avi Opshenko for his very useful remarks, and the anonymous referees for their suggestions on the first version of this paper.

References

1. Ross Anderson, *On Fibonacci Keystream Generators*, Proceedings of Fast Software Encryption - FSE '95, Springer vol. 1008, pp. 346-352, 1995. 43, 45

2. Alex Biryukov, Adi Shamir, *Real Time cryptanalysis of A5/1*, private communication. 43, 44, 44, 45, 45, 45
3. Alex Biryukov, Adi Shamir, David Wagner, *Real Time Cryptanalysis of A5/1 on a PC*, Preproceedings of FSE '7, pp. 1-18, 2000. 45, 46, 47, 50, 50, 50
4. Marc Briceno, Ian Goldberg, David Wagner, *A Pedagogical Implementation of A5/1.* 43
5. Marc Briceno, Ian Goldberg, David Wagner, *A Pedagogical Implementation of the GSM A5/1 and A5/2 "voice privacy" encryption algorithms*. Available at *http://www.scard.org/gsm/a51.html.* 43, 43
6. Jovan Golic, *On the Security of Nonlinear Filter Generators*, Proceedings of Fast Software Encryption - FSE '96, Springer vol. 1039, pp. 173-188, 1996.
7. Jovan Golic, *Cryptanalysis of Alleged A5 Stream Cipher*, Proceedings of Eurocrypt '97, Springer LNCS vol. 1233, pp. 239-255, 1997. 43, 45, 45, 45, 45, 47, 50, 50, 50

On Bias Estimation in Linear Cryptanalysis

Ali Aydın Selçuk

Maryland Center for Telecommunications Research
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
Baltimore, MD, 21250, USA
aselcu1@csee.umbc.edu

Abstract. Security analysis of block ciphers against linear cryptanalysis has virtually always been based on the bias estimates obtained by the Piling-Up Lemma (PUL) method. Despite its common use, and despite the fact that the independence assumption of the PUL is known not to hold in practice, accuracy of the PUL method has not been analyzed to date. In this study, we start with an experimental analysis of the PUL method. The results on RC5 show that the estimates by the PUL method can be quite inaccurate for some non-Feistel ciphers. On the other hand, the tests with SP-structured Feistel ciphers consistently show a much higher degree of accuracy.
In the second part, we analyze several theories for an alternative method for bias estimation, including correlation matrices, linear hulls, and statistical sampling. We show a practical application of the theory of correlation matrices, where better estimates than the PUL method are obtained. We point out certain problems in some current applications of linear hulls. We show that the sample size required for a reliable statistical estimator is an impractically large amount for most practical cases.

1 Introduction

Estimating the bias of a given linear approximation is one of the most important problems in linear cryptanalysis: the success rate of a linear attack is directly related to the bias of the approximation it uses, therefore, security analysis of block ciphers against linear cryptanalysis is exclusively based on the estimation of the bias of their linear approximations.

In practice, estimation of the bias is almost exclusively based on the Piling-Up Lemma (PUL) [11], which is a very practical tool for bias estimation on iterated block ciphers. To estimate the bias of a multi-round approximation, the round approximations are assumed independent and the bias of the combined approximation is calculated by the PUL. We will refer to this application of the PUL as the *PUL method*.

In the first part of this study, we analyze the bias estimates obtained by the PUL method. Although the PUL method has been widely used, and although it is known that this method's assumption of independent round approximations is virtually never true, the accuracy of the estimates obtained by this method has almost never been the subject of a study.[1] Our analysis concentrates on two cases: DES-like SP-structured Feistel ciphers and RC5. The Feistel ciphers represent the class of ciphers that the PUL method was originally applied on. RC5 represents a cipher that has a totally different structure (which is based on a mixture of arithmetic operations and data-dependent rotations, instead of the traditional substitution and permutation structures). In the study of Feistel ciphers, we analyze the accuracy of the estimated values with respect to various factors, including the number of active s-boxes in a round, presence/absence of a bit expansion function, etc.

The analysis results show that the PUL method gives quite accurate estimates with SP-structured Feistel ciphers, especially for approximations with at most a single active s-box at each round, as long as the estimated values are significantly higher than $2^{-\frac{n}{2}}$. With RC5, the estimates turn out to have a much lesser accuracy.

In the second part of this study, we look for an alternative estimation method which would give more accurate estimates than the PUL method in general (e.g., for non-Feistel ciphers, or for large number of rounds where the PUL method gives too small values). For this purpose, we analyze the theories of correlation matrices, linear hulls, and statistical sampling. We give an example application of correlation matrices for bias estimation, which gives consistently better estimates than the PUL method on RC5. We review the theory of linear hulls, which has also been used as an alternative technique for bias estimation. We point out certain problems with some current applications of linear hulls where the application has no basis in theory. Finally, we look at the prospects of estimating the bias by statistical techniques over a randomly generated sample of plaintext/ciphertext blocks. It turns out that the statistical techniques do not provide any practical solutions for bias estimation, especially when the inverse square of the bias is an impractically large amount for a sample size.

Notation: Throughout the paper, we use n to denote the block size and r to denote the number of rounds in an iterated block cipher. K_i denotes the ith round key, L_i and R_i denote the left and right halves of the round output. p is used for the probability of an approximation, where $|p - 1/2|$ is the bias. Bits in a block are numbered from right to left, beginning with 0. The "$\cdot$" operator denotes the bitwise dot product.

2 Experiments with RC5

During some linear cryptanalysis experiments with small block sizes of RC5, we noticed significant differences between the actual bias of a linear approximation and the values that were estimated by the PUL method. We summarize these findings in this section.

The RC5 encryption function is:

[1] One exception in this regard is [2], where the accuracy of PUL estimates was studied in the specific context of combining two neighbor s-box approximations in DES.

```
$L_1 = L_0 + K_0$
$R_1 = R_0 + K_1$
for $i = 2$ to $2r + 1$ do
    $L_i = R_{i-1}$
    $R_i = ((L_{i-1} \oplus R_{i-1}) \lll R_{i-1}) + K_i$
```

The best currently-known linear approximation of RC5 [8] is

$$R_0[0] \oplus L_{2r}[0] = K_1[0] \oplus K_3[0] \oplus \cdots \oplus K_{2r-1}[0], \tag{1}$$

The probability of this approximation is estimated as $1/2+1/2w^{r-1}$ by the PUL method where w is the word size in bits (in RC5, half of a block is called a *word*).

We computed the bias of Approximation (1) by exhaustively going over all plaintext blocks for various values of w and r. The test results are summarized in Table 1. The results show quite significant differences between the actual and the estimated values of the bias. Another remarkable point is that increasing the number of rounds does not affect the bias after a certain point, and the bias does not get much smaller than 2^{-w-1}. We further discuss these results in Section 4.

r	Bias	PUL
2	$2^{-3.0}$	2^{-3}
3	$2^{-4.5}$	2^{-5}
4	$2^{-5.1}$	2^{-7}
5	$2^{-5.3}$	2^{-9}
10	$2^{-5.3}$	2^{-19}

(a) $w = 4$

r	Bias	PUL
3	$2^{-5.8}$	2^{-7}
4	$2^{-7.7}$	2^{-10}
5	$2^{-8.8}$	2^{-13}
6	$2^{-9.1}$	2^{-16}
10	$2^{-9.2}$	2^{-28}

(b) $w = 8$

r	Bias	PUL
4	$2^{-10.4}$	2^{-13}
5	$2^{-12.1}$	2^{-17}
6	$2^{-14.6}$	2^{-21}
7	$2^{-16.3}$	2^{-25}
10	$2^{-17.3}$	2^{-37}

(c) $w = 16$

Table 1. Average actual bias of Approximation (1) and the bias estimated by the PUL for various values of w and r, with 500 randomly chosen keys for each w and r. The results show a significant difference between the actual bias values and the PUL estimates. The difference increases sharply with increasing number of rounds.

3 Experiments with Feistel Ciphers

Following the findings on RC5 described in Section 2, we performed similar tests with Feistel ciphers, which is the type of cipher the PUL method was originally used for [11]. In this section, we describe these tests and summarize their results.

3.1 Design

The ciphers used in these experiments are Feistel ciphers with 32-bit block sizes. The encryption function of a Feistel cipher is of the following form:

```
for $i = 1$ to $r$ do
    $L_i = R_{i-1}$
    $R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$
```

The F function we used in the experiments has a structure similar to that in DES. It has a sequence of key addition, substitution, and permutation stages. In the key addition stage, R_{i-1} is XORed by the round key K_i, with a possible bit expansion before the XOR. The Feistel ciphers used in the experiments include both those with an expansion function E and those without one. In ciphers with a bit expansion, the expansion function E is the equivalent of the expansion function in DES, reduced to 16×24 bits.

The substitution stage is also similar to that in DES, with four parallel distinct s-boxes. The s-boxes are either 4×4 or 6×4, depending on the presence or absence of the bit expansion. The 4×4 s-boxes are chosen from the s-boxes of Serpent [1], and the 6×4 s-boxes are chosen from the s-boxes of DES [6].

The permutation stage uses a 16×16 permutation function P to mix the output bits from the s-boxes. In the Serpent s-boxes, unlike the DES s-boxes [6], there is no distinction among the role of input bits. So, with the Serpent s-boxes we use the following simple permutation P which guarantees that each output bit from an s-box affects a different s-box in the next round: $P =$ (15 11 7 3 14 10 6 2 13 9 5 1 12 8 4 0).[2] With the DES s-boxes, we use the following permutation which guarantees that the output of each s-box has an effect on two outer bits, two inner non-middle bits, and two middle bits (of different s-boxes) in the next round: $P =$ (12 10 3 6 14 11 4 2 15 9 7 1 13 8 5 0).

Notation: Each Feistel cipher used in the tests will be identified by the presence/absence of the expansion in key addition, and by the numbers of the s-boxes used (ordered from left to right). For the numbering of the s-boxes, the numbering in their original ciphers will be used. E.g. FC$_{NE}$0214 will denote the Feistel cipher with no bit expansion and with the Serpent s-boxes S_0, S_2, S_1, S_4. FC$_E$8735 will denote the Feistel cipher with the bit expansion E and with the DES s-boxes S_8, S_7, S_3, S_5.

We start our experiments with the FC$_{NE}$ ciphers which are the simpler case since there is no issue of duplicate bits. We first look at the approximations with at most a single active s-box at each round. Then we go to the approximations with multiple s-boxes in the same round.

3.2 Approximations with Single Active S-Box

First, we look at how the PUL method performs on the approximations with at most a single active s-box at every round, which is the most basic type of linear approximations of an SP-structured Feistel cipher. We denote the round approximations in terms of the s-box approximations, as in [11,12].

We consider the 4-round iterative approximations of the form ABC–, which can be combined by itself as ABC–CBA–ABC–..., where A, B and C are some

[2] The numbers show the new location of the bits after permutation.

non-trivial approximations of the F function, and "–" denotes the nil approximation. This is the form of the approximations which gave the highest bias on DES [11,12], and which also gives the highest bias on our FC_{NE} ciphers. Here we present the results for three of our test cases. The results are summarized in Table 2. The bias is denoted by b.

Case 1.1: Cipher: $\mathrm{FC}_{NE}1745$, A: $4{\cdot}x = 11{\cdot}S_1(x)$, $b = 1/4$, B: $8{\cdot}x = 8{\cdot}S_7(x)$, $b = 1/8$, C: $4{\cdot}x = 15{\cdot}S_1(x)$, $b = 1/4$.
Case 1.2: Cipher: $\mathrm{FC}_{NE}6530$, A: $2{\cdot}x = 13{\cdot}S_5(x)$, $b = 1/4$, B: $4{\cdot}x = 4{\cdot}S_3(x)$, $b = 1/8$, C: $2{\cdot}x = 15{\cdot}S_5(x)$, $b = 1/4$.
Case 1.3: Cipher: $\mathrm{FC}_{NE}0214$, A: $4{\cdot}x = 8{\cdot}S_1(x)$, $b = 1/8$, B: $2{\cdot}x = 2{\cdot}S_2(x)$, $b = 1/8$, C: $4{\cdot}x = 12{\cdot}S_1(x)$, $b = 1/8$.

r	Bias	PUL
4	$2^{-5.00}$	2^{-5}
8	$2^{-9.00}$	2^{-9}
12	$2^{-12.96}$	2^{-13}
16	$2^{-16.51}$	2^{-17}
20	$2^{-17.30}$	2^{-21}
24	$2^{-17.31}$	2^{-25}

(a) Case 1.1

r	Bias	PUL
4	$2^{-5.00}$	2^{-5}
8	$2^{-9.00}$	2^{-9}
12	$2^{-12.86}$	2^{-13}
16	$2^{-16.83}$	2^{-17}
20	$2^{-17.26}$	2^{-21}
24	$2^{-18.28}$	2^{-25}

(b) Case 1.2

r	Bias	PUL
4	$2^{-7.00}$	2^{-7}
8	$2^{-12.94}$	2^{-13}
12	$2^{-16.75}$	2^{-19}
16	$2^{-17.35}$	2^{-25}
20	$2^{-17.84}$	2^{-31}
24	$2^{-17.42}$	2^{-37}

(c) Case 1.3

Table 2. Test results for single-sbox approximations. PUL estimates are quite accurate, as long as they are above $2^{-\frac{n}{2}-1}$. Like the results on RC5, the bias does not go much below $2^{-\frac{n}{2}-1}$.

3.3 Approximations with Multiple S-Box

In this section, we look at how having multiple active s-boxes in the same round affects the accuracy of PUL estimation. We focus our experiments on approximations with two s-boxes, because in our miniature ciphers the bias with three or four active s-boxes drop too fast to draw any useful conclusions.

We work with the 3-round iterative approximations of the form AB–, which can be combined by itself as AB–BA–AB–..., where A and B are approximations of the F function with two active s-boxes. Three such approximations are given below. The results are summarized in Table 3.

Case 2.1: Cipher: $\mathrm{FC}_{NE}5614$, A: ($3{\cdot}x = 3{\cdot}S_4(x)$, $b = 1/4$) AND ($3{\cdot}x = 3{\cdot}S_5(x)$, $b = 1/4$) B: ($9{\cdot}x = 9{\cdot}S_1(x)$, $b = 1/4$) AND ($9{\cdot}x = 9{\cdot}S_4(x)$, $b = 1/4$)
Case 2.2: Cipher: $\mathrm{FC}_{NE}4250$, A: ($10{\cdot}x = 10{\cdot}S_0(x)$, $b = 1/4$) AND ($10{\cdot}x = 10{\cdot}S_5(x)$, $b = 1/4$) B: ($3{\cdot}x = 3{\cdot}S_4(x)$, $b = 1/4$) AND ($3{\cdot}x = 3{\cdot}S_5(x)$, $b = 1/4$)
Case 2.3: Cipher: $\mathrm{FC}_{NE}5014$, A: ($3{\cdot}x = 3{\cdot}S_4(x)$, $b = 1/4$) AND ($3{\cdot}x = 3{\cdot}S_5(x)$, $b = 1/4$) B: ($9{\cdot}x = 9{\cdot}S_1(x)$, $b = 1/4$) AND ($9{\cdot}x = 9{\cdot}S_4(x)$, $b = 1/4$)

r	Bias	PUL
3	$2^{-5.00}$	2^{-5}
6	$2^{-8.94}$	2^{-9}
9	$2^{-12.94}$	2^{-13}
12	$2^{-16.99}$	2^{-17}
15	$2^{-17.48}$	2^{-21}
18	$2^{-18.29}$	2^{-25}

(a) Case 2.1

r	Bias	PUL
3	$2^{-5.00}$	2^{-5}
6	$2^{-8.90}$	2^{-9}
9	$2^{-12.90}$	2^{-13}
12	$2^{-16.54}$	2^{-17}
15	$2^{-17.17}$	2^{-21}
18	$2^{-17.04}$	2^{-25}

(b) Case 2.2

r	Bias	PUL
3	$2^{-5.00}$	2^{-5}
6	$2^{-8.91}$	2^{-9}
9	$2^{-12.88}$	2^{-13}
12	$2^{-16.94}$	2^{-17}
15	$2^{-17.19}$	2^{-21}
18	$2^{-17.37}$	2^{-25}

(c) Case 2.3

Table 3. Test results for approximations with two active s-boxes in a round. PUL estimates are somewhat less accurate than those in Table 2 for single-sbox approximations, but still better than those on RC5.

3.4 Approximations with Expansion

Here we look at the effect of having an expansion function at the key addition stage. When there is an expansion at the key addition stage like the E function in DES and our FC$_E$ ciphers, an approximation of an s-box not only affects the input to that active s-box, but also affects the two shared input bits with the neighbor s-boxes. Therefore, the output of the neighbor s-boxes will also be more or less affected by an s-box approximation.

We tested the accuracy of the PUL estimates with certain approximations of the FC$_E$ ciphers. The tests are focused on approximations with a single active s-box at every round, because the bias of approximations with multiple active s-boxes drops too fast in FC$_E$s. The approximations used in the tests are iterative approximations of the form ABC–CBA–ABC–... The tested approximations are listed below, and the results are summarized in Table 4.

Case 3.1: Cipher: FC$_E$5216, A: $16{\cdot}x = 15{\cdot}S_5(x)$, $b = 20/64$, B: $8{\cdot}x = 8{\cdot}S_1(x)$, $b = 4/64$, C: $16{\cdot}x = 14{\cdot}S_5(x)$, $b = 10/64$.

Case 3.2: Cipher: FC$_E$8735, A: $16{\cdot}x = 7{\cdot}S_5(x)$, $b = 8/64$, B: $4{\cdot}x = 2{\cdot}S_8(x)$, $b = 2/64$, C: $16{\cdot}x = 15{\cdot}S_5(x)$, $b = 20/64$.

3.5 Other Approximations

The ciphers and approximations considered in these tests are by no means exhaustive, and in fact there are many different Feistel ciphers and approximations possible. The purpose of the tests is not to exhaustively prove a result about the bias of Feistel ciphers, but to obtain a general view of the accuracy of PUL estimation on Feistel ciphers. As we will discuss in Section 4, these results indeed give a general idea on the subject.

r	Bias	PUL
4	$2^{-6.36}$	$2^{-6.36}$
8	$2^{-11.71}$	$2^{-11.71}$
12	$2^{-16.74}$	$2^{-17.06}$
16	$2^{-17.31}$	$2^{-22.42}$
20	$2^{-17.62}$	$2^{-27.78}$

(a) Case 3.1

r	Bias	PUL
4	$2^{-7.68}$	$2^{-7.68}$
8	$2^{-14.40}$	$2^{-14.36}$
12	$2^{-17.26}$	$2^{-21.03}$
16	$2^{-17.70}$	$2^{-27.71}$
20	$2^{-17.47}$	$2^{-34.39}$

(b) Case 3.2

Table 4. Test results for single-sbox approximations with the expansion E. PUL estimates are slightly less accurate than those without E, given in Table 2.

4 Discussion on the Results

The test results on Feistel ciphers show that the PUL method is quite effective for bias estimation with SP-structured Feistel ciphers, as long as the estimated values are significantly higher than $2^{-\frac{n}{2}}$. The results are best when there is at most a single active s-box in each round approximation, and when there is no bit expansion. When there are more affected s-boxes in a round approximation, the number of bits affected by the approximation increases, and so does the effect it has on the following round approximations (i.e. dependence among round approximations).

The test results on RC5 show that accuracy of the PUL estimates may not be so good with ciphers that are not SP-structured Feistel ciphers. In the test results with RC5, there is a considerable difference between the estimated and actual values even for smaller number of rounds. With larger number of rounds, the bias may be significantly higher than $2^{-\frac{n}{2}}$ even after the estimated values become lower than $2^{-\frac{n}{2}}$. We can say, looking at the test results, that larger differences should be expected in practice with larger block sizes (i.e. with 64- and 128-bit blocks).

It is not easy to explain the difference in the accuracy of the estimates with RC5 and with Feistel ciphers: The source of inaccuracy of a PUL estimate is the dependence between round approximations, which is a factor that has to be neglected by the PUL method by its very definition. Both the RC5 round approximations and the single-sbox FC_{NE} round approximations affect (i.e., give information on) 4 out of 16 bits of R_{i-1}. Moreover, in the FC_{NE} approximations, there are three non-trivial round approximations in every four rounds, whereas in the RC5 approximation there are only two non-trivial approximations in four (half)rounds. So, it would be natural to expect that there would be more dependence and interaction among the FC_{NE} round approximations, which turns out not to be the case. For now, we accept the accuracy of the PUL estimates on Feistel ciphers as an experimental result and do not go into an in-depth analysis of the factors underlying it.

Similarly, we cannot give a simple explanation for why the actual and the estimated bias values for the Feistel ciphers go so closely until they reach the $2^{-\frac{n}{2}}$ threshold, and become so divergent after that point. It is not possible to simply explain this with the accumulation of the dependence affect with more rounds, since a comparison of the test results with low-bias and high-bias approximations suggests that the point where the actual and the estimated values diverge is not determined by the number of rounds, but is mostly determined by the proximity to the $2^{-\frac{n}{2}}$ threshold.

4.1 Stabilization of the Bias

Here we give two theorems related to the stabilization of the bias around $2^{-\frac{n}{2}-1}$. Although the theorems do not explain how the sharp change in accuracy of PUL estimates at $2^{-\frac{n}{2}}$ is related to the dependence between round approximations, they provide some important facts about the stabilization of the bias. The first theorem gives a lower bound for the bias of the best approximation. The second theorem suggests that when the bias of the best approximation approaches the theoretical lower bound, the bias of almost all linear approximations of the cipher should be around $2^{-\frac{n}{2}-1}$. The first theorem follows from Theorem 4 in [4] with $p = q = n$ (see Appendix A). The second theorem is observed in the proof of that theorem. F_k is an n-bit block cipher with key $K = k$; $p_{a,b}$ denotes the probability of the approximation $a \cdot X \oplus b \cdot F_k(X) = 0$.[3]

Theorem 1. *$\exists\ a, b \in \{0,1\}^n$, such that $|p_{a,b} - 1/2| \geq 2^{-\frac{n+1}{2}}$.*

Theorem 2. $\sum_{a,b \in \{0,1\}^n} |p_{a,b} - 1/2|^2 = 2^{n-2}$.

5 Alternative Methods for Bias Estimation

In this section, we analyze several theories for an alternative method for bias estimation, including correlation matrices, linear hulls and statistical sampling.

5.1 Correlation Matrices

For a function $f : \{0,1\}^m \to \{0,1\}^n$, the *correlation matrix* $C^{(f)}$ is a $2^n \times 2^m$ matrix whose (b, a)th entry $c_{ba}^{(f)}$ is the *correlation coefficient* $2P_X(a \cdot X = b \cdot f(X)) - 1$ [7]. The relationship between the correlation coefficients and the bias is straightforward: If F_k is a block cipher with key $K = k$, bias of $a \cdot X \oplus b \cdot F_k(X) = d \cdot K$ (for any d) equals $|c_{ba}^{(F_k)}|/2$. For $f : \{0,1\}^\ell \to \{0,1\}^m$, $g : \{0,1\}^m \to \{0,1\}^n$, we have $C^{(g \circ f)} = C^{(g)} \times C^{(f)}$. So, if F_k is an iterative cipher and f_i is the ith round with its respective round key, we have $C^{(F_k)} = \prod_{i=1}^r C^{(f_i)}$. Then $c_{ba}^{(F_k)}$ equals $\sum_{a_1, a_2, \ldots, a_{r-1}} (\prod_{i=1}^r c_{a_i a_{i-1}}^{(f_i)})$ where $a_0 = a$, $a_r = b$, and each $\prod_{i=1}^r c_{a_i a_{i-1}}^{(f_i)}$

[3] The bias does not depend on the key mask here, because the key is a fixed parameter (which is also the case in a linear attack).

is known as the *correlation contribution coefficient (CCC)* of the *linear trail* $a_0 \to a_1 \to \ldots \to a_r$.

So, the theory of correlation matrices tells that the bias of a linear approximation is equal to the sum of the PUL biases (without absolute value) of all linear trails that lead to the given approximation. Hence, correlation matrices provide a generalization of the PUL method: Instead of using the PUL bias of a single linear trail, the bias can be estimated by summing up the PUL bias of as many linear trails as possible which lead to the given approximation. We will refer to this generalization of the PUL method as the *CM method*.

An Example Application: As an example, we apply the CM method to our RC5 approximation (1). All effective RC5 approximations obtained so far [8,9,16,3] are based on round approximations with a single active input and output bit. To be expandable over multiple rounds, the 1-bit round approximations should be of the form $R_i[m_i] \oplus L_{i-1}[m_i'] = S_i[m_i] \oplus c$ where $m_i, m_i' < \lg w$ and c is a constant. These approximations are analyzed in detail by Borst et al. [3]. Probability of the 1-round approximation

$$R_i[m_i] \oplus L_{i-1}[m_i'] = S_i[m_i] \oplus (m_i - m_i')[m_i'],$$

where $(m_i - m_i')[m_i']$ denotes the m_i'th bit of $(m_i - m_i') \bmod w$, is equal to $\frac{1}{2} + \frac{1}{w}\left(\frac{1}{2} - \frac{s}{2^{m_i}}\right)$, where s denotes S_i mod 2^{m_i}. Hence, the correlation coefficient of $R_i[m_i] = L_{i-1}[m_i']$ is equal to $(-1)^{\delta}\frac{1}{w}\left(1 - \frac{s}{2^{m_i-1}}\right)$, where $\delta = S_i[m_i] \oplus (m_i - m_i')[m_i']$. The 1-bit trails that lead to Approximation (1) are those which satisfy,

$$\begin{aligned} &m_1 = 0, \\ &m_i = m_{i+2}' < \lg w, \quad \text{for } i = 3, 5, \ldots, 2r-3, \\ &m_{2r-1} = 0. \end{aligned}$$

We computed the bias by adding up the correlation contribution coefficients of all 1-bit linear trails of this form. The results are given in Figure 1 and are compared to the actual bias and the PUL estimate values.

	Bias		
r	Actual	PUL	CM
2	2^{-5}	2^{-5}	2^{-5}
4	$2^{-10.4}$	2^{-13}	$2^{-12.2}$
6	$2^{-14.6}$	2^{-21}	$2^{-19.2}$
8	$2^{-17.2}$	2^{-29}	$2^{-26.2}$
10	$2^{-17.3}$	2^{-37}	$2^{-33.3}$

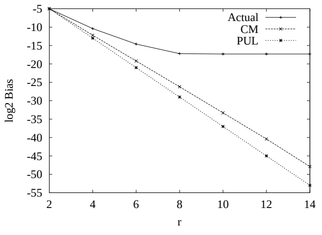

Fig. 1. Comparison of the CM, PUL and actual bias values for $w = 16$ over the key sample used in Table 1. The CM estimates are consistently better than the PUL estimates. But their accuracy too drops exponentially with the number of rounds.

We would like to note that this example application on RC5 is intended to illustrate the practical usage of the theory of correlation matrices; it does not show the limits of the theory. In fact, it is possible to obtain better estimates than those given in Figure 1 by including multiple-bit trails in bias estimation as well as the single-bit trails. But eventually the accuracy of the estimates should be expected to drop with the increasing number of rounds, since the number of trails that can be considered in bias estimation can be no more than a certain tractable number; but the number of all trails that contribute to the bias of an approximation increases exponentially with the number of rounds.

5.2 Linear Hulls and Correlation Matrices

Like correlation matrices, linear hulls [14] can also be used to combine the bias of linear trails. But unlike correlation matrices, this kind of application of linear hulls is proven specifically for DES-like ciphers[4]. In fact, in the Fundamental Theorem of Linear Hulls (see Appendix B), the summation for the average squared bias is over different key masks, not over linear trails. But since in a DES-like cipher there is a one-to-one correspondence between a linear trail and a key mask, the summation can be transformed into a summation over linear trails (see Theorem 2 in [14]).[5] However, this argument is not equally applicable to all ciphers. For example, for the addition operation $X = Y + K$, the input and output masks for the round are not uniquely determined by the key mask; i.e., for a given $c_i > 1$, there are many values of a_i and b_i such that the approximation $a_i \cdot X = b_i \cdot Y + c_i \cdot K$ has a non-zero bias. So, for example in RC5, there may be many linear trails corresponding to the same key mask c.

In short, we would like to point out that the application of linear hulls to combine bias from linear trails has been proven specifically for DES-like ciphers, and it should not be used with arbitrary ciphers unless a proof is given for that application. In this respect, certain bias studies with linear hulls (e.g. [3,5]) have no theoretical basis.[6]

Another confusion with the application of linear hulls is that, linear hulls are often taken as the exact analog of differentials in differential cryptanalysis; i.e., it is assumed that $|bias| = \sum_{LT(a,b)} |PUL\ bias|$ where $\sum_{LT(a,b)}$ denotes the summation over all linear trails with plaintext mask a, ciphertext mask b. Obviously, this equation has no basis in the theory of linear hulls.[7] A similar but correct equation is the one given by correlation matrices where bias is taken without absolute value; $bias = \sum_{LT(a,b)}(PUL\ bias)$. So, even though it is wrong to use $\sum_{LT(a,b)} |PUL\ bias|$ for bias estimation, it can be used as an upper-bound for bias in analyzing the security of a cipher against linear cryptanalysis.

[4] For a formal definition of DES-like ciphers for linear hulls, see [14,13].

[5] This theorem on combining the squared bias of linear trails in a DES-like cipher is recently given an alternative proof by Nyberg [15], which is based on correlation matrices rather than linear hulls.

[6] A mid-solution for these applications can be possible if each trail used in bias estimation can be shown to match a different key mask.

[7] Simply note that every equation in linear hulls is in terms of squared bias rather than the bias itself.

However, the correct reference for this kind of application should be correlation matrices rather than linear hulls.

5.3 Estimation by Sampling

To estimate a parameter of a population where the population is too large to count every member, a random sample from the population can be used to estimate the desired parameter. For a typical block cipher size (e.g. 64 or 128 bits), there are too many plaintext/ciphertext blocks to calculate the actual bias by going over every block; so, a random sample of blocks can be used to estimate the bias. Here we look at a number of alternative statistical estimators for estimating the bias over a random sample of plaintext blocks. Throughout this section, N is the sample size, T is the number of ciphertexts in the sample satisfying the approximation. $E[.]$ denotes the expected value, $Var[.]$ denotes the variance, θ denotes the bias $|p-1/2|$. $\hat{\theta}$ is used for estimators for θ. MSE denotes the mean squared error, $E[(\hat{\theta}-\theta)^2]$.

The UMVUE: One of the most important point estimators in statistics is the uniform minimum variance unbiased estimator (UMVUE), which is the (unique) unbiased estimator that has the minimum variance among all unbiased estimators, under all values of the parameter to be estimated [10]. Regarding the UMVUE, we prove the following negative result:

Theorem 3. *No unbiased estimator exists for $|p-1/2|$ over a random plaintext sample.*

Proof. T is binomially distributed. Assume $\hat{\theta}_N(T)$ is an unbiased estimator[8]:

$$E[\hat{\theta}_N] = \sum_{T=0}^{N} \hat{\theta}_N(T)\binom{N}{T}p^T(1-p)^{N-T} = |p-1/2|, \tag{2}$$

for all $0 \le p \le 1$. Now, define $\rho = p/(1-p)$ so that $p = \rho/(1+\rho)$ and $1-p = 1/(1+\rho)$. For $1/2 \le p < 1$, Equation (2) becomes

$$\sum_{T=0}^{N} \hat{\theta}_N(T)\binom{N}{T}\rho^T = (1+\rho)^{N-1}(\rho-1)/2 = \sum_{T=0}^{N}(\binom{N-1}{T-1} - \binom{N-1}{T})/2\rho^T,$$

$1 \le \rho < \infty$. A comparison of the coefficients of ρ^T on the left and right sides leads to $\hat{\theta}_N(T) = T/N - 1/2$. Similarly, for $p < 1/2$ we obtain $\hat{\theta}_N(T) = 1/2 - T/N$. Obviously, $\hat{\theta}_N$ cannot satisfy both of these equations. □

Corollary 1. *The UMVUE does not exists for $|p-1/2|$.*

[8] We can denote the estimator as a function of T since T is a sufficient statistics [10] for $|p-1/2|$.

The Sample Bias: It may seem like a good idea to use the sample bias $|T/N - 1/2|$ as an estimator for the actual bias $|p-1/2|$. In this section we show that the sample bias cannot be used to estimate the bias when the sample size is much smaller than $|p-1/2|^{-2}$.

$T/N - 1/2$ approximately follows a normal distribution with mean $\mu = p - 1/2$ and variance $\sigma^2 = p(1-p)/N \approx 1/4N$. For $\hat{\theta} = |T/N - 1/2|$, it can be shown

$$\begin{aligned}
E[\hat{\theta}] &= |\mu|(1 - 2\Phi(-|\mu|/\sigma)) + 2\sigma\phi(|\mu|/\sigma) \\
Var[\hat{\theta}] &= \mu^2 + \sigma^2 - E[\hat{\theta}]^2 \\
MSE(\hat{\theta}) &= E[(\hat{\theta} - \theta)^2] \;=\; Var[\hat{\theta}] + (E[\hat{\theta}] - |\mu|)^2 \\
&= \sigma^2 + 4|\mu|^2\Phi(-|\mu|/\sigma) - 4\mu\sigma\phi(|\mu|/\sigma)
\end{aligned}$$

where ϕ and Φ denote respectively the probability density function and the cumulative distribution function for the standard normal distribution. As it can be seen from Figure 2, to have the standard error $\sqrt{MSE}$ at least comparable to $|p-1/2|$, a sample size comparable to $|p-1/2|^{-2}$ will be needed. Therefore, when $|p-1/2|^{-2}$ is an intractably large number (which should be the case for a secure cipher), it will not be possible to obtain a reliable estimator from $|T/N - 1/2|$ with any practical sample size.[9]

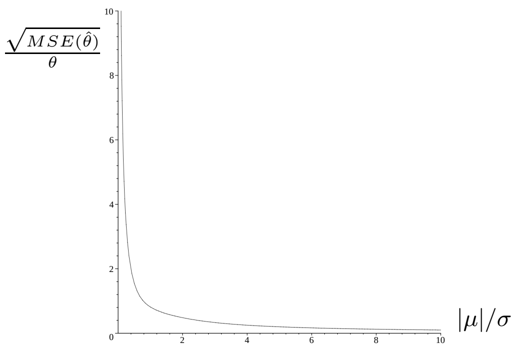

Fig. 2. Standard error rate vs. $|\mu|/\sigma$, for $\hat{\theta} = |T/N - 1/2|$. It converges to $1/(|\mu|/\sigma)$ in both directions. Sample size for a desired error rate $\sqrt{MSE}/\theta \le e$ can be computed from $|\mu|/\sigma = |p - 1/2|\sqrt{4N} \ge 1/e$, hence $N \ge \frac{1}{4e^2}|p-1/2|^{-2}$.

If a sample size much smaller than $|p-1/2|^{-2}$ is used, then $E[\hat{\theta}] \approx 1/\sqrt{2\pi N}$, independent of $|p-1/2|$. As an example, Table 5 gives the results of a computation of the sample bias of the RC5 approximation (1) for $w = 32$ with $N = 10^7$ plaintexts. For this sample size, we have $1/\sqrt{2\pi 10^7} = 2^{-12.95}$, which explains the stabilization of the bias around 2^{-13}.

[9] This sample size requirement should not be confused with the similar plaintext requirement for an attack.

r	2	4	6	8	10	12
Bias	$2^{-6.0}$	$2^{-12.4}$	$2^{-13.0}$	$2^{-13.0}$	$2^{-12.9}$	$2^{-13.0}$

Table 5. Average sample bias of the RC5 approximation (1) for $w = 32$ with 10^7 plaintexts, on 500 randomly chosen keys for each r. The results show an alarmingly high bias for a 64-bit block cipher.

The MLE: Another important point estimator in statistics is the maximum likelihood estimator (MLE). The MLE for $|p - 1/2|$ would be the value $\hat{\theta}^*$ that maximizes the likelihood function

$$L(\hat{\theta}) = \begin{cases} (1/2 - \hat{\theta})^T (1/2 + \hat{\theta})^{N-T} + (1/2 - \hat{\theta})^{N-T} (1/2 + \hat{\theta})^T, & \text{if } \hat{\theta} \neq 0 \\ (1/2)^N, & \text{if } \hat{\theta} = 0 \end{cases} \tag{3}$$

Unfortunately, there is no easy way to compute $\hat{\theta}^*$. Nevertheless, we can obtain a bound on the reliability of the MLE by assuming availability of some extra information, such as whether or not $p > 1/2$. If we know $p > 1/2$, then

$$\hat{\theta}^* = \begin{cases} 0, & \text{if } T < N/2 \\ T/N - 1/2, & \text{otherwise} \end{cases}$$

and vice versa for $p < 1/2$; which is not any more reliable than the sample bias.

6 Conclusions

Looking at the tests summarized in this paper, we conclude that the PUL method gives quite accurate estimates with SP-structured Feistel ciphers, especially for approximations with a single active s-box per round. With increasing number of rounds, the actual bias values follow the PUL estimates quite closely until the PUL estimates become much less than $2^{-\frac{n}{2}}$. After that point the actual bias remains stabilized around $2^{-\frac{n}{2}-1}$ and does not get much lower.

The experiments on RC5 show that the performance of the PUL method may not be as good with other kinds of ciphers. In the case of the RC5 approximation tested, there is a considerable difference between the estimated and actual values even for small number of rounds. At certain cases, the bias is significantly higher than $2^{-\frac{n}{2}}$ even after the estimated values become lower than $2^{-\frac{n}{2}}$. The inaccuracy of the PUL estimates increases with larger block sizes, so even greater differences between actual and estimated values should be expected with 64- and 128-bit blocks.

We analyzed several other techniques for an alternative estimation method that would give more accurate estimates than the PUL method in general. Our attempts to obtain good estimators by statistical techniques from a random sample of plaintext blocks did not provide any useful results, especially when the inverse square of the bias is an impractically large amount for a sample size.

The theory of correlation matrices provides some opportunities for an alternative estimation method. By this theory, it may be possible to obtain improvements over the PUL method by using more than a single trail for bias estimation.

We gave an example of such an application on RC5. The method gave some improvements over the PUL method. But eventually its estimates also fell far from the actual bias values with increasing number of rounds. The main reason for this deviation is that the number of trails that can be considered in bias estimation can be no more than a certain tractable number; but the number of all trails that contribute to the bias of an approximation increases exponentially with the number of rounds.

Another theory used as an alternative method for bias estimation is the theory of linear hulls. In Section 5.2, we pointed out some problems with the current applications of this theory. The main problem with the current practice is that, the theoretical results on linear hulls regarding combining the bias of different linear trails is proven only for DES-like ciphers, whereas in practice these results are used for different kinds of ciphers (e.g., RC5, RC6, SAFER).

We conclude that the PUL method is quite an effective method for bias estimation with SP-structured Feistel ciphers, especially for approximations with at most one active s-box at each round and with a bias considerably higher than $2^{-\frac{n}{2}}$. It is an open problem to find an equally effective method for non-Feistel ciphers and for the ciphers with too many rounds for the PUL method to give a meaningful value.

Acknowledgments

I would like to thank Kaisa Nyberg for her comments on linear hulls, to Erkan Türe for his help with the statistics and especially for reference [10], and to Ali Bıçak, Matt Robshaw, and Lisa Yin for many helpful comments on the paper.

References

1. Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A Proposal for the Advanced Encryption Standard. Available from http://www.nist.gov/aes. 55
2. Uwe Blöcher and Markus Dichtl. Problems with the linear cryptanalysis of DES using more than one active S-box per round. In *Fast Software Encryption*, 1994. 53
3. Johan Borst, Bart Preneel, and Joos Vandewalle. Linear cryptanalysis of RC5 and RC6. In *Fast Software Encryption, 6th International Workshop*, 1999. 60, 60, 61
4. Florent Chabaud and Serge Vaudenay. Links between differential and linear cryptanalysis. In *Advances in Cryptology—Eurocrypt'94*. Springer-Verlag, 1994. 59, 66
5. S. Contini, R. Rivest, M. Robshaw, and L. Yin. The Security of the RC6 Block Cipher. Available from http://www.rsasecurity.com/rsalabs/aes. 61
6. Don Coppersmith. The Data Encryption Standard (DES) and its strength against attacks. *IBM Journal of Research and Development*, May(3):243–250, 38 1994. 55, 55
7. Joan Daemen, René Govaerts, and Joos Vandewalle. Correlation matrices. In *Fast Software Encryption, Second International Workshop*. Springer-Verlag, 1994. 59
8. Burton S. Kaliski Jr. and Yiqun Lisa Yin. On differential and linear cryptanalysis of the RC5 encryption algorithm. In *Advances in Cryptology—Crypto'95*. 54, 60

9. Burton S. Kaliski Jr. and Yiqun Lisa Yin. On the security of the RC5 encryption algorithm. Technical Report TR-602, Version 1.0, RSA Laboratories, 1998. 60
10. E. L. Lehmann and George Casella. *Theory of Point Estimation.* Springer Texts in Statistics. Springer-Verlag, 2nd edition, 1998. 62, 62, 65
11. Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology—Eurocrypt'93.* Springer-Verlag, 1993. 52, 54, 55, 56
12. Mitsuru Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In *Advances in Cryptology—Crypto'94.* Springer-Verlag, 1994. 55, 56
13. Mitsuru Matsui. New structure of block ciphers with provable security against differential and linear cryptanalysis. In *Fast Software Encryption*, 1996. 61
14. Kaisa Nyberg. Linear approximation of block ciphers. In *Advances in Cryptology—Eurocrypt'94.* Springer-Verlag, 1994. 61, 61, 61, 66, 66
15. Kaisa Nyberg. Correlation theorems in cryptanalysis. To appear in *Discrete Applied Mathematics.* 61
16. Ali Aydın Selçuk. New results in linear cryptanalysis of RC5. In *Fast Software Encryption, 5th International Workshop.* Springer-Verlag, 1998. 60

A Chabaud-Vaudenay Theorem on Max. Non-Linearity

Theorem 4 (in Chabaud-Vaudenay [4]) *For $K = \{0,1\}$ and $F : K^p \to K^q$,*

$$\Lambda_F \geq \frac{1}{2}\left(3 \times 2^p - 2 - 2\frac{(2^p-1)(2^{p-1}-1)}{2^q-1}\right)^{1/2}$$

where $\Lambda_F = \max_{b\neq 0,a} \left|\ |\{x \in K^p \ : \ a{\cdot}x \oplus b{\cdot}F(x) = 0\}| - \frac{|K^p|}{2}\ \right|$.

B Fundamental Theorem of Linear Hulls

Theorem 1 (in Nyberg [14]) *For $X \in \{0,1\}^m$, $K \in \{0,1\}^\ell$, $F : \{0,1\}^m \times \{0,1\}^\ell \to \{0,1\}^n$, if X and K are independent random variables, and K is uniformly distributed, then for all $a \in \{0,1\}^m$, $b \in \{0,1\}^n$*

$$2^{-\ell} \sum_{k\in\{0,1\}^\ell} |P_X(a \cdot X \oplus b \cdot F(X,k) = 0) - \frac{1}{2}|^2 \ = \sum_{c\in\{0,1\}^\ell} |P_{X,K}(a \cdot X \oplus b \cdot F(X,K) \oplus c \cdot K = 0) - \frac{1}{2}|^2$$

During a linear attack, the key K is a fixed parameter, so the bias of interest is the bias on the left side of the equation; i.e., $|P_X(a{\cdot}X \oplus b{\cdot}F(X,k) = 0) - \frac{1}{2}|$.[10] The summation on the right is the squared bias with a random-variable key, over all key masks c. For DES-like ciphers, this right-hand side summation can be turned into a summation of the PUL bias over linear trails (Theorem 2 in [14]).

[10] The key mask does not matter here since the key is fixed.

On the Incomparability of Entropy and Marginal Guesswork in Brute-Force Attacks

John O. Pliam

Laboratory for Security and Cryptography (LASEC),
Swiss Federal Institute of Technology, Lausanne (EPFL),
LASEC–DSC–EPFL, CH–1015 Lausanne, Switzerland
john.pliam@epfl.ch

Abstract. We discuss measures of statistical uncertainty relevant to determining random values in cryptology. It is shown that unbalanced and self-similar Huffman trees have extremal properties with respect to these measures. Their corresponding probability distributions exhibit an unbounded gap between (Shannon) entropy and the logarithm of the minimum search space size necessary to be guaranteed a certain chance of success (called marginal guesswork). Thus, there can be no general inequality between them. We discuss the implications of this result in terms of the security of weak secrets against brute-force searching attacks, and also in terms of Shannon's uncertainty axioms.

A Introduction

It has become popular folklore that any "uncertainty" in cryptology can always be measured by (Shannon) entropy. This belief likely traces back to a result of Shannon's [15] that entropy is the only quantity which satisfies a set of *uncertainty axioms*. While information theory continues to be of great importance to cryptology (see e.g. [10,17] and references therein), in this paper we challenge the less-rigorous folklore. Specifically we compare entropy with another eminently reasonable uncertainty measure, *marginal guesswork*, which we define as the optimal number of trials necessary to be guaranteed a certain chance of guessing a random value in a brute-force search. Our main result is Theorem 1 below, which basically says that there can be *no* general relationship (i.e. no inequality can hold) between entropy and the logarithm of marginal guesswork.

It is now well-established that there are a variety of different uncertainty measures important to cryptology. Recent scholarship suggests a hierarchy of inequalities surrounding entropy (see in particular Cachin's summary of [3, Table 3.1]), and counterexamples exist (see e.g. an important one due to Massey [9]) which show that many of these inequalities are not tight. Our result adds to this overall picture in a rather negative way: marginal guesswork, which we shall argue to be as meaningful a measure of uncertainty as any within the secret-guessing paradigm (see Sect. B.2 and Remark 1 below), can not exist with entropy in any hierarchy of inequalities. This rules out even vague notions that entropy may uniquely measure uncertainty at *some* level of granularity.

In Sect. B, we recall some basic definitions in order to precisely state our main result in Sect. C. We conclude in section Sect. D with a discussion about why Shannon's uncertainty axioms don't apply in such an extreme way to the secret-guessing paradigm.

B Preliminaries

In this section, we recall the definitions and some basic properties of the two uncertainty measures studied in this paper: entropy and marginal guesswork. We assume throughout that X is a discrete random variable taking on values in set a $\mathscr{X} = \{x_1, x_2 \ldots\}$. Following both Shannon [15] and Massey [9], we assume no bound on the size of $\mathscr{X}$. For our purposes however, we need only consider random variables which have finite support (only finitely many probabilities $p_i = \mathsf{P}[X = x_i]$ are nonzero). It will be clear from the context whether occasionally $\mathscr{X}$ itself should be assumed finite.

Both uncertainty measures we consider are related to determining the value of X given certain oracles. Speaking loosely for now:

– Entropy measures how difficult it is to determine X given *single* queries to *multiple* oracles which answer questions of the form, "Is X an element of subset $\mathscr{U} \subseteq \mathscr{X}$?".
– Marginal guesswork measures how difficult it is to determine X given *multiple* queries to a *single* oracle which answers the question, "Is $X = x$?".

It turns out that these two querying strategies amount to different ways of traversing a Huffman tree as discussed below.

B.1 Entropy: The Uncertainty of Description

Definition. The *entropy* of X is defined by

$$H(X) = -\sum_i p_i \log p_i,$$

where these and other logarithms are taken to be base 2. When there are only two nonzero probabilities, p and $(1-p)$, the *binary entropy function* is sometimes written $H(p) = -p \log p - (1-p)\log(1-p)$.

Informally, entropy is a measure of redundancy of natural languages and other information sources. It is used extensively in the theory of communication systems to quantify the optimal use of bandwidth in noiseless and noisy channels [1,4]. Since plaintext redundancy limits cipher security, entropy is often useful in cryptology [16,17].

Defining Algorithm: Prefix Codes and Huffman Trees. A *code tree* is any binary tree for which the directions left and right are assigned binary digits 0 and 1, respectively. Following the path from the root to a leaf ℓ determines a unique *codeword* $c(\ell) \in \{0,1\}^*$. The mapping c defines a *prefix code* because no proper prefix of a codeword is a codeword, and so arbitrary concatenations of codewords may be uniquely decoded[1]. Certain code trees correspond to optimal compression, because the average codeword length is minimal.

For our purposes, a *Huffman tree* is a binary tree, all of whose non-leaves have two children, and whose leaves are arranged, from left to right, in order of non-decreasing distance from the root of the tree. Not every binary tree can be transformed into a Huffman tree by permuting its branches. Each Huffman tree with leaves $\mathscr{Y} = \{y_1, \ldots, y_n\}$ corresponds to a $\mathscr{Y}$-valued random variable Y with distribution $q_i = \mathsf{P}[Y = y_i] = 2^{-|c(y_i)|}$, where $|c(y_i)|$ is the length of the codeword corresponding to y_i, or equivalently the distance from the root of the tree to y_i. For this random variable, the entropy is *exactly* the average codeword length, which is optimal. In other words,

$$H(Y) = \sum_i q_i |c(y_i)| \leq \sum_i q_i |c^*(y_i)|,$$

for any other prefix code c^*.

That the converse is nearly true constitutes one of beautiful elementary results of information theory [1,4]; every $\mathscr{X}$-valued random variable X corresponds to a Huffman tree with leaves $\mathscr{X}$ and its prefix code $c : \mathscr{X} \longrightarrow \{0,1\}^*$ is optimal (called the *Huffman code* for X). Furthermore, $H(X)$ is within a single bit of the optimal average codeword length, i.e.

$$H(X) \leq \sum_i p_i |c(x_i)| < H(X) + 1.$$

Thus entropy characterizes optimal encoding, and so can be seen as a measure of how difficult it is to determine a random value X given oracles which tell us whether, "X an element of $\mathscr{U} \subseteq \mathscr{X}$?". Indeed, by collecting into subsets those codewords which share common prefixes, we may formalize this characterization of entropy in the following algorithm.

Algorithm 1 traverses the Huffman tree of X from its root to the leaf which corresponds to the value of X. The average time complexity of this algorithm is clearly just the optimal average codeword length and hence is within 1 bit of $H(X)$.

B.2 Marginal Guesswork: The Uncertainty of Searching

Definition. For a random variable X, it is convenient and common [8] to rearrange the probabilities $p_i = \mathsf{P}[X = x_i]$ into non-increasing order

$$p_{[1]} \geq p_{[2]} \geq \cdots \geq p_{[n]} > p_{[n+1]} = \cdots = 0, \tag{1}$$

[1] The family of prefix codes does not constitute all uniquely decodable codes, but codes outside of this family are not relevant to this paper.

A recursive algorithm to determine the value of X given: (i). its Huffman code $c : \mathscr{X} \longrightarrow \{0,1\}^*$, and (ii). oracles that answer whether $c(X)$ has prefix $w \in \{0,1\}^*$. The codeword corresponding to X is ultimately returned when the empty word is passed as the initial argument, i.e. $X = \mathsf{Hfind}(\varepsilon)$.

```
function Hfind($s$):
  if $s \in c(\mathscr{X})$ then
    return $c^{-1}(s)$.
  endif
  if $s \cdot 0$ is a prefix of $c(X)$ then
    return Hfind($s \cdot 0$).
  else
    return Hfind($s \cdot 1$).
  endif
```

where n is the least integer satisfying (1). Thus if $x_{[i]}$ is the i-th most probable event, we have $p_{[i]} = \mathsf{P}\left[X = x_{[i]}\right]$. The *marginal guesswork* (or when necessary, the *α-marginal guesswork*) of X is defined by

$$w_\alpha(X) = \min \left\{ i \,\middle|\, \sum_{j=1}^{i} p_{[j]} \geq \alpha \right\} .$$

Informally, marginal guesswork measures the maximum cost of determining the value of X when we wish to be guaranteed an appreciable chance of success (the probability α) in a brute-force search. In the case of searching for the secret key of a cipher given known plaintext-ciphertext pairs, it has been quantified under certain circumstances [13].

More generally, a plot of the cumulative probability of success as a function of the minimum number of trials (i.e. the transpose of the profile of $w_\alpha(X)$) is essentially a measure of the non-uniformity of X which predates information theory by a half century. Indeed, the *Lorenz curve* of economics, which is a plot of cumulative wealth as a function of percentile group, has been widely used to quantify non-uniformity in the distribution of wealth [7,12], and has deep connections to *majorization* in the theory of inequalities [8].

Defining Algorithm: Optimal Brute-Force Searches. Many situations in computer security force an adversary to conduct a brute-force search for the value of X by enumerating *some* elements of $\mathscr{X}$ and testing for a certain success condition. The only possible luxury afforded to the adversary is that he may know which events are more likely, and hence be able to search in order of decreasing likelihood. The necessary success condition is abstracted as an oracle which answers whether $X = x$, and often occurs in practice. For example, UNIX passwords are routinely guessed in this manner [11], because for any candidate password x, it can be checked whether its hash $f(x)$ exists in a table of such hashes (which is often available publicly).

In cryptology, the secret key X of a symmetric key cipher almost always comes along with such an oracle. The reason is that given a small amount of ciphertext, a candidate key can be used to decrypt and produce candidate plaintext. Most plaintext – whether compressed or not – has certain features which suffice to rule out false keys. But such oracles are not limited to symmetric key ciphers. Even elaborate multipass protocols which employ public key primitives can be subverted in order to yield an oracle of this form sufficient to break them [10,2]. The safest bet for the cryptographer is to assume that the adversary has complete knowledge of the probabilities $\{p_i\}$ and will conduct any search optimally, i.e. in the order given by (1). This suggests the following optimal *brute-force search* algorithm. Clearly, the probability of success of Algorithm B.2 is

Optimal brute-force search for the value of X which guarantees a chance of success of α. Assumes an oracle which answers whether $X = x$. The algorithm may not succeed, in which case $\emptyset$ is returned.

```
function Wfind($\alpha$):
  for $i \in \{1, \ldots, w_\alpha(X)\}$ do
    if $X = x_{[i]}$ then
      return $x_{[i]}$.
    endif
  done
  return $\emptyset$.
```

$\sum_{i=1}^{w_\alpha(X)} p_{[i]} \geq \alpha$, and this is the least costly algorithm to guarantee chance α. Notice also that since the Huffman tree of X is arranged, from left to right, in order of non-decreasing length (non-increasing probability), Algorithm B.2 traverses the Huffman tree of X along the leaves from left to right.

Guesswork and Optimal Exhaustive Searches. When $\alpha = 1$, we shall call the optimal search of Algorithm B.2 an *exhaustive search*, because it will exhaust all *possible* elements of $\mathscr{X}$ until it succeeds. In practice, not all brute-force searches are exhaustive. The search for passwords and is rarely exhaustive [11], while the search for cipher keys is often exhaustive [5]. The choice of α in a brute-force attack depends, of course, on the goals of the adversary and the nature of the problem.

The average time complexity of Algorithm B.2 in an exhaustive search is given by

$$W(X) = \sum_i i p_{[i]}, \tag{2}$$

which we shall call simply the *guesswork*[2] of X. It is readily verified that the guesswork of X is the average of its α-marginal guesswork, over all values of α. Thus

$$W(X) = \int_0^1 w_\alpha(X)\, d\alpha,$$

so that guesswork can be seen as the area under the cryptologic Lorenz curve, $w_\alpha(X)$.

Guesswork has been used to study cipher security in [13,14]. It was first introduced in cryptology by Massey [9] in a context similar to our own, and he obtained a bound

$$W(X) \geq 2^{H(X)-2} + 1 \tag{3}$$

when $H(X) \geq 2$. Massey didn't give the quantity of (2) a name, but in light of its Lorenz curve interpretation, it is not surprising that it also appeared in economics nearly a century ago as the crucial term in the *Gini coefficient* [6,12,8].

Remark 1. The guesswork $W(X)$ is not the principle object of study here because like any average it could easily obscure the region of greatest interest. Cryptanalytic adversaries often don't care if $W(X)$ is intractably large, say 2^{128}, when $w_{0.001}(X)$ is relatively small, say around 2^{30}. They might be content to carry out 1,000 separate and independent attacks, expending a total effort of about 2^{40} guesses and be guaranteed a 63% chance of success (from standard calculus).

Thus the complete picture about the security of a secret X must include the entire profile of $w_\alpha(X)$ rather than merely its average $W(X)$. $\square$

B.3 When Marginal Guesswork and Exponentiated Entropy Agree

When X is uniform on some subset of $\mathscr{X}$, we get a kind of agreement between entropy and marginal guesswork given by,

$$H(X) \approx \log w_\alpha(X). \tag{4}$$

This occurs in several interesting cases.

Minimum Uncertainty. When X is deterministic, i.e. it takes on only one value with nonzero probability, it is easily seen that $H(X) = 0 = \log w_\alpha(X)$.

Maximum Uncertainty. When X is uniform over finite $\mathscr{X}$ it is easily seen that $H(X) = \log|\mathscr{X}|$, and $w_\alpha(X) = \lceil \alpha|\mathscr{X}| \rceil$. Equation (4) follows for fixed α since $H(X) = \log|\mathscr{X}| = O(\log|\mathscr{X}| + \log(\alpha)) = O(\log w_\alpha(X))$.

[2] Guesswork is sometimes called *guessing entropy*, though as far as we are aware, never without enclosing quotation marks.

Long Random Sequences (The Asymptotic Equipartition Property). Consider the random tuple $X = (Y_1, \dots, Y_m)$, where $\{Y_i\}$ are independent and identically distributed $\mathscr{Y}$-valued random variables for some $\mathscr{Y} = \{y_1, y_2 \dots\}$. The AEP [1,4] informally tells us that for large m, X behaves as if it were uniform on a subset of $\mathscr{Y}^m$, and we would again expect (4) to hold.

The AEP is analogous to the law of large numbers and can be formally stated as follows. Suppose each Y_i is drawn according to probability distribution $q : \mathscr{Y} \longrightarrow \mathbb{R}$. Following [4], we define the (strangely self-referential) $\mathbb{R}$-valued random variables $q(Y_i)$ by $\mathsf{P}[q(Y_i) = q(y_j)] = \mathsf{P}[Y_i = y_j] = q(y_j)$, and similarly $q(Y_1, \dots, Y_m) = q(Y_1) \cdots q(Y_m)$. Note that $q(Y_i)$ takes on at most $|\mathscr{Y}|$ real values, namely the actual probabilities of Y_i. Writing $H(Y) = H(Y_i)$ for any i, we have, $-\log q(Y_1, \dots, Y_m)/m \to H(Y)$ in probability, as $m \to \infty$ (see [1] and cf. [4]). Thus for every $\varepsilon > 0$, there is an m such that

$$\mathsf{P}\left[\left|-\frac{1}{m}\log q(Y_1, \dots, Y_m) - H(Y)\right| < \varepsilon\right] \geq 1 - \varepsilon.$$

In essence, drawing a value of $q(Y_1, \dots, Y_m)$ means drawing the probability of a tuple $X = (Y_1, \dots, Y_m)$. For fixed m and ε, it is natural to divide the set of all $|\mathscr{Y}|^m$ sequences into the *typical sequences* $T_m^\varepsilon \subseteq \mathscr{X}$, for which

$$2^{-m(H(Y)+\varepsilon)} < \mathsf{P}[X = (y_1, \dots, y_m)] < 2^{-m(H(Y)-\varepsilon)},$$

and the remaining *atypical sequences*. Thus it is clear that we recover (4) for sufficiently large m, $H(X) \approx mH(Y) \approx mH(Y) + \log(\alpha) \approx \log w_\alpha(X)$.

Evidently, random sequences of i.i.d. values constitute special random variables for which entropy and (logarithmic) marginal guesswork essentially agree as in (4). However as we shall see in the sequel, no equivalent of the AEP must hold for a general random variable.

C The Main Result

It is easy to see that there are random variables for which $2^{H(X)}$ is much greater than $w_\alpha(X)$; simply choose $p_{[1]} = \alpha$ and all remaining probabilities sufficiently small. This is essentially the example used by Massey [9] to show that the guesswork inequality of (3) is not tight. The details for marginal guesswork are given in the proof to Theorem 1 below.

The following lemma shows that the opposite can happen as well. The cumulative probability of the most likely $2^{H(X)}$ events can be arbitrarily close to zero.

Lemma 1. *For any real $\varepsilon > 0$, there exists a random variable X with finitely many nonzero probabilities $p_i = \mathsf{P}[X = x_i]$, such that*

$$\sum_{i=1}^{2^{\lceil H(X) \rceil}} p_{[i]} < \varepsilon.$$

Proof. We define a family of random variables $X_{j,k}$ parameterized by integers j and k. Specifically for each $j, k \geq 1$, write $a = 2^j$ and define $X_{j,k}$ by the sequence of probabilities $a, a^{-2}, a^{-3}, \ldots, a^{-k}$, followed by m copies of the last value a^{-k}, where m must be chosen so that the probabilities add to 1. These probabilities come from self-similar Huffman trees as discussed in Fig. 1 below. It is easy to show that m must be given by

$$m = \frac{1 + (a-2)a^k}{a-1}.$$

Now let us examine the entropies of this family.

$$\begin{aligned} H_{j,k} \triangleq H(X_{j,k}) &= \sum_{i=1}^{k} \frac{1}{a^i} \log a^i + \left[\frac{1+(a-2)a^k}{a-1}\right] \frac{1}{a^k} \log a^k \\ &= j \sum_{i=1}^{k} \frac{i}{a^i} + jk \left[\frac{1+(a-2)a^k}{(a-1)a^k}\right] \\ &= j \left[\frac{a^{k+1} - (k+1)a + k}{(a-1)^2 a^k}\right] + jk \left[\frac{1+(a-2)a^k}{(a-1)a^k}\right] \\ &= j \left[\frac{ka^{k+2} + (1-3k)a^{k+1} + 2ka^k - a}{a^k (a-1)^2}\right] \\ &= jk \left(\frac{a-2}{a-1}\right) + h_{j,k}, \end{aligned}$$

where

$$h_{j,k} = \frac{j(a^k - 1)}{a^{k-1}(a-1)^2}.$$

Let us fix a lower bound $2 < j$ so that $a > 4$ as well. Then it is easy to see that

$$jk \left(\frac{a-2}{a-1}\right) > \log k,$$

and thus that $2^{\lceil H_{j,k} \rceil} \geq 2^{H_{j,k}} \geq 2^{jk\left(\frac{a-2}{a-1}\right)} > k$. In this case, we may calculate the cumulative sum

$$s_{j,k} \triangleq \sum_{i=1}^{2^{\lceil H_{j,k} \rceil}} p_{[i]} \leq \sum_{i=1}^{k} \frac{1}{a^i} + (2^{\lceil H_{j,k} \rceil} - k)\frac{1}{a^k} = \frac{1}{2^j - 1} + \sigma_{j,k},$$

where

$$\sigma_{j,k} = \frac{(a-1)(2^{\lceil H_{j,k} \rceil} - k) - 1}{a^k (a-1)}.$$

We claim that for fixed j, $\sigma_{j,k} \to 0$ as $k \to \infty$. If that were true, then for any $\varepsilon > 0$, we may fix a $j > 2$ such that

$$\frac{1}{2^j - 1} < \varepsilon,$$

and find a k such that $s_{j,k} < \varepsilon$. Thus the proof would be complete.

We now turn our attention to finding the limit of $\sigma_{j,k}$ as $k \to \infty$. For some $\widehat{k}(j)$, $h_{j,k} < 1$ for all $k \geq \widehat{k}(j)$, because $h_{j,k} \to j/a$ as $k \to \infty$. Thus for $k \geq \widehat{k}(j)$,

$$\lceil H_{j,k} \rceil \leq jk \left(\frac{a-2}{a-1} \right) + 2,$$

allowing us to give the following upper bound on $\sigma_{j,k}$.

$$\sigma_{j,k} \leq \frac{(a-1)(4\beta^k - k) - 1}{a^k(a-1)},$$

where $\beta = 2^{j\left(\frac{a-2}{a-1}\right)} < a$. Finally, by two applications of L'Hospital's rule we see that $\sigma_{j,k} \to 0$ as $k \to \infty$. □

Remark 2. Each random variable $X_{j,k}$ in the proof of the previous lemma corresponds to a self-similar Huffman tree, $t_{j,k}$. Fig. 1 shows $t_{2,3}$. The trees are self-similar in that each $t_{j,k}$ contains subtrees isomorphic to $t_{j,i}$, for all $i < k$. The self-similarity is also characterized by a zig-zag pattern of the path starting from the root and going alternately through the maximal proper codeword prefixes and the roots of the trees isomorphic to $t_{j,i}$, for $i < k$. □

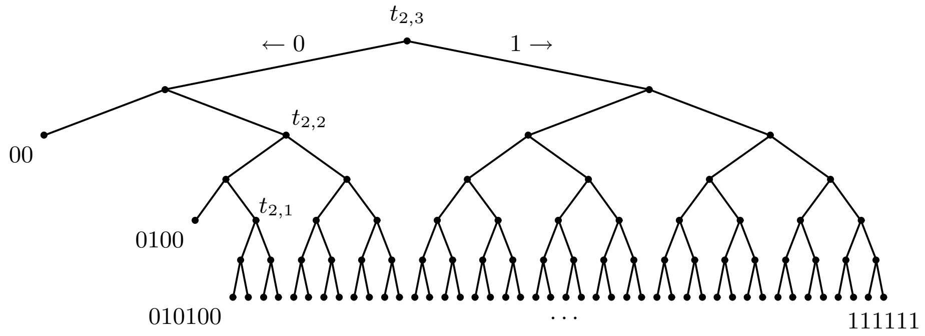

Fig. 1. The random variables in the proof of Lemma 1 correspond to self-similar Huffman trees as described in Remark 2.

Our main result formalizes the notion that there are random variables for which $H(X)$ and $\log w_\alpha(X)$ could be arbitrarily far away from one another. In other words, to determine a secret, the effective search space size – measured in bits – could be arbitrarily more or less than the entropy of that secret.

Theorem 1. *For each $0 > \alpha > 1$ and every integer $N > 0$, there are random variables X and Y with finite support satisfying $\log w_\alpha(X) - H(X) > N$, and $H(Y) - \log w_\alpha(Y) > N$.*

Proof. First we find X. Write $H = H(X)$ and choose the ε of Lemma 1 satisfying $\varepsilon = \alpha/2^N$. Then

$$\sum_{i=1}^{2^N 2^{\lceil H \rceil}} p_{[i]} \le 2^N \sum_{i=1}^{2^{\lceil H \rceil}} p_{[i]} < 2^N \varepsilon = \alpha.$$

We conclude that $w_\alpha(X) > 2^N 2^{\lceil H \rceil} \ge 2^{N+H}$, and thus that $\log w_\alpha(X) - H > N$.

We now turn to finding Y. Consider the probabilities $q_i = \mathsf{P}\left[Y = y_i\right]$ given by $q_{[1]} = \alpha$ followed by $q_{[i]} = (1-\alpha)/2^k$, $2 \le i \le 2^k + 1$. In other words, the same smallest probability is repeated 2^k times. This corresponds to an unbalanced Huffman tree as in Fig. 2. It is easy to see that $w_\alpha(Y) = 1$, and

$$H(Y) = -\alpha \log \alpha - (1-\alpha) \log \frac{(1-\alpha)}{2^k} = (1-\alpha)k + H(\alpha).$$

The desired result is obtained when $H(Y) > N$, which is easily achieved if we choose

$$k > \frac{N - H(\alpha)}{1-\alpha}.$$

□

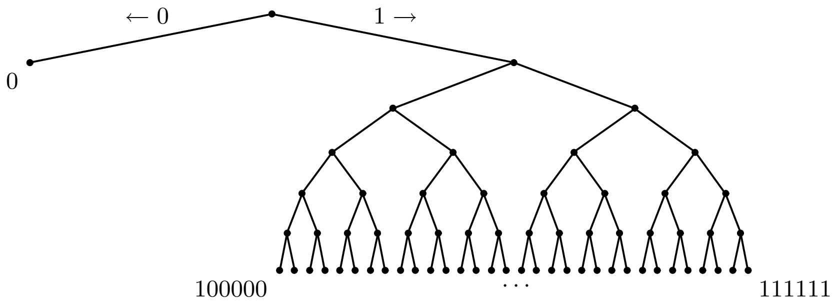

Fig. 2. An unbalanced Huffman tree demonstrating $H(X) \gg \log w_\alpha(X)$, as in Theorem 1.

D Discussion: Shannon's Axioms

Any measure of uncertainty about X should indicate the difficulty of determining the value of X under a certain set of rules. In fact, entropy and marginal guesswork are the average and worst-case time complexities of querying strategies as formalized in Algorithms 1 and B.2, respectively. Furthermore these algorithms represent two optimal ways of traversing the same combinatorial object, namely the code tree corresponding to the Huffman code of X.

While there are a variety of cases where they agree, we have seen that $H(X)$ is not, in general, a good indicator of $\log w_\alpha(X)$. Yet Shannon's uncertainty axioms along with a great deal of folklore seem to admit a single, unique (logarithmic) uncertainty, namely entropy. Let us recall these axioms, one or more of which must be inconsistent with the guessing paradigm for determining a secret. Paraphrasing from [16], an uncertainty measure U must satisfy (cf. [1,4]):

1. $U(X)$ must be continuous function of the probabilities $p_i = \mathsf{P}[X = x_i]$.
2. For uniformly distributed X, $U(X)$ must be a monotonically increasing function of $|\mathscr{X}|$.
3. (*grouping axiom*) Suppose $\mathscr{X}$ is decomposed into mutually disjoint events $y_j \subseteq \mathscr{X}$, $1 \leq j \leq m$. Let $\mathscr{Y} = \{y_1, \ldots, y_m\}$ and define the $\mathscr{Y}$-valued random variable Y by $q_j = \mathsf{P}[Y = y_j] = \mathsf{P}[X \in y_j]$. The uncertainty $U(X)$ must satisfy
$$U(X) = U(Y) + \sum_{j=1}^{m} q_j U(X|y_j),$$
where for each j, $U(X|y_j)$ is the uncertainty corresponding to the *a posteriori* probabilities given by $p(x_i|y_j) = p_i/q_j$ if $x_i \in y_j$, and 0 otherwise. (Note that for entropy, this axiom is a special case of the *chain rule* of information theory [4]: $H(X,Y) = H(Y) + H(X|Y)$.)

Even though $\log w_\alpha(X)$ is not continuous, it could be replaced by a similar continuous function, so axiom 1 is really not a problem. Since $\log \lceil \alpha|\mathscr{X}| \rceil$ is monotonically increasing, axiom 2 holds for logarithmic marginal guesswork. However, we claim that the grouping axiom 3 cannot hold for $\log w_\alpha(X)$. If we assume otherwise, we must have

$$\frac{w_\alpha(X)}{w_\alpha(Y)} = \prod_{j=1}^{w_1(Y)} [w_\alpha(X|y_j)]^{q_j} \tag{5}$$

But let $\mathscr{X}$ be the leaves of a Huffman tree T and let $\mathscr{Y}$ be the leaves of any proper Huffman subtree of T with the same root. As described in Sect. B.1, we get $\mathscr{X}$ and $\mathscr{Y}$-valued random variables X and Y, whose distributions are related to the various codeword and prefix lengths, respectively. In the case $\alpha = 1/2$, we can compare (5) with an exact expression because $w_{\frac{1}{2}}(X)$ is just the number of leaves in the left half-tree, and thus

$$\frac{w_{\frac{1}{2}}(X)}{w_{\frac{1}{2}}(Y)} = \frac{1}{w_{\frac{1}{2}}(Y)} \sum_{j=1}^{w_{\frac{1}{2}}(Y)} w_1(X|y_j) \tag{6}$$

There are obvious cosmetic differences between (5) and (6): The arithmetic mean of (6) is not necessarily the geometric-like mean of (5), the number of terms is different, and the weighting factors aren't necessarily the same, i.e. $q_j \neq 1/w_{\frac{1}{2}}(Y)$. But we hold that the heart of the disparity is the fact that the individual values being "averaged" have essentially no relation to one another. For sufficiently large trees, it is easy to construct examples (e.g. using Fig. 2 on branches) where the $w_{\frac{1}{2}}(X|y_j)$ of (5) is arbitrarily less than the $w_1(X|y_j)$ of (6). By carefully choosing which y_j to introduce the disparity, we can drive the r.h.s.'s of (5) and (6) arbitrarily far away from each other – in either direction.

Shannon's grouping axiom cannot hold for $\log w_\alpha(X)$, not even as an inequality. Yet we have argued that marginal guesswork is a reasonable and useful measure of uncertainty. We must conclude that the axiomatic premise which can serve to identify uncertainty as entropy, is inappropriate for characterizing the uncertainty of a secret against an adversary who is equipped to conduct a brute-force searching attack.

Acknowledgments

Special thanks are due to the anonymous reviewers of *Indocrypt 2000*, whose detailed comments greatly helped to enhance the clarity of this paper. I am also indebted to Serge Vaudenay for suggesting the term "marginal guesswork."

References

1. Robert B. Ash. *Information Theory.* Dover, New York, 1965. 68, 69, 73, 73, 77
2. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proc. of the 1992 IEEE Comp. Soc. Symp. on Res. in Secur. and Priv.*, pages 72–84. IEEE Press, 1992. 71
3. Christian Cachin. *Entropy Measures and Unconditional Security in Cryptography.* PhD thesis, ETH Zürich, 1997. 67
4. Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* John Wiley & Sons, New York, 1991. 68, 69, 73, 73, 73, 77, 77
5. Electronic Frontier Foundation. *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design.* O'Reilly & Associates, Sebastopol, CA, 1998. 71
6. C. Gini. *Variabilitàe Mutabilità.* Anno 3, Part 2, pp. 80. Studi Economico-Giuridici della R. Università de Cagliari, 1912. 72
7. M. O. Lorenz. Methods of measuring concentration of wealth. *J. Amer. Statist. Assoc.*, 9:209–219, 1905. 70
8. Albert W. Marshall and Ingram Olkin. *Inequalities: Theory of Majorization and Its Applications.* Academic Press, San Diego, 1979. 69, 70, 72
9. James L. Massey. Guessing and entropy. *Proc. 1994 IEEE Int'l Symp. on Information Theory*, page 204, 1994. 67, 68, 72, 73
10. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, 1997. 67, 71
11. Alec Muffett. *Crack Version 5.0a User Manual.* URL: `ftp://ftp.cert.org/pub/tools/crack/`. 70, 71
12. David W. Pearce. *The MIT Dictionary of Modern Economics.* The MIT Press, Cambridge, MA, fourth edition, 1992. 70, 72
13. John O. Pliam. *Ciphers and their Products: Group Theory in Private Key Cryptography.* PhD thesis, University of Minnesota, July 1999. URL: `http://www.ima.umn.edu/~pliam/doc`. 70, 72
14. John O. Pliam. Guesswork and variation distance as measures of cipher security. In *Selected Areas in Cryptography - SAC'99*, LNCS 1758, pages 62–77, Berlin, 2000. Springer-Verlag. 72

15. Claude E. Shannon. A mathematical theory of communication. *Bell System Tech. Jour.*, 27:379–423, 623–656, 1948. 67, 68
16. Claude E. Shannon. Communication theory of secrecy systems. *Bell System Tech. Jour.*, 28:656–715, 1949. 68, 77
17. Douglas R. Stinson. *Cryptography: Theory and Practice.* CRC Press, Boca Raton, 1995. 67, 68

Improved Impossible Differentials on Twofish [⋆]

Eli Biham and Vladimir Furman

Computer Science Department,
Technion - Israel Institute of Technology,
Haifa 32000, Israel
biham@cs.technion.ac.il
http://www.cs.technion.ac.il/~biham/
vfurman@cs.technion.ac.il

Abstract. Twofish is one of the finalists for the Advanced Encryption Standard selection process (AES). The best up-to-date analysis of Twofish was presented by one of its designers, who showed that its complexity on 6-round Twofish with 128-bit keys is $4.6 \cdot 2^{128}$ one-round computations. In this paper we present an improvement of the attack on 6-round Twofish, whose complexity is $1.84 \cdot 2^{128}$ one-round computations. For other key sizes, our results have the complexity $13 \cdot 2^{160}$ one-round computations for 192-bit keys and $24.2 \cdot 2^{192}$ one-round computations for 256-bit keys. For 7-round Twofish, the designers mentioned an attack, and estimated its complexity to be about 2^{256} simple steps for 256-bit keys, and for other key sizes have complexities that exceed exhaustive search. We present an improvement of the attack on 7-round Twofish, whose complexity is $2^{226.48}$ one-round computations for 256-bit keys. We also show, for the first time, that this attack is faster than exhaustive search for 192-bit keys for which it breaks 7-round Twofish in $2 \cdot 2^{192}$ one-round computations, while exhaustive search takes $7 \cdot 2^{192}$.

Keywords: cryptanalysis, impossible differential, Twofish.

A Introduction

Twofish [4] is a block cipher designed by Counterpane Systems Group as a candidate for the Advanced Encryption Standard selection process, and was accepted as one of the five finalists.

The best up-to-date attacks on Twofish breaking 6 rounds for all key sizes (128, 192 and 256) and 7 rounds for 256-bit keys only were presented by Ferguson in [2]. They use a 5-round impossible differential (see [1,3] for more details on attacks using impossible differentials). In this paper we present an improvement, based on the same 5-round impossible differential and an additional 4-round impossible differential. Our improvement reduces the total complexity of the attack on 6-round Twofish for all key sizes and on 7-round Twofish for 256-bit

⋆ This work was supported by the European Union fund IST-1999-12324 - NESSIE.

keys. In addition, we show, for the first time, that the attack on 7-round Twofish for 192-bit keys is efficient as well.

Note that Ferguson made the full analysis only for 6-round Twofish with 128-bit keys and for other cases gave only his estimations about the complexity of the attacks. All results of Ferguson presented in this paper are computed by us, to allow fair comparison of his and our methods.

Our paper is organized as follows: Section B describes the attack on 6-round Twofish with 128-bit keys. Section C describes the attacks on 6-round Twofish with 192-bit and 256-bit keys. Finally, Section D describes the known plaintext attacks on 7-round Twofish with 192-bit and 256-bit keys.

B The 6-Round Attack on Twofish with 128-bit Keys

Twofish is a 16-round block cipher with pre- and post-whitenings. Figure 1 outlines round i of Twofish, where

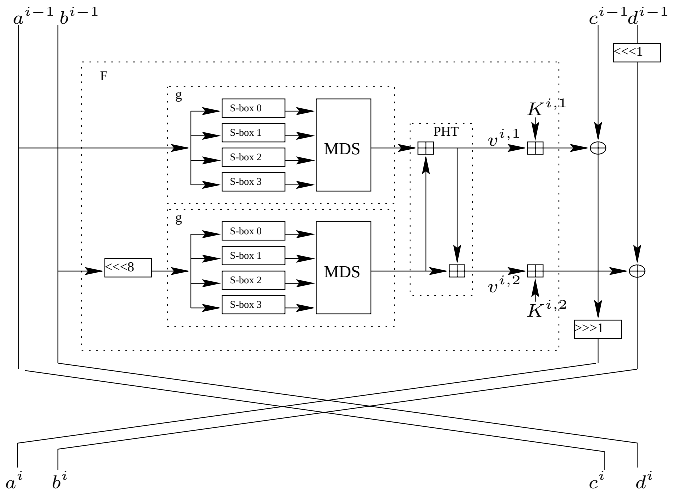

Fig. 1. The round i

- (a^i, b^i, c^i, d^i) is the 128-bit intermediate data after round i,
- $(K^{i,1}, K^{i,2})$ is the subkey of round i,
- The S-boxes are key dependent,
- MDS is a linear operation defined by a constant matrix,
- PHT is a layer that mixes two 32-bit words x, y by $x' = x + y$, $y' = 2 \cdot y + x$,
- $(v^{i,1}, v^{i,2})$ is the 64-bit output of the PHT in round i.

More detailed description of Twofish may be found in [4]. The following analysis is performed on an equivalent variant of Twofish without the one-bit rotations of the right half of the data. In [2] the author shows the equivalence between this variant of Twofish and the original Twofish variant.

First, we describe the detailed analysis of Ferguson, as it is shown in [2], and then we describe our detailed analysis.

B.1 Ferguson's Analysis

We know the 5-round impossible differential

$$(0, 0, \delta_1, \delta_2) \overset{5\ rounds}{\not\mapsto} (\delta_1, \delta_2, 0, 0),$$

where $(\delta_1, \delta_2) \neq (0, 0)$ [denoted in the equivalent variant].

Using this impossible differential a 6-round Twofish is attacked as follows: Look for plaintext pairs with a difference $(0, 0, \delta_1, \delta_2)$, where $(\delta_1, \delta_2) \neq (0, 0)$, that have a difference $(\delta_1, \delta_2, 0, 0)$ after the 5th round when decrypting by some guessed subkey of the last round. If such a pair is found, the corresponding guessed subkey cannot be the correct subkey, because the differential is impossible. Therefore such a subkey is certainly wrong, and can be discarded from the list of the possible subkeys. If a sufficient number of such pairs are analyzed, most subkeys can be discarded, and the remaining keys can be exhaustively searched.

A more detailed description of the attack is:

1. Fix a 64-bit value (a^0, b^0), and let (c^0, d^0) take all the 2^{64} possible values. Note that for this attack only a single such structure is needed.
2. Ask for the ciphertexts $(a_i^6, b_i^6, c_i^6, d_i^6)$ of the 2^{64} plaintexts (a^0, b^0, c_i^0, d_i^0), and insert them in a hash table indexed by $(c_i^0 \oplus c_i^6, d_i^0 \oplus d_i^6)$.
3. Any pair of encryptions without the same $(c_i^0 \oplus c_i^6, d_i^0 \oplus d_i^6)$ cannot make the differences before the last round to be as required by 5-round impossible differential. These pairs are ignored, and only the pairs of encryptions with the same $(c_i^0 \oplus c_i^6, d_i^0 \oplus d_i^6)$ are analyzed. As there are 2^{127} pairs in total and 64-bit restrictions, the expectation is that about 2^{63} pairs remain.
4. The 64-bit S-box key S is guessed and the values $\hat{v}_i^{6,1}$, $\hat{v}_i^{6,2}$ (from c_i^6, d_i^6) are computed for each plaintext in the structure. Thus, for each remaining pair, the situation is as described in Figure 2, where the values $\hat{v}_i^{6,1}$, $\hat{v}_i^{6,2}$, $\hat{v}_j^{6,1}$, $\hat{v}_j^{6,2}$ and the differences $a_i^6 \oplus a_j^6$, $b_i^6 \oplus b_j^6$ are known. The subkeys $(\hat{K}^{6,1}, \hat{K}^{6,2})$ that satisfy the described requirement can be easily found. There is one such subkey $(\hat{K}^{6,1}, \hat{K}^{6,2})$ on average for each analyzed pair and each S-box key S guessing.
5. Each remaining pair eliminates a fraction of 2^{-64} of the possible keys for the last round. For each last round subkey there exists one original key on average. Thus, a single structure of 2^{64} plaintexts reduces the number of the keys to $(1 - 2^{-64})^{2^{63}} = 1/\sqrt{e} = 0.6$ of the original number.
6. Test the remaining keys exhaustively.

The complexity of this attack is as follows: 2^{128} times (2^{64} possible S-box keys and 2^{64} plaintexts) one-round computations are spent, which reduces the number

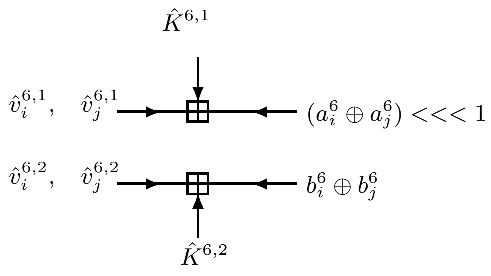
$\hat{K}^{6,1}$
$\hat{v}_i^{6,1}, \hat{v}_j^{6,1}$
$(a_i^6 \oplus a_j^6) <<< 1$
$\hat{v}_i^{6,2}, \hat{v}_j^{6,2}$
$b_i^6 \oplus b_j^6$
$\hat{K}^{6,2}$

Fig. 2. *The equation for finding last round subkeys, that are wrong due to 5-round impossible differential*

of the keys to 0.6 of the original number. So, there remain $0.6 \cdot 2^{128}$ undiscarded keys, whose exhaustive search takes $6 \cdot 0.6 \cdot 2^{128}$ one-round computations. In total, the complexity is $2^{128} + 3.6 \cdot 2^{128} = 4.6 \cdot 2^{128}$ one-round computations, while an exhaustive search takes $6 \cdot 2^{128}$ one-round computations. On average, these values are $2.8 \cdot 2^{128}$ and $3 \cdot 2^{128}$ respectively.

B.2 Our Analysis

We use the 5-round impossible difference,

$$(0, 0, \delta_1, \delta_2) \overset{5\ rounds}{\not\to} (\delta_1, \delta_2, 0, 0),$$

as does Ferguson, together with the following additional 4-round impossible differential:

$$(0, 0, \delta_1, \delta_2) \overset{4\ rounds}{\not\to} (\delta_3, \delta_4, \delta_1, \delta_2),$$

for any δ_3, δ_4 such that $(\delta_3, \delta_4) \neq (0, 0)$[1].

Similar to the earlier analysis, for each pair of plaintexts $(a_i^0, b_i^0, c_i^0, d_i^0)$ and $(a_j^0, b_j^0, c_j^0, d_j^0)$ with a difference of $(0, 0, \delta_1, \delta_2)$, where $(\delta_1, \delta_2) \neq (0, 0)$, we ask for the ciphertexts $(a_i^6, b_i^6, c_i^6, d_i^6)$ and $(a_j^6, b_j^6, c_j^6, d_j^6)$ respectively.

We continue the analysis for each pair for which one of the following conditions is satisfied:

- $c_i^0 \oplus c_i^6 = c_j^0 \oplus c_j^6$ and $d_i^0 \oplus d_i^6 = d_j^0 \oplus d_j^6$ (we can use the 5-round impossible differential). This is the case studied in the earlier analysis.
- $c_i^0 \oplus c_i^6 = c_j^0 \oplus c_j^6$ and $d_i^0 \oplus d_i^6 = d_j^0 \oplus d_j^6 \oplus 2^{31}$ (we can use the 4-round impossible differential).
- $c_i^0 \oplus c_i^6 = c_j^0 \oplus c_j^6 \oplus 2^{31}$ and $d_i^0 \oplus d_i^6 = d_j^0 \oplus d_j^6$ (we can use the 4-round impossible differential).
- $c_i^0 \oplus c_i^6 = c_j^0 \oplus c_j^6 \oplus 2^{31}$ and $d_i^0 \oplus d_i^6 = d_j^0 \oplus d_j^6 \oplus 2^{31}$ (we can use the 4-round impossible differential).

[1] If $(\delta_3, \delta_4) = (0, 0)$ we get back to the 5-round impossible differential.

The last three cases extend the earlier analysis using the 4-round impossible differential.

Thus, in the 5th round, the output difference of the F function is either: $(0,0)$ (first condition), $(0,2^{31})$ (second condition), $(2^{31},0)$ (third condition) or $(2^{31},2^{31})$ (fourth condition). All these differences are preserved when they pass through addition with the 5th round subkey $(K^{5,1},K^{5,2})$, so the output difference of the PHT in the 5th round is: $(0,0)$, $(0,2^{31})$, $(2^{31},0)$ or $(2^{31},2^{31})$, respectively. The PHT affects these differences in a predetermined way, thus the output differences of the g functions in the 5th round (i.e., the input difference of the PHT) are $(0,0)$, $(2^{31},2^{31})$, $(0,2^{31})$ and $(2^{31},0)$, respectively. The MDS matrix is linear, so we can compute the output difference of each S-box in the 5th round (by multiplication of the MDS^{-1} with the output difference of the g function). So we know the output difference of each S-box in the 5th round.

We then guess the S-box key S and compute the values $\hat{v}_i^{6,1}$, $\hat{v}_i^{6,2}$ (from c_i^6, d_i^6) for each plaintext in the structure. Thus, for each remaining pair, we get the situation, described in Figure 3, in which we know $\hat{v}_i^{6,1}$, $\hat{v}_i^{6,2}$, $\hat{v}_j^{6,1}$, $\hat{v}_j^{6,2}$, a_i^6, b_i^6,

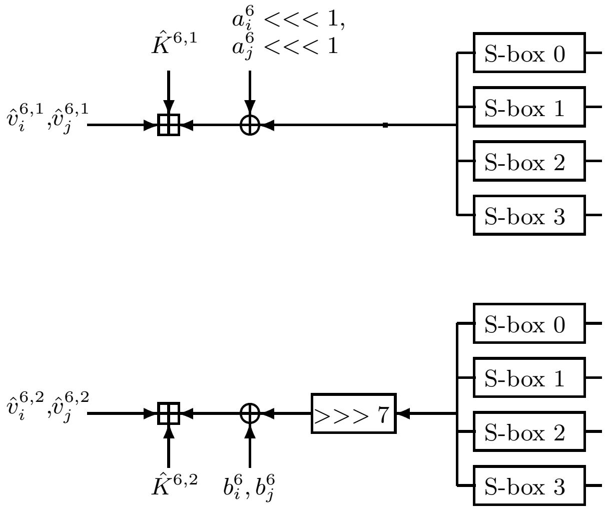

Fig. 3. *The equation for finding last round subkeys, that are wrong due to 4-round impossible differential*

a_j^6, b_j^6, the S-box key S and the output differences of all the S-boxes. The only information that we do not know in Figure 3 is the 6th round subkey $(\hat{K}^{6,1},\hat{K}^{6,2})$ that would satisfy these restrictions. Appendix A shows how we can easily find it by one operation on average.

A more detailed description of the attack is:

1. Fix a 64-bit value (a^0,b^0), and let (c^0,d^0) take all the 2^{64} possible values.

2. Ask for the ciphertexts $(a_i^6, b_i^6, c_i^6, d_i^6)$ of the 2^{64} plaintexts $(a_i^0, b_i^0, c_i^0, d_i^0)$, and insert them in a hash table indexed by $((c_i^0 \oplus c_i^6) \bmod 2^{31}, (d_i^0 \oplus d_i^6) \bmod 2^{31})$.
3. We have in total 2^{127} pairs and 62-bit restrictions, so we expect to get about 2^{65} correct pairs (satisfying the condition described earlier in this section).
4. We guess the 64-bit S-box key S and calculate the supposed $(\hat{v}^{6,1}, \hat{v}^{6,2})$ for each plaintext from the structure.
5. Each analyzed pair eliminates a fraction of 2^{-64} of the possible keys for the last round. For each last round subkey there exists one original key on average. Thus, a single structure of 2^{64} plaintexts reduces the number of keys to $(1 - 2^{-64})^{2^{65}} = 1/e^2 = 0.14$ of the original number.
6. Test the remaining keys exhaustively.

The total complexity is 2^{128} (2^{64} plaintexts $\cdot 2^{64}$ S-box keys) $+6 \cdot 0.14 \cdot 2^{128}$ (an exhaustive search of $0.14 \cdot 2^{128}$ keys through 6 rounds) $= 1.84 \cdot 2^{128}$ one-round computations and 2^{42} preprocessing computations.

Actually, we can use a smaller structure of 2^{63} plaintexts. In this case, we get about 2^{63} pairs, satisfying the condition described in the beginning of this section. Thus, a single structure of 2^{63} plaintexts reduces the number of undiscarded keys to $(1 - 2^{-64})^{2^{63}} = 1/\sqrt{e} = 0.6$ of the original number. Hence, the total complexity is $2^{127} + 6 \cdot 0.6 \cdot 2^{128} = 4.1 \cdot 2^{128}$ one-round computations and 2^{42} preprocessing computations. This is still a smaller complexity than Ferguson describes, despite the fact that we use a smaller number of plaintexts.

We observe that we can enhance the attack using several structures of 2^{64} plaintexts. This enhancement can be applied both to our variant of attack and to the variant of Ferguson. Denote the number of structures used in the attack by p. Then, the total complexity of the attack is

– Based on our method: $p \cdot 2^{128} + 6 \cdot e^{-2 \cdot p} \cdot 2^{128}$.
– Based on Ferguson's method: $p \cdot 2^{128} + 6 \cdot e^{-p/2} \cdot 2^{128}$.

Figure 4 shows the relationship between the total complexity and the number of the structures for Ferguson's method and for our method (the 1/2 structure point refers to the small half-size structure).

C The 6-Round Attacks on Twofish with 192-bit and 256-bit Keys

The attack on 6-round Twofish, described earlier, works in a similar way and with a similar reduction in the complexity when one structure is used and the key size is either 192 bits or 256 bits. However, for these cases, the same attack used in conjunction with several additional structures can further reduce the total complexity of the attack.

C.1 192-bit Keys

The total complexity of the attack on 192-bit keys variant using p structures is

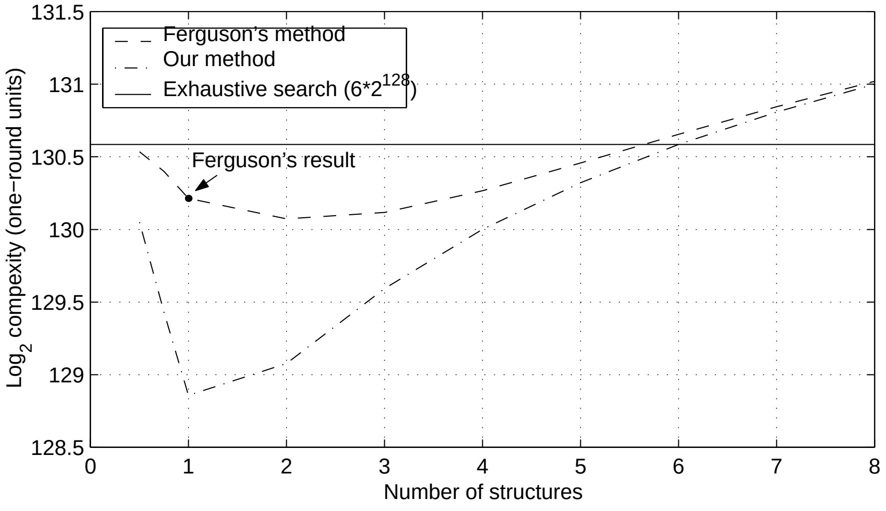

Fig. 4. *The total complexity of attack on 6-round Twofish with 128-bit keys*

- Based on our method: $p \cdot 2^{160} + 6 \cdot e^{-2 \cdot p} \cdot 2^{192}$.
- Based on Ferguson's method: $p \cdot 2^{160} + 6 \cdot e^{-p/2} \cdot 2^{192}$.

Figure 5 shows the relation between the number of structures used in the attack and the total complexity of the attack for the attack presented by Ferguson and for our improvement. From Figure 5, we can see that our best result is achieved given 12 structures ($12 \cdot 2^{64}$ chosen plaintexts), and the complexity in this case is about $2^{163.7}$ one-round computations. Figure 5 also shows that the complexity of our analysis is increased when more than 12 structures are used (and similarly for more than 48 structures with Ferguson analysis). This phenomenon is due to the larger number of chosen plaintexts, which cause the time required to discard the wrong keys to be longer than the time required to exhaustively search the remaining keys.

C.2 256-bit Keys

The total complexity of the attack on 256-bit key variant using p structures is

- Based on our method: $p \cdot 2^{192} + 6 \cdot e^{-2 \cdot p} \cdot 2^{256}$.
- Based on Ferguson's method: $p \cdot 2^{192} + 6 \cdot e^{-p/2} \cdot 2^{256}$.

Figure 6 is similar to Figure 5 for the case of 256-bit keys. Our best result in this case is about $2^{196.6}$ one-round computations using 24 structures.

Table 1 compares our best results with Ferguson's best results (computed by us, as noted in the introduction) for each possible key size.

Note that these attacks are preserved when the 6-round variant of Twofish includes pre-whitening.

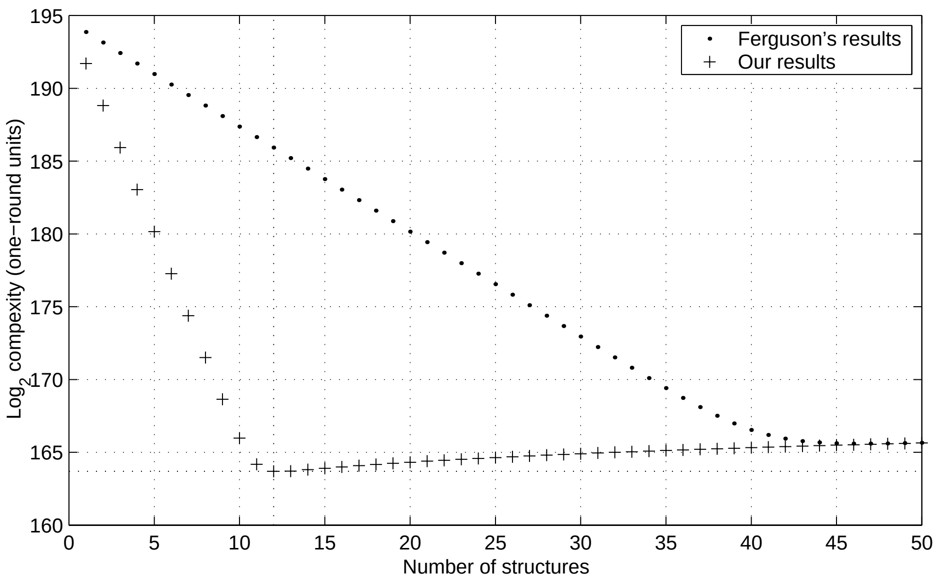

Fig. 5. *The total complexity of attack on 6-round Twofish with 192-bit keys*

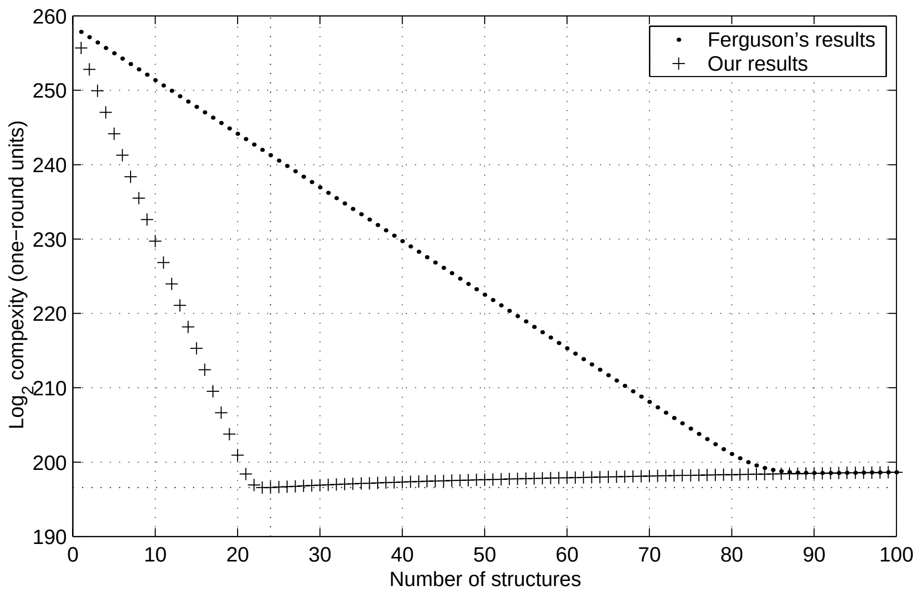

Fig. 6. *The total complexity of the attack on 6-round Twofish with 256-bit keys*

Key size in bits	Our results complexity	Our results chosen plaintexts	Ferguson's results complexity	Ferguson's results chosen plaintexts
128	$1.84 \cdot 2^{128}$	2^{64}	$4.6 \cdot 2^{128}$	2^{64}
192	$13 \cdot 2^{160}$	$12 \cdot 2^{64}$	$48.6 \cdot 2^{160}$	$47 \cdot 2^{64}$
256	$24.2 \cdot 2^{192}$	$24 \cdot 2^{64}$	$93 \cdot 2^{192}$	$91 \cdot 2^{64}$

Table 1. Comparison between ours and Ferguson's best results

D The 7-Round Known Plaintext Attacks on Twofish with 192-bit and 256-bit Keys

The 7-round attack is based on the same impossible differentials as the 6-round attack. In this attack we use these impossible differentials in the middle rounds (starting from the second round), in contrast to the 6-round attack, in which the impossible differentials start in the first round.

It is convenient to consider this attack as a variant of the 6-round attack, in which an additional round is prepended before the 6-round cipher. We want to get the difference $(0, 0, \delta_1, \delta_2)$ before the second round for some $(\delta_1, \delta_2) \neq (0, 0)$. Therefore, the difference before the first round must be $(\delta_1, \delta_2, \epsilon_1, \epsilon_2)$, where $(\epsilon_1, \epsilon_2) \neq (0, 0)$. In addition, the right half of the difference after the seventh round must be one of the following values: (δ_1, δ_2), $(\delta_1 \oplus 2^{31}, \delta_2)$, $(\delta_1, \delta_2 \oplus 2^{31})$ or $(\delta_1 \oplus 2^{31}, \delta_2 \oplus 2^{31})$ (otherwise we cannot use the impossible differentials). Hence, a pair of plaintexts may be used for this attack with probability 2^{-62} (the left part of the input difference must be equal to the right part of the output difference, except for their most significant bits). We call such a pair *a matching pair*.

For every matching pair, we guess the S-box key S and find the subkeys of the first and the last rounds, that leads to the impossible differential in the middle rounds. Such subkeys are wrong, and thus the combination of the initial keys that lead to these subkeys together with the guessed key S is wrong as well. We discard these keys from the list of the possible keys. Each matching pair eliminates about 2^{-128} fraction of the possible keys. Therefore, we need about 2^{128} matching pairs, i.e., about 2^{190} $(2^{128} \cdot 2^{62})$ pairs in total, which can be generated using $2^{95.5}$ known plaintexts.

In [2], Ferguson maintains that such an attack (using a 5-round impossible differential, of course) may be used only when the key size is 256 bits and it takes about 2^{256} steps. In Subsection D.1 we show that such an attack is more efficient than exhaustive search even when the key size is 192 bits, and even if we use only a 5-round impossible differential, as Ferguson does. In Subsection D.2 we show that the complexity of the attack on 256-bit key variant is significantly less than 2^{256}.

D.1 192-bit Keys

As in the previous attack, the analysis consists of two stages: discarding most keys and exhaustively searching the remaining ones. We denote the number of the known plaintexts required for the attack by 2^p. Clearly, p must be close to 95.5 (according to the above explanation). So, the first stage of the analysis takes 2^p (known plaintexts) $\cdot 2^{96}$ (guessing of the S-box key S) $\cdot 2$ (for the first and the last round) $= 2^{p+96+1} = 2^{p+97}$ one-round computations. It discards most keys as wrong, except for a fraction of $(1-2^{-128})^{2^{2\cdot p-1-62}} = e^{-2^{2\cdot p-63-128}} = e^{-2^{2\cdot p-191}}$ of the keys (where $2^{2\cdot p-1}$ pairs are received from 2^p known plaintexts and the probability of a pair being a matching pair is 2^{-62}). Hence, the second stage of the attack takes $7 \cdot e^{-2^{2\cdot p-191}} \cdot 2^{192}$ one-round computations (where 7 is the number of rounds and $e^{-2^{2\cdot p-191}} \cdot 2^{192}$ is the number of the remaining keys, as described earlier). The total complexity of the attack is $2^{p+97}+7\cdot e^{-2^{2\cdot p-191}}\cdot 2^{192}$ one-round computations.

We can improve this result by observing the symmetry of the cipher: We set the 4-round impossible differential in the decryption direction from round 6 backwards and use a similar analysis replacing the first and last rounds. This improvement allows us to discard twice as many keys using the 4-round impossible differential (used in 3/4 of the cases), or a total of 7/4 of the number of keys discarded by each pair in the first stage of the attack. So it reduces the fraction of remaining keys to $(1-7/4\cdot 2^{-128})^{2^{2\cdot p-1-62}} = e^{-7/4\cdot 2^{2\cdot p-1-62-128}} = e^{-7\cdot 2^{2\cdot p-193}}$. Hence, the total complexity of the attack is $2^{p+97}+7\cdot e^{-7\cdot 2^{2\cdot p-193}}\cdot 2^{192}$ one-round computations.

If the attack uses only the 5-round impossible differential (as in Ferguson's method), then the probability for some pair to be a matching pair is 2^{-64}, and the total complexity of the attack is $2^{p+97} + 7 \cdot e^{-2^{2\cdot p-193}} \cdot 2^{192}$ one-round computations.

Figure 7 shows the total complexity of the three variants of the attack:

1. Using only the 5-round impossible differential – the best result requires $4.95\cdot 2^{192}$ one-round computations using $2^{96.95}$ chosen plaintexts.
2. Using the 5-round and the 4-round impossible differentials – the best result requires $2.79 \cdot 2^{192}$ one-round computations using $2^{96.25}$ chosen plaintexts.
3. Improved result using the 5-round and the 4-round impossible differentials – the best result requires $2 \cdot 2^{192}$ one-round computations using $2^{95.9}$ chosen plaintexts.

Note that an exhaustive search takes $7 \cdot 2^{192} = 2^{194.81}$ one-round computations.

D.2 256-bit Keys

In this case, the analysis is similar to the previous one. The first stage of the analysis takes $2^p \cdot 2^{128} \cdot 2 = 2^{p+128+1} = 2^{p+129}$ one-round computations. This reduces the fraction of undiscarded keys to $(1-2^{-128})^{2^{2\cdot p-1-62}} = e^{-2^{2\cdot p-63-128}} = e^{-2^{2\cdot p-191}}$. Hence, the second stage of the attack takes $7\cdot e^{-2^{2\cdot p-191}}\cdot 2^{256}$ one-round computations. The total complexity of the attack is $2^{p+129} + 7 \cdot e^{-2^{2\cdot p-191}} \cdot 2^{256}$ one-round computations. In the improved analysis, the total complexity of the attack is $2^{p+129}+7\cdot e^{-7\cdot 2^{2\cdot p-193}}\cdot 2^{256}$ one-round computations. In the attack that uses only the 5-round impossible differential the total complexity is $2^{p+129} + 7 \cdot$

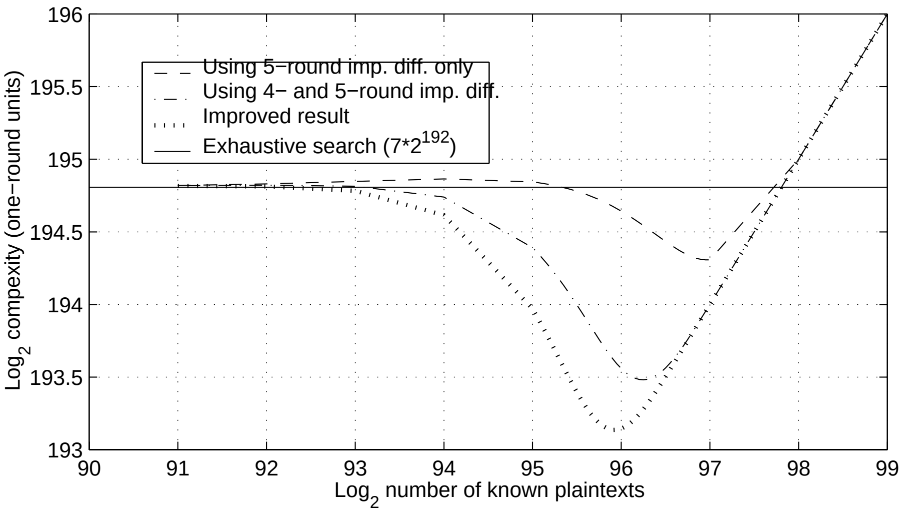

Fig. 7. *The total attack complexity for 7-round Twofish with 192-bit keys*

$e^{-2^{2\cdot p-193}} \cdot 2^{256}$ one-round computations. Note that an exhaustive search takes $7 \cdot 2^{256} = 2^{258.81}$ one-round computations.

Figure 8 shows the total complexity of all these cases, and its best results are shown in Table 2.

	one-round computations	known plaintexts
5-round imp. diff. only	2^{228}	2^{99}
5-round and 4-round imp. diff.	$2^{226.88}$	$2^{97.85}$
Improved using the 5-round and 4-round imp. diff.	$2^{226.48}$	$2^{97.46}$

Table 2. *The best results for the attack on 7-round Twofish with 256-bit keys*

References

1. Eli Biham, Alex Biryukov, Adi Shamir, *Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials*, LNCS 1592, Advanced in Cryptology - Proceeding of EUROCRYPT'99, pp. 12-23, Springer-Verlag 1999. 80
2. Niels Ferguson, *Impossible Differentials in Twofish*, Twofish Technical Report 5, Counterpane Systems, October 1999. 80, 82, 82, 88
3. Lars Knudsen, *DEAL - A 128-bit Block Cipher*, NIST AES Proposal, June 1998. 80

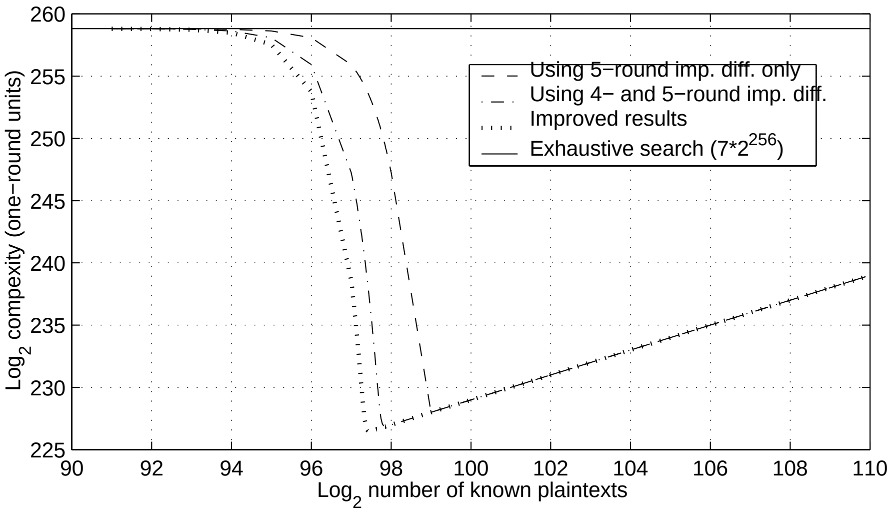

Fig. 8. *The total attack complexity for 7-round Twofish with 256-bit keys*

4. Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, *Twofish: A 128-bit Block Cipher*, NIST AES Proposal, June 1998. 80, 81

A The Subkey Computation in a Unit of Time

From Figure 3 we can see that the $\hat{K}^{6,1}$ and $\hat{K}^{6,2}$ have independent computations. Thus, we can compute $\hat{K}^{6,1}$ and in parallel $\hat{K}^{6,2}$. Without loss of generality, we show how to find the supposed values of $\hat{K}^{6,1}$, and $\hat{K}^{6,2}$ can be found similarly.

As we discussed earlier, the 32-bit MDS output difference may be either 0 or 2^{31}. In the first case (a difference 0), the 32-bit MDS input difference is 0 as well. It means that the 8-bit output differences of all the S-boxes are zeroes, and thus the 8-bit input differences of all the S-boxes are zeroes as well. So we get back to the situation described in Figure 2 for one of the words (rather than for both words as shown there). This information suffices to find all the matching subkeys directly.

In the other case the 32-bit MDS output difference is 2^{31}. The 32-bit MDS input difference is some fixed value that may be calculated by multiplying MDS^{-1} by 2^{31}. We denote the received 8-bit output differences of the S-boxes by Δ_{S0}, Δ_{S1}, Δ_{S2} and Δ_{S3}. During the attack, when we are looking for $\hat{K}^{6,1}$, we know $\hat{v}_i^{6,1}$, $\hat{v}_j^{6,1}$, a_i^6, a_j^6, Δ_{S0}, Δ_{S1}, Δ_{S2}, Δ_{S3} and S-box key S, but it still takes 2^{64} guesses to find the subkeys matching the condition. A more efficient method is to use a precomputed table, which reduces the time needed during the attack.

We consider the conditions on the right hand side of the $\hat{K}^{6,1}$ (XOR with a^6 and S-boxes), as shown in Figure 3. Before the attack we don't know the a_i^6, a_j^6 and S-box key S, but we know the Δ_{S0}, Δ_{S1}, Δ_{S2} and Δ_{S3}. We want to find all possible pairs of 32-bit values that may lead to the difference (Δ_{S0}, Δ_{S1}, Δ_{S2}, Δ_{S3}) for some a_i^6, a_j^6 and S-box key S. For each 32-bit difference there exist 2^{32} pairs satisfying the difference. We guess a S-box key S and values a_i^6, a_j^6, and build a table which, for any such guessing, gives all possible input pairs, that together with guessed values lead to required output difference (Δ_{S0}, Δ_{S1}, Δ_{S2}, Δ_{S3}). To do it in such a way takes 2^{32} (possible output pairs) $\cdot 2^{64}$ (possible S-box key S guessing) $\cdot 2^{64}$ (possible a_i^6, a_j^6 values) $= 2^{160}$ computations, that is too much for the 128-bit keys case. But the S-boxes are independent and the XOR operation of 32-bit values may be represented as four independent XOR operations of 8-bit values, so we can make the above computations in four independent computations, on 8-bit values each one. Thus we build four tables, and each of them takes 2^8 (possible output pairs) $\cdot 2^{16}$ (possible S-box key S guessing) $\cdot 2^{16}$ (possible a_i^6, a_j^6 values) $= 2^{40}$ computations, that takes in total $4 \cdot 2^{40} = 2^{42}$ computations. In addition, we hash the received input pairs by their additive difference. This reduces the time taken during the attack to find the $\hat{K}^{6,1}$, because when we know the $\hat{v}_i^{6,1}$ and $\hat{v}_j^{6,1}$, we may calculate their additive difference which is preserved when it passes through the addition operation.

During the attack we know the values: a_i^6, a_j^6, S-box key S and additive difference of the $\hat{v}_i^{6,1}$ and $\hat{v}_j^{6,1}$, so we get the possible pairs from the four tables, and starting from the least significant 8 bits to the most significant 8 bits we get $\hat{K}^{6,1}$. There is one such subkey on average, so it takes one operation on average.

Note that the fact that an additive difference may have a "carry" does not disturb us. If there is an input carry to some 8-bit additive difference, we just consult the table with the original additive difference minus one.

An Online, Transferable E-Cash Payment System

R. Sai Anand[⋆] and C.E. Veni Madhavan

Department of Computer Science & Automation
Indian Institute of Science
Bangalore, India
{anand,cevm}@csa.iisc.ernet.in

Abstract. Most electronic cash (e-cash) based payment systems that have been proposed do not possess the property of transferability. Transferability in an e-cash based system means that when a payee receives an electronic coin in a transaction he may spend it without depositing the coin first and getting a new coin issued from a bank. Usually electronic coins that are transferred in a transaction have a lifetime of the transaction itself. In this paper, we propose a payment system where coins can be transferred over multiple hands, spread over various transactions, similar to physical cash. Detection or prevention of double spending of coins is a critical issue in *online* e-cash payment systems. In our system the verification is distributed across multiple entities as opposed to the case of a coin-issuing entity or a central bank *alone* being responsible for the verification. A resolution mechanism for handling disputes is also presented. The proposed system provides guarantees of anonymity, fairness and transferability.

Key words: E-cash, Transferability, Fairness, Anonymity, Double Spending.

A Introduction

With the exponential increase in the number of users on the Internet and the market place shifting to the Internet, the role of an electronic payment system is crucial. Anonymity, transferability, fungibility (breaking large denominations to smaller ones) are some of the properties of paper cash, which an electronic payment system should possess. From security considerations, cryptographic primitives are used in electronic payment systems. If a payment system is to succeed on the Internet then the computational efforts in using these primitives need to be optimized.

In this work, we consider electronic payment systems in which the payment instrument is e-cash. Such systems are classified into two types, viz, *online* and *offline*. Online e-payment systems are those in which the transfer of electronic money between the *payer* and *payee* takes place in the presence of a third party, usually a bank, that guarantees the authenticity of the *coins* being transferred. In

[⋆] Presently at IBM India Research Lab, New Delhi, India contrast, in offline systems the transaction occurs between the two parties, payer and payee, alone. The money transferred is verified when the payee *deposits* the coins with a bank. Transferability of coins is a missing feature in most of the systems that have been proposed so far, whether online or offline. The *lifetime* of a coin is the lifetime of the transaction it is involved in, in such payment schemes. This is in contrast to paper cash where the money retains its value over several transactions and merely changes hand. An obvious advantage of transferable cash is that a coin-issuing authority need not issue new coins for every transaction that takes place.

A disadvantage, that has often been pointed out, of online systems is the unfavourable load that would be placed on the central bank or coin-issuing authority that would check the authenticity of the coin. This is certainly a bottleneck since each coin in a transaction needs to be verified by a central server. In the proposed scheme, we do away with a central verifying server and balance the load across several entities. We also require a dispute resolving mechanism in place for the electronic payment system. Given the fact that the payer and payee do not even know each others' real identities as they transact over the Internet, the payment system should be able to give guarantees to both the parties in the transaction. We outline such a dispute resolution protocol as a part of our proposed payment system.

We briefly review some of the existing e-cash based payment systems. For brevity, we choose the principal systems [3,6]. Certain other related and interesting systems are [2,5,7,8,10,11].

David Chaum's Ecash[6] is a fully anonymous, secure online electronic cash system. It implements anonymity using blind signature techniques. The Ecash system consists of three main entities:

- Banks who issue coins, validate existing coins and exchange real money for Ecash.
- Buyers who have accounts with a bank, from which they can withdraw and deposit Ecash coins.
- Merchants who can accept Ecash coins in payment for information, or hard goods. Merchants can also run a pay-out service where they can pay a client Ecash coins.

To withdraw a coin, the user generates a coin(message), m, consisting of a random serial number, r, multiplied by a blinding factor, b, and the denomination. This message, m, is signed by the user using his private key and sent to the bank after encrypting the message using the bank's public key. The bank signs the blinded coin and debits the user's account. The user unblinds the coin by dividing by an appropriate blinding factor. Thus, the bank cannot link the Ecash to the user.

While spending, the coins are securely transferred to the merchant. The merchant verifies the coins by sending them to the bank. After ascertaining that the coins are not double spent, the bank credits the merchant's account and the coin is destroyed. If the coin is double spent the bank sends an appropriate message to abort the transaction.

The advantage of Ecash is that it is fully anonymous and secure as it uses public key cryptography. The downside is that the database of spent coins gets bigger and new coins have to be issued for every transaction.

Stefan Brands proposed an offline e-cash payment system[3]. In this scheme, three participants are involved : the computer at the bank, computer of an Internet service provider and the machine of the user. The user's machine is interfaced with a tamper resistant device. The tamper resistant device increases the counter at withdrawal time by the amount that is withdrawn and decreases the counter when a payment is made. To make a payment from the user to the Internet service provider and for the latter to verify that the payment is genuine, a secret key is installed in the device. When a specified amount is transferred this key is used to sign the amount. The service provider can now use the bank's public key to verify the authenticity of the electronic money so transferred. The user does not know the secret key and hence cannot produce the signature. After the digital signature is verified, the service provider accepts and provides the requested service to the user.

The advantage of this system is that transactions do not require the presence of a third party for verification. Thus offline operation provides lesser communication overheads. However, if the device is broken by anybody, a change of device for every user of the system will be necessary.

Our system, combines the feature of anonymity provided by E-cash with the feature of transferability. This obviates the withdrawal and deposit protocols for each transaction, as in the case of [6]. For handling disputes during a transaction, we propose a resolution protocol. In an online system, a single entity is responsible for verification of coins. In the system that we propose, this load is balanced across several verifying authorities.

This paper is organized as follows. In Sect. 2 we describe the e-payment setup and describe the basic coin exchange, coin-withdrawal, coin deposit and dispute resolution protocols. In Sect. 3 we describe the features - anonymity, fairness, transferability - of our proposed payment system and how it achieves these. In Sect. 4 we discuss some optimizations and extensions to the proposed scheme.

B The e-Payment System

In this section, we present our online payment system. Apart from ensuring security and anonymity, the system will also incorporate the feature of transferability.

B.1 Setup and Notations

There are three parties involved in the basic coin exchange mechanism: the Payer, designated as the Customer (C), the Payee, called a Merchant (M) and a Verifying Authority (VA). In a transaction the coins are transferred from C to M and the coins are verified by the VA. The VA's job is two-fold: First, he has to verify that the coin has not been spent previously and next, he needs to affix his signature along with requisite information on the coin to allow the merchant M to spend the coin later.

Before describing the protocol, we set up notations for the rest of the paper. A signature on a message, MESG, by an entity X will be denoted as S_X(MESG). H(.) denotes a strong collision resistant one-way hash function. A symmetric key between two entities X and Y will be denoted by K_{XY}.

In our system the coin, COIN, is a bit string consisting of three parts. The format of the coin is shown in Fig. 1 below.

$$\{\ S_B(\text{SNO, DENM, EXPD, TS }),\ S_{VA_0}(\text{VA}_1,\ \text{TS}_0,\ \text{H(SNO, TS)}),\ \text{VA}_0\ \}$$

Fig. 1. Coin Format

The first part has the fields, serial number (SNO), denomination of the coin (DENM), expiry date of the coin (EXPD) and the timestamp of issue (TS). This part is signed by the bank using its private key. The second part consists of the name of the next verifying authority (VA_1), a timestamp (TS_0) at which verification is done and the hash of serial number (SNO) and timestamp of issue (TS) which are fields of the first part. The second part is then signed by the present verifying authority (VA_0). At issue stage, the present verifying authority is the bank B itself. Also, the timestamp (TS_0) at issue stage is the timestamp of issue, TS. The third part of the coin is the name of the present verifying authority (VA_0). The first part of the coin remains unchanged throughout the lifetime of the coin, i. e. till its expiry date.

B.2 Coin Exchange Protocol

The basic coin exchange protocol is explained below. Figure 2 shows the messages exchanged between C, M and VA_i, the current verifying authority of the coin. We now detail the steps.

Step 1: The customer C requests for goods from merchant M.

Step 2: M sends a signed message containing the transaction number (TID), the description (DESC) of the goods, price (PRICE) and a time stamp (TS).

Step 3: C sends an encrypted message, signed using the public key of VA_i at which the coins are to be verified. This message consists of COIN and a hash of the TID, DESC, PRICE and TS. Along with the encrypted message, the name of VA_i is also sent unencrypted so that M knows where he needs to send the coin for verification.

Step 4: M sends to VA_i two items. (i) The encrypted message received from C in Step 3 and (ii) A composite message consisting of the name of the next verifying authority, VA$_{i+1}$, the above hash of the TID, DESC, PRICE and TS and a symmetric key K$_{MV}$. VA$_{i+1}$ is the name of the next verifying authority, to which M would like to send the coin for verification, when M spends it. K$_{MV}$ is generated by M for one session of coin exchange protocol. This is used for sending the signed coin back to M. This composite message is encrypted with the public key of VA$_i$.

Step 5: VA$_i$ decrypts the encrypted message and verifies that the coins have not been spent previously after consulting his database and signs the coin.

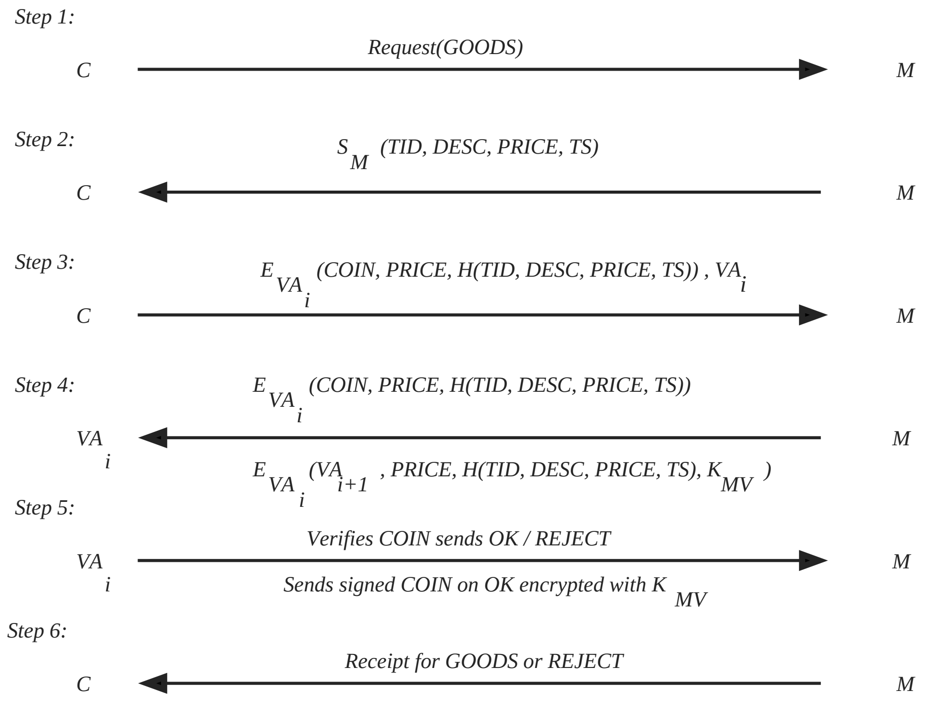
Step 1:
C
Request(GOODS)
M
Step 2:
C
S_M (TID, DESC, PRICE, TS)
M
Step 3:
C
E_{VA_i} (COIN, PRICE, H(TID, DESC, PRICE, TS)) , VA_i
M
Step 4:
VA_i
E_{VA_i} (COIN, PRICE, H(TID, DESC, PRICE, TS))
E_{VA_i} (VA_{i+1} , PRICE, H(TID, DESC, PRICE, TS), K_{MV})
M
Step 5:
VA_i
Verifies COIN sends OK / REJECT
Sends signed COIN on OK encrypted with K_{MV}
M
Step 6:
C
Receipt for GOODS or REJECT
M

Fig. 2. Basic Coin Exchange Protocol

The verification of the COIN is done as follows: Firstly, VA$_i$ verifies from the second part that he is indeed the current valid verifying authority. Next, VA$_i$ checks if the SNO appearing on the COIN is listed in his database. If not, then the COIN is authentic and proceeds to sign the COIN. If the COIN is listed in his database, he checks the timestamp appearing on the second part of the COIN. If this timestamp is greater than the time stamp in the database corresponding to the SNO of the COIN, then the COIN is authentic. If both of the above conditions are violated then the COIN is treated as double spent and a REJECT signal is conveyed to M. The entries that need to be made in the verifying authority's database when a COIN is found to be authentic are: (a)

The SNO, if it is not already in the database and (b) the timestamp (TS) of previous verification which appears in the second part.

If the COIN is verified as being authentic, the verifying authority, VA_i, needs to sign it. This signing is done by replacing the second part of the COIN. The second part will now contain the new timestamp (TS_i) at which the COIN is being verified, the next verifying authority's name (VA_{i+1}) and the hash value that existed in the replaced part. Of course, he may also verify that the hash value is correct by doing the hash computation. The third part of the COIN is replaced by the name of the present verifying authority, VA_i. Figure 3 shows the coin being signed by VA_i.

Coin being Verified by VA_i:

$$\{S_B(\text{SNO, DENM, EXPD, TS }), S_{VA_{i-1}}(VA_i, TS_{i-1}, H(\text{SNO, TS})), VA_{i-1}\}$$

Coin after Signing by VA_i:

$$\{S_B(\text{SNO, DENM, EXPD, TS }), S_{VA_i}(VA_{i+1}, TS_i, H(\text{SNO, TS})), VA_i\}$$

Fig. 3. Coin verification/signing at VA_i

After verification and signing, VA_i sends an OK signal and the signed COIN back to M encrypting it with the key K_{MV} sent by the merchant. If the coin is not authentic a REJECT signal is sent to M.

Step 6: M on receipt of an OK signal sends a signed receipt for the goods to C. Similarly, on receipt of a REJECT signal M informs C accordingly.

Thus, if the COIN is not double spent, then it can be used by the current owner M for his next transaction.

B.3 Withdrawal/Deposit Protocols

These protocols are executed infrequently since the payment mechanism guarantees transferability of coins. This implies that the coin issuing authority need not create new coins for every transaction as in e-payment mechanisms which do not possess this transferability property. The deposit and withdrawal protocol are now described below (Figs. 4 & 5).

Coin Withdrawal

Step 1: The user C sends a request for coins, indicating the amount, the change required and his account number with the bank/coin issuing authority B.

Step 2: B verifies the outstanding balance in the account and if the requested amount is available in the account, issues the coins to the customer. The bank is provided with a public key of C when the account is opened which is used to encrypt the coins while sending them to C. It is possible to include the blind signature mechanism to provide for anonymity of C as in [1,4,9].

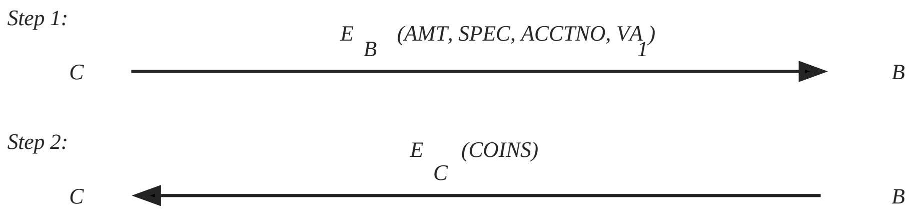

Fig. 4. Coin Withdrawal Protocol

Coin Deposit

Step 1: The customer C, who wants to deposit the coins, packs the coins and encrypts them using bank B's public key. He provides the account number to which the deposit is to be made.

Step 2: The bank B verifies the authenticity of the coins with the appropriate verifying authority VA_k. On confirmation from VA_k, the bank credits the amount to the account specified by C.

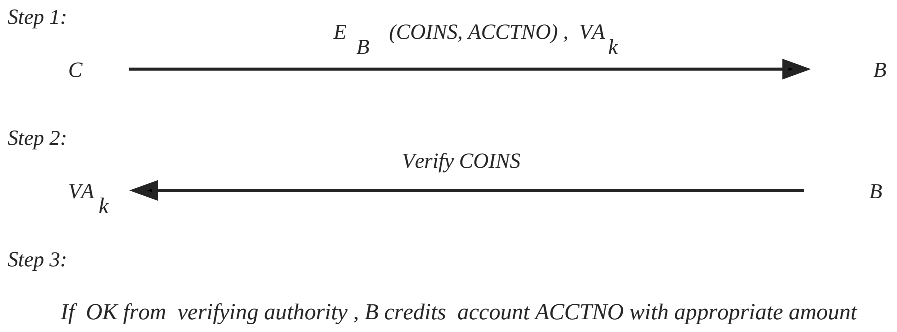
Step 1:
E_B (COINS, ACCTNO) , VA_k
C
B
Step 2:
Verify COINS
VA_k
B
Step 3:
If OK from verifying authority , B credits account ACCTNO with appropriate amount

Fig. 5. Coin Deposit Protocol

B.4 Resolution Protocol

The e-payment mechanism proposed provides guarantees of fairness to both the payer and payee. Thus, if coins are transferred from one entity to another in a transaction and a dispute arises, neither party loses any money or goods. In this subsection we detail the resolve protocol that has to be executed in case of disputes. This protocol also handles aborted transactions due to system failures.

The payee in our system is guaranteed an authentic payment since the verifying authority authenticates the coins. When the payee receives an OK signal from some VA_i it also receives the properly time-stamped coins which he may use later for future transactions. On a REJECT signal from VA_i, the same may be used by the payee to prove that the transaction was not completed and therefore, refuse transfer of goods.

The second part of the guarantee is what happens if a payee refuses to transfer goods even though valid coins have been handed over by the verifying authority. In Step 2 of the basic coin exchange protocol, the merchant M returns a signed message containing the description of goods (DESC), transaction ID (TID), the price (PRICE) and the time stamp (TS) at which the transaction occurs. The merchant cannot deny sending the message since it carries his signature. If the customer C can prove that the coins he had sent to M were valid coins and were authenticated by the verifying authority, then he can lay claim to the goods negotiated during the transaction. To this end, he executes the following resolution protocol:

Resolution Protocol

Step 1: C sends a resolve request with the signed message obtained from merchant M and a hash of the coins he had used in the transaction to the verifying authority VA_i at which the coin was sent.

Step 2: VA_i on receipt of the request checks the signature of M and if found valid checks if the coins spent were authenticated by it. If the claim of C is found correct, VA_i directs M to transfer the goods. If M does not accept then the coins transferred to it are invalidated by sending an appropriate message to VA_{i+1}. Also, the coins spent by C are restored to it so that C does not lose money in the aborted transaction.

C Features

In this section we describe the main guarantees of our proposed system and how it achieves them.

C.1 Anonymity

Anonymity is provided to both payer and payee as long as both interact truthfully. At no point during the transaction does the payee come to know about the payer's identity. The payer merely transfers the encrypted version of coins used during the transaction and the coins themselves do not contain the identity of the payer. Thus, neither the payee nor the verifying authority can break the anonymity of the payer. In any case, a payee with honest intentions will be satisfied if he gets paid for the goods being transferred and has no advantage in getting to know the identity of the payer. Similarly, the verifying authority is concerned about the validity of the coins being spent and the identity of the payer is immaterial for this purpose.

However, the identity of the payee is known to the payer. Without this knowledge the payer cannot transfer the coins to the payee. What the payee will be interested in is whether he remains anonymous to the verifying authority. This is guaranteed by the protocol since the merchant's identity is never revealed in the basic coin exchange protocol. If the merchant tries to abort the transaction on receipt of valid coins from the verifying authority then the resolution protocol initiated by the customer would reveal the identity of the payee to the verifying authority. Thus, the merchant remains anonymous to the verifying authority as long as he involves himself in the transaction in a fair manner.

C.2 Fairness

Our proposed system guarantees fairness to both the parties, the customer C and the merchant M. By *fairness* we mean that irrespective of the final outcome of a transaction, whether completed or aborted, none of the parties involved in a transaction suffer a loss in terms of money or goods. The merchant M transfers goods to C only on receipt of coins authenticated by the verifying authority. Thus the e-payment mechanism guarantees that he does not have to deliver goods if proper payment is not made. A resolution protocol initiated by a customer on a "double spent" coin will be detected by the verifying authority. Also, the merchant himself will hold a REJECT signal signed by the verifying authority for the invalid transaction. This is proof enough for the merchant to not deliver the goods. Hence, the payment mechanism ensures fairness to the merchant.

The Customer C is guaranteed fairness by the resolution protocol. If he has indeed transferred valid coins, that are not double spent ones, then by initiating the resolution protocol the erring merchant M can be made to deliver the goods. A merchant will not dishonour a transaction since he will run the danger of being blacklisted. This may adversely affect his future business prospects.

C.3 Transferability

A major and significant feature of our proposed mechanism is the transferability of coins. Transferability of a coin is achieved by time stamping it at every transaction by a verifying authority. Transferability would imply that there would be no need for the withdrawal and deposit protocols for every transaction. Also, for achieving this feature there is no dependence on a unique trusted third party.

The load of verification of the coins is evenly spread across several verifying authorities so that congestion does not become a bottleneck.

D Extensions/Optimizations

In this section we outline some extensions to the basic coin exchange protocol to cover the situation where more than one coin is spent in a transaction. We also discuss a hierarchy within the verifying authorities to optimize the coin verification stage.

D.1 Payment with Multiple Coins

When more than one coin is spent in a transaction it is not practical to encrypt all the coins using a public key of the verifying authority. Instead, the customer C generates a symmetric key, K_{CV}, and encrypts this key and the hash value as described in Step 3 of the basic coin exchange protocol using the public key of the verifying authority. The coins are then encrypted using the symmetric key, K_{CV}, generated. This key could also be used to send the change back to C if exact change cannot be tendered by C. Since, the verifying authority does not know the identity of the customer the "change" coins will be sent via the merchant who will piggyback these along with the signed receipt of the goods.

D.2 Fungibility

Fungibility (i.e. the feature of breaking a coin into coins of smaller denominations) can be achieved in the following manner. The user is provided with a tamper resistant device which will take as input, from the user, the coin that is to be split into smaller denominations. The exact change necessary is specified by the user. The device has a secret key embedded within it that it uses to sign the smaller denominations. The coins that are signed do not have the sanctity of a coin signed by the bank. But, the verifying authority can ensure that the change produced is from a genuine coin. The change produced by the device contains all the information of the coin that is being "changed" and also the details of the change itself. It should also contain a nonce so that the same coin cannot be used to make a different set of change later. Finally, the change is signed using the key installed within the device before returning it to the user.

D.3 Organization of Verifying Authority

In practice, most transactions would involve exchange of more than a single coin. In this case, for improving the efficiency of the coin verification stage a distribution of the work across a hierarchy is necessary. For each denomination of coins a different entity verifies coins of that denomination. During peak hours, depending on the load at each of these entities the verification could be distributed evenly across them.

E Conclusions

We have described an electronic payment system with the transferability property. This eliminates the withdrawal and deposit protocols. Also, new coins need not be issued for every transaction as in payment schemes without the transferability property. We have removed the unfavourable load that would be placed on the authority that verifies coins by distributing the load across several entities. The computations involved compare favourably with schemes of similar nature. Fairness to both merchant and customer is ensured by the resolution protocol. Thus, the system ensures that neither money nor goods are lost by either party. This holds good for transactions that are completed normally or aborted. The proposed system also provides security and anonymity guarantees to the participants in a transaction.

References

1. Abe, M. , Fujisaki, E.: How to Date Blind Signatures. Advances in Cryptology - ASIACRYPT '96, LNCS, Volume 1163.
2. Anderson, R., Manifavas, C., Sutherland, C.: Netcard - a practical electronic cash system. Fourth Cambridge Workshop on Security Protocols, LNCS, April 1996.
3. Brands, S.: Electronic Cash on the Internet. Proceedings of the Internet Society 1995 Symposium on Network and Distributed System Security, February 1995.
4. Chaum, D.: Blind Signatures for Untraceable Payments. Advances in Cryptology - Crypto '82 Proceedings, Plenum Press, 1983.
5. Chaum,D.: Security without Identification: Transaction Systems to Make Big Brother Obsolete. Communications of the ACM v.28, n.10, October 1985.
6. Chaum, D., Fiat, A., Naor, M.: Untraceable Electronic Cash. Advances in Cryptology - CRYPTO'88, LNCS, Volume 403, 1990.
7. Hauser, R., Steiner, M., Waidner, M.: MicroPayments based on iKP Technical report. IBM Research Division, Zurich Research Laboratory, January, 1996.
8. Medvinsky, G., Neumann, B.C.: NetCash: A design for practical electronic currency on the Internet. First ACM Conference on Computer and Communication Security, November 1993.
9. Pointcheval, D., Stern, J.:Provably Secure Blind signature Schemes. Advances in Cryptology - ASIACRYPT '96, LNCS, Volume 1163. preprint, MIT.
10. Rivest, R.L.: Electronic Lottery Tickets as Micropayments.
11. Rivest, R.L., Shamir, A.: PayWord and Micromint - Two simple Micropayment Schemes. preprint, MIT, May 1996.

Anonymity Control in Multi-bank E-Cash System

Ik Rae Jeong and Dong Hoon Lee

Dept. of Computer Science, Korea University,
Chochiwon,Chungnam, 339-700, KOREA

Abstract. Most electronic cash systems in the literature have been developed in the single bank model in which clients and merchants have accounts at the same bank. In the real world, electronic cash may be issued by a large group of banks, monitored by the Central Bank. Thus not only client anonymity but also bank anonymity should be considered to simulate anonymity of real money. Because anonymity could be in conflict with law enforcement, anonymity of both clients and banks must be contollable such that the identity of the bank is concealed but identifiable by the Central Bank in case of dispute and the client is anonymous but revocable by a trusted third party. In this paper we study electronic cash system in the multiple bank model in which clients and merchants have their accounts at different banks, especially from the viewpoint of anonymity control. Client anonymity control and bank anonymity control are achieved by fair blind signatures and group signatures, respectively. By merging fair blind signatures and group signatures, we provide both client anonymity control and group anonymity control efficiently.

A Introduction

Most proposed electronic cash systems in the literature have been developed on the assumption that clients and merchants have their accounts at the same bank. Several electronic cash requirements such as anonymity, transferability, divisibility, anonymity control have been analyzed on this assumption. But in the real world there exist several banks issuing real cash and so electronic cash system in more complicated model, so called the multiple bank model, should be considered to reflect real cash closer. In the paper we develop an electronic cash system in the multiple bank model and analyze the system, especially from the viewpoint of anonymity control.

Anonymity in electronic cash systems is considered useful with the argument that real cash is also anonymous and that clients of the systems prefer to keep their everyday payment activities private. But anonymity could be used for blackmailing or money laundering by criminals without revealing their identities. To make e-cash systems acceptable to government, anonymity control should be provided such that a trusted third party, called a *trustee*, has an ability to revoke the identities of clients in case of unlawful or suspicious transactions.

In the single bank model all of the clients have their accounts at the same bank and hence anonymity means client anonymity. But in the multiple bank model clients and merchants may have their accounts at different banks, so electronic cash system should provide both client anonymity and bank anonymity like real cash, therefore both client anonymity control and bank anonymity control. Without bank anonymity, client anonymity could be restricted since whoever sees cash issued at the certain bank knows that the owner of the cash is someone who has the account at the bank.

Anonymity control of clients: An electronic cash system providing client anonymity control is called a fair electronic cash system. It is important from an operational point of view that the trustee is *off-line*, i.e., neither involved in transactions nor in the opening of accounts. Two fair off-line payment systems were studied in [4,8] independently. The system in [4] uses "coin identifier" and is more efficient in the communication and computation overhead than one in [8]. The system in [8] embeds the identity of a client in coins and hence searches the relatively small account database using the revoked identity, while the system in [4] has to search the withdrawal transaction database to trace the owner of coins. Recently some efficient fair off-line payment systems [9] were suggested by combining techniques in [4] and [8].

Anonymity control of banks: For bank anonymity control we use a group signature. It allows members of a group to sign messages anonymously on behalf of the group. The signed messages are then verified by a group public key. In case of dispute, only a designated group manager can reveal the identity of the member. Various group signature schemes have been investigated to develop the efficient scheme of which the length of signatures and the size of the group public key are independent of the size of the group. But only a few schemes [3,5] satisfy both requirements. Group signature schemes should be coalition resistant. In other words no subset of group members and the group manager is able to collude and generate valid group signatures that are untraceable or from which the trustee revokes the identity of another group member. The scheme in [3] is the provable coalition resistant group signature scheme.

In this paper we develop a generic fair off-line e-cash system in the multiple bank model with both client and bank anonymity controls. For client anonymity control, the generic fair e-cash system is first constructed. For bank anonymity control we modify the blind group signature scheme in [10] in order to make coalition attacks hard. The scheme in [10] is the blinded version of [5] which is known to be vulnerable to a coalition attack [1]. This modified blind group signature scheme, the protocol **W** in this paper, is merged with the generic fair off-line e-cash system developed, thus providing both client and bank anonymity controls in the multiple bank model efficiently.

B Signatures of Knowledge

To build the system proposed in the paper, we use proof systems in which one party can convince other parties that he knows certain values without leaking useful information. In this section the well-known signature of knowledge systems introduced in [4,5] are defined. Since they are used for the purpose of both signing a message and providing knowledge of a secret, they are called signatures of knowledge.

Definition 1. A signature of the knowledge of representations of $y_1, ..., y_w$ with respect to the bases $g_1, ..., g_v$ on the message m is denoted as follows :

$$SKREP[(\alpha_1, ..., \alpha_u) : (y_1 = \prod_{j=1}^{l_1} g_{b_{1j}}^{\alpha_{e_{1j}}}) \wedge ... \wedge (y_w = \prod_{j=1}^{l_w} g_{b_{wj}}^{\alpha_{e_{wj}}})](m),$$

where the indices $e_{ij} \in \{1, ..., u\}$ refer to the elements $\alpha_1, ...\alpha_u$ and the indices $b_{ij} \in \{1, ..., v\}$ refer to the base elements $g_1, ..., g_v$. The signature consists of an $(u + 1)$ tuple $(c, s_1, ..., s_u) \in \{0, 1\}^k \times Z_n^u$ satisfying the equation

$$c = H(m||y_1||...||y_w||g_1||...||g_v||y_1^c \prod_{j=1}^{l_1} g_{b_{1j}}^{s_{e_{1j}}} ||...||y_w^c \prod_{j=1}^{l_w} g_{b_{wj}}^{s_{e_{wj}}}).$$

For example, $SKREP[(\alpha_1, \alpha_2) : y_1 = g_2^{\alpha_1} \wedge y_2 = g_1^{\alpha_1} g_2^{\alpha_2}](m)$ is used for proving the knowledge of the discrete logarithm of y_1 to the base g_2 and the representation of y_2 to the bases (g_1, g_2) without revealing any information about (α_1, α_2), where the g_1 part of this representation is equal to the discrete logarithm of y_1. The $SKREP$ can be computed only if the secrets (α_1, α_2) are known.

The following two signatures of knowledge are based on the double discrete logarithm and the root of the discrete logarithm problem, respectively.

Definition 2. Let $l \leq k$ be a security parameter. An $(l+1)$ tuple $(c, s_1, ..., s_l) \in \{0, 1\}^k \times Z^l$ satisfying the equation

$$c = H(m||y||g||a||t_1||...||t_l), \ where\ t_i = \begin{cases} g^{(a^{s_i})} & if\ c[i] = 0 \\ y^{(a^{s_i})} & otherwise \end{cases}$$

is a signature of the knowledge of a double discrete logarithm of y to the bases g and a, and is denoted $SKLOGLOG[\alpha : y = g^{(a^{\alpha})}](m)$.

$SKLOGLOG[\alpha : y = g^{(a^{\alpha})}](m)$ is used for proving the knowledge of the double discrete logarithm of y without revealing any information about α. This $SKLOGLOG$ can be computed only if the secret α is known.

Definition 3. Let $l \leq k$ be a security parameter. An $(l+1)$ tuple $(c, s_1, ..., s_l) \in \{0, 1\}^k \times Z_n^{*l}$ satisfying the equation

$$c = H(m||y||g||e||t_1||...||t_l), \ where\ t_i = \begin{cases} g^{(s_i^e)} & if\ c[i] = 0 \\ y^{(s_i^e)} & otherwise \end{cases}$$

is a signature of the knowledge of an $e-$th root of the discrete logarithm of y to the bases g, and is denoted $SKROOTLOG[\alpha : y = g^{\alpha^e}](m)$.

$SKROOTLOG[\alpha : y = g^{\alpha^e}](m)$ is used for proving the knowledge of an e-th root of the discrete logarithm of y without revealing any information about α. This $SKROOTLOG$ can be computed only if the secret α is known.

C A Generic Fair Off-Line Electronic Cash System in the Multiple Bank Model

In this section we describe a generic fair off-line e-cash system using a *restrictive blind group signature scheme* as a building block.

A group signature provides signer's anonymity control since a group manager, in case of later dispute, can reveal the identity of the signer. A blind group signature scheme [10] adds a blindness property to group signatures such that if the signer later sees a message he has signed, he will not be able to determine for whom he signed it. So the blind group signature scheme is for both signer's anonymity control and receiver's anonymity. A restrictive blind group signature scheme adds a restrictive blindness property to blind group signatures. The restrictive blind signature protocol is defined by Brands in [2] as follows. It is implemented using Chaum-pedersen's blind signature scheme[6].

Definition 4. Let $\tilde{m} \in G$ (in general, it can be a vector of elements) be the blinded value of m such that the receiver at the start of a blind signature protocol knows a representation $(a_1, ..., a_k)$ of $\tilde{m}$ with respect to a generator-tuple $(g_1, ...g_k)$. The signer verifies that the internal structure of the blinded message $\tilde{m}$. Let $(b_1, ..., b_k)$ be the representation the receiver knows of the number m after the protocol has finished. If there exist two functions I_1 and I_2 such that

$$I_1(a_1, ..., a_k) = I_2(b_1, ..., b_k),$$

regardless of m and the blinding transformations applied by the receiver, then the protocol is called a *restrictive blind signature* protocol. The functions I_1 and I_2 are called *blinding-invariant functions* of the protocol with respect to $(g_1, ..., g_k)$.

We construct the restrictive blind group signature scheme, the protocol **W**, in the following section. It is the modification of the blind group signature scheme in [10]. In our scheme, the restrictive blindness property in the restrictive blind group signature enables that the correct identity of a client is embedded in coins, which is used in owner tracing.

C.1 System Setup and Making Subprotocol

We assume for the sake of simplicity that there is only one coin denomination, one group manager and one trustee. The extension to multiple denominations, several group managers and several trustees is easy.

The group manager chooses a cyclic group G of order $|G|$, generators g, g_1, g_2 and g_3 such that computing discrete logarithm is infeasible, and then publishes

(G, g, g_1, g_2, g_3). The bank chooses a secret key and gets a membership certificate from the group manager. The trustee chooses its secret key $x_T \in Z^*_{|G|}$ and computes $y_T = g_2^{x_T}$ and $h_T = g_2^{x_T^{-1}}$, and publishes y_T and h_T.

To open an account at the bank, a client first proves his identity by means of, say, a passport. The client generates a random number u_1 and computes $I = g_1^{u_1}$, then sends I to the bank. The bank stores the identifying information of the client and I in the account database. I is served as the account number of the client.

In the construction of the system, a group signature scheme is converted into a restrictive blind group signature scheme. To control anonymity, the order of a cyclic group G of the base group signature should be known. The scheme in [5] publishes the group order while the schemes in [3] do not. Although it seems to be possible to convert group signature schemes in [3] into blind group signature schemes, it seems to be impossible to convert those schemes to restrictive blind group signature schemes.

C.2 Procedure to Construct the Generic Fair Cash System

$$
\begin{array}{lcr}
\text{Client} & & \text{Bank} \\
\hline
k \in_R Z_{|G|}, \alpha \in_R Z^*_{|G|} & & \\
D_1 = g_2^k, D_2 = I \cdot y_T^k & & \\
m_0 = D_2 \cdot g_3 & & \\
\tilde{m_0} = m_0^\alpha = (D_2 \cdot g_3)^\alpha & & \\
U = \text{proof of } \tilde{m_0} \text{ structure} & \xrightarrow{\tilde{m_0}, U} & \text{verify } U \\
 & & \\
\tilde{m_0} & & \tilde{m_0} \\
\hline
 & \text{Restrictive Blind Group Signature Scheme} & \\
\hline
S_B(m_0) & &
\end{array}
$$

Coin : $\{\ D_1, S_B(m_0), V_U, V_E\ \}$,
where $S_B(m_0) = S_B(D_2 \cdot g_3)$,
V_U = proof of unblinding of the commitment ,
V_E = proof of the correct Elgamal encryption of I

Fig. 1. The generic fair scheme in distributed banking

Let g_1 and g_2 be generators and y_T be the trustee's public key. The Elgamal encryption of the client identity $I(= g_1^{u_1})$ with the trustee's public key consists of $\{D_1, D_2\}$ where

$$D_1 = g_2^k, D_2 = I y_T^k.$$

The trustee can recover the client identity from $\{D_1, D_2\}$ with his secret key x_T as follows :

$$D_1^{-x_T} D_2 = g_2^{-kx_T} I g_2^{kx_T} = I.$$

In our generic fair cash scheme, the coin consists of three parts:

$$Coin = \{D_1, S_B(D_2 \cdot g_3), V\}.$$

D_1 and D_2 are the Elgamal encryption of the client identity. $S_B(D_2 \cdot g_3)$ is generated through the restrictive blind signature scheme. V is the proof which concatenates the proof V_U of unblinding of the commitment and the proof V_E of the correct Elgamal encryption of I. Now we describe the procedures to construct the generic fair e-cash system. The generic fair off-line e-cash system is shown in Figure 1.

1. *Commitment construction* : $(D_2 \cdot g_3)^\alpha$ which is the blinded value of $D_2 \cdot g_3$ with the random value α and is sent to the bank or constructed using committed values by the bank. Another commitment to be used in coin tracing is also sent to the bank.
2. *Proof of the commitment structure* : The blind group signature scheme is converted into the restrictive blind group signature scheme by making the bank check the structure of the commitments. The proof is generated to show that $(D_2 \cdot g_3)^\alpha$ is really the blinded value of $(D_2 \cdot g_3 = I \cdot y_T^k \cdot g_3)$. Another proof which makes sure that the trustee can trace the coin from the withdrawal instance is also sent to the bank. Two proofs are merged in our scheme.
3. *Signature generation through the restrictive blind group signature scheme*: When the client withdraws coins, the bank generates signatures through the restrictive blind group signature scheme. So the client is unable to blind the internal structure of the commitment.
4. *Proof of unblinding of the commitment* : To make the trustee be able to recover the client identity from the coin, the client must unblind the commitment $(D_2 \cdot g_3)^\alpha$ into $D_2 \cdot g_3$. The proof of the unblinding is satisfied by showing that the client knows the representation of $\frac{D_2 \cdot g_3}{g_3}$ with respect to (g_1, y_T). In our scheme V is the proof of the unblinding.
5. *Proof of the correct Elgamal encryption of I* : The Elgamal encryption consists of D_1 and D_2. To make the trustee recover the client identity from the coin, the exponent of g_2 in the representation of D_1 must be equal to the exponent of y_T in the representation of $D_2 \cdot g_3$. In our scheme the proof V also acts as the proof of the correct Elgamal encryption.

The generic fair e-cash system provides three functionalities in the single bank model, i.e., owner tracing, coin tracing and recovering the identity of the double spender. To make owner tracing possible, we insert into the coin the Elgamal encryption of the client identity with the trustee's public key. The trustee can recover the client identity from the coin by decrypting the Elgamal encryption.

To make coin tracing possible, the bank must look into the structure of the commitment in the withdrawal protocol. The trustee can trace the coin from the commitments. To make the merchant recover the client identity when double spending happens, the client is required to prove the knowledge of random values which are used in the coin by responding the challenge from the merchant. If double spending happens, the bank can know the random values and calculate the client identity.

In the multiple bank model another functionality, i.e. signer tracing, is provided by the blind group signature scheme which is used as subprotocol in the generic fair cash system.

D The Implementation of the Generic Fair Electronic Cash System in the Multiple Bank Model

In this section, we implement a fair off-line e-cash system where a group of banks distribute electronic cash. First the restrictive blind group signature scheme, the protocol **W**, is constructed. Using the protocol **W** as subprotocol, we implement a generic fair off-line e-cash system in the multiple bank model.

D.1 System Setup

To set up the group signature scheme, the group manager chooses a security parameter l and computes the following values:

1. An RSA public key (n, e), secret key d such that $ed \equiv 1(mod\ n)$, where the length of n is at least $2l$ bits.
2. A cyclic group G of order n, generators g, g_1, g_2 and g_3 such that computing discrete logarithm is infeasible. In particular, we can choose G to be a cyclic subgroup of Z_P^* where P is a prime and $n|(P-1)$.
3. An element $a \in Z_n^*$ where a has large multiplicative order modulo all the prime factors of n.
4. An element t where the logarithm of t to the base a is unknown to group members.
5. An upper bound λ on the length of the secret keys and a constant $\mu > 1$.

The group's public key is $(n, e, G, g, g_1, g_2, g_3, a, \lambda, \mu, t)$.

When a bank wants to join the group, it picks a *secret key* $x \in \{0, 1, ..., 2^\lambda - 1\}$ and calculates $y = a^x(mod\ n)$ and the *membership key* $z = g^y$. The bank commits to y and sends (y, z) to the group manager and proves that it knows x (without actually revealing x) using techniques similar to the signature of knowledge of discrete logarithm [4]. If the group manager is convinced that the bank knows x, the group manager gives the bank a *membership certificate* $v \equiv (y+t)^d(mod\ n)$. (Note that this certificate structure is different from the original certificate structure in [5] to prevent a coalition attack. This form is appeared in [1].) It is an additional security assumption that computing v without factoring n is hard.

A client has his account I at the bank. To make his account, he selects a random number u_1 and computes his account $I = g_1^{u_1}$.

The trustee chooses its secret key $x_T \in Z_n^*$ such that $y_T = g_2^{x_T}$ and $h_T = g_2^{x_T^{-1}}$. The trustee publishes y_T and h_T.

D.2 The Protocol W

Player A $(m, \tilde{g}, w)$		Player B $(x, y, v, \tilde{g})$
		$\tilde{z} = \tilde{g}^y$
		$2^\lambda \le r_i^{LL} \le 2^{\lambda+\mu} - 1$
		$r_i^{RL} \in Z_n^*$
		$P_i^{LL} = \tilde{g}^{(a^{r_i^{LL}})}$
$\hat{g} = \tilde{g}^w$, $\hat{z} = \tilde{z}^w$	$\xleftarrow{\tilde{z}, P_i^{LL}, P_i^{RL}}$	$P_i^{RL} = \tilde{g}^{(r_i^{RL})^e}$
$\hat{P}_i^{LL} = (P_i^{LL})^w$		
$\hat{P}_i^{RL} = (P_i^{RL})^w$		
$Q_i^{LL} = \hat{P}_{\sigma(i)}^{LL}$		
$Q_i^{RL} = \hat{P}_{\sigma(i)}^{RL}$		
$2^{\lambda+\mu} \le b_i^{LL} \le 2^{\lambda+2\mu} - 1$		
$b_i^{RL} \in Z_n^*$		
$R_i^{LL} = Q_i^{LL(a^{b_i^{LL}})}$		
$R_i^{RL} = Q_i^{RL(b_i^{RL})^e}$		
$c^{LL} = H_l(m \Vert \hat{z} \Vert \hat{g} \Vert a \Vert R_1^{LL} \Vert ... \Vert R_l^{LL})$		
$c^{RL} = H_l(m \Vert \hat{z}\hat{g}^t \Vert \hat{g} \Vert e \Vert R_1^{RL} \Vert ... \Vert R_l^{RL})$		
$c^{'}[i]^{LL} = c[\sigma^{-1}(i)]^{LL}$		
$c^{'}[i]^{RL} = c[\sigma^{-1}(i)]^{RL}$	$\xrightarrow{c'^{LL}, c'^{RL}}$	$t_i^{LL} = r_i^{LL}$ if $c^{'}[i]^{LL} = 0$,
		$r_i^{LL} - x(mod\ n)$ otherwise.
		$t_i^{RL} = r_i^{RL}$ if $c^{'}[i]^{RL} = 0$,
verify t_i^{RL} with $c^{'RL}$	$\xleftarrow{t_i^{LL}, t_i^{RL}}$	$r_i^{RL}/v(mod\ n)$ otherwise.
verify t_i^{LL} with $c^{'LL}$		
$s_i^{LL} = t_{\sigma[i]}^{LL} + b_i^{LL}$		
$s_i^{RL} = t_{\sigma[i]}^{RL} \cdot b_i^{RL}$		
$(m, \hat{g}, \hat{z}, V_1, V_2)$		

Fig. 2. The protocol **W**

The protocol **W** is a restrictive blind group signature scheme which is a modification of the blind group signature scheme in [10]. To make the protocol **W** a restrictive blind group signature scheme, the commitment $\tilde{g}$ is generated by Player A while it is generated by Player B in the blind group signature scheme in [10]. The modified version of the underlying group signature scheme in [10] is also used in the protocol **W** in order to make coalition attack hard. The protocol **W** is shown in Figure 2.

Player A gets player B's blind group signature which consists of $(m, \hat{g}, \hat{z}, V_1, V_2)$ where V_1 and V_2 are the blind signatures of knowledge. The blind signature of knowledge

$$V_1 = (c^{LL}, s_i^{LL}\ for\ 1 \leq i \leq l) = SKLOGLOG[x : \hat{z} = \hat{g}^{(a^x)}](m)$$

assures that the group manager can determine the identity of the signer. The blind signature of knowledge

$$V_2 = (c^{RL}, s_i^{RL}\ for\ 1 \leq i \leq l) = SKROOTLOG[v : \hat{z}\hat{g}^t = \hat{g}^{(v^e)}](m)$$

proves that the signer is one of the group members.

In this protocol w is used in blinding $\hat{g}$ and $\hat{z}$. σ(a permutation) is used in blinding c^{LL} and c^{RL}. $b_i^{LL}(1 \leq i \leq l)$ and $b_i^{RL}(1 \leq i \leq l)$ are used to blind $s_i^{LL}(1 \leq i \leq l)$ and $s_i^{RL}(1 \leq i \leq l)$, respectively. So σ and $b_i^{LL}(1 \leq i \leq l)$ are blinding the proof $V_1 = (c^{LL}, s_i^{LL}\ for\ 1 \leq i \leq l) = SKLOGLOG[x : \hat{z} = \hat{g}^{(a^x)}](m)$ and σ and $b_i^{RL}(1 \leq i \leq l)$ are blinding the proof $V_2 = (c^{RL}, s_i^{RL}\ for\ 1 \leq i \leq l) = SKROOTLOG[v : \hat{z}\hat{g}^t = \hat{g}^{(v^e)}](m)$.

D.3 The Withdrawal Protocol

Client		Bank
$(g, g_1, g_2, g_3, h_T, y_T, t)$		$(g, x, y, v, g_1, g_2, g_3, h_T, y_T, t)$
$k, b \in_R Z_n, \alpha \in_R Z_n^*$ $U = SKREP[(\alpha, k) : M_1 = I^\alpha$ $\wedge M_2 = y_T^\alpha \wedge M_3 = g_3^\alpha$ $\wedge M_4 = M_2^k \wedge M_5 = h_T^k]$		
	$\xrightarrow{M_1, M_2, M_3, M_4, M_5, U}$	verify U
make $\tilde{m}_0 = M_1 \cdot M_4 \cdot M_3$ $= I^\alpha \cdot (y_T^k)^\alpha \cdot g_3^\alpha = (I y_T^k g_3)^\alpha$		make $\tilde{m}_0 = M_1 \cdot M_4 \cdot M_3$ $= I^\alpha \cdot (y_T^k)^\alpha \cdot g_3^\alpha = (I y_T^k g_3)^\alpha$
$(g_2^b, \tilde{m}_0, \alpha^{-1})$		$(x, y, v, \tilde{m}_0)$
	The protocol **W**	
$(g_2^b, m_0, m_0^y, V_1, V_2)$		

Fig. 3. The withdrawal protocol in distributed banking

When the client wants to withdraw a coin, he first must prove ownership of his account by any means. Then the withdrawal protocol is performed. We use the protocol **W** as subprotocol of the withdrawal protocol. The withdrawal protocol is shown in Figure 3. The client first constructs commitment $(D_2 \cdot g_3)^\alpha$

and generates the signature U of the commitment structure as described in Section 4. The signature of knowledge

$$\begin{aligned} U &= (c, s_1, s_2) \\ &= SKREP[(\alpha, k) : M_1 = I^{\alpha} \wedge M_2 = y_T^{\alpha} \wedge M_3 = g_3^{\alpha} \wedge M_4 = M_2^{k} \wedge M_5 = h_T^{k}] \end{aligned}$$

demonstrates two facts to the bank. First, $(D_2 \cdot g_3)^{\alpha}$ is really the blinded value of $(D_2 \cdot g_3 = I \cdot y_T^k \cdot g_3)$. Second, the exponent k in h_T^k is equal to the exponent k in $D_2 = I \cdot y_T^k$, which allows the trustee trace the coin.

The client commits $(D_2 \cdot g_3)^{\alpha}$ and the bank uses the restrictive blind group signature protocol to generate the signature of $D_2 \cdot g_3$. By the characteristic of restrictiveness, the client is unable to blind the internal structure of the commitment. This assures that the structure of commitment $(D_2 \cdot g_3)^{\alpha}$ and the structure of $D_2 \cdot g_3$ is identical. That is,

$$I_1(\alpha \cdot u_1, \alpha \cdot k, \alpha \cdot 1) = I_2(u_1, k, 1) = u_1 : k : 1$$

where I_1 and I_2 are blinding invariant functions of the protocol with respect to (g_1, g_2, g_3).

To allow the trustee to recover the client's identity I from the coin, the client unblinds the commitment $(D_2 \cdot g_3)^{\alpha}$ into $D_2 \cdot g_3$ and embeds it into the coin. This unblinding is proved by V :

$$V = (c, s_1, s_2) = SKREP[(k, u_1) : D_1 = g_2^k \wedge D_2 = \frac{m_0}{g_3} = \frac{g_1^{u_1} y_T^k g_3}{g_3} = g_1^{u_1} y_T^k].$$

The signature of knowledge V shows that the client knows the representation of $\frac{D_2 \cdot g_3}{g_3}$ with respect to (g_1, y_T) and hence assures that the client correctly unblinds the commitment. Simultaneously, V proves that exponent k in D_1 is equal to exponent k in $D_2 \cdot g_3$. Note that D_2 can be extracted from $D_2 \cdot g_3$ and D_1 is made by the client himself. Since the Elgamal encryption of the client's identity I consists of D_1 and D_2, V eventually proves that I is revocable from the coin by the trustee.

D.4 The Payment Protocol

With the coin and random numbers k, b which was generated in the withdrawal protocol, the client initiates the payment protocol. The client gets $ID_{Merchant}$, and $Date$ and calculates (c_p, s_p) using the hash function H:

$$\begin{aligned} c_p &= H(ID_{Merchant} \| Date) \\ s_p &= b - kc_p \pmod{n}. \end{aligned}$$

The payment protocol is shown in Figure 4. The client sends $\{Coin, Date, (c_p, s_p)\}$ to the merchant. The merchant then checks if $Date$ is within allowed time interval and verifies (V_1, V_2) and V. The merchant also verifies that equation $g_2^{s_p} \cdot (g_2^k)^{c_p} = g_2^b$ is satisfied. If all verifications are successful, the merchant checks if the paid coin is double-spent by looking into its coins. When the merchant deposits coins, the bank also checks if the deposited coins are double-spent by looking into its coins.

Client		Merchant
$(Coin, k, b, ID_{Merchant}, Date)$		$(n, e, G, g, g_1, g_2, g_3, a, \lambda, \mu, t)$
$c_p = H(ID_{Merchant} \| Date)$		
$s_p = b - kc_p (mod\ n)$		
	$\xrightarrow{Coin, Date, (c_p, s_p)}$	
		verify $(V_1, V_2), V, Date$
		verify $g_2^{s_p} \cdot (g_2^k)^{c_p} \overset{?}{=} g_2^b$
		$\{Coin, (c_p, s_p)\}$

Fig. 4. The Payment protocol

D.5 Anonymity Revocation

In this implementation, the coin is as follows :

$$\begin{aligned} Coin &= \{D_1, S_B(D_2 \cdot g_3), V\} \\ where\ \ S_B(D_2 \cdot g_3) &= \{g_2^b, D_2 \cdot g_3, (D_2 \cdot g_3)^y, V_1, V_2\}. \end{aligned}$$

Double Spending Detection When the client double spends the coin to the same merchant, two *Date*s are different, and therefore two (c_p, s_p)s are different. Then the merchant calculates the client's identity, I, from (c_1, s_1) and (c_2, s_2) by following calculation :

$$\frac{s_1 - s_2}{c_2 - c_1} = k,$$
$$\frac{D_2}{y_T^k} = I.$$

In case of double spending of the same coin to different merchants, two $ID_{Merchant}$'s are different and hence two (c_p, s_p) are different. After two merchants deposit each coin, the bank knows that the double spending has occurred. Then the bank calculates the client's identity, I, by the calculation above.

Owner Tracing The trustee can recover the client identity from the coin by computing

$$D_2 \cdot D_1^{-x_T} = I.$$

Coin Tracing The trustee can trace the coin from the record of the withdrawal protocol which is kept in the bank by computing

$$M_5^{x_T} = (h_T^k)^{x_T} = (g_2^{x_T^{-1} k})^{x_T} = g_2^k = D_1.$$

Signer Tracing Given a signature $(\hat{g}, \hat{z}, V_1, V_2)$ for a message m, the group manager can determine the signer by testing if $\hat{g}^{y_P} = \hat{z}$ for every group member P (where $y_P = \log_g z_P$ and z_P is P's membership key). Thus the running time of this scheme is linear in the size of the group.

D.6 Efficiency and Security

In our generic e-cash system, owner tracing is bank account based as in [9]. Thus from the viewpoint of coin tracing and owner tracing our scheme is as efficient as the one in [9], which is one of the most efficient schemes to date. With regard to signer tracing our scheme is as efficient as the one in [10] in opening the signature to recover the signer's identity.

The size of the fair group signature depends on the size of the underlying blind group signature, which is the size of $S_B(D_2 \cdot g_3)$ plus the size of D_1 and V.

The security of our fair off-line e-cash system depends on the security of the underlying blind group signature schemes and the signatures of knowledge. The security of the scheme is based on the assumptions that the discrete logarithm, double discrete logarithm, and the roots of discrete logarithm problems are hard. In addition, it is based on the security of Schnorr and the RSA signature schemes and on the assumption that computing membership certificates is hard. The group signature scheme in [5] is not coalition resistant, but can be easily fixed. So in our implemented scheme we use fixed version of the group signature scheme in [5], which is mentioned in [1]. The fixed group signature scheme seems to be coalition resistant, but is not provably coalition resistant.

E Conclusion

In this paper we have proposed efficient and secure fair off-line electronic cash a group of banks can distribute. Distributed electronic banking reflects the real world closer than the single bank model and was considered by Lysyanskaya and Ramzan in [10]. They suggested that banks form a group and clients form another group to provide both client and bank anonymity control through blind group signatures. In our approach, first a generic fair off-line e-cash system is developed to provide client anonymity control and then the blind group signature scheme is applied to a group of banks to provide bank anonymity control. Our scheme provides coin tracing which the scheme suggested in [10] does not provide. Furthermore, when double spending is occurred in the scheme in [10], merchants or banks have to ask the trustee to revoke the identity of the double spender from the double spent coin. In our scheme merchants or banks can revoke the identity of the double spender from the instances of payment with the same coin. Our scheme is more efficient since the clients need not to be involved in extra group signature protocols, and hence the size of cash in our scheme is half of that in the scheme suggested in [10].

References

1. G. Ateniese, M. Joye, and G. Tsudik. On the Difficulty of Coalition-Resistance in Group Signature Schemes. Security in Computer Networks 1999. 105, 110, 115

2. S. Brands. Untraceable off-line cash in wallets with observers. *Advances in Cryptology - CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 302-318. Springer-Verlag, 1993. 107
3. J. Camenish and M. Michels. A group signature scheme with improved efficiency. *Advances in Cryptology - ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 160-174. Springer-Verlag, 1998. 105, 105, 108, 108
4. J. Camenisch, U. Maurer, and M. Stadler. Digital payment systems with passive anonymity-revoking trustees. *Computer Security - ESORICS '96*, volume 1146 of *Lecture Notes in Computer Science*, pages 33-43, Springer-Verlag, 1996. 105, 105, 105, 105, 106, 110
5. J. Camenisch and M. Stadler. Efficient group signature scheme for large groups, *Advances in Cryptology - CRYPTO '97*, volume 1296 of *Lecture Notes in Computer Science*, pages 410-424. Springer-Verlag, 1997. 105, 105, 106, 108, 110, 115, 115
6. D. Chaum and T. Pedersen. Wallet databases with observers. *Advances in Cryptology-CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 89-105. Springer-Verlag, 1992. 107
7. L. Chen and T. P. Pedersen. New group signature schemes. *Advances in Cryptology-EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 171-181. Springer-Verlag, 1995.
8. Y. Frankel, Y. Tsiounis, and M. Yung. "Indirect discourse proofs": Achieving fair off-line e-cash. *Advances in Cryptology-ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 286-300. Springer-Verlag, 1996. 105, 105, 105, 105
9. Y. Frankel, Y. Tsiounis, and M. Yung. Fair off-line e-cash made easy. *Advances in Cryptology-ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 257-270. Springer-Verlag, 1998. 105, 115, 115
10. A. Lysyanskaya and Z. Ramzan. Group Blind Digital Signatures: A Scalable Solution to Electronic Cash, *Proceedings of the Second International Conference on Financial Cryptography*, volume 1465 of *Lecture Notes in Computer Science*, pages 184-197, Springer-Verlag, 1998. 105, 105, 107, 107, 111, 111, 111, 115, 115, 115, 115, 115
11. M. Stadler, J. M. Piveteau, and J. Camenisch. Fair blind signatures. *Advances in Cryptology-EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 209-219. Springer-Verlag, 1995.

Efficient Asynchronous Secure Multiparty Distributed Computation

K. Srinathan and C. Pandu Rangan

Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Chennai - 600036, India
srinath@meenakshi.iitm.ernet.in, rangan@iitm.ernet.in

Abstract. This paper significantly improves the message complexity of perfect asynchronous secure computations among n players tolerating a computationally unbounded active adversary that corrupts up to $t < \frac{n}{4}$ players. The protocol presented in this paper communicates $\mathcal{O}(mn^3 \lg |\mathbb{F}| + mn^3 \lg n)$ bits and broadcasts $\mathcal{O}(mn^2)$ bits, where m is the number of multiplication gates in the circuit. This is to be compared with the most efficient perfect secure asynchronous protocol known so far, namely the protocol of [5], which requires $\mathcal{O}(mn^4 \lg |\mathbb{F}| + mn^4 \lg n)$ bits of communication apart from $\mathcal{O}(mn^4 \lg n)$ bits of broadcast.

A Introduction

Consider the scenario where there are n players (or processors) $\mathcal{P} = \{P_1, P_2, \ldots P_n\}$, each P_i with a local input x_i. These players neither trust each other nor trust the channels by which they communicate. Nevertheless, they wish to correctly compute a function $\mathcal{F}(x_1, \ldots, x_n)$ of their local inputs, while keeping their local data as private as possible. This is the problem of *secure multiparty computation*. The problem of secure multiparty computation has been extensively studied in several models of computation. The *communication* facilities assumed in the underlying network differ with respect to whether secure communication channels are available [6,11] or not available [16,10], whether or not broadcast channels are available [18,2,13] and whether the communication channels are synchronous or asynchronous [5,7]. The *correctness* requirements of the protocol differ with respect to whether exponentially small probability of error is allowed (*unconditional*) or not allowed (*perfect*). The corrupted players are usually modeled via a central *adversary* that can corrupt up to t players. The adversary may be computationally bounded (*computational setting*) or unbounded (*secure channels setting*). One also generally distinguishes between actively corrupted players (*Byzantine*), passively corrupted players (*Eavesdropping*).

We consider the problem of perfect asynchronous secure multiparty computation over a fully connected network of n players (processors) in which up to t Byzantine faults may occur, where every two players are connected via a secure and reliable communication channel (secure channels setting). The network is *asynchronous*, meaning that any message sent on these channels can have arbitrary (finite) delay. We model the faulty players' behavior through a computationally unbounded adaptive adversary. This adversary may choose, at any stage of the protocol, additional players to corrupt, as long as he does not exceed the limit of t faulty players. To model the network's asynchrony, we further assume that the adversary can schedule the communication channel, i.e. he can determine the time delays of all the messages (however, he can neither read nor change those messages). We call such an adversary a *t-adversary.*

It was shown in [5] that perfect asynchronous secure multi party computation is possible in the above setting if and only if $t < \frac{n}{4}$. In [7], an $\left(\lceil \frac{n}{3} \rceil - 1\right)$-resilient protocol is described that securely computes any function $\mathcal{F}$ when exponentially small probability of error is allowed. Although theoretically impressive, these results lack in the area of practical feasibility. The complicated exchanges of messages and zero-knowledge proofs in protocols like [5,7] might render them impractical.

In the synchronous model of communication one has often attempted to reduce the communication complexity of multiparty protocols [1,4,3,14,15]. More recently, [17] significantly improved the message complexity of secure synchronous unconditional multi party computations through a generic framework that strictly distinguishes between fault-detection and fault-correction and uses a technique called *player elimination* to efficiently transform a distributed protocol with fault detection into a fully resilient protocol. The problem of reducing the communication complexity of secure multiparty computation over an asynchronous network was left open in [17].

We concentrate on reducing the communication complexity of perfect asynchronous secure protocols. The notion of communication complexity of secure computations in an asynchronous network has not received much attention. The only known solution for error-less asynchronous secure computation was proposed by [5]. Even for this protocol the communication complexity was not fully analyzed. We provide a simple analysis of the communication complexity of [5] and then show that our protocol has significantly smaller communication complexity.

B Towards Efficient Asynchronous Secure Protocols

Each player P_i holds a private input x_i, and the players wish to securely compute the exact value of a function $\mathcal{F}(x_1, \cdots, x_n)$. However, since the network is asynchronous, and t players may be faulty, the players can never ever wait for more than $n - t$ of the inputs to be entered to the computation. Furthermore, the missing inputs are not necessarily of the faulty players. Formalizing this concept is a very delicate issue and we refer the reader to [5,9] for complete formal definitions of asynchronous secure computations.

Most distributed resilient protocols in the literature are derived from a private protocol processing the circuit in steps where after each step some consistency checks are performed. If an inconsistency is detected, then a fault-recovery procedure is invoked. These protocols are in general much less communication-efficient than their private (but non-resilient) counterparts since the adversary can provoke fault-recovery after every step even by corrupting only a single player. In this section, we describe an efficient resilient protocol with a substantially reduced number of invocations of the fault-recovery procedure. Each time a fault is detected, at least one corrupted player is eliminated from the further computation [17]. Hence the number of fault-recovery invocations is bounded by the maximal number of corrupted players and is independent of the circuit size.

Furthermore, fault detection requires that all players agree on whether or not a fault occurred. If broadcast is not available as a primitive a Byzantine agreement (BA) protocol is required.[1] Since protocols simulating broadcast have substantial communication complexity, we need to amortize them over a sequence of several steps. For this, we divide the circuit into **segments** each consisting of a sequence of steps. In general, if a fault occurs in a step within a segment the players must know this immediately after that step to prevent a violation of privacy in subsequent steps. But it is sufficient that a player who detects a fault informs all other players (without using broadcast). This is only a **partial fault detection**. Each player informed about a fault finishes the computation of the current segment with dummy values. Only at the end of the segment, a strict fault detection is performed and if required fault localization, player elimination, and fault correction protocols are executed. When a segment fails and fault localization yields a (r,p)-localization[2] $\mathcal{D} \subset \mathcal{P}$ (our protocol yields a (1,3)-localization) a first idea could be to eliminate these players and re-share all the intermediate values of the current computation according to the new setting. However, the number of intermediate values is potentially a super-polynomial in n, hence such an approach is potentially inefficient. Rather we use a lazy re-sharing: whenever a (r,p)-localization takes place, the set $\mathcal{P}$ of players is restricted to $\mathcal{P} \backslash \mathcal{D}$, and t is reduced to $t-r$ but no re-sharing is performed. For each gate, immediately before their evaluation, only that gates' inputs are re-shared, if necessary. It is important that this lazy re-sharing preserves the ability for continuing the computation. That our implementation of lazy re-sharing has the above property will be clear in the sequel.

We analyze the communication complexity of our protocols. The computation complexity is not analyzed because only the communication complexity is the bottleneck of such protocols (such protocols evaluate circuits where the number of linear gates (requiring no communication) is not very large compared to the number of multiplication gates). The considered protocols make extensive use of

[1] Clearly, broadcast can be implemented by Byzantine Agreement (BA). Nevertheless, Bracha [8] describes a simple $\left(\lceil \frac{n}{3} \rceil - 1\right)$-resilient Broadcast protocol for a Byzantine setting. The termination property of Broadcast is much weaker than the termination property of BA: for broadcast, we do not require that the good parties complete the protocol in case that the sender is corrupted. We do not distinguish BA from Broadcast since in the complexity analysis we use the sub-protocol for simulating broadcast channels only as a black-box.

[2] An (r, p)- localization is a set $\mathcal{D}$ with $|\mathcal{D}| = p$ players, guaranteed to contain at least r cheaters.

a BA primitive and hence their efficiency depends heavily on the communication complexity of the applied BA protocol. Therefore, we count the BA message complexity (BAC) and the message complexity (MC) separately.

C The Protocol Construction

Efficient protocols for perfect asynchronous secure computation can be constructed based on sub-protocols for *Input Sharing, Agreement on a Common Subset, Linear Combination, Multiplication, Degree Reduction, Segment Fault Localization and Output Reconstruction.*

In a nutshell, an agreed function is computed as follows: Let x_i be the input of P_i. Let $\mathbb{F}$ be a finite field known to all parties with $|\mathbb{F}| > n$, and let $\mathcal{F} : \mathbb{F}^n \rightarrow \mathbb{F}$ be the computed function. We assume that the players have an arithmetic circuit computing $\mathcal{F}$; the circuit consists of linear combination gates and multiplication gates of in-degree 2. All the computations in the sequel are done in $\mathbb{F}$.

The outline of the protocol follows: First each player secret-shares its input among the players using the sub-protocol for *Input Sharing*. This sub-protocol runs in two phases. In the first phase it uses a technique similar to Shamir's secret sharing scheme [19], extended to a two-dimensional sharing [6,5,17]. In the second phase, the players agree, using the protocol for *Agreement on a Common Subset*, on a core set C of players that have successfully shared their input. Once C is computed, the parties proceed to compute the function in the following way. First, the input values of the players not in C are set to a default value, say 0; then the players evaluate the given circuit gate by gate using the sub-protocols for *Linear Combination* and *Multiplication* and finally the output value is reconstructed towards the each of the players using the sub-protocol for *Output Reconstruction.* The sub-protocol for multiplication applies the player-elimination technique; either the outcome is a proper sharing of the correct product, or a fault is detected. In the latter case, the sub-protocol for *Segment Fault Localization* is applied to determine a localization $\mathcal{D}$. Then the players in $\mathcal{D}$ are eliminated from the further protocol, and the shared values are re-shared for this new setting involving fewer players using the sub-protocol for *Degree Reduction.*

C.1 Agreement on a Common Subset

In a perfect asynchronous resilient computation, very often the players in $\mathcal{P}$ need to decide on a subset of players, that satisfy some property, of size at least $(n - t) \geq 3t + 1$, where $n = |\mathcal{P}|$. It is known that all the honest players will eventually satisfy this property, but some faulty players may satisfy it as well. In our context, we need to agree on the set C of players who have completed correctly sharing their input. For implementing this primitive, we adopt directly the protocol presented in [7]. The idea is to execute a *BA* (Byzantine Agreement) protocol for each player, to determine whether it will be in the agreed set. Notice that if some P_j knows that P_i satisfies the required property, then eventually all the players will know the same. Thus, we can suppose the existence of a predicate Q that assigns a binary value to each player P_i, denoted $Q(i)$, based on whether P_i has satisfied the property as yet. The protocol is denoted by $AgreeSet[Q, \mathcal{P}, t]$. For a detailed description of the protocol, see [7].

Theorem 1 ([7]). *Using the protocol $AgreeSet[Q, \mathcal{P}, t]$ the players indeed agree on a common subset of players, denoted by C such that $|C| \geq (|\mathcal{P}| - t)$. Moreover, for every P_j in C, we have $Q(j) = 1$. The $AgreeSet[Q, \mathcal{P}, t]$ protocol has a Byzantine Agreement Complexiy of $\mathcal{O}(|\mathcal{P}|)$.*

C.2 Input Sharing and Output Reconstruction

The protocol for Input Sharing has two phases: first, each party shares its input using an Asynchronous Verifiable Secret Sharing (AVSS)[3] scheme; next, the parties use the AgreeSet protocol to agree on a set C of at least $(|\mathcal{P}| - t)$ parties who have shared their inputs properly. Our implementation of the protocols *InputShare* and *OutputReconstruction* are quite similar to the ones existing in the literature. In our protocol, each player P_i uses a variation of the V-Share protocol of [5], denoted *AV-Share*, to share its input in the first phase of the Input Sharing protocol. The second phase is implemented using the *AgreeSet* protocol. The protocols *AV-Share*, *InputShare* and *OutputReconstruction* are a formally specified in Fig. 1.

Implementation To implement the input sharing, each player P_i uses a variation of the V-Share protocol of [5], denoted *AV-Share*, to share its input in the first phase of the Input Sharing protocol. The *AV-Share* protocol differs from the V-Share protocol in the following ways:

- In *AV-Share* protocol, the dealer (the player with the secret that is to be shared among all players) chooses a polynomial $p(x, y) = \sum_{i,j=0}^{t} r_{ij} x^i y^j$ of degree t in two variables, with $r_{ij} = r_{ji}$[4], such that $p(0, 0) = s$ and sends the polynomial $f_i(x) = p(x, \alpha_i)$ to player P_i(for $i = 1, 2, \ldots |\mathcal{P}|$), unlike the V-Share protocol where r_{ij} need not be equal to r_{ji} and hence the dealer needs to send both the polynomials $f_i(x) = p(x, \alpha_i)$ and $g_i(x) = p(\alpha_i, x)$ to player P_i(for $i = 1, 2, \ldots |\mathcal{P}|$).
- In the V-Share protocol, the share of player P_i is $s_i = f_i(0)$, and the second dimension of sharing is not used. In the *AV-Share* protocol, each P_i (locally) outputs $f_i(0)$, which is P_i's share of s, and also the share-shares $s_{ji} = f_i(\alpha_j)$.

The second phase is implemented using the *AgreeSet* protocol. The *InputShare* protocol and the *OutputReconstruction*[5] protocol are given in Fig 1.

Theorem 2. *The pair* (AV-Share,OutputReconstruction) *is a t-resilient AVSS scheme in a network of $|\mathcal{P}|$ parties, provided that $|\mathcal{P}| \geq 4t + 1$.*

[3] For a formal specification of an AVSS scheme, see [5]

[4] This helps one to achieve an efficiency gain of a factor 2. One can prove that privacy is not violated by this technique. See [12] for details.

[5] Note that this protocol needs neither error correction nor broadcast.

```
                         Protocol InputShare[$\mathcal{P}, t$]
Code for party $P_i$ with secret $s_i$

1. Initiate $AV\text{-}Share_i[\mathcal{P}, t, P_i, s_i]$. For $1 \le j \le |\mathcal{P}|$, participate in $AV\text{-}Share_j$. Let
   $f_i^j(x), s_{ki}^j, 1 \le k \le |\mathcal{P}|$ be the output $AV\text{-}Share_j$.
2. Execute protocol $AgreeSet[Q, \mathcal{P}, t]$ with the boolean predicate : $Q(j) = 1$ if $P_j$
   has completed $AV\text{-}Share_j$ successfully. Let $CommonSet$ be the output of the
   protocol.
3. Output $(CommonSet, \{f_i^j(x), s_{ki}^j, 1 \le k \le |\mathcal{P}|, j \in CommonSet\})$

                 Protocol OutputReconstruction[$\mathcal{P}, t, P$]

1. Each party $P_i$ sends the polynomial $f_i(x)$ and the share-shares $s_{ji}$ to party $P$.
2. Party $P$ reconstructs the output as follows:
   For $0 \le k \le t$ do:
     Wait until $(|\mathcal{P}| - 2t + k)$ parties' messages are received.
     Check if at least $(|\mathcal{P}| - 2t)$ $f_i(x)$'s are consistent with all but $t$ share-shares.
     If yes, interpolate and output $s$ from $s_i = f_i(0)$ of consistent polynomials.
     If not, go to the next iteration.
```

Fig. 1. Input Sharing Protocol and Output Reconstruction Protocol

Proof. We assert the Termination, Correctness and Secrecy requirements of the above scheme. As per the definition, if the dealer is honest, an AVSS scheme has to terminate with certainty for all uncorrupted players. In case of a corrupted dealer no requirements are posed on the termination. The protocol *AV-Share* terminates with probability 1 for an honest dealer(this follows directly from the proof of the V-Share protocol given in [5]). The protocol *OutputReconstruct* terminates with certainty since P will eventually receive at least $(|\mathcal{P}| - t)$ messages. The correctness of the *AV-Share* protocol follows from the results of [5,12] where they prove that the shares and share-shares of the honest players indeed define an unique polynomial of degree t in two variables. The correctness of the protocol *OutputReconstruct* can be proven as follows: Assume that a player hands a bad polynomial $f_i^{'} \neq f_i$. Of the $(|\mathcal{P}| - t)$ messages received, this polynomial is inconsistent with the share-shares of at least $(|\mathcal{P}| - 2t)$ players. At least $(|\mathcal{P}| - 3t)$ players gave their correct share-shares to P. Player P will detect at least $(|\mathcal{P}| - 3t) > t$ inconsistencies and ignore this polynomial. On the other, if f_i is the correct polynomial, at most t share-shares(of the corrupted players) will be inconsistent. Hence P interpolates only correct shares and computes the correct secret s. The secrecy of protocol *AV-Share* follows from the fact that the degree of the chosen polynomial is always greater than the number of corrupted players. The privacy of the *OutputReconstruct* protocol is obvious as no player but P receives any information. □

C.3 Linear Combination and Multiplication

Let $\mathcal{P}$ be the set of players, at most t of which are corrupted, and let $\mathcal{L}$ be a linear function. Futhermore, assume that the values a,b,... are d-shared[6] with polynomials $p(x,y), q(x,y), \ldots$ respectively, where $t \leq d < (n-3t)$ and $n = |\mathcal{P}|$. Note that each player P_i has $f_i(x) = p(x,\alpha_i), a_{ji} = p(\alpha_i,\alpha_j), g_i(x) = q(x,\alpha_i), b_{ji} = q(\alpha_i,\alpha_j), \ldots$ (the α_i's are public random field elements known to all players). Due to linearity of $\mathcal{L}$, in order to compute $\mathcal{L}(a,b,\ldots)$, each player P_i locally computes $h_i = \mathcal{L}(f_i, g_i, \ldots)$ $c_{ji} = \mathcal{L}(a_{ji}, b_{ji}, \ldots)$, for $j = 1, \ldots, |\mathcal{P}|$.

A t-resilient protocol for multiplication starts with t-sharings of a, b and ends with a t-sharing of $c = a.b$ (if there are no inconsistencies) or with a partial fault detection (if there exist inconsistencies). The t-adversary view of the computation is distributed independent of the initial t-sharings for any t-adversary. Our implementation of the multiplication protocol is given in Fig. 2.

C.4 Degree Reduction

Let $\mathcal{P}$ be the set of players of which at most t are corrupted, and assume that a value s is d-shared among the players, where $t \leq d < (n-2t)$ and $n = |\mathcal{P}|$. Assume that s is d-shared with the polynomial $p(x,y)$ with player P_i holding $f_i(x) = p(x,\alpha_i), s_{ji} = p(\alpha_i,\alpha_j), 1 \leq j \leq n$. The goal of Degree Reduction is to transform this sharing into a proper and independent t-sharing of s. A t-resilient protocol for Degree Reduction for $|\mathcal{P}|$ players starts with a d-sharing of s where $t \leq d < (|\mathcal{P}| - 2t)$ and ends with either a t-sharing of s (if there are no inconsistencies) or with a partial fault detection (if there exist inconsistencies). Also, the t-adversary view of the protocol is distributed independent of the initial d-sharings for any t-adversary. We implement the Degree Reduction protocol as follows: first, every player P_i t-shares $s_i = f_i(0)$ using the protocol *EfficientAVShare*(see Fig. 2) and proves that the value shared is in fact s_i using the protocol *ACheckShare*(see Fig. 2). Then, every player locally computes the linear combination according to Lagrangian interpolation which results in a t-sharing of s [15]. The formal description of the implementation of the protocol Degree Reduction is given in Fig. 2.

Theorem 3. *The protocol DegreeReduction is a t-resilient protocol for the degree reduction functionality.*

Proof. The protocol *EfficientAVShare* terminates because there are at most t corrupted players and hence at least $(|\mathcal{P} - t|)$ players will distribute consistent shares and so the set G is well defined. On similar lines, we can see that the protocol *ACheckShare* terminates as well. The correctness of the *DegreeReduction* protocol follows from the following. If the protocol *EfficientAVShare* succeeds, then it implies that there exists a set of players G of size at least $(|\mathcal{P}| - t)$ such

[6] A value s is d-shared among the players if there exists a polynomial $p(x,y)$ in two variables of degree d such that $p(0,0) = s$. Each party P_i has $s_i(x) = p(x,\alpha_i), s_{ji} = p(\alpha_i,\alpha_j), 1 \leq j \leq n$.

Protocol EfficientAVShare$[\mathcal{P}, t, P, s]$

The dealer chooses a random polynomial $h(x, y) = \sum_{i,j=0}^{t} r_{ij} x^i y^j$ of degree t in two variables, with $r_{ij} = r_{ji}$ such that $h(0, 0) = s$ and sends the polynomial $f_i(x) = h(x, \alpha_i)$ to player P_i(for $i = 1, 2, \ldots |\mathcal{P}|$).

Code for party P_i :

1. Upon receiving $f_i(x)$, for each $1 \le j \le |\mathcal{P}|$, send $f_i(\alpha_j)$ to party P_j.
2. Upon receiving $(|\mathcal{P}| - t)$ messages m_{ji} check if $m_{ji} = f_i(\alpha_j)$. If equality holds for all m_{ji}, send a $CheckMessage_i$:= "OK'', else, send $CheckMessage_i := j$, where j denotes the smallest index such that the value received from P_j was not equal to $f_i(\alpha_j)$, to all parties.
3. Execute protocol $AgreeSet[Q, \mathcal{P}, t]$ with the boolean predicate: $Q(j) = 1$ if $CheckMessage_j$ has been received. Let G be the output of this protocol. If $CheckMessage_j$ = "OK'' $\forall$ j such that $P_j \in G$, then the protocol succeeds with output $(f_i(x), f_i(\alpha_j), 1 \le j \le |\mathcal{P}|)$. Else, set $FaultDetected_i := TRUE$.

Protocol ACheckShare$[\mathcal{P}, d, t, P, s]$

1. The dealer sets $g(x) := \frac{(h(x)-f(x))}{x}$ and distributes shares on $g(x)$ which is of degree $d - 1$ to every player P_i using the protocol *EfficientAVShare*. If this fails, then the whole verification protocol fails.
2. Every player P_i checks that $f(\alpha_i) + \alpha_i.g(\alpha_i) = h(\alpha_i)$. If consistent, the player sends $CheckBit_i := 1$, else he sends $CheckBit_i := 0$, to all players.
3. Every player executes protocol $AgreeSet[Q, \mathcal{P}, t]$ with $Q(j) = 1$ if $CheckBit_j$ has been received. Let G be the output of this protocol. If $CheckBit_j = 1$ for all $j \mid P_j \in G$, then the verification was successful. Else, set $FaultDetected_i := TRUE$.

Protocol DegreeReduction$[\mathcal{P}, d, t, s]$

Code for party P_i:

1. Initiate $EfficientAVShare_i[\mathcal{P}, t, P_i, s_i]$. For $1 \le j \le |\mathcal{P}|$, participate in $EfficientAVShare_j$. If $FaultDetected_i = FALSE$ then let $(f_i^j(x), s_{ki}^j, 1 \le k \le |\mathcal{P}|)$ be the output of $EfficientAVShare_j$. Else terminate with output 'NULL'.
2. Run $ACheckShare_i[\mathcal{P}, d, t, s_i]$. For $1 \le j \le |\mathcal{P}|$, participate in $ACheckShare_j$. If $FaultDetected_i = TRUE$ then terminate with output 'NULL'.
3. Execute protocol $AgreeSet[Q, \mathcal{P}, t]$ with the boolean predicate: $Q(j) = 1$ if P_j has completed $EfficientAVShare_j$ successfully to get the output say $CommonSet$.
4. Locally compute the linear combination of the shares and share-shares received from players in $CommonSet$, as in [15].

MUL$[\mathcal{P}, t, a, b]$

Each player P_i does the following for $j = 1, \ldots, n$:

– Computes $c_i(x) = f_i(x).g_i(x)$.
– Evaluates $c_{ji} = a_{ji}.b_{ji}$.
– Calls $DegreeReduction[\mathcal{P}, 2t, t, c]$ which starts with a $2t$-sharing of c.
– If the above *DegreeReduction* call outputs 'NULL' then terminates with output 'FAIL', else ends with the t-sharing of c.

Fig. 2. Input Sharing with Fault Detection, Degree Reduction and Multiplication

that each player $P_i \in G$ is consistent with the share-shares of at least $(|\mathcal{P}| - t)$ players of which at least $(|\mathcal{P}| - 2t)$ are honest. Since $(|\mathcal{P}| - 2t) > 2t + 1$, the players in G have verifiable shares and the $(|\mathcal{P}| - 2t)$ honest players in G define a t-sharing of s. Also, if the sharing phase of the protocol *ACheckShare* succeeds, then indeed the shares of all players lie on a polynomial of degree $d - 1$, and the degree of the polynomial $f(x) + x.g(x)$ is at most d. Hence, if there exists a set of players G of size at least $(|\mathcal{P}| - t)$ such that each honest player $P_i \in G$ is consistent with $f(\alpha_i) + \alpha_i.g(\alpha_i) = h(\alpha_i)$, then the polynomial $f(x) + x.g(x)$ and $h(x)$ have at least $(|\mathcal{P}| - 2t) > d$ points in common, which implies that $f(0) = h(0)$ (i.e. same secrets). The secrecy of this protocol is due to the independence of the the sharings. □

C.5 Segment Fault Localization

The purpose of fault localization is to find out which players are corrupted or, because agreement about this can usually not be reached, at least to narrow down the set of players containing the cheaters. The output of an (r, p)-localization is a set $\mathcal{D}$ with $|\mathcal{D}| = p$ players, guaranteed to contain at least r corrupted players.

The n shares of a d-shared value define a unique codeword of a code with Hamming distance $(n - d)$. Hence over an asynchronous network, less than $(n - t - d)$ faults can be detected, or less than $(\frac{n-t-d}{2})$ faults can be corrected. Therefore, computing linear functions requires $t \leq d < n - 3t$. For our lazy re-sharing procedure, to preserve the ability for continuing the computation, it is required that after a (sequence of) (r, p)-localizations and player eliminations (without degree reduction), still $t < \frac{n}{4}$ holds, and the degree d of each sharing still satisfies $t \leq d < n - 3t$. It immediately follows that $3r \geq p$. In the sequel, we use both $(1, 2)$-localizations as well as $(1, 3)$-localizations.

In our protocol, if a segment has detected a fault, the first faulty sub-protocol is found and we invoke the corresponding fault localization procedure. In what follows, we outline the fault localization methodology for the various sub-protocols that are allowed to fail with a partial fault detection, viz. *EfficientAVShare, ACheckShare* and *DegreeReduction.*

1. **Sharing Protocol (*EfficientAVShare*):** From the corresponding common set G, among all the players who complained about an inconsistency, each of the uncorrupted parties can agree on the player P_i with the smallest index i. From P_i's $CheckMessage_i$ all the parties know some P_j that P_i complained about. The parties execute a BA protocol (with their $CheckMessage_i$, that P_i sent to them, as input) to agree on a single P_j. Then every player sets $\mathcal{D} := \{P, P_i, P_j\}$, where P is the corresponding dealer. It is obvious that all players find the same set $\mathcal{D}$, and at least one player in $\mathcal{D}$ must be corrupted, hence $\mathcal{D}$ is a $(1, 3)$-localization.
2. **Verifying Shares Protocol (*ACheckShare*):** Let P_i be the player with the smallest index in the common set G who complained. Then the set $\mathcal{D} := \{P, P_i\}$ is the $(1, 2)$-localization.

3. **Degree Reduction Protocol (*DegreeReduction*):** Failure of *DegreeReduction* protocol is due to the failure of either or both of the above sub-protocols. Then the same $\mathcal{D}$ is determined as in the first failed sub-protocol.

D The Top-Level Protocol

Let $\mathcal{F} : \mathbb{F}^n \rightarrow \mathbb{F}$ be given by an arithmetic circuit **A**. Our protocol for t-securely computing $\mathcal{F}(x_1, x_2, \ldots, x_n)$ is described in Fig. 3.

Theorem 4. *The protocol* AsyncSecureCompute *allows a set of n players, with at most $t < \frac{n}{4}$ of them being corrupted, to securely compute a function over a finite field $\mathbb{F}$, with a message complexity (excluding broadcast) of $\mathcal{O}(mn^3 \lg |\mathbb{F}|) + mn^3 \lg n)$ and $\mathcal{O}(mn^2)$ Byzantine Agreement simulations.*

The complexity analysis of our protocol is given in Section E. This result should be compared with the most efficient perfect asynchronous secure computation protocol known before, namely the protocol of [5]. The protocol of [5] is briefly analyzed in Appendix A.

E Complexity Analysis of our Protocol

Complexity Analysis of Initial Sharing: In the *AV-Share$_i$*$[\mathcal{P}, t, P]$ protocol the distribution phase communicates $MC = nt \lg |\mathbb{F}| + n^2 \lg |\mathbb{F}|$ bits. In the verification phase each party needs to send a star in the graph which requires $\mathcal{O}(n \lg n)$ bits. So, this phase has $MC = \mathcal{O}(n^3 \lg n)$ and $BAC = \mathcal{O}(n \lg n)$. The *InputShare*$[\mathcal{P}, t]$ protocol runs *AV-Share$_i$* n times followed by an *AgreeSet* protocol. Hence, $MC = \mathcal{O}(n^3 \lg |\mathbb{F}| + n^4 \lg n)$ and $BAC = \mathcal{O}(n^2 \lg n)$.
Complexity Analysis of Receiving Output: The *OutputReconstruction*$[\mathcal{P}, t, P]$ protocol requires each party to send a polynomial of degree $t = \mathcal{O}(n)$ and n other field elements. This requires $MC = \mathcal{O}(n^2 \lg n)$.
Complexity Analysis of our Efficient VSS: In the *EfficientAVShare$_i$* protocol the distribution phase communicates $MC = nt \lg |\mathbb{F}| + n^2 \lg |\mathbb{F}|$ bits. The pairwise consistency checks need another $MC = \mathcal{O}(n^2 \lg n)$ bits to be sent. The agreement on G has $BAC = \mathcal{O}(n)$. The *ACheckShare* protocol runs an *EfficientAVShare$_i$* followed by n^2 bits of communication and an *AgreeSet* protocol. Hence, $MC = \mathcal{O}(n^2 \lg |\mathbb{F}| + n^2 \lg n + n^2)$ and $BAC = \mathcal{O}(n)$.
Complexity Analysis of Degree Reduction: The *DegreeReduction* protocol amounts to running each of the above two protocols, namely *EfficientAVShare* and *ACheckShare* n times followed by an *AgreeSet* protocol. This requires $MC = \mathcal{O}(n^3 \lg |\mathbb{F}|$ $+ n^3 \lg n)$ bits and $BAC = \mathcal{O}(n^2)$ bits.
VSS with Fault Localization uses $BAC = \mathcal{O}(\lg n)$.
Analysis of a Segment with l Multiplications: Every multiplication gate takes up to three degree reductions, re-sharing of arguments and the actual multiplication, where the re-sharings can be performed in parallel. In every degree reduction only the actual computation with partial fault detection is performed.

```
                    Protocol AsyncSecureCompute$[n, \mathbf{A}, x_1, \ldots, x_n]$
Code for party $P_i$:

 1. Initialization: Set $\mathcal{P} := \{P_1, P_2, \ldots, P_n\}$ and $t \leq \lceil \frac{n}{4} \rceil - 1$.
 2. Input Sharing: Set $(G_{share}, \{f_i^j(x), s_{ki}^j, \}) := InputShare[\mathcal{P}, t]$
    $1 \leq k \leq |\mathcal{P}|, j \in G_{share}$
    The secret $s_i$ of player $P_i$ in the above sub-protocol is $x_i$. For a line $l$ in the
    circuit, let $l^{(i)}$ denote the share of party $P_i$ in the value of this line. If $l$ is the
    $j^{th}$ input line of the circuit, then: Set $l^{(i)} := f_i^j(x)$ if $j \in G_{share}$ and $l^{(i)} := 0$
    otherwise.
 3. Computation:

    For each segment of the circuit :
      Repeat
        For each $P_i$: Set $FaultDetected_i := FALSE$.
        For each gate $g$ in the segment:
          Each player $P_i$ acts as follows:
            Wait until the $i^{th}$ shares of all input lines of $g$ are computed.
            If $g$ is linear with output line $l$ and input lines $l_1, l_2, \ldots$:
              Set $l := LinearCombination[\mathcal{P}, t, g, l_1, l_2, \ldots]$.
              Players $P_i$ with $FaultDetected_i = TRUE$ use random shares.
            If $g$ is a multiplication gate with output line $l$ and input lines $l_1, l_2$:
              If $l_k$, $k = 1$ or $2$ is shared with degree $d$ greater than $t$:
                Call $DegreeReduction[\mathcal{P}, d, t, l_k]$
                Every player $P_i$ with $FaultDetected_i = TRUE$ uses
                random shares in the sub-protocol. If the sub-protocol
                outputs 'NULL', then $P_i$ sets $FaultDetected_i := TRUE$
              Set $l := Multiplication[\mathcal{P}, t, g, l_1, l_2]$
              Every player $P_i$ with $FaultDetected_i = TRUE$ uses
              random shares in the sub-protocol.If the sub-protocol
              outputs 'FAIL', then $P_i$ sets $FaultDetected_i := TRUE$
        For each $P_i$, broadcast $FaultDetected_i$.
        Set $C := AgreeSet[Q, \mathcal{P}, t]$,$Q(j) = 1$ for $P_j$ if it received the broadcast from $P_j$.
        If at least one player in $C$ has $FaultDetected_i = TRUE$:
          Each $P_i$ broadcasts the index of the first sub-protocol that failed.
          Agree on the smallest sub-protocol that failed.
          Invoke the fault localization procedure for that sub-protocol to get $\mathcal{D}$.
          Set $\mathcal{P} := \mathcal{P} \backslash \mathcal{D}$ and $t := t - 1$.
      Until $((FaultDetected_i = FALSE)$ for all $i \in C)$.
    For each player $P$ needing output:
      Call $OutputReconstruction[\mathcal{P}, t, P]$.
```

Fig. 3. Protocol for Asynchronous Secure Computation

At the end of the segment, during (strict) fault detection, n bits are broadcast as well as an *AgreeSet* is run. If there are faults, extended fault localization is performed. There are at most $\mathcal{O}(nl)$ partial fault detections, hence to localize the first one which reported some failures $\mathcal{O}(n\lg(nl))$ bits are broadcast. The total complexities are $MC = \mathcal{O}(ln^3 \lg|\mathbb{F}| + ln^3 \lg n)$ and $BAC = \mathcal{O}(ln^2 + n\lg(nl))$.
Cumulative Complexity Analysis: The protocol *AsyncSecureCompute* $[n, \mathbf{A}, x_1, \ldots, x_n]$ uses *InputShare*$[\mathcal{P}, t]$ sub-protocol followed by the computation of $\frac{m}{l}$ segments, each of which has l multiplications. In all, at most t segments may fail and require repetition. Finally the protocol *OutputReconstruction*$[\mathcal{P}, t, P]$ is performed $\mathcal{O}(n)$ times, possibly in parallel. The overall complexity is as follows. $MC = \{\mathcal{O}(n^3 \lg|\mathbb{F}| + n^4 \lg n\} + \left\{\left(\frac{m}{l} + t\right) \mathcal{O}(ln^3 \lg|\mathbb{F}| + ln^3 \lg n)\right\} + \{\mathcal{O}(n^3 \lg n)\}$ and $BAC = \{n^2 \lg n\} + \left\{\left(\frac{m}{l} + t\right)(\mathcal{O}(ln^2)\right\}$. When setting $l = \frac{m}{n}$ and $t = \mathcal{O}(n)$, we have $MC = \mathcal{O}(mn^3 \lg|\mathbb{F}|) + mn^3 \lg n)$ and $BAC = \mathcal{O}(mn^2)$.

F Conclusions

In this paper we study the communication complexity, which is the bottleneck in most distributed applications, of perfect asynchronous secure computation protocols. We rigorously distinguish between the actual protocol with fault detection and additional procedures for fault correction. This, when combined with the technique of player elimination for reducing the costs of repetitive malicious faults, allows a substantial reduction in the overall communication complexity of the protocol.

References

1. Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In *Proceedings of 8th ACM PODC*, pages 201–210, August 1989. 118
2. Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, pages 75–122, 1991. 117
3. Donald Beaver, Joan Feigenbaum, Joe Kilian, and Phillip Rogaway. Security with low communication overhead (extended abstract). In *CRYPTO '90*, pages 62–76, 1990. 118
4. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols(extended abstract). In *Proceedings of 22nd ACM STOC*, pages 503–513, 1990. 118
5. M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computations. In *Proceedings of 25th ACM STOC*, pages 52–61, 1993. 117, 117, 118, 118, 118, 118, 118, 120, 121, 121, 121, 122, 122, 126, 126, 129
6. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of 20th ACM STOC*, pages 1–10, 1988. 117, 120
7. M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computation with optimal resilience. In *Proceedings of 13th ACM PODC*, pages 183–192, 1994. 117, 118, 118, 120, 121, 121

8. G. Bracha. An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In *Proceedings of 3rd ACM PODC*, pages 154–162, 1984. 119
9. R. Canetti. *Studies in Secure Multiparty Computation and Applications.* PhD thesis, The Weizmann Institute of Science, June 1995. 118
10. R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure computation. In *Proceedings of 28th ACM STOC*, 1996. 117
11. D. Chaum, C. Crepeau, and I. Damgard. Multiparty unconditionally secure protocols. In *Proceedings of 20th ACM STOC*, pages 11–19, 1988. 117
12. R. Cramer, I. Damgard, and U. Maurer. Efficient general secure multiparty computation from any linear secret sharing scheme. In *EUROCRYPT2000.* Springer-Verlag, 2000. 121, 122
13. Ronald Cramer, Ivan Damgard, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *EUROCRYPT '99.* Springer-Verlag, 1999. 117
14. Matthew K. Franklin and Moti Yung. Communication complexity of secure computation. In *Proceedings of 24th ACM STOC*, pages 699–710, 1992. 118
15. Rosario Gennaro, Micheal O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of 17th ACM PODC*, 1998. 118, 123, 124
16. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *19th ACM STOC*, pages 218–229. ACM Press, 1987. 117
17. Martin Hirt, Ueli Maurer, and Bartosz Przydatek. Efficient multi-party computation. In *ASIACRYPT 2000.* Springer-Verlag, December 2000. 118, 118, 119, 120
18. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of 21st ACM STOC*, pages 73–85, 1989. 117
19. A. Shamir. How to share a secret. *CACM*, 22:612–613, 1979. 120

Appendix A : Brief Complexity Analysis of [5]

The AVSS scheme is implemented using the protocol *V-Share* which communicates $MC = \mathcal{O}(n^2 \lg |\mathbb{F}| + n^3 \lg n)$ bits and $BAC = \mathcal{O}(n \lg n)$ bits. The *V-Recon* protocol takes only $\mathcal{O}(\lg |\mathbb{F}|)$ bits for a single player to reconstruct the output. The input sharing phase uses n *V-Share*'s and an ACS sub-protocol which sends $\mathcal{O}(n^2 \lg^2 n)$ bits and broadcasts $\mathcal{O}(n)$ bits, leading to a total complexity of $MC = \mathcal{O}(n^3 \lg |\mathbb{F}| + n^4 \lg n)$ and $BAC = \mathcal{O}(n^2 \lg n)$. The computation phase uses the protocol $BMUL$ for evaluating a single multiplication gate, in which each party runs t *V-Share*'s followed by an ACS, another *V-Share*, an AIS which communicates $\mathcal{O}(n^3 \lg^2 n + n^2 \lg |\mathbb{F}|)$ bits and broadcasts $\mathcal{O}(n^2)$ bits, and n V-Recon's. This leads to the following overall complexities for m multiplication gates. $MC = \mathcal{O}(mn^4 \lg |\mathbb{F}| + mn^4 \lg n)$ and $BAC = \mathcal{O}(mn^4 \lg n)$.

Tolerating Generalized Mobile Adversaries in Secure Multiparty Computation

K. Srinathan and C. Pandu Rangan

Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Chennai - 600036, India
srinath@meenakshi.iitm.ernet.in, rangan@iitm.ernet.in

Abstract. We study a distributed adversarial model of computation in which the faults are non-stationary and can move through the network (like viruses) as well as non-threshold (there is no specific bound on the number of corrupted players at any given time). We show how to construct multiparty protocols that are perfectly secure against such generalized mobile adversaries. The key element in our solution is devising *non-threshold proactive verifiable secret sharing schemes* that generalize the secret sharing schemes known in the literature.

A Introduction

Consider a fully connected network of n players (processors) who do not trust each other. Nevertheless they want to compute some agreed function of their inputs in a secure way. This is the secure multiparty computation problem. The players' distrust in each other and in the underlying network is usually modeled via an *adversary* that has control over some of the players and communication channels. Adversaries are classified according to their *computational resources* (limited (cryptographic) or unlimited (information theoretic)), their control over communication (secure, insecure, or unauthenticated channels), their control over corrupted players (eavesdropping (passive), fail-stop, or Byzantine (active)), their mobility (static, adaptive, or mobile) and their corruption capacity (threshold or non-threshold). In the information theoretic model one can distinguish between protocols with small (*unconditional*) or zero failure probability (*perfect*).

A.1 Contributions of this Paper

We propose a framework for constructing *non-threshold proactive verifiable secret sharing* schemes that strictly generalize the secret sharing schemes known in the literature. We study the notion of *mobile adversary structures* in the field of multiparty computation, and we construct protocols that provide perfect security (with zero error probability) tolerating such generalized mobile adversaries. The primary emphasis of this paper is on the existence of protocols. The designed protocols provide perfect security, that is, we consider a passive or an active generalized mobile adversary with unbounded computing power, in the classical model with pairwise synchronous secure communication channels between players, but not assuming a broadcast channel (which can be simulated, see [4]).

The threshold models of adversaries were found insufficient for capturing general scenarios of mutual trust and distrust among the players. This was illustrated in [7,8]. But, note that even the protocols that are designed to withstand generalized (non-threshold) adversaries cannot tolerate the (more realistic) adversary that gradually breaks the security of the protocol by "releasing" an already corrupted player and corrupting another player in its place, that is, the adversary corrupts different set of players at different times during the computation. Since even a momentary access to a player (processor) can expose that player's share for the rest of the duration of the protocol, it is important to provide protocols that are secure against such strong determined adversaries.

In this work, we define the security of a multiparty computation protocol with respect to a *mobile adversary structure*, a monotone set of subsets of the players, where the adversary is allowed to corrupt the players of *one* set in this mobile adversary structure in any *single* **time period**[1] and allow the players of different sets in this mobile adversary structure to be corrupted in different time periods. A mobile adversary structure is monotone in the sense of being closed with respect to taking subsets.

A.2 The Approach

We follow the solutions proposed in the many of the previous efforts in the literature e.g. [2]. The function to be computed by the protocol is without loss of generality specified by a circuit over a finite field $(\mathcal{F}, +, \star)$. The protocol can easily be constructed based on subprotocols for *providing input*, *computing arbitrary linear functions of shared values*, *multiplying (using degree reduction) the shared values*, and *receiving output*. In our scheme, the input is shared using the non-threshold proactive verifiable secret sharing protocol described in Section B which also facilitates output reconstruction. This step is the key to our solution. Other subprotocols like, for computing linear functions, and, for multiplications, are modifications of the ones already known in the literature to our setting.

B Non-threshold Proactive (Verifiable) Secret Sharing

Non-Threshold Proactive (Verifiable) Secret Sharing strictly generalizes the secret sharing schemes known in the literature. In such a sharing scheme, the initial sharing is based on an access structure and the other core properties of proactive secret sharing, like *periodic share renewal* and *share recovery*, are also incorporated. Given a monotone access structure $\mathcal{S} \subseteq 2^{\mathcal{P}}$, the problem is to design a secret sharing protocol that is *secure* against a *generalized mobile adversary* that is characterized by the *mobile adversary structure* $\mathcal{M}$, where $\mathcal{M}$

[1] Time is divided into *time periods* which are determined by the common global clock.

is, by definition, the compliment of the closure (with respect to taking supersets) of $\mathcal{S}$. Our construction of protocols for non-threshold proactive verifiable secret sharing is based on the subprotocols for the following tasks: *Initial Secret Sharing, Periodic Share Renewal, Share Recovery, and Share Reconstruction.* In our solution, first, the dealer shares the secret s using the *initial secret sharing* scheme which deals with withstanding the static non-threshold adversaries. This sharing is then "proactivized" to tolerate generalized mobile adversaries, that is, after the initialization, at the beginning of every time period, all honest players trigger an update phase consisting of the *share recovery* subprotocol followed by the *share renewal* subprotocol and *erase* all the share information of the previous time period. This "ensures" that the scheme is secure against *mobile* adversaries. For simplicity, we assume that if a player is corrupted during an update phase, then the player is corrupted during both periods adjacent to that update phase.

B.1 Initial Secret Sharing

To the mobile adversary structure $\mathcal{M}$, we attach an attribute, called the *corruptibility index*, CI, that is the minimum number of time periods required for any adversary characterized by $\mathcal{M}$ to corrupt every player at least once.[2] We show how to construct the initial secret sharing subprotocol for any given $\mathcal{M}$ provided $CI \geq 4$ (we will see later that if $CI \not\geq 4$ then it is *impossible* to design secure protocols for distributed computations). We start by defining a structure called the *share distribution tree (SDT)* which is a tree in which each node has some information (its *share* etc.) about the secret s that is to be distributed among the players in $\mathcal{P}$. The root node has full information about s. Let L denote the set of leaf nodes of the SDT. Every player P_i in $\mathcal{P}$ is assigned a (mutually disjoint) set of leaf nodes in L. That is there exists a *Player Share Function* Ψ that maps every player to some subset of leaf nodes in a way that no leaf node is attributed to more than one player. P_i's **share** of the secret s is the set of shares defined by the leaf nodes in $\Psi(P_i)$. The **cardinality** of a player $P_i \in \mathcal{P}$ is defined as the number of elements in the set $\Psi(P_i)$.

A Suitable SDT Topology and Corresponding $\Psi(\cdot)$: A protocol will work correctly on any SDT that satisfies the following property irrespective the set in $\mathcal{M}$ that has been corrupted. An adversary can choose to corrupt any set in the structure $\mathcal{M}$: say he chooses the set $\{P_{i_1}, P_{i_2}, \ldots, P_{i_k}\}$. Let $L' = \{l_1, l_2, l_3, \ldots l_N\}$ be the corresponding corrupted leaf nodes, that is $L' = \bigcup_{j=1}^{k} \Psi(P_{i_j})$. Next, any non-leaf node w in the SDT, with children $u_1, u_2, \ldots, u_m$, is iteratively marked as corrupted if more than $\lfloor \frac{m-1}{3} \rfloor$ of the u_i's are corrupted. This process is repeated for all the nodes in the SDT in a bottom-up fashion (start from the lowest level and go up to the root). The required property is that notwithstanding which set in the structure $\mathcal{M}$ has been corrupted the root of the SDT should not be marked as corrupted. It can be immediately seen that seen that constructing a

[2] $CI = \infty$ if $\mathcal{M}$ does not cover $\mathcal{P}$.

suitable SDT has a large number of degrees of freedom. As a first step, we note that the player simulation tree of [8] (based on the ideas of [1,3,7]) satisfies the required property. We use their construction to get the topology of the SDT.

Implementation: Without much security concern, we assume that for the given mobile adversary structure $\mathcal{M}$, the (same) topology of a suitable SDT and its corresponding $\Psi(\cdot)$ function are known to all players. First, the dealer (locally) distributes the secret s across the SDT as follows: Assign s to the root (at level 0) of the SDT. Each (non-leaf) node at level l with share s_l recursively shares it with its $k = 4$ children[3] at level $l + 1$ using a variation of the Shamir's 1-threshold secret sharing scheme [10], extended to two-dimensions. That is, for each node u at level l the dealer chooses at random a polynomial, $p_u(x, y) = s_l + a_1(x + y) + b_1(x \cdot y)$ and assigns the polynomial $f_{u,m}(x) = p_u(x, m) = s_l + a_1 m + x(a_1 + b_1 m)$ to node r_m, which is u's m^{th} child at level $l + 1$, for $m = 1, \ldots, 4$. The share of the node r_m is $s_m = f_{u,m}(0) = s_l + am$. Its share-shares are $s_{jm} = f_{u,m}(j) = s_l + a_1 m + j(a_1 + b_1 m)$, for $j = 1, \ldots, 4$. Only the share, not the share-shares, is used for further recursive sharing.[4]

The dealer distributes the share-shares of all the internal nodes $u_1, u_2, \ldots, u_N$ of the SDT. For distributing the share-shares associated with the node u_d the dealer does the following. Let w be the parent of u_d and let v_1, v_2, v_3 and v_4 be w's children such that $u_d = v_r$ (i.e. u_d is the r^{th} child of w). The dealer distributes the four share-shares associated with the internal node u_d, $d = 1, 2, \ldots$, by recursively sharing (using Shamir's secret sharing scheme) each of these four share-shares $(x_{1r}, x_{2r}, x_{3r}, x_{4r})$ through the subtree rooted at u_d. That is, for each internal node u_d, the dealer chooses at random four polynomials (in one dimension), $q_{u_d,j}(y) = x_{jr} + a_{u_d} y$, $j = 1, \ldots, 4$ and assigns the share-share-shares $x_{jr} + am$ to node r_m, which is u_d's m^{th} child, for $m = 1, \ldots, 4$. If any of u_d's children is not a leaf node then that child's share-share-shares are further recursively shared (using random polynomials) in the same manner. Apart from this, the dealer associates with all the v_is the share-share-shares that resulted during the sharing the share-share x_{ir} through the subtree rooted at u_d. For all the other non-leaf v_i's, the dealer, instead of just sending the share-share-shares, shares the share-share-share (resulting from sharing x_{ir}) of every leaf node in the subtree rooted at u_d, along the subtree rooted at v_i (using random one degree polynomials). We call the values received by the leaves of the subtree rooted at v_i as *share-coefficients*. Furthermore, the dealer sends the random polynomials used to distribute, through the subtree rooted at v_i, the share-share-share of some leaf node l, to the player associated with l. After thus locally spreading the secret in the SDT, the dealer *sends*, through secure communication channels, to each player $P_i \in \mathcal{P}$ the shares, share-shares, share-share-shares, share-coefficients and polynomial coefficients associated with the leaf nodes that occur in $\Psi(P_i)$.

[3] k is always 4 in the case where SDT topology of [8] is used. But the implementation can be modified to make it compatible with any other (more efficient) topology.

[4] If the share-shares too are recursively shared in the manner similar to the shares, the communication complexity increases geometrically.

When the adversary is active (*Byzantine*), for each non-leaf node w in the SDT with children u_1, u_2, u_3, u_4, the following check is adopted. Let x_{ji} denote the share-shares assigned to the node u_i, $i, j = 1, \ldots, 4$. Each pair of nodes u_a, u_b (for $1 \leq a, b \leq 4$) check whether $x_{ab} \overset{?}{=} x_{ba}$.

Verifying $x_{ab} \overset{?}{=} x_{ba}$ for nodes u_a, u_b: Case 1 (Both u_a and u_b are leaf nodes): The player P_d (associated with u_b) sends x_{ab} to P_e (associated with u_a) who does the verification and broadcasts the result (consistent or inconsistent).
Case 2 (Node u_a is a leaf node but not u_b): The players $P_{b_1}, \ldots, P_{b_{c'}}$ associated with the leaves of the subtree T_b rooted at u_b send their share-share-shares of x_{ab} to player P_d (associated with u_a) who broadcasts the result.
Case 3 (Node u_b is a leaf node but not u_a): The player P_e associated with u_b shares x_{ab} through the subtree T_a rooted at u_a exactly as the dealer did (P_e has the information about the share-share-shares of the leaves of T_a). Each of the leaf node of T_a just simply locally verify that the share-share-share he has due to x_{ab} is the same as that sent by P_e. In case of any mismatch, the corresponding player broadcasts a *doubt* and that leaf is marked as *doubtful*. If any of the leaves are *doubtful*, then iteratively, each non-leaf node w of T_a that has two or more of its children as doubtful is also marked doubtful. If u_a is doubtful, then the result is considered *inconsistent* else it is consistent.
Case 4 (Neither u_a nor u_b is a leaf node): For each leaf node $l_{b_i}, i = 1, \ldots, e$ in T_b, let t_{b_i} denote its respective share-share-share. Each player $P_j \in \{P_{b_1}, \ldots, P_{b_{c'}}\}$ (players corresponding to the leaves of T_b), shares the value t_{b_i} among the players $P_{a_1}, \ldots, P_{a_c}$ (players associated with the leaves of T_a) along T_a using randomly chosen one degree polynomials, say, $c_{j'}$, $j' = 1, 2, \ldots, N_{int}^{u_a}$ (where N_{int}^{u} is the number of non-leaf nodes in the subtree rooted at u). We know that the share-share-share t_{b_i} has already been shared along T_a by the dealer using the polynomials whose coefficients are say, $e_{j'}$, $j' = 1, 2, \ldots, N_{int}^{u_a}$ and are known to P_j. Let sc_{a_k} be the share-coefficient of P_{a_k}, $k = 1, \ldots, c$ corresponding to the sharing of the share-share-share t_{b_i}. The player P_j also sends to each of the players P_{a_m}, $m = 1, \ldots, c$ some additional check values (explained below). Let $r_1, r_2, \ldots, r_t$ be the path from node u_a to the leaf node l_{a_m} associated with P_{a_m} and let e_x (given by the dealer) and c_x (using now) , $x = 1, \ldots, t$ be the corresponding sharing polynomials' coefficients. The player P_j sends to P_{a_m} the values $(e_q - c_q)$, for $q = 1, \ldots, t$. Let $t_{a_k}^{b_i}$ and $EC_{a_k}^{b_i}$ denote the respective values that the player corresponding to the node l_{a_k} received from the player P_j when sharing t_{b_i}, $i = 1, \ldots, e$, $k = 1, \ldots, d$. Next, the players $P_{a_1}, \ldots, P_{a_c}$ work with the received $t_{a_k}^{b_i}$'s and $EC_{a_k}^{b_i}$'s as follows. Each player P_{a_k} locally does the following verification. He checks whether $sc_{a_k} \overset{?}{=} t_{a_k}^{b_i} + \sum_{q=1}^{t} r_q \cdot (e_q - c_q)$ and broadcasts the result. Once these checks have been completed, it is easy to verify whether $x_{ab} \overset{?}{=} x_{ba}$. For doing this verification, each of the leaf nodes $l_{a_1}, l_{a_2}, \ldots, l_{a_d}$ just simply locally verifies that the values sc_{a_k} and $t_{a_k}^{b_i} + \sum_{q=1}^{t} r_q \cdot (e_q - c_q)$ are the same for sufficiently "large" number of i's. By sufficiently "large", we mean the following. From the bottom of T_b, the player P_{a_k} verifies the following. Assign the value 1 to the leaf node l_{b_i} if $sc_{a_k} = t_{a_k}^{b_i} + \sum_{q=1}^{t} r_q \cdot (e_q - c_q)$, else assign

0. Iteratively for all non-leaf nodes w in T_b assign the value 1 to w if three or more of its children are assigned the value 1 else assign 0 to w. If the node u_b is assigned 0, then the player P_{a_k} broadcasts a *doubt* and that node (l_{a_k}) is marked as *doubtful.* If any of the leaf nodes in T_a is *doubtful*, then iteratively, each non-leaf node w of T_a that has two or more of its children marked as doubtful is also marked doubtful. If u_a is doubtful, then the result of $x_{ab} \overset{?}{=} x_{ba}$ is considered *inconsistent* else it is consistent. This ends the verification procedure.

After the above procedure has been repeated for all the children of each non-leaf node w, the following is done. In case of any inconsistency, the player broadcasts the corresponding index and the dealer answers by broadcasting the corresponding correct values. If more than one node triggers inconsistency for some node u_i or discovers that the dealer's answers contradict its own values, an accusation is registered by the players which the dealer defends publicly by broadcasting the polynomial that it originally had assigned to u_i in the sharing phase leading to further accusations. If two or more accusations are registered (due to child nodes of the non-leaf node w) then the node w is marked *damaged.* If any of the nodes have been marked *damaged*, then iteratively, each non-leaf node w of the SDT that has two or more of its children marked as damaged is also marked damaged. If the root of the SDT is marked damaged or if the dealer did not answer all the inconsistencies and accusations, the dealer is corrupted and a default sharing of secret 0 is taken. The initial secret sharing subprotocol is thus implemented. The following theorem succinctly summarizes the outcome of this subsection.

Theorem 1. *The above construction constitutes a initial secret sharing subprotocol secure against $\mathcal{M}_{static}$ for any given mobile adversary structure $\mathcal{M}$ of corruptibility index CI. For an active adversary we require that $CI \geq 4$.*

Proof. See [11] □

Periodic Share Renewal After the initialization, each player $P_i \in \mathcal{P}$ holds his share(s), share-shares, share-share-shares, share-coefficients and polynomial coefficients of the secret s defined by the leaves that occur in $\Psi(P_i)$.

Passive Adversary: The shares computed in the period t are denoted by using the superscript (t). To renew the shares at period $t = 1, 2, \ldots$, we modify the ideas in [9,6] to the non-threshold setting. In our system, each player P_i secret shares the value 0 using the *initial secret sharing subprotocol.* That is, each player P_i selects at random polynomials (for each node u), $h_{u,i}(x, y) = b_{u,i}(x \cdot y) + a_{u,i}(x + y) + secret$ (when u is the root, $secret = 0$) as the secret sharing polynomials and for each node u selects at random additional four polynomials $g_{u,d}(y) = c_{u,i}(y) + secret$ $d = 1, \ldots, 4$ for the distribution of the share-shares of node u. The player P_i then locally constructs an SDT_i and then sends to each other player $P_j \in \mathcal{P}$ the relevant shares, share-shares, share-share-shares, share-coefficients and polynomial coefficients associated. Let $h_{qi,m}$ (and $h_{qij,m}$ and $h_{qij,(u)}$ and $v_{qi,(u)}$ and $pc_{qi,g,(u)}$) denote P_i's m^{th} share (respectively four share-shares, $j = 1, \ldots, 4$, and the share-share-share(s) of the j^{th} share-share of the

internal node u and the relevant share-coefficients and polynomial coefficients) sent by player P_q corresponding to the leaf node at level l_m, for $1 \leq m \leq n_i$. Each player P_i (locally) computes its new share ($s_{i,m}^{(t)}$), share-shares ($s_{ij,m}^{(t)}$), share-share-shares ($s_{ij,(u)}^{(t)}$), share-coefficients ($v_{i,(u)}^{(t)}$) and polynomial coefficients ($pc_{i,g,(u)}^{(t)}$) as follows and *erases* all the variables not pertaining to the new time period: $\left(s_{i,m}^{(t)} \longleftarrow s_{i,m}^{(t-1)} + \sum_{q=1}^{n} h_{qi,m}\right)$, $\left(s_{ij,m}^{(t)} \longleftarrow s_{ij,m}^{(t-1)} + \sum_{q=1}^{n} h_{qij,m}\right)$, $\left(s_{ij,(u)}^{(t)} \longleftarrow s_{ij,(u)}^{(t-1)} + \sum_{q=1}^{n} h_{qij,(u)}\right)$, $\left(v_{i,(u)}^{(t)} \longleftarrow v_{i,(u)}^{(t-1)} + \sum_{q=1}^{n} v_{qi,(u)}\right)$,
and $\left(pc_{i,g,(u)}^{(t)} \longleftarrow pc_{i,g,(u)}^{(t-1)} + \sum_{q=1}^{n} pc_{qi,g,(u)}\right)$
where $j = 1, \ldots, 4, g = 1, 2, \ldots, N_{int}^{u}$ and $m = 1, \ldots, n_i$.
Active Adversary: In the above passive share renewal protocol an *active* adversary controlling a player can cause the destruction of the secret by dealing inconsistent share updates or just by choosing a polynomial with constant term $\neq 0$. We describe our method that helps circumvent the above problem. Each player P_i selects at random polynomials (for each node u in the SDT), $h_{u,i}(x, y) = b_{u,i}(x \cdot y) + a_{u,i}(x + y) + secret$ as the secret sharing polynomial and for each node u selects at random additional four polynomials $g_{u,d}(y) = c_{u,i}(y) + secret$ $d = 1, \ldots, 4$ for the distribution of the four share-shares of node u. Let $z_{qi,m}$ (z_{qijm}, $z_{qij,(u)}$, $v_{qi,(u)}$ and $pc_{qi,g,(u)}$) denote P_i's m^{th} share (respectively four share-shares, $j = 1, \ldots, 4$, the share-share-share(s) of the j^{th} share-share of the internal node u, the relevant share-coefficients and the polynomial coefficient values of SDT_q) sent by player P_q using SDT_q, corresponding to the leaf node l_m, $1 \leq m \leq n_i$. Let t_{r_m} be the first step in the path from the root to the leaf node l_m, that is the node l_m is in the subtree rooted at the root's $t_{r_m}^{th}$ child. First, P_i locally computes $H_{qi,m} = t_{r_m} \cdot z_{qi,m}$, $H_{qij,m} = t_{r_m} \cdot z_{qij,m}$, $H_{qij,(u)} = t_{r_m} \cdot z_{qij,(u)}$, $V_{qi,(u)} = t_{r_m} \cdot v_{qi,(u)}$ and $PC_{qi,g,(u)} = t_{r_m} \cdot pc_{qi,g,(u)}$. Next, the player P_i applies the degree reduction protocol (as will be explained in the sequel) to the shares $H_{qi,m}$, $H_{qij,m}$, $H_{qij,(u)}$, $V_{qi,(u)}$ and $PC_{qi,g,(u)}$ to get $h'_{qi,m}$, $h'_{qij,m}$, $h'_{qij,(u)}$, $v'_{qi,(u)}$ and $pc'_{qi,g,(u)}$ respectively. Then the new shares are calculated by adding the sum of these values to the original share (as in the passive case). The main result of this subsection is summarized in the following theorem.

Theorem 2. *The new shares, share-shares, share-share-shares, share-coefficients and polynomial coefficients computed by the above procedure at the end of the update phase correspond to the secret s, and the active (or passive) adversary $\mathcal{A}_{active}$ (or $\mathcal{A}_{passive}$) that at any single time period corrupts no more that* one *set in the mobile adversary structure $\mathcal{M}$ learns nothing about the secret.*

Proof. See [11] □

Share Recovery Scheme In a non-threshold proactive secret sharing system, the participating players must be able to make sure whether shares of other participating players have not been corrupted or lost, and restore the correct share if necessary. Otherwise, an adversary could cause the loss of the secret by gradually destroying the shares.

Detection and Localization of the lost/corrupt nodes: For each non-leaf node w in the SDT with children u_1, u_2, u_3 and u_4 with share-shares $x_{\cdot,\cdot}$, each pair of nodes u_a, u_b (for $1 \leq a, b \leq 4$) check whether $x_{ab} \overset{?}{=} x_{ba}$. For this, the players adopt the verification procedure outlined in page 134. Each non-leaf node w whose children-nodes triggered two or more inconsistencies are detected as *lost*, in the passive model, or *corrupted*[5], in the active model. Also, for every non-leaf node that is detected as lost/corrupted, all the nodes in the subtree rooted at that node is marked as lost/corrupted as well. We denote these lost/corrupted nodes by $\mathcal{B} = \{u_{b_1}, u_{b_2}, \ldots, u_{b_d}\}$, where d is the total number of corrupted nodes in the current time period. It is not possible to exactly as well as perfect-privately categorize and globally agree on the leaf nodes that are corrupted. Therefore, all the leaf nodes (i.e. their associated players) undergo the recovery procedure illustrated below.

Share Recovery: In this section we will show how to recover the corrupted shares, share-shares, share-share-shares, share-coefficients and the polynomial coefficients of the corrupted players.

Passive Adversary: *Recovering the shares:* To recover the share $s^{(t)}_{xi,m}$ of player P_{x_i} corresponding to the leaf node l_m, where l_m is the child of w in the SDT (let v_1, v_2 and v_3 be the other children of w and let l_m be the r^{th} child of w), the following method is used depending on whether any corrupted node exists in the path from l_m to the root.
Case 1: No node in the path from l_m to the root is in $\mathcal{B}$: Each player P_y, $y = 1, \ldots, n$ secret-shares some random secret of his choice using SDT_y (of the same topology as the SDT used initially) in such a way that the share of the node l_m is zero (this is achieved using the scheme outlined later in the sequel). After this step, each player P_y has n additional random shares, share-shares, share-share-shares, share-coefficients and polynomial coefficients (corresponding to the SDT_y's $y = 1, \ldots, n$) for each of the (original) values respectively. Next, each player P_y randomizes his (original) share, share-shares, share-share-shares, share-coefficients and polynomial coefficient values by respectively adding to them the sum of the n corresponding random ones. Subsequently, player P_{x_i} first reconstructs (as will be explained below) the randomized share of each (other) child of node w, namely v_i, $i = 1, 2, 3$ and then interpolates these randomized shares to recover the share $s^{(t)}_{xi,m}$ of node u_{b_j}. The former step is executed as follows. Each player P_x, that is associated with a leaf of the subtree rooted at v_i, sends his randomized shares, share-shares and share-share-shares corresponding the nodes v_1, v_2 and v_3 to the player P_{x_i} who then interpolates the share $s^{(t)}_{xi,m}$ from the random shares of all those v_i's that are consistent with all but at most one random share-share.
Case 2: Some node in the path from l_m to the root is in $\mathcal{B}$: Let w' be the first corrupted node in the path from the root to the leaf node l_m. It is clear that such a $w' \neq root$ (eventually) exists since the root is maintained uncorrupted. Let l'_i ,

[5] Since the sharing was originally 1-threshold, two inconsistencies imply corruption.

$i = 1, 2, \ldots$ be the leaf nodes of the subtree rooted at w' and let $l'_{i_m} = l_m$. Let the node u be w''s parent and let v'_i, $i = 1, \ldots, 4$ be u' children such that $v'_{i_m} = w'$. Then, by the definition of w', the node u is uncorrupted. And since u is not corrupted we can recover the information associated with v'_{i_m} as done in case 1.[6] The only difference from the methodology from case 1 for achieving the recovery of the share information associated with the node v'_{i_m} is that, each player P_y, $y = 1, \ldots, n$ secret-shares some random secret of his choice using SDT_y in such a way that the share of the node v'_{i_m} is zero (instead of the share of some leaf node being zero). After this step, the subsequent steps are all same as in case 1. After the recovery, the player P_{x_i} to re-shares the recovered information of v'_{i_m} through the subtree rooted at v'_{i_m} using the procedure outlined in the implementation of the initial sharing scheme (See Subsection B.1).

Thus, the shares of any corrupted player can be recovered. It is easily seen that to recover the share-shares, share-share-shares, share-coefficients of player P_{x_i}, a similar randomize-then-reconstruct procedure can be adopted. The only major difference will be in each player P_y, $y = 1, \ldots, n$ secret-sharing some random secret of his choice using SDT_y in such a way that the corresponding share-share (respectively share-share-shares, and share-coefficients) of the relevant node is zero. Once all the share-coefficients (of that node) are recovered, the players associated with the leaves of the subtree rooted at that relevant node re-share their new share-share-shares respectively through the corresponding subtrees thereby, in effect, "recovering" (actually, newly creating) the polynomial coefficients.

Secret sharing such that the some required information associated with some given node w is zero: Let $j_1, j_2, \ldots, j_r$ be the path from the root of the SDT to the node w, that is the node w is the root's j_1^{th} child's j_2^{th} child's $\ldots j_r^{th}$ child, and let w_l, $l = 1, \ldots, r$ denote these intermediate nodes respectively, (note that $w = w_r$). Let the sharing polynomial used at node w_l, $l = 1, \ldots, r$ be $p_l(x, y) = b_l(x \cdot y) + a_l(x + y) + secret$ and let the polynomials used at w_l for sharing the four share-shares (or share-share-shares) associated with w_l (respectively associated with the j^{th} share-share of u) be $q_{lj,(u)}(y) = c_{lj,(u)}y + secret$, $j = 1, \ldots, k$. (that is, $w_l = u$ when sharing the share-shares). The dealer, who wants the *share* of node w to be zero, chooses to share the secret s where $s = -\left(\sum_{l=1}^{r} a_l \cdot j_l\right)$. To get the m^{th} share-share of node w to be zero, $s = -\left(\left(\sum_{l=1}^{r} a_l \cdot j_l\right) + m(a_r + b_r)\right)$. For the share-share-share of the m^{th} share-share of node u present in node w to be zero, chooses the value of s to be $s = -\left[\left(\sum_{l=r_u+1}^{r}(c_{lm,(u)} \cdot j_l)\right) + \left(\sum_{l=1}^{r_u}(a_l \cdot j_l)\right) + (a_{r_u} \cdot m) + (b_{r_u} \cdot m)\right]$ where $j_1, j_2, \ldots, j_{r_u}$ is the path from root to the node u. For the share-coefficient of node w (in the subtree rooted at u_a), due to the share-share-share of node u (in the subtree rooted at u_b) for checking equality of the share-shares of u_a and u_b, to be zero, the dealer chooses r random one degree polynomials with coefficients $e_1, \ldots, e_r$ to share sss_u along the subtree rooted at u_a such that

[6] Since v'_{i_m} is already corrupted, without any loss in security, the recovery of the share information associated with the node v'_{i_m} can be performed by any player, in particular by the player P_{x_i}.

$e_r = -\left(\frac{(\sum_{l=1}^{r-1} e_l \cdot j_l)+sss_u}{j_r}\right)$, where the share-share-share of node u is sss_u and $j_1, j_2, \ldots, j_r$ is the path from u_a to w.

Active Adversary: We note that using the above passive share recovery protocol in the presence of an *active* adversary gives rise to two problems namely, an active adversary controlling a player can cause the destruction of the secret by dealing inconsistent random shares just by choosing the random secret sharing polynomial such that the share at the required node w is not equal to zero, and secondly, in the case 2 (where there exists some corrupted node v'_{i_m} in the path from l_m to the root) the recovery and re-sharing of the information in the node v'_{i_m} cannot be performed by any player as if it were corrupted, it could re-share incorrectly. The second problem can be solved by reconstructing towards all the leaves of the subtree rooted at v'_{i_m} and run an Byzantine Agreement among them to 'fix' a sharing. Our method to solve the first problem is similar to the method used in the active share renewal protocol. Each player P_q, $q = 1, \ldots, n$ shares a randomly chosen value giving rise to the SDT_q. The player P_q sends to each other player $P_j \in \mathcal{P}$ the shares, share-shares, share-share-shares, share-coefficients and the polynomial coefficients associated with the leaf nodes that occur in $\Psi(P_q)$ denoted $z_{qi,m}$, z_{qijm}, $z_{qij,(u)}$, $v_{qi,(u)}$ and $pc_{qi,g,(u)}$, $g = 1, \ldots, N^u_{int}$) respectively. To get a sharing with the *share* of node w (where w is the r^{th} child of u; w_1, w_2, w_3 and w_k be the children of u ($w = w_r$)) as zero, each player P_{t_i} associated with the leaf node l_{t_i} , $i = 1, \ldots, T$ that is in the subtree rooted at w_x, $x \in \{1, \ldots, 4\}$ locally computes $H_{qt_i,m} = (x - r) \cdot z_{qt_i,m}$, $H_{qt_ij,m} = (x - r) \cdot z_{qt_ij,m}$, $H_{qt_ij,(u)} = (x - r) \cdot z_{qt_ij,(u)}$, $V_{qt_i,(u)} = (x - r) \cdot v_{qt_i,(u)}$. After this step, the players P_{t_i}, $i = 1, \ldots, T$ apply the degree reduction protocol for the subtree rooted at u to and compute the random share, share-shares, share-share-shares, share-coefficients and polynomial coefficients in the manner similar to the share renewal protocol. To get a sharing with the d^{th} *share-share* of node w as zero, the procedure is almost the same as in the case 1, with the only variation being that each player P_{t_i} uses $(x - d)$ instead of $(x - r)$. To get a sharing with the share-share-share of the d^{th} share-share of node u that is with node w to be zero, only the share-share-shares and the share-coefficients need to be multiplied by $(x - r)$.

Theorem 3. *If the adversary $\mathcal{A}_{active}$ (or $\mathcal{A}_{passive}$) compromises no more than one set in the mobile adversary structure $\mathcal{M}$, with corruptibility index $CI \geq 4$, in any time period, then every recovering player that follows the above protocol recovers its correct share, share-shares, share-share-shares, share-coefficients, and polynomial coefficients while the adversary $\mathcal{A}_{active}$ (or $\mathcal{A}_{passive}$) learns nothing about the original secret shared among the players.*

Secret Reconstruction Let P be the designated player supposed to receive a value s that is non-threshold proactively shared among the players. Let N_l denote the last level of the SDT (root is at level 0).

1. Every player $P_i \in \mathcal{P}$ sends to P his share $s^{(t)}_{i,m}$, his share-shares $s^{(t)}_{ij,m}$, $j = 1, \ldots, 4$, $m = 1, 2, \ldots, n_i$, and his share-share-shares of the j^{th} share-share of internal nodes $s^{(t)}_{ij,(u)}$, $j = 1, \ldots, 4$.
2. The player P recursively reconstructs the shares for all the nodes at levels $N_l - 1$ down to 0: For each node w at level $N_l - 1$, let u_1, u_2, u_3 and u_4 be its children (clearly, all u_i's are leaf nodes). He reconstruct w's share by the Lagrangian interpolation of the shares of the u_i's that are consistent with all but at most one share-share. This is followed by the reconstruction of the share-shares of the each node w at level $N_l - 1$ from the share-share-shares $s^{(t)}_{ij,(w)}$, $j = 1, \ldots, 4$. This is done by recursively reconstructing the share-share of w from the share-share-shares of only those u_i's that were (previously) found consistent with all but at most one share-share. Player P can then interpolate a unique straight line on these values [7] and pick the constant term as the required share-share of w. Player P is now ready with the shares and share-shares of all nodes at level $N_l - 1$ and can in a similar fashion interpolate those of nodes at level $N_l - 2$ and so on till the required secret s is recovered.

B.2 Non-Threshold Proactive Verifiable Secret Sharing: The Combined Protocol

Adhering all the above pieces we get our full protocol for *non-threshold proactive verifiable secret sharing* tolerating generalized mobile adversaries.

1. The dealer shares the secret s using the *initial secret sharing* protocol.
2. At the beginning of every time period, the update phase is triggered, consisting of
   (a) the lost share detection and localization.
   (b) the lost/corrupted share recovery protocol.
   (c) the share renewal protocol.
3. Share reconstruction protocol (whenever required).

C Secure Multiparty Computation

The function to be computed by the protocol is without loss of generality specified by a circuit over a finite field $(\mathcal{F}, +, \star)$. We follow the solutions in the previous literature in the sense that the protocol consists of three stages: *providing input*, *computation stage* (computation of arbitrary linear functions of shared values, and multiplication using degree reduction of the shared values), *receiving output*. In our scheme, the input is provided using the non-threshold proactive

[7] The adversary may corrupt only the share-share-shares leaving the share-shares intact leading the player P use the corrupted share-share-share. This is not much of a problem because, even if all the four share-share-shares (with one corrupted) were to be used, P can easily find those three values that are collinear.

verifiable secret sharing protocol described in the previous section which also facilitates output reconstruction.

Linear Combination: Let $\mathcal{L}$ be a linear function and assume that the values $a, b, \ldots$ are non-threshold proactively shared. Due to the linearity of $\mathcal{L}$, in order to get the sharing of $c = \mathcal{L}(a, b, \ldots)$, it is sufficient if each player locally computes the linear combination of its shares (and share-shares, share-share-shares, share-coefficients and polynomial coefficients) of a and b.

Multiplication: Assume that two values a and b are non-threshold proactively shared. First, each player just locally multiplies the corresponding share, share-shares, and internal node share-shares of the two sharings. This defines a sharing of $c = a \cdot b$ but the used polynomials are of double the degree. Next, the players perform degree reduction. The key idea of *degree reduction* is to non-threshold proactively verifiably share the shares $c_{ij,m}^{(t)}$ (using the initial sharing protocol on the same SDT topology) and prove that the value shared in the previous step is indeed $c_{ij,m}^{(t)}$ and then locally compute the linear combination to get the degree reduced sharing. This protocol can be implemented using a straight-forward adaptation of the protocol of [2] improved by [5], to the non-threshold setting.

From the results of [8], it is clear that the lower bounds of 3 (in the passive model) and 4 (in the active model) are *necessary.* In [8] these bounds are proved as necessary for their adversary model ("stationary" generalized adversary) and since our adversary is a generalization ("non-stationary" generalized adversary) of theirs, these lower bounds trivially hold in our case too. In this work, we have shown that these bounds are also *sufficient* in the case of active adversaries by constructing secure multiparty protocols that are resilient against generalized mobile active adversaries characterized by a mobile adversary structure of corruptibility index $CI \geq 4$. Due to the "rebooting" sort of mechanisms involved in the removal of the adversary's effect on a corrupted player, there is loss of share information even in the case of a passive adversary leading to our solution requiring $CI \geq 4$ even for the passive case.

Theorem 4. *Let CI denote the corruptibility index of a mobile adversary structure $\mathcal{M}$. A player set $\mathcal{P}$ can compute any function perfectly $\mathcal{M}$-securely in the active model, where players corrupted during an update phase are considered as corrupted in both the adjacent time periods, if and only if, $CI \geq 4$.*

D Conclusion

We have studied the problem of secure multiparty computation in a generalized model where the adversary is, simultaneously, both mobile [9] and non-threshold [7,8]. We have proposed constructions that, for any admissible adversary, yield secure protocols offering perfect security. Moreover, as a by-product we have developed a non-threshold verifiable proactive secret sharing scheme that may have further applications on its own.

References

1. S. Akers and T. Robbins. Logical design with three-input majority gates. *Computer Design*, pp. 12–27, March 1963. 133
2. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th ACM STOC*, pp. 1–10, 1988. 131, 141
3. M. Fitzi. Erweiterte zugriffstrukturen in multiparty computation, 1996. Student's Project. 133
4. M. Fitzi and U. Maurer. Efficient Byzantine agreement secure against general adversaries. In *DISC '98*, vol. 1499 of *LNCS*, pp. 134–148. Springer-Verlag, 1998. 131
5. R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *17th ACM PODC*, 1998. 141
6. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing. In *CRYPTO 95*, vol. 963 of *LNCS*, pp. 339–352. Springer-Verlag, 1995. 135
7. M. Hirt and U. Maurer. Complete characterization of adversaries tolerable in secure multiparty computation. In *16th ACM PODC*, pp. 25–34, August 1997. 131, 133, 141
8. M. Hirt and U. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, April 2000. 131, 133, 133, 141, 141, 141
9. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *10th ACM PODC*, pp. 51–61, 1991. 135, 141
10. A. Shamir. How to share a secret. *CACM*, 22:612–613, 1979. 133
11. K. Srinathan and C. Pandu Rangan. Tolerating Generalized Mobile Adversaries in Secure Multiparty Computation. Technical Report, Dept. CSE, Indian Institute of Technology, Madras, August 2000. 135, 136

Codes Identifying Bad Signatures in Batches

Jarosław Pastuszak[1], Josef Pieprzyk[2], and Jennifer Seberry[2]

[1] Systems Research Institute
Polish Academy of Sciences
Warsaw, POLAND
jarek.pastuszak@bsb.com.pl
[2] Centre for Computer Security Research
School of IT and Computer Science
University of Wollongong
Wollongong, NSW 2522, AUSTRALIA
Josef_Pieprzyk@uow.edu.au
Jennifer_Seberry@uow.edu.au

Abstract. The work is concerned with identification of bad signatures in a sequence which is validated using batching. Identification codes (id-codes) are defined and their general properties are investigated. The generic construction for a wide range of id-codes is given and its instantiation using mutually orthogonal Latin squares is described. Hierarchical identification is studied for two cases when the identification procedure uses a family of id-codes and when there is a single underlying id-code. Remarks about future research conclude the work.

Keywords: Digital Signatures, Batch Verification, Identification Codes.

A Introduction

The reliance of e-Commerce on digital money has a dramatic impact on the computing load imposed on the bank. The bank has become the focal point where all electronic money (digital signatures) are flowing. Observe that before the transaction is approved, the electronic money must be validated. Batch verification is an attractive short cut for signature validation saving time and computing resources. It is applicable whenever the verifier gets a large number of digital signatures generated by the same signer provided the signature exhibits the homomorphic property allowing signatures to be validated in batches in the expense of a single exponentiation.

If the batch passes the validation, all signatures are considered correct and are accepted. If however, the batch fails to pass the validation test, the verifier must identify invalid signatures in the batch. Clearly, rejection of the whole batch is not an option. A natural question arises: how to identify invalid signatures in the batch so the valid signatures can be accepted ? Additionally, one would expect that the identification process should be as efficient as possible.

B Background

Batch verification makes sense if the signatures in a batch are related or generated by the same signer. There are two types of signatures which can be batched: RSA signatures and DSA (DSS) signatures. The RSA signatures use the fixed exponent (public key of the signer) for verification. Assume that we have n messages and their signatures. The signatures can be verified independently at the expense of n exponentiations. The batch containing all signature can be verified at the expense of a single exponentiation plus $(n-1)$ modular multiplications.

DSA signatures are based on exponentiation when the base is fixed and publicly known. Again n signatures can be verified one by one at the expense of n exponentiations. The batch of n signatures are validated using a single exponentiation and $(n-1)$ modular additions.

Bellare, Garay and Rabin [1] developed verification tests which are secure against any attacker. The security of the test is measured by the probability that a contaminated batch passes it making the verifier to accept all invalid or bad signatures contained in the batch. The probability of slipping bad signatures through the test can be traded with efficiency.

The problem we address in this work is an efficient identification of bad signatures after the test fails. There is a general method of bad signature identification which is called "cut and choose" in [3] or "divide and conquer" in [4]. It takes a contaminated batch and splits it repeatedly until all bad signatures are identified. The efficiency of this method depends on the degree of contamination (or how many bad signatures are in the batch) and also on how the bad signatures are distributed in the batch.

Note that identification of bad signatures resembles the problem of error correction. To be able to correct errors, the code must clearly identify all positions on which errors have occurred. As observed in [4], error correcting codes can be applicable for bad signature identification. There is a major difference between error correcting codes and *identification codes* or *id-codes* which allow to identify bad signatures. Computations in error correcting codes are done in the binary field with EXCLUSIVE-OR addition (XOR). The interaction among valid and invalid signatures within the batch are governed by INCLUSIVE-OR (logical OR).

The work is structured as follows. The model for id-codes is studied in Section C. Section D investigates general properties of id-codes. The general construction based on OR-checking matrix and its instantiation based on mutually orthogonal Latin squares are given in Section E. Hierarchical identification is described in Section F. A discussion about further work on id-codes closes the work.

C The Model

The problem we are dealing with is bad signature identification in a batch which has failed to pass the test. The test $\mathcal{T}$ is a probabilistic algorithm which takes a batch of an arbitrary length and produces a binary outcome accept/reject. Any clean batch always passes the test. A dirty batch (which contains one or more bad signatures) fails the test with an overwhelming probability.

Definition 1. *Given a batch $\mathcal{B}^u = \{(m_i, s_i) | i = 1, \ldots, u\}$ of signed documents (m_i is the i-th document and s_i its signature). The identification code $IC(u,t)$ able to identify up to t bad signatures is a collection of sub-batches $(\mathcal{B}_1, \ldots, \mathcal{B}_v)$ where $\mathcal{B}_i \subset \mathcal{B}^u$ such that for any possible pattern of up to t bad signatures, the outcomes (the syndrome) $S = (\mathcal{T}(\mathcal{B}_1), \ldots, \mathcal{T}(\mathcal{B}_v))$ uniquely identifies all bad signatures.*

The identification code $IC(u,t)$ can be equivalently represented by its $v \times u$ test-checking matrix $A = [a_{ij}]$ such that

$$a_{ij} = \begin{cases} 1 & \text{if } (m_i, s_i) \in \mathcal{B}_j \\ 0 & \text{otherwise} \end{cases}$$

Clearly, for a fixed size u of the batch, one would like to obtain a code $IC(u,t)$ with the parameter v as small as possible. Note that v indicates how many tests $\mathcal{T}$ must be run to identify all bad signatures and it can be considered as the parameter characterising the efficiency of the code. The parameter v is upper bounded by u as it is always possible to design a trivial code whose matrix A is the $u \times u$ identity matrix. This code is equivalent to serial validation of signatures one by one.

The following notation is introduced. The code $IC(u,t)$ is uniquely identified by its $(v \times u)$ test-checking matrix A. The entries of A are binary. Columns of the matrix A are indexed by u signatures in a batch. So the matrix A can be seen as a sequence of columns of the form $A = (A_1, \ldots, A_u)$. The index of the i-th signature in the batch $\mathcal{B}^u$ is the i-th column A_i. A row specifies the corresponding sub-batch which includes all signatures for which the entries are 1.

Note that if the i-th signature is bad the syndrome produced for a batch contaminated by it is equal to A_i or $S(i) = A_i$. Given a batch $\mathcal{B}^u$ with t bad signatures. Assume further that the bad signatures have occurred on positions $(b_1, \ldots, b_t)$ in the batch $\mathcal{B}^u$. Their corresponding indices are $(A_{b_1}, \ldots, A_{b_t})$. Denote the syndrome produced for the batch as $S(b_1, \ldots, b_t) = A_{b_1} \vee \ldots \vee A_{b_t}$, where $\vee$ is bit-by-bit inclusive (logical) OR. For example, if

$$A_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \text{ and } A_2 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \text{ then } A_1 \vee A_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

D Properties of Id-Codes

Using an information-theoretic arguments, we argue that there is a lower bound on the v parameter.

Theorem 1. *Given an id-code $IC(u,t)$ which always identifies correctly any t bad signatures in the batch of the size u. Then the number of tests (and the number of collections) v satisfies the following inequality*

$$v \geq \log_2 \sum_{i=0}^{t} \binom{u}{i} \tag{1}$$

Proof. Given a batch $\mathcal{B}^u$ of u elements contaminated by at most t bad signatures. The identification of bad signatures is possible if the syndromes are distinct for all patterns of i bad signatures ($i \leq t$) so knowing the syndrome, it is possible to determine the positions of bad signatures in the batch. Note that there are

$$\sum_{i=0}^{t} \binom{u}{i}$$

different identifiable patterns (including the pattern with no bad signature). Now if we have v sub-batches $(\mathcal{B}_1, \ldots, \mathcal{B}_v)$, then the test $\mathcal{T}$ applied for a single sub-batch $\mathcal{B}_j$; $0 \leq j \leq t$, provides a binary outcome (pass/fail) so the number of possible syndromes is 2^v. Clearly

$$2^v \geq \sum_{i=0}^{t} \binom{u}{i}$$

and the bound described by Equation (1) holds.

Obviously, searching for id-codes makes sense if they are better (take less tests) than the naive id-code which tests batches containing single signatures. >From Theorem 1 we can derive an interesting corollary.

Corollary 1. *Id-codes better than the naive id-code exist only if $t < u/2$.*

Proof. If $t \geq n/2$, the number of tests

$$2^v \geq \sum_{i=0}^{t} \binom{u}{i} \geq \sum_{i=0}^{u/2} \binom{u}{i} \geq \frac{1}{2} \sum_{i=0}^{u} \binom{u}{i} = 2^{u-1}$$

Thus the number of tests v must be at least $u-1$ which is almost the same as for the naive id-code which requires u tests.

Definition 2. *An index A_i includes A_j if $A_i \vee A_j = A_i$.*

Given the matrix A of an id-code. Observe that if there are two columns $i \neq j$ such that the index A_i includes A_j, then the code is unable to identify whether there are a single bad signature with the syndrome A_i or two bad signatures with the syndrome $A_i \vee A_j$. In other words, the matrix A with such indices is not able to identify bad signatures with indices A_j and A_i. We say that the two indices *collide.*

Lemma 1. *Given identification coding with a $(v \times u)$ test-checking matrix A. Assume further that there is an index A_i (column A_i) such that its Hamming weight $wt(A_i) = r$, then the number of colliding indices with A_i is $C_\#(A_i) = 2^r + 2^{v-r} - 2$.*

Proof. There are two cases where collision may occur

– the index A_i includes other indices $(A_i \vee A_k = A_i)$ for some k,
– the index A_i is included in other indices $(A_i \vee A_k = A_k)$.

For a given index A_i with its Hamming weight r, we can create $2^r - 1$ indices which are included in A_i – the first case. We can also create $2^{v-r} - 1$ indices which include A_i – the second case. In effect, we have to exclude $2^r + 2^{v-r} - 2$ indices.

Corollary 2. *To increase effectiveness of identification codes we should select weights of indices so the number of colliding indices is minimal. The smallest number of colliding indices occurs when the Hamming weight of all indices is $\frac{v}{2}$.*

Assume that we have two indices A_i and A_j. We can define the intersection of the two as $A_i \wedge A_j$ where $\wedge$ is bit-by-bit logical AND.

Lemma 2. *Given two indices A_i and A_j such that $wt(A_i) = r_1$ and $wt(A_j) = r_2$. Denote $A_c = A_i \wedge A_j$ – the maximal index which is contained in both A_i and A_j and $wt(A_c) = r$. Then the number of indices which collide with the pair (A_i, A_j) is*

$$C_\#(A_i, A_j) = 2^{v-r_1} + 2^{v-r_2} + 2^{r_1+r_2-r} - 2^{v+r-r_1-r_2} - 2^{r_1-r} - 2^{r_2-r}.$$

Proof. Denote $\mathcal{A} = \{A_1, \ldots, A_u\}$. Note that $C_\#(A_i, A_j) \geq C_\#(A_i \vee A_j)$ and becomes the equality only if $r = 0$. ¿From Lemma 1, we can write

$$C_\#(A_i \vee A_j) = 2^{r_1+r_2-r} + 2^{v+r-r_1-r_2} - 2.$$

Denote $\#A_i$ and $\#A_j$ the numbers of colliding indices from A_i and A_j, respectively, which have not been considered among the indices from $A_i \vee A_j$. Thus, we have

$$C_\#(A_i, A_j) = C_\#(A_i \vee A_j) + \#A_i + \#A_j.$$

There are the following cases, the index

– collides with A_i – there are 2^{r_1} such indices,
– collides with $A_j \setminus A_i$ – there are 2^{r_2-r} such indices,
– collides with $\mathcal{A} \setminus (A_i \wedge A_j)$ – there are $2^{v+r-r_1-r_2}$ such indices.

Observe that indices colliding with $A_j \setminus A_i$ have been already counted in $C_\#(A_i \vee A_j)$. Further on, note that the zero index (all bits are zero) has been counted. Therefore

$$\#A_i = (2^{r_2-r} - 1)(2^{v+r-r_1-r_2} - 1) \text{ and } \#A_j = (2^{r_1-r} - 1)(2^{v+r-r_1-r_2} - 1).$$

Adding the numbers we obtain the final result.

Lemma 3. *Given identification code determined by its $(v \times u)$ matrix A. If there is a parameter $k \leq u$ and a sequence of indices $(A_{i_1}, \ldots, A_{i_k})$ such that*

$$\bigvee_{j=1}^{k} A_{i_j} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \overset{def}{=} \mathbf{1}_v,$$

then the id-code can identify no more than k bad signatures. Where $\bigvee_{j=1}^{k}$ stands for bit-by-bit logical OR and $\mathbf{1}_v$ is a binary vector of length v containing ones only.

Proof. Denote $\mathcal{A} = \{A_1, \ldots, A_u\}$ as the set of all indices (columns) of the matrix A. Create the following two sets: $\mathcal{A}_1 = \{A_{i_1}, \ldots, A_{i_k}\}$ and $\mathcal{A}_2 = \mathcal{A} \setminus \mathcal{A}_1$. The proof proceeds by contradiction. Assume that any $t = k + 1$ bad signatures can be identified. Now we take a sequence of k bad signatures with their indices $(A_{i_1}, \ldots, A_{i_k})$. Their syndrome is $\mathbf{1}_v$. Now if there is an extra bad signature than the collection of t bad signatures have the same syndrome – there is a collision and we have obtained the contradiction.

E Constructions of Id-Codes

As we know, one would wish to have an identification code which allows for gradual increment of t with a possible re-use of all tests conducted for smaller ts. Now we present our main construction.

Definition 3. *A $(k+1)n \times n^2$ matrix A with binary elements is a* OR-checking matrix *if there are $k+1$ ones per column, n ones per row, and the inner product of any pair of columns is either zero or one.*

Lemma 4. *Given a $(k+1)n \times n^2$ OR-checking matrix A. Then the OR of any subset of k columns is unique for $k = 1, \ldots, n-1$.*

Proof. For convenience in typesetting we will write these columns as rows by transposing the matrix – so we are going to consider A^T. We consider any k rows but permute them so that the ones are moved to the left of each row as far as possible. We now consider a simple counting argument to look at the intersection patters of the rows. If any two rows have an intersection $+1$, the ones (written as x) will use a total of $\frac{1}{2}(k+1)(k+2) - 1$ columns and be able to be represented as:

```
  k+1          k        k-1      ...  | 2
x x x ... x  0 0 ... 0  0 ... 0   ...  |00
x 0 0 ... 0  x x ... x  0 ... 0   ...  |00
0 x 0 ... 0  x 0 ... 0  x ... x   ...  |00
                     .
                     .
                     .
0 0 0 ... x  0 0 ... x  0 ... x   ...  |xx
```

If any pair of rows do not have intersection $+1$ then more than $\frac{1}{2}(k+1)(k+2)-1$ columns will be needed to represent the patters of ones but the last row will always have at least 2 elements $+1$ at the right of the row which have no element in the column above either of them which is non-zero.

Now suppose that the matrix yielded that any $k-1$ rows corresponding to bad signatures gave a unique OR but that there are two solutions which give the same result for k rows indicating bad signatures. We rearrange the rows in our pattern representative, if necessary, so one of these two solutions is the last row. We now consider the other solution. For the first $k-1$ vectors and the second solution to cover the same number of columns the second solution must have two $+1$ at the right of the row which have no element in the column above either of them non-zero. But this means the first and second solution have at intersection at least 2 ones contradicting the definition of the OR-checking matrix. Hence any collection of k rows produces OR sums which are distinct.

We note that this proof does not extend to a collection of $k+1$ rows because in that case we could only assume the last row to have more than one elements $+1$ at the right of the last row which has no element in the column above it which is non-zero. This does not lead to any contradiction.

Corollary 3. *Given a $(k+1)n \times n^2$ OR-checking matrix A whose every two column intersection is either zero or one. Then there is an $IC(u,t)$ code which is capable to identify up to $t = n-1$ bad signatures within a batch of size $u = n^2$.*

The identification code based on OR-checking matrices is efficient as it allows to re-use all previous results if the guess about the parameter t has been wrong. Given a batch $\mathcal{B}^u$ of the size $u = n^2$. The $(n \times u)$ OR-checking matrix A is created. Denote $A^{(t)}$ as a shortened version of A containing first $(t+1)n$ rows of A; $t = 1, 2, \ldots, n-1$.

1. The identification process starts from the assumption that $t = 1$. First collection of $2n$ tests $\mathcal{T}$ are run for batches defined by rows of the matrix $A^{(1)}$. If the bad signatures are not correctly identified (i.e. the batch without bad signatures still fails the test $\mathcal{T}$), then it is assumed that $t = 2$. Otherwise the process ends.
2. Assume that the identification using $A^{(t)}$ has failed to identify bad signatures $(t = 2, 3, \ldots, n-1)$. The collection of necessary tests are defined by $A^{(t+1)}$. Note that $A^{(t+1)}$ differs from $A^{(t)}$ in that it contains n additional rows. The identification process can be accomplished by running n additional tests corresponding to the batches defined by rows in $A^{(t+1)}$ which are not in $A^{(t)}$. If the identification has not been successful, t is increment by 1 and the process continues.

The identification fails if $t \geq n$.

The construction also gives the upper bound on the number v of necessary tests to identify t bad signatures.

Corollary 4. *The number v of tests necessary to identify t bad signatures in the batch of size u satisfies the following inequality $v \leq (t+1)\sqrt{u}$.*

There are many combinatorial structures which can be used to give the required OR-checking matrices for example transversal designs and group divisible designs. However we give a rich construction based on Latin squares.

A *Latin square* of order n is an $n \times n$ array in which n different symbols, say a, b, … each occur once in each row and column. Two Latin squares are said to the *mutually orthogonal* if when the squares are compared element by element each of the distinct pairs occurs exactly once. Formally, two Latin squares, L and L' are said to be mutually orthogonal if $L(a,b) = L(c,d)$ and $L'(a,b) = L'(c,d)$, implies $a = c$ and $b = d$. For further information, refer to [2].

Lemma 5. *Suppose there are k mutually orthogonal Latin squares of order n. Then there is a $(k+1)n \times n^2$ OR-checking matrix.*

Proof. We use the auxiliary matrices described in [2].

Example 1. Let

$$M_1 = \begin{bmatrix} \text{a b c d} \\ \text{b a d c} \\ \text{c d a b} \\ \text{d c b a} \end{bmatrix}, \; M_2 = \begin{bmatrix} \text{a b c d} \\ \text{c d a b} \\ \text{d c b a} \\ \text{b a d c} \end{bmatrix}, \; M_3 = \begin{bmatrix} \text{a b c d} \\ \text{d c b a} \\ \text{b a d c} \\ \text{c d a b} \end{bmatrix}$$

be three mutually orthogonal Latin squares of order 4 on the symbols $x_1 = a$, $x_2 = b$, $x_3 = c$ and $x_4 = d$. Define M_{ij}, $1 \leq i \leq k$, by

$$(M_{ij})_{ef} = \begin{cases} 1 \; (M_i)_{fj} = x_e, \\ 0 \text{ otherwise.} \end{cases}$$

where $1 \leq e, f \leq 4$. So M_{ij}, $1 \leq i \leq 4$ and $1 \leq j \leq 4$ can be written as

$$\begin{array}{cccc|cccc|cccc|cccc}
1 & & & & 1 & & & & 1 & & & & 1 & & & \\
 & 1 & & & & 1 & & & & 1 & & & & 1 & & \\
 & & 1 & & & & 1 & & & & 1 & & & & 1 & \\
 & & & 1 & & & & 1 & & & & 1 & & & & 1 \\
\hline
1 & & & & & 1 & & & & & 1 & & & & & 1 \\
 & 1 & & & 1 & & & & & & & 1 & & & 1 & \\
 & & 1 & & & & & 1 & 1 & & & & & 1 & & \\
 & & & 1 & & & 1 & & & 1 & & & 1 & & & \\
\hline
1 & & & & & & & 1 & & 1 & & & & & 1 & \\
 & & & 1 & 1 & & & & & & 1 & & & 1 & & \\
 & 1 & & & & & 1 & & 1 & & & & & & & 1 \\
 & & 1 & & & 1 & & & & & & 1 & 1 & & & \\
\hline
1 & & & & & & 1 & & & & & 1 & & 1 & & \\
 & & 1 & & 1 & & & & & 1 & & & & & & 1 \\
 & & & 1 & & 1 & & & 1 & & & & & & 1 & \\
 & 1 & & & & & & 1 & & & 1 & & 1 & & &
\end{array}$$

Corollary 5. *Let $q > 2$ be a prime power then there are $q - 1$ mutually orthogonal Latin squares of order q*

Many other results are also known, for example for every $n \geq 3$ except 6 there are at least two orthogonal Latin squares of order n and for $n > 90$ there are at least 6.

F Hierarchical Identification

Identification codes are designed to work with a batch of fixed size. In practice, one would expect to have an identification scheme which is going to work with a batch of arbitrary length. Hierarchical identification provides such a scheme. Consider a family of id-codes $\mathcal{F} = \{IC(v,t)\}$ with some well defined parameters (v,t).

Definition 4. *Given a batch $\mathcal{B}^u$ of arbitrary length u. Hierarchical identification based on the family of identification codes $\mathcal{F}$ is a procedure defined recursively:*

- *stopping case – if the size of the batch u is smaller or equal to some parameter v so we can use the identification code $IC(v,t) \in \mathcal{F}$, then we apply it (flat identification), otherwise*
- *recursive step – if the size of the batch u is bigger than the highest parameter v_{max} in the family $\mathcal{F}$, then it is divided into ℓ sub-batches such that $\ell \leq v_{max}$ and there is some $IC(v,t) \in \mathcal{F}$ which can be used to identify contaminated sub-batches where $\ell \leq v$ and $(t' \leq t)$.*

The hierarchical identification is denoted by $HI(\mathcal{F})$.

Hierarchical identification can be based on different collections of id-codes. There are two extreme cases:

- $\mathcal{F}$ consists of infinite sequence of id-codes,
- the family $\mathcal{F}$ is reduced to a single id-code.

No matter what is the underlying family $\mathcal{F}$, one would ask the following questions:

- What is the minimum (maximum, average) number of tests which is necessary to identify all bad signatures ?
- Given a family $\mathcal{F}$ and the number t of bad signatures in the batch, is there any procedure which minimises the number of tests ?

F.1 Hierarchical Identification with Infinite $\mathcal{F}$

Consider id-codes defined in Section E. Each id-code can be uniquely indexed by a prime power $p > 2$. For this index, the code is $IC(p^2, p-1)$. The family

$$\mathcal{F} = \{IC(p^2, p-1) | p \text{ is the prime power}; p \neq 2\}$$

Note that $IC(p^2, p-1)$ can be used to identify up to $p-1$ bad signatures. If the number of bad signatures is $t \leq p-1$, the code will use $(t+1)p+1$ tests. If $t > p-1$, then the code fails.

Let $\#\mathcal{T}(\mathcal{F})$ be the number of tests necessary to identify all t bad signatures in a batch $\mathcal{B}^u$. Now we are trying to evaluate lower and upper bound for the number $\#\mathcal{T}(\mathcal{F})$. Assume that the size of the batch $u = p^2$ where p is a prime power. Now we choose somehow $p_1 < p$ and divide the batch $\mathcal{B}^u$ into p_1^2 sub-batches. Each sub-batch contains $\frac{u}{p_1^2}$ elements. Note that we have to consider only codes for which $t > p_1 - 1$ as otherwise the code may fail.

Let the t bad signatures be clustered into r sub-batches each containing t_i bad signatures so $t = \sum_{i=1}^{r} t_i$, where $r \leq p_1 - 1$ and naturally, $t_i \leq \frac{u}{p_1^2}$. The number $\#\mathcal{T}(\mathcal{F})$ has two components:

1. the number of tests necessary to identify all contaminated sub-batches – this takes $\alpha = (r + 1)p_1 + 1$,
2. the number of tests necessary to identify bad signatures within the sub-batches. For a given sub-batch, we count the number of necessary tests. First we choose a prime power p_2 such that $p_2^2 \geq \frac{u}{p_1^2}$. As the sub-batch contains t_i bad signatures we need
$$\beta_i = (t_i + 1)p_2 + 1$$
tests.

The number of tests $\#\mathcal{T}(\mathcal{F}) = \alpha + \sum_{i=1}^{r} \beta_i$ which after simple transformations gives $\#\mathcal{T}(\mathcal{F}) = (r + 1)p_1 + p_2(t + r) + (r + 1)$. The number $\#\mathcal{T}(\mathcal{F})$ depends on the random parameter r and grows linearly with r so $\#\mathcal{T}(\mathcal{F})$ is smallest for $r = 1$ when all bad signatures occur in a single sub-batch. $\#\mathcal{T}(\mathcal{F})$ takes on the maximum for $r = t = p_1 - 1$. So we have the following corollary.

Corollary 6. *Given a batch $\mathcal{B}^u$ with t bad signatures. Hierarchical identification with infinite $\mathcal{F}$ will consume $\#\mathcal{T}(\mathcal{F})$ tests where*
$$2p_1 + (t + 1)p_2 + 2 \leq \#\mathcal{T}(\mathcal{F}) \leq p_1^2 + 2p_1p_2 + p_1 - 2p_2.$$

F.2 Hierarchical Batching with a Single $IC(v, t)$

In some applications, one would like to keep the identification procedure as simple as possible which is using a single identification code or in other words the family $\mathcal{F}$ contains a single element. Again, knowing the number t of bad signatures in a batch $\mathcal{B}^u$, one would like to see how the number of necessary tests to identify all signatures varies (lower and upper bounds) as a function of the u and t.

Assume that $v = p^2$ and we apply the id-code $IC(p^2, p - 1)$. Given a batch $\mathcal{B}^u$. There are two ways bad signatures can be identified:

– Serial identification – a batch is divided into $\frac{u}{p^2}$ sub-batches. For each sub-batch, the id-code is used. This is a serial application of flat identification.
– Hierarchical identification – a batch is divided into v sub-batches and the id-code is applied for the sub-batches and identifies the contaminated sub-batches. The process is repeated for contaminated sub-batches as many times as necessary to identify bad signatures.

Consider serial identification. Note that if a batch $\mathcal{B}^{p^2}$ is clean ($t = 0$), it takes one test to verify it. If the batch is contaminated by $t < p$ bad signatures, the identification will take $(t + 1)p + t + 1$ tests. Assume that a batch $\mathcal{B}^u$ has been divided into $R = \frac{u}{p^2}$ sub-batches (if u is a multiple of p^2) among which r sub-batches are dirty and the other $R - r$ are clean. All clean sub-batches consume one test each. A dirty sub-batch $\mathcal{B}_i$ takes $(t_i + 1)(p + 1)$ tests where $\sum_{i=1}^{r} t_i = t$. So the number of tests required to identify bad signatures is

$$\frac{u}{p^2} - r + (p + 1)(t + r)$$

Note that the number of tests is a random variable which ranges from $r = 1$ when all bad signatures happen to be in one sub-batch, to $r = t$ when there are t sub-batches each containing a single bad signature.

Consider the second case of hierarchical identification. To simplify our deliberations, assume that $u = p^{2j}$ for some integer j. Denote $\#\mathcal{T}(j, t)$ the number of tests needed to identify t bad signatures in a batch $\mathcal{B}^{p^{2j}}$ when the id-code is applied to the sub-batches each containing $p^{2(j-1)}$ signatures. The following recursive equation is easy to derive

$$\#\mathcal{T}(j, t) = (r + 1)p + r + \sum_{i=1}^{r} \#\mathcal{T}(j - 1, t_i),$$

where r is a random variable which indicates the number of contaminated sub-batches and t_i are numbers of bad signatures in the corresponding contaminated sub-batches; $i = 1, \ldots, r$.

G Conclusions

The generic class of id-codes has been defined using the test-checking matrix A. The $(u \times v)$ matrix A determines the necessary tests. The syndrome is the binary vector which gives the test results for sub-batches defined by rows of A. The syndrome is also equal to bit-by-bit inclusive-OR of indices which correspond to bad signatures. We have investigated interaction of indices and found out that to maximise the identification capability of an id-code, one would need to choose indices of their Hamming weight equal to $v/2$.

The main construction of id-codes uses the so-called OR-checking matrix. The id-code takes a sequence of n^2 signatures and allows to identify up to $n - 1$ bad signatures. The nice characteristic of the code is that the number of tests can be reduced if the batch contains less than $n - 1$ bad signatures. To identify a single bad signature, it takes $2n$ tests. Any additional bad signature, adds n additional tests necessary for correct identification. There are many combinatorial structures which can be used to design id-codes. We have shown how mutually orthogonal Latin squares can be applied to construct id-codes.

We have not discussed the identification procedure of bad signatures in our id-code. The problem is far less complicated than for example in error correcting codes, mainly because the monotonicity of the Hamming weight of the syndrome. In other words, indices of bad signatures must be included in the syndrome. The implementation of this process can be done by

– checking all signatures one by one and marking those whose index collides with the syndrome,
– removing all signatures belonging to those sub-batches which have passed the test (they identified by zeros in the syndrome). In other words, all bad signatures are in the set
$$\mathcal{B} \setminus \bigcup_{\mathcal{T}(\mathcal{B}_i)=0} \mathcal{B}_i$$
where $\mathcal{B}_i$ is the sub-batch determined by the i-th row of the id-code.

Id-codes can be used directly to a contaminated batch. We called this flat identification. Alternatively, a contaminated batch can be first grouped into sub-batches and the id-code is applied to sub-batches and identifies contaminated sub-batches. This process can be done many times until bad signatures are identified. This is the hierarchical identification.

There are still many open problems. The obvious one is whether the construction given in this work is "optimal", i.e. identification of bad signatures consumes the smallest possible number of tests. Hierarchical identification allows to avoid natural limitations imposed by the size of batch and apply the id-code in hand to a batch of arbitrary length. Is there any strategy for grouping signatures into sub-batches so the number of necessary tests is minimised ?

References

1. M. Bellare, J. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In K. Nyberg, editor, *Advances in Cryptology - EUROCRYPT'98*, pages 236–250. Springer, 1998. Lecture Notes in Computer Science No. 1403. 144
2. C.J. Colbourn and J.H. Dinitz, editors. *The CRC Handbook of Combinatorial Designs.* CRC Press, Boca Raton, FL, 1996. 150, 150
3. J-S. Coron and D. Naccache. On the security of RSA screening. In H. Imai and Y. Zheng, editors, *Public Key Cryptography – Second International Workshop on Practice and Theory in Public Key Cryptography, PKC'99*, pages 197–203. Springer, 1999. Lecture Notes in Computer Science No. 1560. 144
4. J. Pastuszak, D. Michałek, J. Pieprzyk, and J. Seberry. Identification of bad signatures in batches. In H. Imai and Y. Zheng, editors, *Public Key Cryptography – Third International Workshop on Practice and Theory in Public Key Cryptography, PKC'2000*, pages 28–45. Springer, 2000. Lecture Notes in Computer Science No. 1751. 144, 144

Distributed Signcryption

Yi Mu and Vijay Varadharajan

School of Computing and IT, University of Western Sydney, Nepean,
P.O.Box 10, Kingswood, NSW 2747, Australia
{yimu,vijay}@cit.nepean.uws.edu.au

Abstract. This paper proposes a distributed encryption scheme, where any party can "signcrypt" a message and distribute it to a designated group and any member in the receiving group can "de-signcrypt" the message. We also propose a group signcryption, where, given a designated group, any member in the group can signcrypt a message on the group's behalf. A group signcrypted message can be distributed to another group. The proposed schemes have potential applicability in electronic commerce.

Key Words: Signcryption, Public-key Cryptography.

A Introduction

Digital signatures are used to ensure the authenticity of information and its sender, whereas the information confidentiality is achieved using encryption schemes. Hence to achieve both authenticity and confidentiality both signing and encryption techniques are needed. That is, to secure the message, it is first signed and then encrypted. The total computational cost therefore includes the computational costs for performing digital signatures and encryption. Recently, a new scheme referred to as *signcryption* that combines encryption with signature has been proposed [1]. The main advantage of the signcryption scheme is claimed to be the savings in computational cost achieved by combining signature and encryption in a specific way. The total computational cost is less than the sum of the computational costs of encryption and signature. It is important to note that the signcryption scheme is different from the normal "sign then encrypt" method; it forms a special combination of signing and encryption procedures.

This paper proposes a distributed signcryption scheme that can be used for distributing a signcrypted message to a designated group. First, we consider a distributed encryption scheme that allows any member in the system to encrypt a message using the group public key and to send it to members of the group. Any member of the group can decrypt the message using his or her private key. Each member of the group has a private key that matches with the group public key. We assume the existence of a group manager, a member trusted by all the members of the group, who is responsible for constructing the group public key and the management of the group. Then the paper describes the distributed signcryption scheme, where the signcryption scheme allows any member in the system to signcrypt a message and send it to members of a group. Any legal member of the group can de-signcrypt the message. This scheme is computationally efficient in the case of multiple receivers by inheriting the property from the original signcryption scheme. We then extend our scheme to a group signcryption scheme whereby the scheme enables a member of one group to signcrypt a message on behalf of the group and distribute it to any member in another group. This group signcryption scheme preserves the anonymity of the signer/sender of the message. The schemes proposed in this paper is computationally efficient in a distributed environment. This is because only one signcryption is needed for a group of n recipients.

The rest of this paper is organized as follows. Section 2 gives an overview of the basic signcryption scheme and the ElGamal encryption scheme. Section 3 describes our distributed encryption scheme. Section 4 presents our distributed signcryption scheme and section 5 extends our scheme to a group signcryption.

B Preliminaries

In this section, we briefly look at the earlier digital signcryption scheme [1] and the ElGamal encryption scheme [2]. Throughout this paper, we denote by p a large prime number, Z_p^* a multiplicative group of order q for $q = p - 1$, and $g \in \mathcal{Z}_p^*$ a primitive.

B.1 Digital Signcryption

Signcryption refers to a cryptographic method that fulfils both the functions of secure encryption and digital signature, but with the total computational cost smaller than the sum of the computational costs required for signature and encryption.

The basic signcryption scheme is based on the Digital Signature Standard (DSS) [3] with a minor modification that makes the scheme more computationally efficient. The modified DSS is referred to as SDSS and there are two versions of the SDSS.

(1) SDSS1: The signer chooses a number $x \in \mathcal{Z}_q$ at random and computes:

$$r = \mathcal{H}(g^x \bmod p \| m), \quad s = x(r + x_s)^{-1} \bmod q.$$

where $g^q = 1 \bmod p$, x_s is the private key of the signer and $k = g^x \bmod p$ is the encryption key that is used to encrypt m using a symmetric key method to obtain the ciphertext c. The signcrypted text consists of (r, s, c).
The verifier recovers the key by computing

$$k = (y_s g^r)^s \bmod p,$$

where y_s is the signer's public key ($y_s = g^{x_s}$). k can be used to decrypt the ciphertext c to obtain m.

(2) SDSS2: The signer chooses a number x at random and computes:

$$r = \mathcal{H}(g^x \bmod p \| m), \quad s = x(1 + x_s r)^{-1} \bmod q, \quad k = g^x \bmod p.$$

The signcrypted text consists of (r, s, c).
The verifier recovers the key by computing

$$k = (y_s^r g)^s \bmod p$$

Two alternative schemes that can be used in signcryption are ElGamal's digital signature scheme [4,2] and Schnorr's digital signature scheme [5].

The complete signcryption protocol is derived from the SDSS and is described below. In the protocol below, we denote by E_k a symmetric encryption under key k, D_k the corresponding decryption. We also denote by $\mathcal{H}(.)$ a strong one-way hash function such as Secure Hash Standard (SHS) [6]. The symmetric encryption and decryption algorithm can be an algorithm such as the DES [7]. We have also used a keyed one-way hash function denoted by $\mathcal{H}_k(.)$. The characteristic of a keyed hash function lies in the fact that the hashed value can be computed only when the key is known and it is computationally infeasible to find message m_1 from m_2 if $m_2 = \mathcal{H}_k(m_1)$. A is the person who signcrypts the message m. The key k is partitioned into k_1 and k_2 of appropriate length and this partitioning procedure is assumed to be public and known to both parties A and B. The encryption of the message is done using the key k_1. The key k_2 is used in the keyed hash function. The signature part s is based on either SDSS1 or SDSS2. Upon receipt of the signcrypted text (r, s, c), B can recover the key k, split it into k_1 and k_2 and use them to decrypt c and verify r.

$$\begin{array}{lcl}
\qquad A & & \qquad B \\
x \in \mathcal{Z}_q & & \\
k = y_b^x \bmod p & & \\
\text{Split } k \text{ into } k_1 \text{ and } k_2 & & \\
r = \mathcal{H}_{k_2}(m) & & \\
s = x(r + x_a)^{-1} \bmod p \ \ \text{(SDSS1)} & & \\
s = x(1 + r x_a)^{-1} \bmod p \ \ \text{(SDSS2)} & & \\
c = E_{k_1}(m) & & \\
& \xrightarrow{r,s,c} & k = (y_a g^r)^s \bmod p \ \ \text{(SDSS1)} \\
& & k = (y_a^r g)^s \bmod p \ \ \text{(SDSS2)} \\
& & \text{Split } k \text{ into } k_1 \text{ and } k_2 \\
& & m = D_{k_1}(c) \\
& & r \overset{?}{=} \mathcal{H}_{k_2}(m)
\end{array}$$

B.2 ElGamal Encryption Scheme

The security of ElGamal's encryption scheme [2] is based on the difficulty of computing discrete logarithms in a finite field. In ElGamal encryption scheme, the recipient is assumed to have a pair of public key and private key $(y =$

$g^x \bmod p, x)$, where g and p are public information and can be shared among a group of users.

To encrypt a message m, the sender first chooses a random number k, such that k is relatively prime to $p-1$. The sender then computes the following:

$$c_1 = g^k \bmod p,$$
$$c_2 = my^k \bmod p.$$

The pair (c_1, c_2) is the ciphertext.

The recipient can decrypt (c_1, c_2) to retrieve m as s/he knows the x.

$$m = c_2 / c_1^x \bmod p$$

C Distributed Encryption

This section describes the distributed encryption scheme. Consider a group with a set of members and with a public key referred to as the group public key. Each member of the group has a private key that matches with the group public key. In distributed encryption, any member in the system (within or outside the group) can encrypt a message using the group public key and can send it to members of the group. Any member of the group can then decrypt the message using his or her private key.

C.1 Construction of A Group

We assume the existence of a group manager who is trusted by the members of the group. The group manager is responsible for constructing the group public key and updating group members.

In order to construct a group, the group manager selects a set of integers, $x_i \in_R \mathcal{Z}_q$ and computes the coefficients $\alpha_1, ..., \alpha_n \in \mathcal{Z}_q$ of the following polynomial:

$$f(x) = \prod_{i=1}^{n}(x - x_i) = \sum_{i=0}^{n} \alpha_i x^i \tag{1}$$

This function has the following property: Define $\beta_i \leftarrow g^{\alpha_i} \bmod p$ for $i = 0, 1, ..., n$, then we have

$$\mathcal{F}(x_l) = \prod_{i=0}^{n} \beta_i^{x_l^i} = 1 \bmod p \tag{2}$$

This is because $\mathcal{F}(x_l) = g^{f(x_l)}$ and $f(x_l) = 0$ in $\mathcal{Z}_{p-1}$.

The above property is important for us to construct our system. However, it is proved that the polynomials are not secure, even if $\{\alpha_i | i = 1, ..., n\}$ are kept secret from a potential adversary[8]. In order to make the polynomials, $\{\beta_i|i = 0, ..., n\}$, tamper-proof, we adopt the method given in [8]:

For the given $\{\alpha_0, \alpha_1, ..., \alpha_n\}$, we define a new set $\{\alpha'_0, \alpha'_1, , ..., \alpha'_n\}$, where $\alpha'_0 = \alpha_0$, $\alpha'_n = \alpha_n$, $\alpha'_1 = ... = \alpha'_{n-1} = \sum_{i=1}^{n-1} \alpha_i$. Define $\{\beta'_0, \beta'_1, ..., \beta'_n\} \leftarrow \{g^{\alpha_0}, g^{\alpha'_1}, ..., g^{a_n}\}$ and $A_l = \sum_{i=1,l=1,i\neq l}^{n-1} \alpha_l x_l^i$, then the property (cf. Equation (2)) still holds:

$$\mathcal{F}'(x_l) = g^{-A_l} \prod_{i=0}^{n} {\beta'}_i^{x_l^i} = g^{-A_l} g^{\sum_{i=0}^{n} \alpha'_i x_l^i} = 1 \pmod{p}, \quad \text{for all } x_l,\ l = 1, ..., n.$$

In order to construct group public key, the group manager picks a number $\gamma \in_R \mathcal{Z}_q$ for member l at random and computes its inverse γ^{-1} and parameters $\rho_l \leftarrow \gamma A_l \bmod q$. The group public key is defined as an $n+1$ tuple, $\{\beta'_1, ..., \beta'_{n+1}\} \leftarrow \{\beta'_1, ..., \beta'_n, g^{\gamma^{-1}}\}$. The group manager keeps γ and all $\{\alpha_i\}$ secret and gives x_l and ρ_l to group member l, who uses x_l as her secret key.

Proposition 1. *The group public key tuple, $\{\beta'_i|i = 0, ..., n+1\}$, can not be illegally modified to add or delete a group member.*

Notice that $\{g^{\alpha_i}|i = 0, ..., n\}$ are not given in a clear form, i.e. they are hidden in $\{g^{\alpha'_i}|i = 1, ..., n-1\}$, and also $\{A_l|l = 1, ..., n-1\}$ remain secret from the public including group members. It is impossible to modify the public key such that $\mathcal{F}'(x_l) = 0$. The detailed proof can be found in Ref. [8]. The above group encryption scheme has the following property.

Proposition 2. *A member is said to belong to the group, iff the member can prove that s/he knows the secret x_l that satisfies $\mathcal{F}'(x_l) = 1 \bmod p$.*

The proof involves the use of non-interactive discrete log proof given in [9]. Using this method, the prover can show that s/he knows the secret x_l without releasing the secret to the verifier. In the non-interactive method, the prover and the verifier do not need to interact with each other. That is, the verifier does not need to send a challenge to the prover as part of the proof process; the challenge is actually computed by the prover. We omit the detail of such proof here.

C.2 Distributed Encryption

Assume that a member Alice wants to send a message m securely to a designated group G. She picks the encryption key, k', computes $k = \mathcal{H}(m)$, and then encrypts m to obtain the ciphertext $c = (c_1, c_2)$ as follows:

$$c_1 \leftarrow \{a_0, ..., a_{n+1}\} \leftarrow \{g^{k'} {\beta'}_0^{k}, {\beta'}_1^{k}, ..., {\beta'}_{n+1}^{k}\},$$
$$c_2 = m g^{k'} \bmod p.$$

Any member in G can decrypt the ciphertext c to obtain m. Let Bob be the member l. Bob does the following:

$$\begin{aligned} c_1' &\leftarrow a_0 \prod_{i=1}^{n} a_i^{x_l^i} a_{n+1}^{\rho_l} \\ &= g^{k'} (\prod_{i=0}^{n} g^{\alpha_i k x_l^i}) \\ &= g^{k'} g^{k f(x_l)} \\ &= g^{k'} \bmod p \\ m &= c_2 / c_1' = (m g^{k'}) / g^{k'}. \end{aligned}$$

Note that $f(x_l) = 0 \bmod q$. Once m is obtained, Bob can verify the correctness of the encryption by checking whether $c_1 = \{g^{k'} \beta'^k_0, \beta'^k_1, ..., \beta'^k_{n+1}\}$ using $g^{k'}$ and $k = \mathcal{H}(m)$.

D Distributed Signcryption

In this section, we present the construction of a signcryption scheme suitable for a group of recipients. That is, any member in the system (within or outside the group) can signcrypt a message and distribute it to the members of the group. We assume that the group is constructed as described in Section 3. The encryption is still assumed to be based on the use of symmetric key techniques such as DES and the digital signature is assumed to be based on SDSS given earlier in Section 2. We will still use the notation E_k to denote symmetric key encryption using key k and D_k to denote the corresponding decryption; also, we will use a keyed one-way hash function $\mathcal{H}_k(.)$.

In the description of the signcryption protocol below, we assume that Alice is the sender who signcrypts a message m and sends the message to the designated group. Bob is a member of the designated group and Bob needs to de-signcrypt and verify the message. Assume that x_a and x_b are private keys of Alice and Bob, respectively. To signcrypt a message m, Alice does the following:

(1) Chooses a number $x \in \mathcal{Z}_q$ at random, computes $k' = g^x$, splits k' into k_1 and k_2 as agreed earlier.
(2) Computes $r = \mathcal{H}_{k_2}(m)$.
(3) Computes $s = x(rk' + x_a)^{-1} \bmod q$ if SDSS1 is used or $s = x(k' + x_a r)^{-1} \bmod q$ if SDSS2 is used.
(4) Computes $k = \mathcal{H}(m)$. The signcrypted text is as follows:

$$\begin{aligned} c_1 &\leftarrow \{a_0, ..., a_{n-1}, a_{n+1}\} \leftarrow \begin{cases} \{g^{k'r} \beta'^k_0, \beta'^k_1, ..., \beta'^k_{n+1}\} & \text{(SDSS1)} \\ \{g^{k'} \beta^k_0, \beta^k_1, ..., \beta'^k_{n+1}\} & \text{(SDSS2)} \end{cases} \\ c_2 &= E_{k_1}(m) \end{aligned}$$

(5) Alice sends to Bob or the designated group the signcrypted text (c_1, c_2, r, s).

Any one of group members can de-signcrypt and verify the signcrypted text by discovering the secret key k'. The procedure is as follows:

(1) Bob recovers k' by computing
For SDSS1:

$$\begin{aligned} k' &\leftarrow (y_a a_0 \prod_{i=1}^{n} a_i^{x_l^i} a_{n+1}^{\rho_l})^s \\ &= (y_a g^{rk'} \prod_{i=0}^{n} g^{(\alpha_i k x_l^i)})^s \\ &= (y_a g^{rk'} g^{kf(x_l)})^s \text{ where } f(x_l) = 0 \bmod q \\ &= g^x \bmod p \end{aligned}$$

For SDSS2:

$$\begin{aligned} k' &= (y_a^r a_0 \prod_{i=1}^{n} a_i^{x_l^i} a_{n+1}^{\rho_l})^s \\ &= (y_a^r g^{k'} \prod_{i=0}^{n} g^{(\alpha_i k x_l^i)})^s \\ &= (y_a^r g^{k'} g^{kf(x_l)})^s \text{ where } f(x_l) = 0 \bmod q \\ &= g^x \bmod p \end{aligned}$$

(2) Bob splits k' into k_1 and k_2 as agreed earlier and computes $m = D_{k_1}(c)$.

Completeness: It is obvious that the recipient can de-signcrypt the message by following the decryption process above. The only way to decrypt is to have one of members' private keys, x_l, along with ρ_l, which gives $\mathcal{F}'(x_l) = 0$.
Soundness: The a_0 is a ciphertext similar to an ElGamal encryption. Alice has to use the group public key, otherwise no one in the group can de-signcrypt the message. Alice also has to use her private signing key to compute s, since her public key is used in the verification of the signcryption. If s does not embed x_a, x_a cannot be removed in the verification in order to obtain k'.

Proposition 3. *Any group member is able to verify the signcrypted text and obtain the embedded message.*

In fact, it is not possible to exclude any particular member of the group from receiving the signcrypted text (and hence the plaintext) that is intended for the group. This can only be achieved by removing the member from the group.

This scheme is computationally efficient considering the distributed environment that requires polynomial computations. Actually, it is obvious that when sending a message to a group of $\mathcal{N}$ members, only one encryption is required. However, $\mathcal{N}$ encryption operations are used the scheme proposed in [1].

E Extension

In this section, we propose a distributed signcryption scheme whereby the system members are partitioned into a set of groups and the scheme enables a member of one group to signcrypt a message on behalf of the group and send it to another member in an other group. This distributed signcryption scheme satisfies the following:

- The anonymity of the member who signcrypts the message is preserved. That is, no one except the group manager is able to find out the identity of the sender.
- At the end of the protocol, the members of the receiving group are confident that the message has been signcrypted by a member of the sending group.
- The member of the sending group who signcrypted the message is confident that the message can only be read by the members of the receiving group.

E.1 The Protocol

Consider two designated groups G_a and G_b and assume that Alice belonging to group G_a wishes to send a signcrypted message m to the group G_b and that Bob is one of recipients belonging to group G_b. Let Alice's personal public key be $y_a = g^{x_a} \bmod p$, where x_a is her group private key satisfying $f_a(x_a) = 0$, where $f_a()$ is the polynomial function of G_a (defined in Section 2). Similarly, Bob has $y_b = g^{x_b} \bmod p$ and x_b. The group public keys for G_a and G_b are respectively $\{\beta'_0, \beta'_1, ..., \beta'_{n+1}\}$ and $\{\bar{\beta}'_0, \bar{\beta}'_1, ..., \bar{\beta}'_{n+1}\}$. Both public keys have the same form as those given earlier.

In order to signcrypt the message, Alice needs to do the following:

- Chooses an integer, x, from $\mathcal{Z}_q$.
- Computes $k' \leftarrow g^x \bmod p$ and splits k' into k_1 and k_2.
- Computes $k = \mathcal{H}(m)$.
- Computes the signing commitment, $u_j \leftarrow \bar{\beta}'^{\,k}_j$ for $(j = 1, ..., n)$
- Computes $r = \mathcal{H}_{k_2}(m)$ and $s_j = k(x_a^j - ru_j) \bmod q \quad (j = 1, ..., n)$.
- The signcryption is then computed as follows:

$$\begin{aligned} c_1 &\leftarrow \{a_0, a_1, ..., a_{n+1}\} \leftarrow \{g^{k'r}\beta'^{\,k}_0, \beta'^{\,k}_1, ..., \beta'^{\,k}_{n+1}\} \\ c_2 &\leftarrow \{\bar{a}_0, \bar{a}_{n+1}\} \leftarrow \{g^{x-rk'}\bar{\beta}'^{\,k}_0,\ \bar{\beta}'^{\,k\rho_a}_{n+1}\ \} \\ c_3 &\leftarrow E_{k_1}(m). \end{aligned}$$

- Sends the signcryption tuple (c_1, c_2, c_3, r, s_j) to group G_b. (x, k, k', k_1, k_2) remain secret.

The public key of the sender or G_a should be known to the recipient or G_b. The verification process includes the following steps:

1: Key recovery: The key, k', can be computed by Bob or another member in the group, G_b:

$$\begin{aligned}
k' &= [a_0 \prod_{i=1}^{n} a_i^{x_b^i} a_{n+1}^{\rho_b}][\bar{a}_0 \prod_{j=1}^{n} u_j^{ru_j} \bar{\beta}'^{\,s_j}_j \bar{a}_{n+1}] \\
&= [(g^{k'r} \beta_0^k \prod_{i=1}^{n} \beta'^{\,kx_b^i}_i \beta'^{\,\rho_b}_{n+1}][g^{x-rk'} \bar{\beta}'^{\,k}_0 \prod_{j=1}^{n} \bar{\beta}'^{\,kx_a^j}_j \bar{\beta}'^{\,k\rho_a}_{n+1}] \\
&= [g^{rk'} \prod_{i=0}^{n} \beta_i^{kx_b^i}][g^{x-rk'} \prod_{j=0}^{n} \bar{\beta}_j^{kx_a^j}] \\
&= g^x \pmod{p} \qquad (3)
\end{aligned}$$

2: Verification of correctness of a_0 and $\bar{a}_0$: This is done by checking a_0 and $\bar{a}_0$ are indeed constructed with β'^{k}_0 and $\bar{\beta}'^{k}_0$ respectively (Notice that k is now known).

Once k' is obtained, Bob splits it to k_1 and k_2, and verifies/computes the message.
Completeness: It is obvious that if Alice follows the signcryption rule, Bob can obtain the correct key, k'.
Soundness: There are two points we intend to make:

– Alice must be a member of G_a and follow the correct signcryption procedure, or Bob will not be able to de-signcrypt the signcryted message.
– If Bob is no a member of G_b, he will not be able to de-signcrypt the message.

To de-signcrypt the message, Bob needs to obtain k' by removing β'^{k}_0 from a_0 and $\bar{\beta}'^{k}_0$ from $\bar{a}_0$. a_0 and $\bar{a}_0$ are constructed similarly to an ElGamal encryption, thus only way to decrypt them is to use group secret keys, in our example, x_b and x_a.

Proposition 4. *The above key discovery process is a witness indistinguishable verification of signcrypted knowledge of m.*

Proof. We omit the detailed proof here but the intuitive argument is as follows: The verification of the knowledge m is based on $\prod_{i=0}^{n+1} \bar{\beta}'^{x_j^i}_j = 1 \bmod p$. It is not possible to find which x_j is used in the signcryption. In particular, all parameters used in the protocol are one-time items and are not linked to the signer; hence the use of this information cannot help to determine the identity of the signer.

F Conclusion

In this paper, we proposed a distributed signcryption scheme that allowed any member in the system to signcrypt a message and distribute it to members of a designated group. We then extended the distributed signcryption scheme to the group signcryption. In this scheme, the system is partitioned into groups of members and this scheme enables a member of one group to signcrypt a message on behalf of the group and distribute it securely to members in the other group. Any member of the receiving group can verify the signcrypted message. This scheme preserved the anonymity of the sender of the message.

The schemes proposed in this paper are computationally efficient in the distributed environment. This is because only one signcryption is needed for a group of n recipients. The schemes proposed in this paper further enhance the practical applicability of signcryption techniques in electronic commerce. For example, in a Pay TV system, the TV programs can be signcrypted and distributed to a group of legitimate subscribers.

References

1. Y. Zheng, "Digital signcryption or how to achieve cost(signature & encryption) $\ll$ cost(signature) + cost(encryption)," in *Advances in Cryptology – CRYPTO '97 Proceedings*, Springer Verlag, 1997. 155, 156, 161
2. T. ElGamal, "A public-key cryptosystems and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. IT-31(4), pp. 469–472, 1985. 156, 157, 157
3. National Institute of Standards and Technology, "Digital encryption standard," Federal Information Processing Standards Publication FIPS PUB 186 U.S. Department of Commerce, May 1994. 156
4. T. ElGamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms," pp. 10–18, Springer-Verlag, Berlin, 1985. 157
5. C. P. Schnorr, "Efficient identification and signatures for smart card," in *Adances in cryptology - CRYPT0'89, Lecture Notes in Computer Secience 435*, pp. 239–251, Springer-Verlag, Berlin, 1990. 157
6. National Institute of Standards and Technology, "Secure hash standard," Federal Information Processing Standards Publication FIPS PUB 180-1 U.S. Department of Commerce, April 1995. 157
7. National Bureau of Standards, "Data encryption standard," Federal Information Processing Standards Publication FIPS PUB 46 U.S. Department of Commerce, January 1977. 157
8. Y. Mu, V. Varadharajan, and K. Q. Nguyen, "Delegated decryption," in *Procedings of Cryptography and Coding, Lecture Notes in Computer Science*, Springer Verlag, 1999. 158, 158, 159
9. J. Camenisch, "Efficient and generalized group signatures," in *Adances in cryptology - EUROCRYPT'97, Lecture Notes in Computer Secience 1233*, pp. 465–479, Springer-Verlag, Berlin, 1997. 159

Fail-Stop Signature for Long Messages
(Extended Abstract)

Rei Safavi-Naini*, Willy Susilo, and Huaxiong Wang

Centre for Computer Security Research
School of Information Technology and Computer Science
University of Wollongong
Wollongong 2522, AUSTRALIA
Email: {rei, wsusilo, huaxiong}@uow.edu.au

Abstract. Security of ordinary digital signature schemes relies on a computational assumption. Fail-stop signature (FSS) schemes provide security for a signer against a forger with unlimited computational power by enabling the signer to provide a proof of forgery, if it occurs. Signing long messages using FSS requires a hash function with provable security which results in a slow signature generation process. In this paper, we propose a new construction for FSS schemes based on linear authentication codes which does not require a hash function and results in a much faster signing process at the cost of slower verification process, and longer secret key and signature. An important advantage of the scheme is that proof of forgery is the same as a traditional FSS and does not rely on the properties of the hash functions.

A Introduction

Security of an *ordinary digital signature* relies on a computational assumption, that is assuming that there is no efficient algorithm to solve a particular problem. This means that if an enemy can solve the underlying hard problem, he can successfully forge a signature and there is no way for the signer to prove that a forgery has occurred. To provide protection against an enemy with unlimited computational power who can always solve the hard underlying problem, fail-stop signature (FSS) schemes have been proposed [20,9]. Loosely speaking, an FSS is a signature scheme augmented by a proof system which allows the signer to prove that a forged signature was not generated by him/her. To achieve this property, the signature scheme has many secret keys that correspond to the same public key and the sender uses a specific one of them. An unbounded enemy who has solved the underlying hard problem and knows the set of all secret keys cannot determine which secret key is actually used by the sender. In the case of a forgery, that is signing a message with a randomly chosen secret key, the sender can use his secret key to generate a second signature for the same message

* This work is in part supported by Australian Research Council Grant Number $A49703076$

which will be different with overwhelming probability from the forged one. The two signatures on the same message can be used as a proof that the underlying computational assumption is broken and the system must be stopped - hence the name *fail-stop*. Thus, FSS schemes offer an information-theoretic security for the signer. However security for the receiver is computational and relies on the difficulty of the underlying hard problem. FSS schemes in their basic form are one time primitives and so the key can be used for signing a single message.

FSS schemes and their variants have been studied by numerous authors (see, for example, [18,19,16,13,14,17]).

Signing long messages
A commonly used method of signing an arbitrary long message using a traditional signature scheme is by employing *hash-then-sign* method. Using this method, the message is first hashed and then the signature scheme is applied to the hash value. In FSS a similar method can be used. However as noted in [9] the proof of forgery will not be based on showing that the underlying assumption of the signature scheme is broken; rather, it will be by showing that a collision for the collision-resistant hash function used for hashing is found. This implies that to have an acceptable proof of forgery, a hash function which is based on a computational assumption must be used. In [2,3] hash functions based on discrete logarithm and factorization assumption are constructed and it is shown that they require on average one multiplication for each bit of the message and the size of the hash value is equal to the size of the modulus. This means that for long messages FSS schemes have a slow signature generation process (for example one million multiplications for a one Mega byte file).

An alternative approach is to have an FSS scheme that can be directly used for arbitrary long messages. A recently proposed measure [17] of efficiency of FSS is *redundancy rate*, $\hat{\rho}$, which is defined as the length of the signature divided by the length of the message. Using FSS for arbitrary long messages and without using hash functions means that FSS with low values for $\hat{\rho}$ must be designed. The lowest value of $\hat{\rho}$ for the known schemes is 1 and is achieved by a scheme proposed in [17]. For other schemes $\hat{\rho} \geq 2$ and so none of the known schemes can be used for direct signing of long messages and the only possible method is hash-then-sign.

Other efficiency measures of FSS schemes are the lengths of the secret key, public key and the signature [9]. The most efficient FSS with respect to the first three parameters is a discrete logarithm based system due to van Heijst and Pedersen [18] (or *vHP scheme*). The scheme in [14] have the same efficiency as vHP.

In this paper we propose a new FSS scheme for which $\hat{\rho}$ can be chosen arbitrarily low and so can be used for direct signing of long messages. This means that no hash function is required and so security of the resulting FSS is the same as a traditional FSS construction (based on the difficulty of DL in this case). The proposed scheme requires much less computation for signing long message compared to the hash-then-sign method and using provably secure hash functions

[2,3]. More specifically, the signing process in our scheme is around $K/2$ times faster than the vHP scheme (using a known provable secure hash functions), where K is the length of the message in the vHP scheme (at least 151 bits [8]). The drawback of our scheme compared to the vHP is that the signature verification process is slower. It also requires larger sizes for the secret key, the public key and the signature. Table 3 compares various parameters of the two schemes. Faster signing process makes the scheme attractive in environments where the client, such as the smart card or mobile phone has limited computational power but the host is a powerful computer. The construction follows a general approach for constructing FSS scheme from unconditionally secure authentication code (A-code) proposed in [13] and uses a special class of A-codes, called linear A-codes, in which the set of encoding rules written as vectors over $\mathcal{F}_q$, form a vector space. These A-codes are of independent interest because of their linearity. We give the construction of a linear A-codes using linearized polynomials over finite fields that can be used to sign arbitrary length messages. Due to the lack of space, some details and proofs are omitted.

Previous Works

The first construction of fail-stop signature [20] is a one-time signature scheme (similar to [7]) and results in bit by bit signing of a message which is very impractical. In [10] an efficient single-recipient FSS to protect clients in an online payment system, is proposed. The main disadvantage of this system is that signature generation is a 3-round protocol between the signer and the recipient and so it has high communication cost. The size of the signature is twice the length of the message. In [18], an efficient FSS that uses the difficulty of the discrete logarithm problem as the underlying assumption is presented. This is the most efficient scheme with respect to the first three parameters and results in a signature which is twice the size of the message. For the rest of this paper, we refer to this scheme as *vHP scheme*. In [17], another scheme which is nearly as efficient as the vHP scheme is proposed.

In [9,11], a formal definition of FSS schemes is given and a general construction using *bundling homomorphism* is proposed. The construction has provable security and all existing FSS with provable security are instances of this construction. It can be proved that for a system with security level δ for the signer, the signature length and the length of secret key required for signing a single message are at least $2\delta - 1$ and $2(\delta - 1)$, respectively.

In [16], an RSA-based FSS scheme is proposed. The construction follows the vHP scheme but is less efficient and produces signatures that are four times the length of the original message.

In [13], a general construction of FSS schemes from authentication codes is proposed. It is shown that a scheme that fits into the general construction of [9] can also be obtained by using this construction. However, it is not known if the two general constructions are equivalent.

B Preliminaries

B.1 Fail-Stop Signature Schemes

Similar to an ordinary digital signature scheme, an FSS scheme consists of three phases

1. *Key generation*: The signer and the center through a two party protocol generate a pair of *secret key, sk,* and *public key, pk*, and makes *pk* public. This is different from ordinary signature schemes where key generation can be performed by the signer individually and without the involvement of other parties.
2. *Sign*: For a message m, the signer uses the signature algorithm *sign* to generate the signature $y = sign(sk, m)$, and sends the pair (m, y) to the receiver(s).
3. *Test*: For a message-signature pair, the receiver(s) uses the public key *pk* and a *test* algorithm test the acceptability of the signature.

It also includes two more polynomial time algorithms:

4. *Proof:* An algorithm for proving a forgery.
5. *Proof-test:* An algorithm for verifying that the proof of forgery is valid.

A secure fail-stop signature scheme must satisfy the following additional properties [19,11,9].

1. If the signer signs a message, the recipient must be able to verify the signature (*correctness*).
2. A polynomially bounded forger cannot create forged signatures that successfully pass the verification test (*recipient's security*).
3. When a forger with an unlimited computational power succeeds in forging a signature that passes the verification test, the presumed signer can construct a proof of forgery and convinces a third party that a forgery has occurred (*signer's security*).
4. A polynomially bounded signer cannot create a signature that he can later prove to be a forgery (*non-repudiability*).

To achieve the above properties, for each public key there exists many matching secret keys such that different secret keys create different signatures on the same message. The real signer knows only one of the secret keys, and can construct one of the many possible signatures. An enemy with unlimited computing power, although can generate all the signatures but does not know which one will be generated by the true signer. Thus, it will be possible for the signer to provide a proof of forgery by generating a second signature on the message with a forged signature, and use the two signatures to show the underlying computational assumption of the system is broken, hence proving the forgery.

An FSS in its basic form is a *one-time signature scheme* that can only be used for signing a single message. However, it is possible to extend an FSS scheme to be used for signing multiple messages [2,18,1].

Security of an FSS is broken if 1) a signer can construct a signature and later provide a proof that is forged; or 2) an unbounded forger succeeds in constructing a signature that the signer cannot prove that it is forged. These two types of forgeries are independent and so two different security parameters, ℓ and δ, are used to show the level of security against the two types of attack. More specifically, ℓ is the security level of the recipient against the forgery of the signer, and δ is that of the signer against the unbounded forger. It is proved [9] that a secure FSS is secure against adaptive chosen plain-text attack and for all $c > 0$ and large enough ℓ, success probability of a polynomially bounded forger is bounded by ℓ^{-c}. For an FSS with security level δ for the signer, the success probability of an unbounded forger is limited by $2^{-\delta}$. In this case we simply call the scheme (ℓ, δ)-*secure*.

B.2 Authentication Codes

In the conventional model of unconditionally secure authentication systems, there are three participants: a *transmitter*, a *receiver* and an *opponent*. The transmitter wants to send a message to the receiver using a public channel which is subject to active attack. The opponent can impersonate the sender by inserting a message into the channel, or substitute a transmitted message with another. To protect against these attacks, transmitter and receiver use an authentication code (A-code).

A *systematic* A-code (or A-code *without secrecy*) is an A-code where the codeword (message) generated for a source state (information to be authenticated) is obtained by concatenation of an authenticator (or a tag) to the source state. The code can be specified by a triple $(\mathcal{S}, \mathcal{T}, \mathcal{E})$ of finite sets together with a (authentication) mapping $f : \mathcal{S} \times \mathcal{E} \rightarrow \mathcal{T}$. Here $\mathcal{S}$ is the set of source states, $\mathcal{E}$ is the set of keys and $\mathcal{T}$ is the set of authenticators. To send a source state $s \in \mathcal{S}$ to the receiver transmitter uses his secret key $e \in \mathcal{E}$ that is shared by the receiver, to construct a message $m = (s, t)$ where $t = f(s, e) \in \mathcal{T}$. When the receiver receives the message $m = (s, t)$, she uses her secret key to check the authenticity by verifying if $t \overset{?}{=} f(s, e)$. If equality holds, the message m is *valid*.

An opponent may insert a message $m' = (s', t')$ into the channel, without observing any previous communication, or substitute a message $m = (s, t)$ sent over the channel with another message $m' = (s', t')$. The two attacks are called *impersonation* and *substitution*, respectively. A message (s, t) is *valid* if there exists a key e such that $t = f(s, e)$. We assume that there is a probability distribution on the source states, which is known to all the participants. Given this distribution, the receiver and the transmitter will choose a probability distribution for $\mathcal{E}$. We will denote the probability of success of the opponent in impersonation and substitution attack, by P_I and P_S, respectively. Let $P(\cdot)$ and $P(\cdot|\cdot)$ denote the probability and conditional probability distribution of the message space $\mathcal{S} \times \mathcal{T}$. Then we have

$$P_I = \max_{s,t} P((s,t) \text{ valid }) \qquad \text{and}$$

$$P_S = \max_{s,t} \max_{s \neq s', t'} P((s', t') \text{ valid} \mid (s,t) \text{ observed }).$$

If we further assume that the keys and the source states are uniformly distributed, then the deception probabilities can be expressed as

$$P_I = \max_{s,t} \frac{|\{e \in \mathcal{E} : t = f(s,e)\}|}{|\mathcal{E}|},$$

$$P_S = \max_{s,t} \max_{s' \neq s, t'} \frac{|\{e \in \mathcal{E} : t = f(s,e), t' = f(s',e)\}|}{|\{e \in \mathcal{E} : t = f(s,e)\}|}.$$

Since the opponent can choose between the two attacks, the overall deception probability of an A-code is defined as $P_D = \max\{P_I, P_S\}$. An A-code is *ϵ-secure* if $P_D \leq \epsilon$. One of the fundamental results in the theory of A-codes is the *square root bound* [4], which states that $P_D \geq 1/\sqrt{|\mathcal{E}|}$ and the equality holds only if $|\mathcal{S}| \leq \sqrt{|\mathcal{E}|} + 1$. The square root bound gives a direct relation between the key size and the protection that we can expect to obtain.

A general construction of FSS from A-codes is given in [13]. The construction is for a single-message and uses two families of A-codes $\mathcal{A} = \{(\mathcal{S}_K, \mathcal{T}_K, \mathcal{E}_K) : K \in N\}$ and $\mathcal{A}' = \{(\mathcal{S}_K, \mathcal{T}'_K, \mathcal{E}'_K) : K \in N\}$, a family of polynomial time collision intractable bundling hash function $\mathcal{H} = \{h_K : K \in N\}$ where $h_K : E_K \mapsto E'_K$, and a family of polynomial time collision intractable hash functions $\mathcal{H}' = \{h'_K : K \in N\}$, where $h'_K : T_K \mapsto T'_K$ and the property that for any choice of key K, and for an arbitrary $e \in E_K$ the following is satisfied for all $s \in S_K$:

$$\text{if} \quad e(s) = t, \text{ and } h_K(e) = e', \text{then } e'(s) = t' \text{ and } h'_K(t) = t'$$

C Construction of FSS from Linear A-Codes

In this section, we introduce a new class of A-codes, called *linear A-codes*, and present a construction of FSS schemes by combining linear A-codes and a one-way functions $f_{p,g}$ based on the discrete logarithm.

In the rest of the paper, let p be a prime and $\mathcal{F}_p$ be the finite field of order p (we may regard $\mathcal{F}_p$ as Z_p). We also use $V(n,q)$ to denote an n-dimension of vector space over a finite field with order q.

Let $(\mathcal{S}, \mathcal{T}, \mathcal{E})$ be an authentication code with the authentication mapping $f : \mathcal{S} \times \mathcal{E} \longrightarrow \mathcal{T}$. To each source state $s \in \mathcal{S}$, we associate a mapping f_s from $\mathcal{S}$ to $\mathcal{T}$ defined by $f_s(e) = f(s,e), \forall e \in \mathcal{E}$. Then the family $\{f_s \mid s \in \mathcal{S}\}$ completely specifies the underlying A-code $(\mathcal{S}, \mathcal{T}, \mathcal{E})$. For our purpose, we shall require that the functions f_s have some additional properties, defined as follows.

Definition 1. *An A-code $(\mathcal{S}, \mathcal{T}, \mathcal{E})$ with the authentication mapping $f : \mathcal{S} \times \mathcal{E} \longrightarrow \mathcal{T}$ is called* linear *(over $\mathcal{F}_p$) if*

1. *Both $\mathcal{E}$ and $\mathcal{T}$ are linear space over $\mathcal{F}_p$;*
2. *For each $s \in S$, f_s is $\mathcal{F}_p$-linear from $\mathcal{E}$ to $\mathcal{T}$.*

In the sequel, we assume that $(\mathcal{S}, \mathcal{T}, \mathcal{E})$ with authentication mapping f is a linear A-code with $\mathcal{E} = V(u,p)$, $\mathcal{T} = V(v,p)$, where p is a prime. We assume that S is a subset of u by v matrices over $\mathcal{F}_p$, and f_s can be defined as $f_s(e) = es = t$, where $e \in V(u,p)$, $t \in V(v,p)$.

To construct our FSS scheme we combine a linear A-code and a one-way function based on the discrete logarithm problem and given by,

$$f_{p,g} : x \to g^x \bmod q$$

where $p|q-1$. The construction works as follows.

- *Prekey Generation:* The center selects primes q and p such that $p|q-1$ and a cyclic subgroup, H_p, of $\mathcal{F}_q^*$ with order p such that the discrete logarithm over H_p is hard. He also chooses two elements $g, h \in H_p$, and publishes (p, q, g, h).
- *Key Generation:* The signer chooses a secret key sk which consists of two authentication keys of a linear A-code $(\mathcal{S}, \mathcal{T}, \mathcal{E})$ over $\mathcal{F}_p$.
  That is, $sk = (e, e')$, $e, e' \in \mathcal{E}$, where $e = [e_1, \ldots, e_u]$, $e' = [e'_1, \ldots, e'_u]$, $\forall e_i, e'_j \in \mathcal{F}_p$. The corresponding public key pk is $g^e \odot h^{e'}$, is defined as
  $$\begin{aligned} g^e \odot h^{e'} &= [g^{e_1}h^{e'_1}, \ldots, g^{e_u}h^{e'_u}] \pmod{q} \\ &= [pk_1, \ldots, pk_u]. \end{aligned}$$
- *Sign:* To sign a message $s \in S$, the signer applies the authentication code $(\mathcal{S}, \mathcal{T}, \mathcal{E})$ to generate two authentication tags t and t' corresponding to the key e and e'. That is, the signature for a message s is
  $$\begin{aligned} (s, f(s,e), f(s,e')) &= (s, es, e's) \pmod{p} \\ &= (s, t, t') \\ &= (s, [t_1, \ldots, t_v], [t'_1, \ldots, t'_v]) \end{aligned}$$
- *Test:* For a message
  $$s = \begin{pmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,v} \\ s_{2,1} & s_{2,2} & \cdots & s_{2,v} \\ \cdots & \cdots & \cdots & \cdots \\ s_{u,1} & s_{u,2} & \cdots & s_{u,v} \end{pmatrix},$$
  (s, t, t') is a valid signed message iff for all $1 \le i \le v$,
  $$g^{t_i}h^{t'_i} = (pk_1)^{s_{1,i}}(pk_2)^{s_{2,i}} \cdots (pk_u)^{s_{u,i}} \pmod{q}.$$
- *Proof of Forgery:* If there is a forged signature $(\tilde{t}, \tilde{t'})$ on a message s, then the presumed signer can produce his own signature on the same message, namely (t, t'), and show that these two signatures collide.

Theorem 1. *Let ℓ be the security parameter of the underlying discrete logarithm problem. If the linear A-code $(\mathcal{S}, \mathcal{T}, \mathcal{E})$ is ϵ-secure, then the above construction results in a (ℓ, δ)-secure FSS scheme, where $\delta = \log(1/\epsilon)$.*

Example 1. Let $\mathcal{S} = \mathcal{T} = \mathcal{F}_p$ and $\mathcal{E} = \mathcal{F}_p \times \mathcal{F}_p$.
We define a function f, $\mathcal{S} \times \mathcal{E} \longrightarrow \mathcal{T}$, by $f(s,(e_1,e_2)) = e_1 + se_2$. Then it is easy to verify that $(\mathcal{S}, \mathcal{T}, \mathcal{E})$ is a linear A-code over $\mathcal{F}_p$. In terms of matrix representation, we can write a source state S as a 2×1 matrix over $\mathcal{F}_p$

$$S = \{s = \begin{pmatrix} 1 \\ w \end{pmatrix} \mid w \in \mathcal{F}_p\},$$

and the authentication mapping f as $f(s,(e_1,e_2)) = (e_1,e_2)\begin{pmatrix} 1 \\ w \end{pmatrix} = e_1 + we_2$. The FSS based on this linear A-code is the same as the vHP scheme.

D Construction of Linear A-Codes

Although the underlying linear A-code of the construction in example 1 is nearly optimal (i.e., nearly meets the square root bound), it requires that the size of key to be double the size of the source, i.e., $\log|\mathcal{E}| = 2\log|\mathcal{S}|$, and so the size of the key grows linearly with the size of the source. In the following, we construct linear A-codes that do not meet the bound and so are not optimal, but allow the size of the source to be much larger than the size of the keys.

A polynomial of the form

$$L(x) = \sum_{i=0}^{n} a_i x^{p^i}$$

with coefficients in an extension field $\mathcal{F}_{p^m}$ of $\mathcal{F}_p$ is called a *p-polynomial* over $\mathcal{F}_{p^m}$. If the value of p is fixed once or is clear from the context, it is also called a *linearized polynomial.* It is well-known that if $\mathcal{F}$ is an arbitrary extension field of $\mathcal{F}_{p^m}$ and $L(x)$ is a linearized polynomial over $\mathcal{F}_{p^m}$, then

$$L(\beta + \gamma) = L(\beta) + L(\gamma) \quad \text{for all } \beta, \gamma \in \mathcal{F},$$

$$L(c\beta) = cL(\beta) \quad \text{for all } c \in \mathcal{F}_p \text{ and all } \beta \in \mathcal{F}.$$

Thus, if $\mathcal{F}$ is considered as a vector space over $\mathcal{F}_p$, then the linearized polynomial $L(x)$ induces a linear operator on $\mathcal{F}$.

Next we construct a linear A-code from linearized polynomial. We note that linearized polynomials have been also used to construct authentication codes for non-trusting parties [6,5] and message authentication codes for multiple authentication [12]. Let p be a prime and assume,

- $\mathcal{S} = \{L_s(x) = \sum_{i=0}^{k-1} a_i x^{p^i} \mid a_i \in \mathcal{F}_{p^r}\}$, that is, we let each source state $s \in \mathcal{S}$ correspond to a linearized polynomial over $\mathcal{F}_{p^r}$, denoted by $L_s(x)$. We use the linearized polynomials up to degree p^{k-1}, resulting in $(p^r)^k$ different polynomials and so $|\mathcal{S}| = p^{rk}$.
- $\mathcal{E} = \{(e_1,e_2) \mid e_1, e_2 \in \mathcal{F}_{p^r}\}$, and so $|\mathcal{E}| = p^{2r}$.

- $\mathcal{T} = \mathcal{F}_{p^r}$.
- The authentication mapping $f : \mathcal{S} \times \mathcal{E} \longrightarrow \mathcal{T}$ is defined by $f(L_s(x), (e_1, e_2)) = e_1 + L_s(e_2)$.

Theorem 2. *The above construction results in a linear A-code $(\mathcal{S}, \mathcal{E}, \mathcal{T})$ over $\mathcal{F}_p$ with the following parameters*

$$|\mathcal{S}| = p^{rk},\ |\mathcal{E}| = p^{2r},\ |\mathcal{T}| = p^r$$

and

$$P_I = p^{-r},\ P_S = p^{-(r-k+1)}.$$

Combining Theorem 1 and 2, we obtain the following corollary.

Corollary 1. *Let p and q be primes such that ℓ is the required security parameter that a polynomially bounded forger cannot break the underlying discrete logarithm problem. Then the above linear A-code results in a (ℓ, δ)-secure FSS scheme such that $\delta = (r - k + 1)\log|p|$.*

E Efficiency Measures of FSS Schemes

In this section, we compare efficiency of our proposed scheme with the most efficient FSS scheme, namely vHP scheme. Firstly, we fix the level of security provided by the two schemes, and then we find the sizes of the secret key, the public key and the signature. Table 1 gives the results of the comparison of FSS schemes when security levels of the receiver and the sender are given by ℓ and δ, respectively. In this comparison, the first two schemes [18,19](first and second column of the table) are chosen because they have provable security. The first scheme (referred as vHP scheme in this paper) is the most efficient provably secure scheme, based on the discrete logarithm problems. The third column is an FSS scheme based on RSA [16]. The fourth column is a factorization based scheme proposed in [17]. Column five corresponds to the scheme from Theorem 2 with $r = k$.

In vHP scheme, given the security parameter (ℓ, δ), first $K = \max(\ell, \delta)$ is found and then the prime p is chosen such that $\log p \geq K$. The value of q is chosen such that $p|q-1$ and $(q-1)/p$ be upper-bounded by a polynomial in K (page 237 and 238 [11]). Since the sizes of p and q can be independently chosen, we use $\hat{K}$ to denote $\log_2 q$.

For our scheme, given the security parameter (ℓ, δ), first $K = \max(\ell, \delta)$ is found, and then the prime p is chosen such that $\log_2 p \geq K$. We also use the notation $\hat{K}$ to denote the size of q.

In the factorization scheme of [19], the security level of the sender, δ, satisfies $\tau = \rho + \delta$ where τ is the *bundling degree* (which determines the number of secret key preimages for a particular public key image) and 2^ρ is the size of the message space. Security parameter of the receiver, ℓ, is determined by the difficulty of factoring the modulus n (where $n = pq$ and p and q are both prime numbers). Now for a given pair of security parameters, (ℓ, δ), the size of the modulus N_ℓ is determined by ℓ but determining τ requires knowledge of the size of the message space. Assume $\rho = \log_2 p \approx \log_2 q = N_\ell/2$. This means that $\tau = \delta + N_\ell/2$. Now the efficiency parameters of the system can be given as shown in the table. In particular the size of secret and public keys are $2(\tau + N_\ell)$ and $2N_\ell$ respectively.

In RSA-based FSS scheme [16], τ is the bundling degree and is defined as $\tau = \log_2 \phi(n)$, and security of the receiver is determined by the difficulty of factoring n (where $n = pq$ and p and q are both prime numbers). This means that $\tau \approx \log_2 n$. To design a system with security parameters (ℓ, δ), first N_ℓ, the modulus size that provides security level ℓ for the receiver is determined and then $K = \max(\delta, N_\ell)$. The modulus n is chosen such that $\log_2 n = K$. With this choice, the system provides adequate security for sender and receiver.

In the factorization scheme of [17], the security level of the sender is $\log_2 q$. Security level of the receiver is determined by the difficulty of discrete logarithm in Z_P^* and factorization of n. Firstly, N_ℓ which is the modulus size for which factorization has difficulty ℓ is chosen. Then, $K = \max(\frac{N_\ell}{2}, \sigma)$ is calculated. Since the size of P can be chosen much greater than $\log_2 n$, we use $\hat{K}$ to denote $\log_2 P$.

	vHP [18]	*Fact.[19]*	*RSA[16]*	*Fact.[17]*	*Our scheme*
Message Size	K	2^ρ	K	$2K$	r^2K
Length of Secret Key	$4K$	$4K + 2\delta$	$4K$	$4K$	$4rK$
Length of Public Key	$2\hat{K}$	$2K$	$2K$	$2\hat{K}$	$2r\hat{K}$
Length of Signature	$2K$	$2K + \delta$	$4K$	$2K$	$2rK$
Underlying Security Assumption	DL	Fact	Fact	Fact & DL	DL

Table 1. Efficiency Parameters Comparison

Note that as we have pointed out in Example 1, if $r = k = 1$, then our scheme coincides with vHP scheme.

Efficiency with respect to the message-length

We also need to consider the relative lengths of the message and the signature. If the length of the signature and the message are denoted by $|y|$ and $|x|$ respectively, then $\hat{\rho} = \frac{|y|}{|x|}$ is a measure of communication efficiency of the scheme.

As pointed out in Table 1, in vHP scheme messages are of length $\log_2 p$ and signatures are of length $2 \log_2 q$. This means that $\hat{\rho} = 2$ and so for every bit authenticated message, 2 bits of signature are required. In our scheme, messages and signatures are of size $r^2 \log p$ and $2r \log p$, respectively. Hence, $\hat{\rho} = \frac{2}{r}$ which is less than or equal to 2 (depending on r). In fact, our scheme is the only FSS that allows $\hat{\rho}$ to change with parameters of the system. Table 2 summarizes these results.

	vHP [18]	*Fact.[19]*	*RSA[16]*	*Fact.[17]*	*Our scheme*
$\hat{\rho}$	2	> 2	4	1	$2/r$

Table 2. Comparison of Communication Efficiency with respect to the Message-Length

Signing long messages
To sign long messages one can use (i) our proposed FSS, or (ii) hash-then-sign approach using one of the existing FSS. In the following we compare two schemes: one based on the construction presented in this paper, and the other one using vHP scheme and a provably secure hash function proposed in [2,3]. The hash function requires on average one multiplication for each bit of the message and the size of the hash value is equal to the size of the modulus. We assume that the length of the message is $r \times kK$ bits for some integers r and k such that $r \geq k$, $K = \log_2 p$ and $\hat{K} = \log_2 q$.

	Hash-then-sign approach*	Our Scheme
Sign (number of multiplications)	$\approx rkK + 2$	$2rk$
Test (number of multiplications)	$\leq 2K$	$\leq (2+k)rK$
Length of Secret Key	$4K$	$4kK$
Length of Public Key	$2\hat{K}$	$2r\hat{K}$
Length of Signature	$2K$	$2rK$
Underlying Security Assumption	Collision-Resistant Hash Function & DL	DL

Table 3. Complexity of the two FSS approaches for signing long messages.

Note: * Hashing uses a provably secure hash functions proposed in [2,3] and signing is by using vHP scheme [18].

The table shows that signing using the new construction is $K/2$ times faster, while verification is approximately $rk/2$ times slower. For example, to achieve adequate security [8], we choose $K = 151$ bits and $\hat{K} = 1881$ bits [8]. Also to simplify the comparison, we assume $r = k$. To sign a 1 Mega byte message using hash-then sign approach (i.e. using vHP scheme with a secure hash function proposed in [3]), the number of multiplications required for signing and testing are 1,065,458 and 302, respectively. However, by using the proposed approach, the number of multiplications required for signing and testing are 14,112 and 1,090,824, respectively. This asymmetry between the amount of computation required for signing and verification is useful in applications where signer has limited computing power, for example uses a smart card, and the verifier has a powerful server, for example is a bank.

F Conclusion

We introduced a new class of A-codes called linear A-codes which allow efficient signing (FSS) of long messages. An important property of the scheme is that $\hat{\rho}$ is a design parameter of the system and so by choosing it to be small, arbitrary length messages can be directly signed without the need for hashing before signing.

References

1. N. Barić and B. Pfitzmann. Collision-Free Accumulators and Fail-Stop Signature Schemes without Trees. *Advances in Cryptology - Eurocrypt '97, Lecture Notes in Computer Science 1233*, pages 480–494, 1997. 168
2. D. Chaum, E. van Heijst, and B. Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. *Interner Bericht, Fakultät für Informatik*, 1/91, 1990. 166, 167, 168, 175, 175
3. I. B. Damgård, *Collision free hash functions and public key signature scheme*, Lecture Notes in Computer Science 304, pages 203 - 216, 1988. 166, 167, 175, 175, 175
4. E. N. Gilbert, F. J. MacWilliams and N. J. A. Sloane. Codes which detect deception. *The Bell System Technical Journal*, Vol.33, No.3, pages 405-424, 1974. 170
5. T. Johansson. *Contributions to unconditionally secure authentication*, Ph.D. thesis, Lund, 1994. 172
6. T. Johansson. Authentication codes for nontrusting parties obtained from rank metric codes, *Designs, Codes and Cryptography*, 6:205-218, 1995. 172
7. L. Lamport. Constructing digital signatures from a one-way function. *PSRI International CSL-98*, 1979. 167
8. A.K. Lenstra, E.R. Verheul, Selecting Cryptographic Key Sizes, online: *http://www.cryptosavvy.com/*. Extended abstract appeared in Commercial Applications, *Price Waterhouse Coopers, CCE Quarterly Journals*, 3, pages 3 - 9, 1999. 167, 175, 175
9. T. P. Pedersen and B. Pfitzmann. Fail-stop signatures. *SIAM Journal on Computing*, 26/2:291–330, 1997. 165, 166, 166, 167, 167, 168, 169
10. B. Pfitzmann. Fail-stop signatures: Principles and applications. *Proc. Compsec '91, 8th world conference on computer security, audit and control*, pages 125–134, 1991. 167
11. B. Pfitzmann. *Digital Signature Schemes – General Framework and Fail-Stop Signatures.* Lecture Notes in Computer Science 1100, Springer-Verlag, 1996. 167, 168, 173
12. R. Safavi-Naini, S. Bakhtiari and C. Charnes. MRD Hashing. *Proceedings of Fast Software Encrytion Workshop, Lecture Notes in Computer Science 1372*, pages 134–149, 1998. 172
13. R. Safavi-Naini and W. Susilo. A General Construction for Fail-Stop Signature using Authentication Codes. *Workshop on Cryptography and Combinatorial Number Theory (CCNT '99)*, 2000 *(to appear).* 166, 167, 167, 170
14. R. Safavi-Naini and W. Susilo. Fail-Stop Threshold Signature Schemes based on Discrete Logarithm and Factorization. The Third International Workshop on Information Security, ISW 2000, 2000 *(to appear).* 166, 166
15. G. J. Simmons. Authentication theory/coding theory. *Advances in Cryptology–Crypto '84, Lecture Notes in Computer Science 196*, pages 411-431, 1984.
16. W. Susilo, R. Safavi-Naini, and J. Pieprzyk. RSA-based Fail-Stop Signature schemes. *International Workshop on Security (IWSec '99), IEEE Computer Society Press*, pages 161–166, 1999. 166, 167, 173, 174, 174, 175
17. W. Susilo, R. Safavi-Naini, M. Gysin, and J. Seberry. An Efficient Fail-Stop Signature Schemes. *The Computer Journal,* 2000 *(to appear).* 166, 166, 166, 167, 173, 174, 174, 175

18. E. van Heijst and T. Pedersen. How to make efficient fail-stop signatures. *Advances in Cryptology - Eurocrypt '92*, pages 337–346, 1992. 166, 166, 167, 168, 173, 174, 175, 175
19. E. van Heijst, T. Pedersen, and B. Pfitzmann. New constructions of fail-stop signatures and lower bounds. *Advances in Cryptology - Crypto '92, Lecture Notes in Computer Science 740*, pages 15–30, 1993. 166, 168, 173, 173, 174, 175
20. M. Waidner and B. Pfitzmann. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. *Advances in Cryptology - Eurocrypt '89, Lecture Notes in Computer Science 434*, 1990. 165, 167

Power Analysis Breaks Elliptic Curve Cryptosystems Even Secure against the Timing Attack

Katsuyuki Okeya[1] and Kouichi Sakurai[2]

[1] Hitachi, Ltd., Software Division,
5030, Totsuka-cho, Totsuka-ku, Yokohama, 244-8555, Japan
okeya_k@soft.hitachi.co.jp
[2] Kyushu University,
Department of Computer Science and Communication Engineering
6-10-1, Hakozaki, Higashi-ku, Fukuoka, 812-8581, Japan
sakurai@csce.kyushu-u.ac.jp

Abstract. We apply power analysis on known elliptic curve cryptosystems, and consider an exact implementation of scalar multiplication on elliptic curves for resisting against power attacks. Our proposed algorithm does not decrease the computational performance compared to the conventional scalar multiplication algorithm, whereas previous methods did cost the performance or fail to protect against power analysis attacks.

Keywords: *Elliptic Curve Cryptosystem, Power Analysis, Timing Attack, Montgomery-form, Efficient Implementation, Scalar Multiplication Algorithm*

A Introduction

A.1 Recent Works for Protecting against Power Attacks on ECC

The usage of Montgomery-like scalar multiplications on ECC While investigating efficient scalar multiplication algorithms on elliptic curves, some researchers [LD99,LH00,OKS00] have independently observed an advantage of Montgomery's method [Mon87] for preventing timing attacks.

- López,J. and Dahab,R. [LD99] presented a fast algorithm for computing scalar multiplications on elliptic curves defined over GF(2^n), which is based on Montgomery's method, and remarked that "Since the complexity of both versions of Algorithm 2 does not depend on the number of 1's (or 0's) in the binary representation of k, this may help to prevent timing attacks."
- Lim and Hwang [LH00] mentioned on Montgomery's method that "..., and the execution time does not depend on the Hamming weight of multipliers, which helps to prevent timing attacks."
- Okeya, Kurumatani, and Sakurai [OKS00] proposed elliptic curves cryptosystems from the Montgomery-form $BY^2 = X^3 + AX^2 + X$ secure against timing-attacks with the technique of randomized projective coordinates.

Coron's generalization of DPA to ECC and suggested Countermeasures Coron [Cor99] generalized DPA to elliptic curve cryptosystems, while the previous power analysis attacks are mainly against DES [DES1] or RSA [RSA78].[1] Moreover, Coron [Cor99] suggested three countermeasures against DPA on elliptic curve cryptosystems: (1) randomization of the private exponent, (2) blinding the point P, (3) randomized projective coordinates.

A.2 Our Contributions

We show that the previous scalar multiplication algorithms [LD99,LH00,OKS00] are vulnerable against power attacks (if we did a wrong implementation) by finding the input-dependency on the executing procedure of scalar multiplication.

We also discuss some weakness of Coron's countermeasures against DPA. This paper claims that the 1st countermeasure by Coron fails to break the secret-key's dependence on the executing procedure, and the Coron's 2nd countermeasure does not succeed in randomizing the expression of computed/computing objects. Though Coron's 3rd countermeasure looks perfect, we also clarify an implicit assumption in the implementation of Coron's designed multiplication algorithm secure against SPA, and remark that SPA could be applicable to this Coron's algorithm if we implemented it wrong.

Then we consider how to implement the scalar multiplication algorithm for resisting against power attacks. A hybrid method, an exact implementation of Montgomery algorithm with Coron's randomized technique, is shown to be secure against power attacks.

We further show that our proposed algorithm does not decrease the computational performance compared to the conventional scalar multiplication algorithm. Whereas previous methods did cost the performance and/or fail to protect against power analysis attacks. Note that if we carefully implement Coron's method in positive manner, power attacks could be prevented. However, Coron's method still costs the computational performance: even the fastest algorithm by Coron achieves less than the half as the performance of the conventional.

A.3 Applied Elliptic Curve Cryptosystems without the y-Coordinate

Our proposed algorithm considers only the x-coordinate of kP, k-time scalar multiplication of the point P on the elliptic curve. However, this is enough for applying to some elliptic curve cryptosystems including the encryption scheme ECES, the key-establishment scheme ECDH, and the signature generation ECDSA-S [IEEEp1363]. We should note that Ohgishi, Sakai and Kasahara [OSK99] developed an implementation of the verifying algorithm of ECDSA without referring the y-coordinate of kP. Thus, our algorithm can be applicable also to such a signature scheme.

[1] Some recent works on DPA are against AES candidates [DPA00,Mes00].

B Power Attacks

Power attacks arise from the actual implementation of the cryptosystems, which differ from the ideal cryptosystems. There are leakages of information in addition to input and output data while the cryptographic devices (e.g. smart card) execute cryptographic transactions (e.g. signature, encryption, decryption). An attacker may use the leakages for his estimate.

In 1996, Kocher proposed timing attack [Koc,Koc96]. Timing attack is one of the power attacks in which an attacker uses timing of execution for his estimate of the secret key. Recently, Kocher et al. proposed DPA (Differential Power Analysis) and SPA (Simple Power Analysis) [KJJ98,KJJ99]. DPA is a power attack in which an attacker uses power consumption and analyzes such data statistically, and SPA is an attack without statistical analysis. Coron [Cor99] generalized DPA to elliptic curve cryptosystems with the following SPA-immune scalar multiplication algorithm.[2]

```
Algorithm 1 : SPA-immune algorithm
  INPUT a scalar value $d$ and a point $P$.
  OUTPUT the scalar multiplication $dP$.
   1. $Q[0] \leftarrow P$
   2. For $i$ from $l-2$ to 0 do the following:    $(|d| = l)$
      2.1 $Q[0] \leftarrow 2Q[0]$
      2.2 $Q[1] \leftarrow Q[0] + P$
      2.3 $Q[0] \leftarrow Q[d_i]$
   3. Output $Q[0]$
```

C Power Analysis Attacks on Known ECC-Schemes

C.1 The Point of DPA Attack

In this subsection, for the purpose of constructing cryptosystems with immunity to DPA, we explain characteristics of DPA how an attacker estimates the secret key in the attack. The point of this attack is "a difference between executing procedures (non-symmetry)" and "an appearance of a predicted special value".

First, the executing procedure of typical cryptographic transaction depends on the secret key. Consequently, the executing procedure of cryptographic transaction differs from secret key to secret key. If an attacker finds the difference of the executing procedure from leakages, he is able to derive the information on the secret key. Actually, since it is hard to find the difference of executing procedure as it is, he treats it statistically and makes its bias big, and finally he finds the difference of the executing procedure.

Next, if an appearance of some specific value on the cryptographic transaction depends on the secret key, an attacker is able to detect the secret key by whether the value appears on the execution or not. That is, it occurs to typical

[2] We denote $|d|$ for the bit-length of the scalar value d and d_i for the i-th bit of d.

cryptographic transaction that a specific value appears in the middle of the computation for some secret key, the specific value does not appear for another secret key. If it is possible for the attacker to predict the specific value from known values beforehand, he is able to detect the secret key from the appearance of it. In Coron's DPA attack to elliptic curve cryptosystems, the predicted specific value is the coordinates of the point $4P$. The predicted specific value does not mean the logical value, but means the expression of the value. For example, $\frac{1}{2}$ and $\frac{2}{4}$ are same logically, but the expressions are different.

Therefore, for the purpose of resisting power analysis attack such as DPA, we have to do the next two things.

1. Break the secret-key's dependence and the executing procedure of cryptographic transaction.
2. Randomize the expression of computed/computing objects.

In the case of elliptic curve cryptosystems, the secret key is a scalar value, and the calculation of cryptographic transaction is the calculation of scalar multiplication. The case that the executing procedure of cryptographic transaction is invariant for any value of secret key is an example of no-dependence. Therefore, for resisting to DPA, the computing procedure of scalar multiplication should be invariant for any scalar value and should be with randomized expression.

C.2 Analysis of Montgomery-Variants

Attack on López-Dahab's method López,J. and Dahab,R. [LD99] presented a fast algorithm for computing scalar multiplications on elliptic curves defined over GF(2^n), which is based on Montgomery's method. Their method has two versions, one uses the affine-coordinates (referred to as LD2A) and the other uses the projective-coordinates (referred to as LD2P).

Algorithm LD2A has immunity against timing attacks. This is because the computation amount does not depend on the value k_i: executing the calculation of $x + t^2 + t$ and $x_j^2 + b/x_j^2$ ($j = 1$ or 2) for both $k_i = 0$ and for $k_i = 1$. However, we show that LD2A is vulnerable against DPA (even against SPA). In Step-4, LD2A first computes $x + t^2 + t$, then $x_2 + b/x_2^2$ in the case of $k_i = 1$, whereas LD2A first computes $x_1^2 + b/x_1^2$, then $x + t^2 + t$ in the case of $k_i = 0$. Therefore, the executing procedure of the calculation depends on the value k_i.[3]

Algorithm LD2P is immune against timing attacks. This is because, Step-4 of LD2P executes the computation of $Madd$, $Mdouble$ in both case of $k_i = 1$ and of $k_i = 0$.[4] Thus, LD2P does not depend on the value k_i, then it has no secret-key's

[3] The computed order of $x + t^2 + t$ and $x_j^2 + b/x_j^2$ is no problem because the order does not effect on the computation-result. Therefore, the problem can be removed by computing first $x + t^2 + t$, then $x_j^2 + b/x_j^2$, which does not depend on the value of k_i.

[4] We should note that $Madd$ must be computed before $Mdouble$. If we compute first $Mdouble$ before $Madd$, the value (X_2, Z_2) or (X_1, Z_1) could change, therefore the output of would be wrong.

dependency nor the executing procedure of cryptographic transaction. Hence, LD2P is immune against SPA as well. However, LD2P does not use the trick of randomizing the expression of computed/computing objects. Therefore, LD2P is vulnerable against DPA.

Attack on the Method by Okeya et al. Okeya, Kurumatani and Sakurai already remarked that the scalar multiplication algorithm on the Montgomery-form elliptic curves is immune against timing attack [OKS00]. However, The immunity against DPA depends on its implementation, that is, some implementation is fragile by DPA.

The computation of the scalar multiplication on the Montgomery-form elliptic curves requires the computations repeatedly that $(2mP, (2m+1)P)$ or $((2m+1)P, (2m+2)P)$ from $(mP, (m+1)P)$ depending on the value of each bit of the scalar value [Mon87]. Each computation requires one addition and one doubling on the elliptic curve. Since the addition and the doubling are independent, the order of the computation is flexible. Okeya et al. did not mention the importance of the order [OKS00]. For example, assume that we implement the algorithm such that "execute addition, and then execute doubling" if the bit is 0 and "execute doubling, and then execute addition" if it is 1. Then, an attacker is able to know the order of the computation, which the addition is first or the doubling is first, by DPA (or SPA) attack. Since the case that the addition is first is that the bit is 0 and the other case is that the bit is 1, he is able to derive the bit. Consequently, this implementation is fragile by DPA (and SPA).

C.3 Analysis of Coron's Countermeasures [Cor99]

Countermeasure 1 : randomization of the private exponent [5] This countermeasure has the following weakness. There exists some bias of d'_i which depends on the scalar value d for $i \geq n$, although this countermeasure uses random number k. Thus, an attacker is able to estimate d by statistical analysis for those values. That is, it does not satisfy 1st requirement.

We give an example of the attack which pricks the weakness. For simplicity, we assume $|k| = 2$, that is, the possibilities of k are $00, 01, 10$ and 11. For simplicity again, we assume that the lower three bits of $\#E$ are equal to 001.[6][7] Then, the possibilities of $d'_2 d'_1 d'_0$ are given by the following table.

$d_2d_1d_0$	000	001	010	011	100	101	110	111
$d'_2d'_1d'_0(k=00)$	000	001	010	011	100	101	110	111
$(k=01)$	001	010	011	100	101	110	111	000
$(k=10)$	010	011	100	101	110	111	000	001
$(k=11)$	011	100	101	110	111	000	001	010
rate of $d'_2 = 1$	0	0.25	0.5	0.75	1	0.75	0.5	0.25

[5] The operation of scalar multiplication inside each countermeasure is done by some usual scalar multiplication algorithm.

[6] It is applicable to any value.

[7] Since $\#E$ is one of the public parameters, the attacker knows its value.

Thus, an attacker is able to derive the information about d from that he detects the rate of $d'_2 = 1$ by statistical treatment of values of d'_2 since the rate varies in the value of d. For example, if the rate of $d'_2 = 1$ is 0.5, then he is able to estimate that $d_2d_1d_0 = 010$ or 110.

Since this countermeasure requires the operation of the scalar multiplication $d'P$ its inside, its security depends on the inside scalar multiplication algorithm. Coron assumed implicitly that the inside scalar multiplication algorithm of Countermeasure 1 resists SPA for resisting against DPA, because the SPA-immune algorithms satisfy 1st requirement. If it does not assume that, this countermeasure allows such an attack by checking the number or execution order of the additions and the doublings on the inside scalar multiplication, namely, the computation for $d'P$.

Next, it does not satisfy 2nd requirement, either, even if an SPA-immune algorithm is used. Because the distributions of the points which appear in the middle of the computation have biases, since the distribution of d'_i has a bias.

We may use Algorithm 1 as an SPA-immune scalar multiplication algorithm. Since $|d'|$ is equal to $|d|+20$, in general, the computation amount of Algorithm 1 is $(|d|+19)A+(|d|+19)D$, and that of Countermeasure 1 is almost same (besides the computation amount of random number generation and the computation of d' is required), where A and D are the computation amount of addition and doubling on the elliptic curve, respectively. If we assume $|k| = |d|$ for resisting the attack above, the computation amount is $(2|d| - 1)A + (2|d| - 1)D$.

Countermeasure 2 : blinding the point *P* Since Countermeasure 2 includes the scalar multiplication $d(R + P)$, the security of this countermeasure depends on the algorithm which computes $d(R + P)$. In this case, the same as Countermeasure 1, this countermeasure requires an SPA-immune algorithm. Otherwise, it is easy for an attacker to detect the scalar value d. All he has to do is to check the number or the execution order of elliptic additions and doublings on the inside scalar multiplication $d(R + P)$. The scalar value d which is used on the computation is invariant, whereas d' of Countermeasure 1 varies.

We may use Algorithm 1 as an SPA-immune scalar multiplication algorithm. For a scalar value of size l, the computation amount of Algorithm 1 is $(l-1)A+(l-1)D$. Besides, Countermeasure 2 requires the addition $R+P$, the subtraction $d(R+P)-Q$ and two doublings for refreshing R and S. The computation amount of Countermeasure 2 is $(|d| + 1)A + (|d| + 1)D$, since the computation amount of elliptic subtraction is almost same to that of elliptic addition.

On the randomization of this countermeasure, we set $R_0 = R$ and $S_0 = S$ for the first R and S, respectively, and set R_j and S_j for the R and S of j-times execution after, respectively. Then, $R_j = (-1)^\alpha 2^j R_0$ and $S_j = (-1)^\alpha 2^j S_0$ for $j \geq 1$, where $\alpha = 0$ or 1. Thus, once this scalar multiplication is done, the possibilities of R_j and S_j does not increase, that is, they remain two possibilities each. It is hard to say that it works well as a randomization. Therefore, this countermeasure does not satisfy 2nd requirement.

We present an attack to Countermeasure 2 with Algorithm 1. The countermeasure is vulnerable to this attack. Let $P, 2P, 4P, \cdots, 2^k P$ be points on the elliptic curve, and $C_j(t)$ be a function of power consumption associated with the execution of $d(2^j P)$. First, an attacker feeds these points to the cryptographic device which is equipped with Countermeasure 2. Then, he gets the functions $C_j(t)$ and calculates the correlation function $g(t)$ [8]. That is,

$$g(t) = \frac{1}{k}\sum_{j=0}^{k-1} \min\left\{\frac{1}{(C_j(t+t_0) - C_{j+1}(t))^2}, MAXVAL\right\},$$

where t_0 is the time required for each round [9] and $MAXVAL$ is some big number as a threshold in case the function take infinity. He distinguishes time t_1 such that $g(t_1)$ does not vanish, and calculates the number n_1 which satisfies $(n_1 - 1)t_0 < t_1 < n_1 t_0$. Then he finds $d_{l-1-n1} = 0$.

The following is the reason why he concludes the bit is true. First, we assume that three significant bits of the scalar value $d_{l-1}d_{l-2}d_{l-3}$ are equal to 100, namely, $d = (100 * * \cdots)_2$. In this case, first round at $(j+1)$-th execution is the exactly same computation as second round at j-th execution if the random-bit b is equal to 0. Thus, the functions of power consumption on the interval are (ideally) identical. On the other hand, the functions of power consumption for random-bit $b = 1$ are not identical. Since it happens with probability 1/2, $g(t)$ takes $(1/2)MAXVAL$ on the interval if k is sufficiently large.

Next, we assume that $d = (101 * * \cdots)_2$. According to Algorithm 1, the computation for the second round at j-th execution, namely, the calculation from $2 * 2^{j-1}(P+R)$ to $5 * 2^{j-1}(P+R)$, and the first round at $(j+1)$-th execution, namely, the calculation from $2^j(P+R)$ to $2 * 2^j(P+R)$, are done as follows :

$$\begin{array}{lccc}
j\text{-th execution :} & 2 * 2^{j-1}(P+R) & \longrightarrow & \text{compute } 4 * 2^{j-1}(P+R) \\
 & \longrightarrow \text{compute } 5 * 2^{j-1}(P+R) & \longrightarrow & \text{choose } 5 * 2^{j-1}(P+R) \\
(j+1)\text{-th execution :} & 2^j(P+R) & \longrightarrow & \text{compute } 2 * 2^j(P+R) \\
 & \longrightarrow \text{compute } 3 * 2^j(P+R) & \longrightarrow & \text{choose } 2 * 2^j(P+R)
\end{array}$$

Thus, the computing procedures are identical on the first half of these rounds. Hence, in this case, $g(t)$ takes $(1/2)MAXVAL$ on the first half of the interval.

Assume that $d = (110 * * \cdots)_2$. The computation for the second round at j-th execution, namely, the calculation from $3 * 2^{j-1}(P+R)$ to $6 * 2^{j-1}(P+R)$, and the first round at $(j+1)$-th execution, namely, the calculation from $2^j(P+R)$ to $3 * 2^j(P+R)$, are done as follows :

$$\begin{array}{lccc}
j\text{-th execution :} & 3 * 2^{j-1}(P+R) & \longrightarrow & \text{compute } 6 * 2^{j-1}(P+R) \\
 & \longrightarrow \text{compute } 7 * 2^{j-1}(P+R) & \longrightarrow & \text{choose } 6 * 2^{j-1}(P+R) \\
(j+1)\text{-th execution :} & 2^j(P+R) & \longrightarrow & \text{compute } 2 * 2^j(P+R) \\
 & \longrightarrow \text{compute } 3 * 2^j(P+R) & \longrightarrow & \text{choose } 3 * 2^j(P+R)
\end{array}$$

[8] The following $g'(t)$ might be better than $g(t)$: $g'(t) = \frac{1}{k}\sum_{j=0}^{k-1} exp\{C_j(t+t_0) - C_{j+1}(t)\}$.

[9] The time required for each round should be constant. Otherwise, the algorithm is subject to timing attacks.

Thus, the computing procedures are not identical on the last half (and the first half) of these rounds, although $6 * 2^{j-1}(P + R)$ is chosen on both cases. The correlation function $g(t)$ should tend to 0 for sufficiently large k in this case.

Finally, we assume that $d = (111 * * \cdots)_2$. The correlation function $g(t)$ should also tend to 0 in this case.

Therefore, a specific bit in the scalar value is detected being equal to 1 by vanishing of $g(t)$ or being equal to 0 by not vanishing of $g(t)$. Consequently, the attacker is able to detect any bit of the scalar value.

Remark 1. This attack stands on the circumstance that the attacker is able to bring some specific point on some execution in another execution. This is because that the number of the possibilities of R after many time execution is small and that the number of the candidates for next R is only two. Thus, in order to avoid the attack, the refreshing procedure of R and S on Countermeasure 2 should be modified to the following.

```
Algorithm 2 : algorithm to refresh $R$ and $S$
    INPUT a secret random point $R$ and its scalar multiplication $S = dR$.
    OUTPUT the refreshed $R$ and $S$.
    1. Generate a random number $k$ of size $n$ bits. (for example, $n = 20$)
    2. $R \leftarrow kR$ and $S \leftarrow kS$
    3. Output $R$ and $S$.
```

Here, computation kR and kS shall be implemented for being immunity against SPA such as Algorithm 1.

Countermeasure 3 : randomized projective coordinates Countermeasure 3 requires an SPA-immune algorithm for the inside scalar multiplication, the same as Countermeasure 2. This countermeasure satisfies 2nd requirement well, since most values in the middle of computation are the points on the elliptic curve, and with randomized expression. Besides the computation amount of Algorithm 1, Countermeasure 3 requires the computation amount of random number generation and three multiplications on the finite field for the randomization of the point P. Thus, the computation amount of Countermeasure 3 is $(|d| - 1)A + (|d| - 1)D + 3M + R$, where M and R are the computation amount of multiplication on the finite field and random number generation, respectively. This countermeasure is the fastest in the Coron's countermeasures, however, it achieves less than the half as the performance of the conventional, which the algorithm in Jacobian coordinates with window method [CMO98] or on the Montgomery-form elliptic curves [Mon87].

D Our Proposed Implementation

The scalar multiplication algorithm on the Montgomery-form elliptic curve needs one addition and one doubling on the elliptic curve, not depending on the specific bit whether it is 0 or it is 1, per bit of the scalar value [OKS00]. For the sake of invariant computing procedure for any scalar value, the execution order of addition and doubling should be fixed, not depending on the specific bit whether it is 0 or it is 1. That is, we should choose "execute addition,and then execute doubling" or "execute doubling, and then execute addition". Coron's Countermeasure 3 works well on 2nd requirement [Cor99] (and Okeya et al. mentioned it, too [OKS00]). Thus, we should adjust it for Montgomery-form.

A scalar multiplication algorithm with immunity to power analysis attack such as DPA using randomized projective coordinates and chosen "execute addition, and then execute doubling"[10] is the following.[11]

```
Algorithm 3 : fast scalar multiplication algorithm against DPA
    INPUT a scalar value $d$ and a point $P = (x, y)$.
    OUTPUT the scalar multiplication $dP$.
     1. Generate a random number $k$.
     2. Express the point $P = (kx, ky, k)$ using projective coordinates.
     3. $i \leftarrow |d| - 1$
     4. Calculate the point $2P$ from the point $P$.
     5. $m \leftarrow 1$
     6. If $i$ is equal to 0 then go to 15. Otherwise go to 7.
     7. $i \leftarrow i - 1$
     8. If $d_i$ is equal to 0 then go to 9. If it is equal to 1 then go to 12.
     9. Calculate the point $(2m + 1)P$. That is, add the point $mP$ and the
        point $(m + 1)P$.
    10. Calculate the point $2mP$. That is, double the point $mP$.
    11. Substitute $m$ for $2m$, and go to 6.
    12. Calculate the point $(2m + 1)P$. That is, add the point $mP$ and the
        point $(m + 1)P$.
    13. Calculate the point $(2m + 2)P$. That is, double the point $(m + 1)P$.
    14. Substitute $m$ for $2m + 1$, and go to 6.
    15. Output the point $mP$ as the scalar multiplication $dP$.
```

Here, "Substitute m for $2m$" (resp. "Substitute m for $2m + 1$") means that the pair of points $(mP, (m + 1)P)$ is substituted by the pair of points $(2mP, (2m + 1)P)$ (resp. $((2m + 1)P, (2m + 2)P)$) and m is substituted by $2m$ (resp. $2m + 1$). The operations on the elliptic curves should be done on the projective coordinates.

[10] We may choose "execute doubling, and then execute addition", and construct the scalar multiplication algorithm.

[11] The proposed scalar multiplication algorithm is applicable not only to Montgomery-form elliptic curves but also to general elliptic curves (including elliptic curves defined over a finite field with characteristic 2, and elliptic curves defined over an optimal extension field [BP98]) which have Montgomery-like scalar multiplication.

D.1 Analysis of the Security

We describe that our proposed scalar multiplication algorithm satisfies 1st and 2nd requirements in 3.1 in order to show that our proposed algorithm resists against power analysis attack such as DPA.

Requirement 1 : On our proposed algorithm, d_i is judged at Step 8, and the following calculation is divided by its value. Step9, Step 10 and Step 11 are executed in order and return to Step 6 if $d_i = 0$. Step 12, Step 13 and Step 14 are executed in order and return to Step 6 if $d_i = 1$. In any case, the addition on the elliptic curve is executed first, and the doubling on the elliptic curve is executed the next. Thus, the executing order of the operations on the elliptic curve in our algorithm is invariant. Therefore, the scalar value and the executing order have no dependence, and 1st requirement is satisfied.

Requirement 2 : On our proposed algorithm, random number k is generated at Step 1, and the point P is expressed in the projective coordinates using random number k at Step 2. The operations on the elliptic curve on the following steps are computed in the projective coordinates. Since the expression of the point P is randomized from the first, an attacker is able to predict a logical value in the middle of the computation but he is not able to predict its expression. Therefore, it is impossible for him to find such a value whose appearance specifies the scalar value, and 2nd requirement is satisfied.

E Analysis of the Efficiency

In this section, we estimate the computation amount of our proposed scalar multiplication algorithm, and show that no other algorithm which resists DPA is faster than ours, and our proposed algorithm is by no means inferior on the speed to other fast scalar multiplication algorithms [CMO98,Mon87].

E.1 Performance of Our Proposed Implementation

The bit d_i is judged at Step 8, then our proposed algorithm executes one addition and one doubling, not depending whether the bit is equal to 0 or it is equal to 1. The repeat time is $|d| - 1$. It requires one doubling at Step 4 and the computation amount of random number generation at Step 1. The expression on the projective coordinates at Step 2 requires one multiplication on the finite field since the following computation does not require the Y-coordinate. Thus, the computation amount of our proposed algorithm is $(|d|-1)A^M + |d|D^M + M + R$, where A^M and D^M are the computation amount of addition and doubling on the Montgomery-form elliptic curve, respectively. Besides, it requires one division on the finite field for output with Affine coordinates.

Addition in the Montgomery-form elliptic curve requires four multiplications and two squarings on the finite field [Mon87], since we may not assume $Z_1 = 1$ for the randomization of the point P at Step 2. Thus, the computation amount of our proposed algorithm is $(7|d| - 3)M + (4|d| - 2)S + R$ because of $A^M =$

$4M+2S, D^M = 3M+2S$. Assume that $S = 0.8M$. Then, it is $(10.2|d|-4.6)M+R$. The computation amount per bit is around $10.2M$.

E.2 Comparison to Other Algorithms

Coron's Countermeasure 3 The computation amount of Coron's Countermeasure 3 with Algorithm 1 is $(|d|-1)A+(|d|-1)D+3M+R$. The computation amount of addition and doubling on the Weierstrass-form elliptic curve with Affine coordinates is $A = 2M + S + I$ and $D = 2M + S + I$, respectively, where I is the computation amount of inversion on the finite field. Since I is estimated that $15M \leq I \leq 40M$, in general, the computation amount per bit is around $40M$.[12] That with projective coordinates is $A = 12M + 2S$ and $D = 7M + 5S$. The computation amount per bit is around $24.6M$. That with Jacobian coordinates is $A = 12M + 4S$ and $D = 4M + 6S$. The computation amount per bit is around $24M$. Therefore, the computation amount of our proposed algorithm is the smallest than that of these.

Some fast algorithms We compare our proposed algorithm to some of the fastest algorithms. The scalar multiplication algorithm with window methods in Jacobian coordinates is one of the fastest algorithms [CMO98]. The computation amount per bit is estimated around $10M$. The computation amount per bit of the algorithm on the Montgomery-form elliptic curve without randomized projective coordinates is around $9.2M$ [Mon87,OKS00]. Compared with the fastest algorithms, our proposed algorithm is by no means inferior on the speed since the computation amount per bit is $10.2M$.

Algorithm LD2A We consider the computational performance of Algorithm LD2A. M_2,S_2 and I_2 respectively denote $GF(2^n)$-field operations of multiplication, squaring, and inversion. Step-4 of Algorithm LD2A requires operations that (1)$M_2 + S_2 + I_2$ for computing t, (2)S_2 for computing $x + t^2 + t$, and (3)$M_2 + S_2 + I_2$ for computing $x_j^2 + b/x_j^2$. We may assume that the computation-time to $GF(2^n)$-field operations of squaring can be ignored. This can be done by cyclic shift with the normal basis. I_2, $GF(2^n)$-field operation of inversion requires (at least) 7 times of $GF(2^n)$-field operation of multiplication, if $n > 128$. Thus, the computational amount per bit of LD2A is estimated to be $16M_2$ [LD99].

Algorithm LD2P A_2, the computation amount of $Madd$ is $A_2 = 4M_2 + S_2$, and D_2, the computation amount of $Mdouble$ is $D_2 = 2M_2 + 4S_2$. With ignoring S_2, we estimate the computational amount per bit of LD2P to be $6M_2$ [LD99].

A table of our comparison The following table shows our comparison on the performance of each method.

[12] Moreover, the effect of randomized projective coordinates vanishes.

Table 1. Computation amount and immunity

Type	Comp. amount/bit	Timing Attack	SPA	DPA
[LD99] LD2A	16 M_2/bit	Immune	Dependent	Vulnerable
[LD99] LD2P	6 M_2/bit	Immune	Immune	Vulnerable
[OKS00] Mon	9.2 M/bit	Immune	Dependent	Vulnerable
[OKS00] Mon+RPC	10.2 M/bit	Immune	Dependent	Dependent
[Cor99] C3P	24.6 M/bit	Immune	Immune	Immune
[Cor99] C3J	24 M/bit	Immune	Immune	Immune
proposed algorithm	10.2 M/bit	Immune	Immune	Immune
[Mon87] Mon	9.2 M/bit	Immune	Dependent	Vulnerable
[CMO98] CMO	10 M/bit	Vulnerable	Vulnerable	Vulnerable

"Dependent" means that the immunity (or vulnerability) of the algorithm depends on the implementation.

References

BDL97. Boneh,D., Demillo,R.A., Lipton,J., *On the Importance of Checking Cryptographic Protocols for Faults*, Advances in Cryptology - EUROCRYPT '97, LNCS1233, (1997), 37-51.

BP98. Bailey,D.V., Paar,C., *Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms*, Advances in Cryptology - CRYPTO '98, LNCS1462, (1998), 472-485. 186

BSS99. Blake,I.F.,Seroussi,G.,Smart,N.P., *Elliptic Curves in Cryptography*, Cambridge University Press,(1999).

CJRR99. Chari,S., Jutla,C.S., Rao,J.R., Rohatgi,P., *Towards Sound Approaches to Counteract Power-Analysis Attacks*, Advances in Cryptology - CRYPTO '99, LNCS1666, (1999), 398-412.

CMO98. Cohen,H., Miyaji,A., Ono,T., *Efficient Elliptic Curve Exponentiation Using Mixed Coordinates*, Advances in Cryptology - ASIACRYPT '98, LNCS1514, (1998), 51-65. 185, 187, 188, 189

Cor99. Coron,J.S., *Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems*, Cryptographic Hardware and Embedded Systems (CHES'99), LNCS1717, (1999), 292-302. 179, 179, 180, 182, 186, 189, 189

DES1. National Bureau of Standards, *Data Encryption Standard*, Federal Information Processing Standards Publication 46 (FIPS PUB 46), (1977). 179

DPA00. Daemen,J., Peeters,M., Assche,G.V., Bitslice Ciphers and Power Analysis Attacks, Fast Software Encryption Workshop 2000 (FSE2000), (2000). 179

Enge99. A. Enge *Elliptic Curves and their applications to Cryptography*, Kluwer Academic publishers,(1999).

IEEEp1363. IEEE P1363 Standard Specifications for Public-Key Cryptography (1999), Available at http://grouper.ieee.org/groups/1363/ 179

Kob87. Koblitz,N., *Elliptic curve cryptosystems*, Math. Comp.48, (1987),203-209.

Koc. Kocher,C., *Cryptanalysis of Diffie-Hellman,RSA,DSS, and Other Systems Using Timing Attacks*, Available at http://www.cryptography.com/ 180

Koc96. Kocher,C., *Timing Attacks on Implementations of Diffie-Hellman, RSA,DSS, and Other Systems*, Advances in Cryptology - CRYPTO '96, LNCS1109, (1996), 104-113. 180

KJJ98. Kocher,C., Jaffe,J., Jun,B., *Introduction to Differential Power Analysis and Related Attacks*, Available at http://www.cryptography.com/dpa/technical/index.html 180

KJJ99. Kocher,C., Jaffe,J., Jun,B., *Differential Power Analysis*, Advances in Cryptology - CRYPTO '99, LNCS1666, (1999), 388-397. 180

Kur98. Kurumatani, H. *A Japanese patent announcement P2000-187438A* (In Japanese) Submitted in 22nd of Dec. (1998), available from http://www.jpo-miti.go.jp/home.htm

LD99. López,J., Dahab,R., *Fast Multiplication on Elliptic Curves over* $GF(2^m)$ *without Precomputation*, Cryptographic Hardware and Embedded Systems (CHES'99), LNCS1717, (1999), 316-327. 178, 178, 179, 181, 188, 188, 189, 189

LH00. Lim, C.H. and Hwang,H.S., *Fast implementation of Elliptic Curve Arithmetic in* $GF(p^m)$, Proc. PKC'00 LNCS1751, (2000), 405-421.

Mes00. Messerges,T.S., Securing the AES Finalists Against Power Analysis Attacks, Fast Software Encryption Workshop 2000 (FSE2000), (2000). 178, 178, 179 179

Mil86. Miller,V.S., *Use of elliptic curves in cryptography*, Advances in Cryptology - CRYPTO '85, LNCS218,(1986),417-426.

Mon87. Montgomery,P.L., *Speeding the Pollard and Elliptic Curve Methods of Factorizations*, Math. Comp. 48, (1987),243-264. 178, 182, 185, 187, 187, 188, 189

OKS00. Okeya,K., Kurumatani,H., Sakurai,K., *Elliptic Curves with the Montgomery-Form and Their Cryptographic Applications*, Public Key Cryptography (PKC2000), LNCS1751, (2000), 238-257. 178, 178, 179, 182, 182, 185, 186, 188, 189, 189

OSK99. Ohgishi,K., Sakai,R., Kasahara,M., *Elliptic Curve Signature Scheme with No y Coordinate*, Proc. SCIS'99,W4-1.3 (1999), 285-287. 179

RSA78. Rivest,R.L., Shamir,A., Adleman,L., *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, Vol.21, No.2, (1978), 120-126. 179

Van97. Vanstone,S.A., *Accelerated finite field operations on an elliptic curve*, GB patent, Application number GB9713138.7 (Date Lodged, 20.06.1997).

Efficient Construction of Cryptographically Strong Elliptic Curves

Johannes Buchmann and Harald Baier

University of Technology, Computer Science Department,
Alexanderstr. 10, 64283 Darmstadt, Germany,
{buchmann, hbaier}@cdc.informatik.tu-darmstadt.de

Abstract. We present a very efficient algorithm which given a negative integer Δ, $\Delta \equiv 1 \bmod 8$, Δ not divisible by 3, finds a prime number p and a cryptographically strong elliptic curve E over the prime field $\mathbb{F}_p$ whose endomorphism ring is the quadratic order $\mathcal{O}$ of discriminant Δ. Our algorithm bases on a variant of the complex multiplication method using Weber functions. We depict our very efficient method to find suitable primes for this method. Furthermore, we show that our algorithm is feasible in reasonable time even for orders $\mathcal{O}$ whose class number is in the range 200 up to 1000.

Keywords: class field theory, complex multiplication, cryptography, elliptic curve, factoring polynomials, finite field

A Introduction

Elliptic curve cryptography is a very efficient basic technology for public key infrastructures. An important computational problem in elliptic curve cryptography is the construction of cryptographically strong curves over finite prime fields. One approach is to randomly generate a curve, determine its cardinality by point counting, and then check whether it is cryptographically strong [MP97]. Unfortunately, point counting over large prime fields is rather slow.

Another method uses complex multiplication. It first searches for the cardinality of a cryptographically strong curve and it then constructs a curve of that cardinality. If the endomorphism ring of the curve has small class number, this is faster than searching for a strong curve by point counting. However, the German National Security Agency (BSI), for example, considers small class numbers of the endomorphism ring as a possible security risk (see [BSI00]) since there are only very few. The BSI recommends using curves whose endomorphism ring is an order in an imaginary quadratic number field of discriminant ≥ 200. For such curves, the standard complex multiplication method [AM93] is rather inefficient since the curve is constructed using a polynomial whose degree is the class number and which has extremely large coefficients.

In this paper we present a variant of the complex multiplication construction. The main theory of this variant can be found in [LZ94] or appendix A.13-A.14 of the IEEE P1363 standard [IEEE]. However, none of these contributions describe how to find suitable prime fields efficiently in the scope of generating cryptographically strong elliptic curves. We develop a very efficient algorithm to find a suitable prime field for this variant. In practice, all generated primes will be different.

As mentioned above, the main reproach to the standard complex multiplication method is that it is restricted to a small set of elliptic curves whose endomorphism ring has a small class number. The practical results of our implementation show that our algorithm is not restricted to this set of elliptic curves. For example, our algorithm constructs curves with endomorphism ring of class number 200 in 33 seconds on a SUN UltraSPARC-IIi. For class number 500 the algorithm requires less than two minutes, and for class number 1000 it takes about 7 minutes. The algorithm uses very little storage. Thus it can be used on any personal computer. This means that the users of a public key infrastructure can generate their own *unique* cryptographically strong elliptic curve.

Input of our algorithm is the discriminant Δ of an imaginary quadratic order $\mathcal{O}$. First, the algorithm searches for a prime number p which is the norm of an element of $\mathcal{O}$, and such that there exists a cryptographically strong curve over the prime field $\mathbb{F}_p$. We show how to make this search fast. Once p is found, the algorithm calculates a zero mod p of a polynomial suggested by Yui and Zagier [YZ97], also used in [LZ94] which has much smaller coefficients than the polynomial from [AM93]. Using that zero, it is easy to construct the cryptographically strong curve and a point of large prime order on that curve.

The paper is organized as follows: In Section B we define cryptographically strong elliptic curves over prime fields. In Section C we present our algorithm. Finally we give examples and running times in Section D.

B Cryptographically Strong Elliptic Curves

In this section we review a few basic facts concerning elliptic curves over finite fields and define cryptographically strong curves.

Let p be a prime number, $p > 3$. An *elliptic curve* over the prime field $\mathbb{F}_p$ of characteristic p is a pair $E = (a, b) \in \mathbb{F}_p^2$ with $4a^3 + 27b^2 \neq 0$. A *point* on E is a solution $(x, y) \in \mathbb{F}_p^2$ of $y^2 = x^3 + ax + b$ or the point at infinity O obtained by considering the projective closure of this equation. The set of points on E over $\mathbb{F}_p$ is denoted by $E(\mathbb{F}_p)$. It carries a group structure with the point at infinity acting as the identity element.

We call the elliptic curve E cryptographically strong if it satisfies the following conditions which make the cryptosystems, in which E is used, secure and efficient.

We first consider security. If the elliptic curve E is used in a cryptosystem, the security of this cryptosystem is based on the intractability of the discrete logarithm problem in the group of points $E(\mathbb{F}_p)$. Several discrete logarithm algorithms are known. To make their application impossible, we require that E satisfies the following conditions.

1. We have $|E(\mathbb{F}_p)| = c \cdot l$ with a prime $l > 2^{160}$ and a positive integer c.

2. The primes l and p are different.
3. The order of p in the multiplicative group $\mathbb{F}_l^\times$ of $\mathbb{F}_l$ is at least $\left\lceil \frac{2000}{\log_2(p)} \right\rceil$.

The first condition excludes the application of discrete logarithm algorithms whose running time is roughly the square root of the largest prime factor of the group order (see for example [vOM99]). The second condition makes the anomalous curve attack impossible (see [Sma99]). The last condition excludes the attack of Menezes, Okamoto, and Vanstone [MOV91] which reduces the discrete logarithm problem in $E(\mathbb{F}_p)$ to the discrete logarithm problem in a finite extension field of $\mathbb{F}_p$. The degree of this extension over $\mathbb{F}_p$ is at least the order of p in $\mathbb{F}_l^\times$. The third condition is based on the assumption that the discrete logarithm problem in a finite field, whose cardinality is a 2000-bit number, is intractable. We remark that the German National Security Agency [BSI00] requires the order of p in $\mathbb{F}_l^\times$ to be at least 10^4.

Let us now consider efficiency. Suppose that an elliptic curve E over a prime field $\mathbb{F}_p$ satisfies the security conditions. If this curve is used in a cryptosystem, the efficiency of this system depends on the efficiency of the arithmetic in $\mathbb{F}_p$. So p should be as small as possible. It follows from a theorem of Hasse that

$$(\sqrt{|E(\mathbb{F}_p)|} - 1)^2 \leq p \leq (\sqrt{|E(\mathbb{F}_p)|} + 1)^2 \ . \tag{1}$$

Hence, we try to make $|E(\mathbb{F}_p)|$ as small as possible. Now the first security condition implies

$$|E(\mathbb{F}_p)| = c \cdot l \tag{2}$$

with a prime number $l > 2^{160}$ and a positive integer c, the so called *cofactor*. The security of the cryptosystem, in which E is used, is based on the intractability of the discrete logarithm problem in the subgroup of order l in $E(\mathbb{F}_p)$. This security is independent of c. Therefore, c can be as small as possible. In our algorithm $c = 1$ is not possible. But we require the following condition which implies the first security condition.

4. We have $|E(\mathbb{F}_p)| = c \cdot l$ with a prime number $l > 2^{160}$ and a positive integer $c \leq 4$.

We say that an elliptic curve E over a finite prime field $\mathbb{F}_p$ is *cryptographically strong* if it satisfies the four conditions from above.

We explain an additional security condition required by the German National Security Agency BSI ([BSI00]). The third condition implies that the endomorphism ring $\mathrm{End}(E(\overline{\mathbb{F}}_p))$ of the elliptic curve over the algebraic closure of $\mathbb{F}_p$ is an imaginary quadratic order. The BSI requires the following.

5. The class number of the maximal order which contains $\mathrm{End}(E(\overline{\mathbb{F}}_p))$ is at least 200.

The reason for this condition is that among all curves over a prime field only very few have endomorphism rings with small class numbers. So those curves may be subject to specific attacks. However, no such attacks are known.

C The Algorithm

Let Δ be a negative integer, $\Delta \equiv 0, 1 \bmod 4$ and let $\mathcal{O}$ be the imaginary quadratic order of discriminant Δ. This order is denoted by $\mathcal{O}_\Delta$. Also, let p be a prime number which is the norm of an element in $\mathcal{O}$ that is, there are positive integers t, y such that

$$t^2 - \Delta y^2 = 4p \ . \tag{3}$$

Using complex multiplication, elliptic curves $E_{1,p}$ and $E_{2,p}$ over $\mathbb{F}_p$ with endomorphism ring $\mathcal{O}$ and

$$|E_{1,p}(\mathbb{F}_p)| = p + 1 - t, \quad |E_{2,p}(\mathbb{F}_p)| = p + 1 + t \tag{4}$$

can be constructed as follows (see [AM93], [BSS99]).

Let $H \in \mathbb{Z}[X]$ be the the minimal polynomial of $j(\frac{\Delta+\sqrt{\Delta}}{2})$ where $j = 12^3 J$ with Klein's modular function J (see [Apo90]). Modulo p the polynomial H splits into linear factors. Let j_p be a zero of H mod p that is, j_p is an integer such that $H(j_p) \equiv 0 \bmod p$. We assume $\Delta < -4$ in what follows. Then we have $j_p \notin \{0; 1728\}$. Let s_p be a quadratic nonresidue mod p. With

$$\kappa_p = \frac{j_p}{1728 - j_p}, \quad (a_p, b_p) = (3\kappa_p, 2\kappa_p) \tag{5}$$

we have

$$\{E_{1,p}, E_{2,p}\} = \{(a_p, b_p), (a_p s_p^2, b_p s_p^3)\}. \tag{6}$$

After this construction it is not known which of the curves is $E_{1,p}$ and which is $E_{2,p}$. However by choosing points on each curve and testing whether their order is a divisor of $p + 1 + t$ or $p + 1 - t$, the curves $E_{1,p}$ and $E_{2,p}$ can be identified.

We can decide whether one of the curves $E_{1,p}$ or $E_{2,p}$ is cryptographically strong before we actually construct those curves. We only need to know the prime number p and its representation (3). Then we know the orders of $E_{1,p}$ and $E_{2,p}$ from (4). Using those orders we can check whether one of the curves is cryptographically strong because the conditions from the previous section only depend on p and the order of the curve.

Our algorithm `cryptoCurve`(Δ, v, w) for computing cryptographically strong curves proceeds as follows. Input is a negative integer Δ with $\Delta \equiv 1 \bmod 8$, $\Delta \not\equiv 0 \bmod 3$, and positive integers v, w with $w > v > 2^{162}$. It uses the algorithm `findPrime`(Δ, v, w) explained below which determines a prime number $p \in [v, w]$ with a representation (3) such that $E_{1,p}$ or $E_{2,p}$ is cryptographically strong. It also returns the prime number l from (2) with $l > 2^{160}$. Once p is found, the algorithm `findCurve`(Δ, p, l) constructs $E_{1,p}$ and $E_{2,p}$ and returns one of those curves which is cryptographically strong. `findCurve` also returns a point P of order l on the curve. We only use discriminants $\Delta \equiv 1 \bmod 8$, $\Delta \not\equiv 0 \bmod 3$ because for those discriminants `findCurve` can be implemented very efficiently.

```
cryptoCurve($\Delta, v, w$)
Input: $\Delta \in \mathbb{Z}_{<0}$, $\Delta \equiv 1 \bmod 8$, $\Delta \not\equiv 0 \bmod 3$, $v, w \in \mathbb{Z}$, $w > v > 2^{162}$.
Output: A prime $p \in [v, w]$.
  A cryptographically strong elliptic curve $E$ over $\mathbb{F}_p$ with endomorphism ring $\mathcal{O}_\Delta$.
  A prime divisor $l$ of $|E(\mathbb{F}_p)|$ with $l > 2^{160}$ such that $|E(\mathbb{F}_p)| = 4l$.
  A point $P$ in $E(\mathbb{F}_p)$ of order $l$.

  $(p, l) \leftarrow$ findPrime($\Delta, v, w$);
  $(E, P) \leftarrow$ findCurve($\Delta, p, l$);
  return $(p, E, l, P)$;
```

By Lemma 1, the order of a curve with endomorphism ring $\mathcal{O}_\Delta$ is then divisible by 4.

In order for the algorithm findPrime(Δ, v, w) to be successful it is necessary that the interval $[v, w]$ is sufficiently large. We use intervals of length 2^{122}.

To satisfy condition 5 of the previous section, we choose Δ such that $\mathcal{O}_\Delta$ is a maximal order and the class number of $\mathcal{O}_\Delta$ is at least 200. Also, if we want to generate many different curves for a fixed Δ, then we choose many pairwise disjoint intervals $[v, w]$.

Next, we explain our algorithm findPrime(Δ, v, w). So let Δ, v, w be as defined in cryptoCurve(Δ, v, w). We have to find positive integers t and y such that $p = (t^2 - \Delta y^2)/4$ is a prime number, $p \in [v, w]$, and $(p + 1 - t)/4$ or $(p + 1 + t)/4$ is a prime $l > 2^{160}$. As the next Lemma shows, those requirements imply congruence conditions for t and y.

Lemma 1. *If $\Delta \equiv 1 \bmod 8$ and t, y are positive integers such that $p = (t^2 - \Delta y^2)/4$ is a prime, then $(t \bmod 4, y \bmod 4) \in \{(0, 2), (2, 0)\}$ and $p + 1 + t \equiv p + 1 - t \equiv 0 \bmod 4$. Also, if $(p + 1 + t)/4$ is prime, then $(t \bmod 8, y \bmod 8) \in \{(2, 0), (6, 4)\}$ and if $(p + 1 - t)/4$ is prime, then $(t \bmod 8, y \bmod 8) \in \{(2, 4), (6, 0)\}$.*

Proof. Since Δ is odd, we have that $t^2 - \Delta y^2 \equiv 0 \bmod 4$ if and only if $t \equiv y \equiv 0 \bmod 2$ or $t \equiv y \equiv 1 \bmod 2$. Let t and y both be odd. Then $t^2 \equiv y^2 \equiv 1 \bmod 8$ and $t^2 - \Delta y^2 \equiv 1 - 1 \cdot 1 \equiv 0 \bmod 8$. Therefore, $(t^2 - \Delta y^2)/4$ is even. Now let t and y both be even. If $t \equiv y \equiv 0 \bmod 4$, then $t^2 - \Delta y^2 \equiv 0 \bmod 16$, and again $(t^2 - \Delta y^2)/4$ is even. The same is true if $t \equiv y \equiv 2 \bmod 4$ since we have $t^2 \equiv y^2 \equiv 4 \bmod 8$ and $t^2 - \Delta y^2 \equiv 0 \bmod 8$. A necessary condition for $(t^2 - \Delta y^2)/4$ to be an odd prime is therefore $(t \bmod 4, y \bmod 4) \in \{(0, 2), (2, 0)\}$. Then it is easy to see that $p+1+t \equiv p+1-t \equiv 0 \bmod 4$. As both t and y are even, there are integers t_0, y_0 with $t = 2t_0$ and $y = 2y_0$. Let $(t \bmod 4, y \bmod 4) = (0, 2)$. Then we have $t_0 = 2t_1$ with an integer t_1 and $y_0^2 \equiv 1 \bmod 8$. It follows that $p+1 \pm t = t_0^2 - \Delta y_0^2 + 1 \pm 2t_0 \equiv 4t_1^2 - 1 + 1 \pm 4t_1 = 4t_1(t_1 \pm 1) \equiv 0 \bmod 8$. Hence, neither $(p+1+t)/4$ nor $(p+1-t)/4$ can be prime. Now let $(t \bmod 4, y \bmod 4) = (2, 0)$. Furthermore, assume that $y_0 \equiv 0 \bmod 4$. Then we have $y_0^2 \equiv 0 \bmod 8$ and $p + 1 \pm t = t_0^2 - \Delta y_0^2 + 1 \pm 2t_0 \equiv 1 - 1 \cdot 0 + 1 \pm 2t_0 = 2(1 \pm t_0) \bmod 8$. If $t_0 \equiv 1 \bmod 4$, that is $(t \bmod 8, y \bmod 8) = (2, 0)$, we get $p+1+t \equiv 4 \bmod 8$ and $p + 1 - t \equiv 0 \bmod 8$. Hence, only $(p + 1 + t)/4$ can be prime. If $t_0 \equiv 3 \bmod 4$, that is $(t \bmod 8, y \bmod 8) = (6, 0)$, we get $p + 1 + t \equiv 0 \bmod 8$ and $p + 1 - t \equiv 4 \bmod 8$. Hence, only $(p+1-t)/4$ can be prime. The case $y_0 \equiv 2 \bmod 4$ is treated analogously. □

It follows from Lemma 1 that the order of a cryptographically strong elliptic curve E over $\mathbb{F}_p$ is $4l$ with a prime $l > 2^{160}$. In the algorithm `findPrime` we first initialize y to a random integer in $[1, \lfloor\sqrt{4v/|\Delta|}\rfloor]$ which is 0 mod 8. We restrict to $y \equiv 0 \bmod 8$ to avoid distinguishing several cases. For all possible $t \equiv 2 \bmod 4$ we try to find a cryptographically strong curve. By Lemma 1 the number $(p+1+t)/4$ can only be prime if $t \equiv 2 \bmod 8$. If this is true, we test whether $p + 1 + t$ is the order of a cryptographically strong curve by using the function `isStrong`. If the order happens to be cryptographically strong, this function returns the prime l such that $p + 1 + t = 4l$. Otherwise, the function returns 0. If $t \equiv 6 \bmod 8$, then we test whether $p + 1 - t$ is the order of a cryptographically strong curve. If no cryptographically strong curve is found, then the algorithm chooses a new y and repeats the above procedure. Since y is chosen randomly, the algorithm returns a different curve each time it is called.

```
findPrime($\Delta, v, w$)
Input: $\Delta \in \mathbb{Z}_{<0}$, $\Delta \equiv 1 \bmod 8$, $\Delta \not\equiv 0 \bmod 3$, $v, w \in \mathbb{Z}$, $w > v > 2^{162}$.
Output: Primes $p \in [v, w]$ and $l > 2^{160}$
  such that $|E(\mathbb{F}_p)| = 4l$ for a cryptographically strong curve $E$ over $\mathbb{F}_p$
  with endomorphism ring $\mathcal{O}_\Delta$.

  while true do
    Choose a random $y$ in $[1, \lfloor\sqrt{4v/|\Delta|}\rfloor]$ with $y \equiv 0 \bmod 8$;
    $t \leftarrow 4\lceil(\sqrt{4v - |\Delta|y^2} - 2)/4\rceil + 2$;
    while $p \leftarrow (t^2 + |\Delta|y^2)/4 \le w$ do
      if $p$ is prime then
        if $t \equiv 2 \bmod 8$ and $l \leftarrow$ isStrong($p, p + 1 + t$) $\neq 0$ then
          return $(p, l)$;
        else if $l \leftarrow$ isStrong($p, p + 1 - t$) $\neq 0$ then
          return $(p, l)$;
        end if
      end if
      $t \leftarrow t + 4$;
    end while
  end while
```

If the interval $[v, w]$ used in `findPrime` is too small then `findPrime` may not terminate. According to our experiments, intervals of length 2^{122} seem to be sufficiently large.

The function `isStrong` decides whether a curve, whose order is determined in `findPrime`, is cryptographically strong. The order of p in the multiplicative group $\mathbb{F}_l^\times$ is denoted by $\mathrm{ord}_{\mathbb{F}_l^\times}(p)$. The algorithm implements the conditions from Section B.

```
isStrong($p, N$)
Input: A prime $p$ and $N \in \mathbb{N}$, $N \equiv 0 \bmod 4$.
Output: A prime $l$ with $N = 4l$ if $N$ is the order of a cryptographically strong elliptic
  curve over $\mathbb{F}_p$,
  $l = 0$ otherwise.

  if $l \leftarrow N/4$ is prime and $l > 2^{160}$ and $p \neq l$ and $\mathrm{ord}_{\mathbb{F}_l^\times}(p) \geq \left\lceil \frac{2000}{\log_2(p)} \right\rceil$ then
    return $l$;
  else
    return 0;
  end if
```

The function `findCurve` implements the construction of the curves as explained in (5) and (6).

```
findCurve($\Delta, p, l$)
Input: $\Delta \in \mathbb{Z}_{<0}$, $\Delta \equiv 1 \bmod 8$, $\Delta \not\equiv 0 \bmod 3$. A prime $p$ such that there exists a
  curve of order $4l$ over $\mathbb{F}_p$.
Output: An elliptic curve $E$ over $\mathbb{F}_p$ with $|E(\mathbb{F}_p)| = 4l$ and endomorphism ring $\mathcal{O}_\Delta$.
  A point $P$ in $E(\mathbb{F}_p)$ of order $l$.

  $j_p \leftarrow$ findRoot($\Delta, p$);
  Select a quadratic nonresidue $s_p$ mod $p$;
  $E_1 \leftarrow (a_p, b_p)$;
  $E_2 \leftarrow (a_p s_p^2, b_p s_p^3)$;
  while true do
    Choose $Q_1$ randomly in $(E_1(\mathbb{F}_p)) \setminus \{O\}$;
    Choose $Q_2$ randomly in $(E_2(\mathbb{F}_p)) \setminus \{O\}$;
    if $4Q_1 \neq O$ and $4lQ_1 = O$ then
      return $(E_1, 4Q_1)$;
    else if $4Q_2 \neq O$ and $4lQ_2 = O$ then
      return $(E_2, 4Q_2)$;
    end if
  end while
```

The function `findRoot`(Δ, p) determines a zero mod p of the polynomial H from above. If the class number of $\mathcal{O}_\Delta$ is large then the coefficients of H are extremely large. Instead, following [YZ97], we use a polynomial W of the same degree which is computed by `findPolynomial(`Δ`)`. This polynomial has much smaller coefficients. For example, if $\Delta = -71$ then

$$\begin{aligned}H = {} & X^7 + 313645809715X^6 - 3091990138604570X^5 + \\ & 98394038810047812049302X^4 - 823534263439730779968091389X^3 + \\ & 5138800366453976780323726329446X^2 - \\ & 425319473946139603274605151187659X + \\ & 737707086760731113357714241006081263\end{aligned}$$

and

$$W = X^7 - X^6 - X^5 + X^4 - X^3 - X^2 + 2X + 1 \ .$$

Also, if w_p is a zero of W mod p then

$$j_p = (w_p^{24} - 16)^3 / w_p^{24}$$

is a zero of H mod p.

```
findRoot($\Delta, p$)
Input: $\Delta \in \mathbb{Z}_{<0}$, $\Delta \equiv 1 \bmod 8$, $\Delta \not\equiv 0 \bmod 3$.
  A prime $p$ which is the norm of an element of $\mathcal{O}_\Delta$.
Output: A root $j_p$ of $H$ mod $p$.

  $W \leftarrow$ findPolynomial($\Delta$);
  $w_p \leftarrow$ find_root($p, W$);
  return $j_p \leftarrow (w_p^{24} - 16)^3 / w_p^{24}$;
```

Next, we explain `findPolynomial`(Δ). In this algorithm, we first compute the set R_Δ of reduced integral primitive forms $(a, b, c) = ax^2 + bxy + cy^2$ of discriminant Δ. An algorithm for computing this set can be found in [Coh95], p.228. We remark that $|R_\Delta|$ is the class number of the quadratic order of discriminant Δ. Using R_Δ we define the polynomial W. For τ in the upper half plane of $\mathbb{C}$ we set $q_\tau = e^{2\pi i \tau}$. The Weber functions (see [Web02]) are

$$\mathfrak{f}(\tau) = q_\tau^{-\frac{1}{48}} \prod_{n=1}^{\infty} (1 + q_\tau^{n-\frac{1}{2}}) \ , \tag{7}$$

$$\mathfrak{f}_1(\tau) = q_\tau^{-\frac{1}{48}} \prod_{n=1}^{\infty} (1 - q_\tau^{n-\frac{1}{2}}) \ , \tag{8}$$

$$\mathfrak{f}_2(\tau) = \sqrt{2} q_\tau^{\frac{1}{24}} \prod_{n=1}^{\infty} (1 + q_\tau^{n}) \ . \tag{9}$$

For $g = (a, b, c) \in R_\Delta$ set $\tau_g = \frac{-b + i\sqrt{|\Delta|}}{2a}$. Then τ_g lies in the upper half plane of $\mathbb{C}$. We set

$$f(g) := \begin{cases} \zeta^{b(a-c-ac^2)} \cdot \mathfrak{f}(\tau_g) & \text{if} \quad 2 \mid a, \ 2 \mid c \ , \\ (-1)^{\frac{\Delta-1}{8}} \cdot \zeta^{b(a-c-ac^2)} \cdot \mathfrak{f}_1(\tau_g) & \text{if} \quad 2 \mid a, \ 2 \nmid c \ , \\ (-1)^{\frac{\Delta-1}{8}} \cdot \zeta^{b(a-c+a^2c)} \cdot \mathfrak{f}_2(\tau_g) & \text{if} \quad 2 \nmid a, \ 2 \mid c \end{cases} \tag{10}$$

with $\zeta = e^{2\pi i/48}$. Then the polynomial W is

$$W = \prod_{g \in R_\Delta} (X - f(g)) \ . \tag{11}$$

This polynomial has integer coefficients. In `findPolynomial`(Δ), we compute W using (11). The zeros $f(g)$ of W, $g \in R_\Delta$ are defined as infinite products. We approximate them by truncated products using the LiDIA type `bigcomplex` with precision (as in [LZ94])

$$F = \frac{G + \frac{h}{4} + 5}{47} + 1$$

where

$$G = \frac{\pi\sqrt{|\Delta|}}{\log 10} \cdot \sum_{(a,b,c)\in R_\Delta} \frac{1}{a} \ .$$

More precisely, we keep multiplying the factors of $f(g)$ in (7), (8), or (9) until five successive partial products are equal within the chosen precision. The following observation can be used to speed up the algorithm. If $(a, b, c) \in R_\Delta$ then $f(a, -b, c) = \overline{f(a, b, c)}$. Hence, if $(a, -b, c)$ also belongs to R_Δ, then we obtain $f(a, -b, c)$ almost for free. By multiplying all linear factors $X - f(g)$ using our truncated products $f(g)$, we obtain a polynomial whose coefficients are close to integers. We determine W by rounding each coefficient to the closest integer.

To compute a zero w_p of $W \bmod p$ we use the LiDIA-function `find_root`(p, W). As input this function requires a prime p and a polynomial $W \in \mathbb{Z}[X]$ which splits into linear factors modulo p. It returns a zero of $W \bmod p$. `find_root` uses the Cantor-Zassenhaus split (see [Coh95]) and a polynomial arithmetic due to Shoup [Sho95].

D Examples and Running Times

We implemented our algorithm in C++ using the library LiDIA 2.0 ([LiDIA]) and the GNU compiler 2.95.2 . The timings were measured on a SUN UltraSPARC-IIi running Solaris 2.6 at 333 MHz and having 512 MB of main memory.

We present three examples. The class numbers in these examples are at least 200 to meet condition 5 of Sect. B. Timings - even for class numbers less than 200 - can be found in Tables 1, 2 and 3.

Let us first consider a fundamental discriminant of class number 200. We take $\Delta = -21311$ as Δ is maximal with this property. We use the floating point precision $F = 48$. Running times can be found in Table 2. We get the following output:

$$\begin{aligned}
p &= 1124388104722008836025816920368001511855982082134 9\\
\lfloor \log_2 p \rfloor + 1 &= 163\\
l &= 281097026180502209006454258315765394457133891369 1\\
\lfloor \log_2 l \rfloor + 1 &= 161\\
E &= (9664333206544914885970277381389792397205945721715,\\
&9712273578800196724738364136255277465294004742516)\\
|E(\mathbb{F}_p)| &= 4 \cdot l\\
P &= (9734858823530657227741101030084051747415204761009,\\
&9869588797805709934476211640090148068685)
\end{aligned}$$

In the second example we use the fundamental discriminant $\Delta = -96599$ with $h = 500$. Again, Δ is maximal with this property. We use $F = 129$. We obtain the following output:

$$\begin{aligned}
p &= 11239364022024445602395071076291409161418004790449\\
\lfloor \log_2 p \rfloor + 1 &= 163\\
l &= 2809841005506111400598767640310076774940711944009\\
\lfloor \log_2 l \rfloor + 1 &= 161\\
E &= (3056338225511409442512995201958098190832410825281,\\
&9530468165023903363272044185499671568166943743820)\\
|E(\mathbb{F}_p)| &= 4 \cdot l\\
P &= (7706333343307924406946634579911418698936319570270,\\
&4046128002100252856695354801672179678 6860)
\end{aligned}$$

To show the limit of the algorithm we also present an example with the fundamental discriminant $\Delta = -10000031$ and $h = 5426$. We use the floating point precision $F = 1986$. We get the following output:

$$\begin{aligned}
p &= 11575224349584783304586589602930035397585956397089\\
\lfloor \log_2 p \rfloor + 1 &= 163\\
l &= 2893806087396195826146647734591894354891848869089\\
\lfloor \log_2 l \rfloor + 1 &= 161\\
E &= (9758151861204570239950038963540169300959330538 8,\\
&6505434574136380159966692642360112867306220359 2)\\
|E(\mathbb{F}_p)| &= 4 \cdot l\\
P &= (4475160835035949933240439608822439678768106791477,\\
&6992349469932163330795423460823116723197169323796)
\end{aligned}$$

The total running time was 1 day 13 hour 41 min 39 sec. The time to compute W and w_p was 1 day 13 hour 21 min 56 sec and 19 min 13 sec, respectively.

Table 1. Timings in seconds for fundamental discriminants of class numbers 20 up to 150. Each discriminant is maximal for a given class number. Each first row contains the running time, each second row the standard deviation.

Δ	h	Number	Time of	Time of	Time of	Time of
		of tests	`cryptoCurve`	`findPrime`	`findPolynomial`	`find_root`
-455	20	100	6.235	5.275	0.05463	0.6242
			5.172	5.166	0.03266	0.05251
-1271	40	100	8.278	5.871	0.1245	2.004
			5.345	5.347	0.04943	0.1157
-2159	60	100	10.03	5.079	0.2492	4.421
			4.901	4.869	0.08125	0.2091
-5183	80	100	15.08	6.238	0.3778	8.174
			7.269	7.213	0.1382	0.3276
-7991	100	100	17.02	5.823	0.6289	10.28
			5.806	5.805	0.2033	0.4606
-11879	150	100	27.52	6.418	1.272	19.46
			5.665	5.654	1932	0.7965

Table 2. Timings in seconds for some fundamental discriminants of class number 200. Each first row contains the running time, each second row the standard deviation.

Δ	h	Number	Time of	Time of	Time of	Time of
		of tests	`cryptoCurve`	`findPrime`	`findPolynomial`	`find_root`
-21311	200	100	33.06	5.808	2.420	24.52
			5.463	5.458	0.009839	0.5377
-31031	200	100	31.98	4.668	2.739	24.26
			4.752	4.770	0.003942	0.5232
-40895	200	100	33.70	6.456	2.720	24.20
			5.793	5.710	0.004976	0.5489
-50183	200	100	33.89	6.198	3.042	24.33
			6.166	6.111	0.008005	0.5736
-100015	200	100	33.45	5.791	3.033	24.29
			4.971	4.888	0.007614	0.5901
-500047	200	100	33.77	4.540	4.368	24.43
			3.831	3.838	0.01136	0.5162

Table 3. Timings in seconds for fundamental discriminants of class numbers 500 up to 1000. Each discriminant is maximal for a given class number. Each first row contains the running time, each second row the standard deviation.

Δ	h	Number	Time of	Time of	Time of	Time of
		of tests	`cryptoCurve`	`findPrime`	`findPolynomial`	`find_root`
-96599	500	25	101.2	8.146	29.30	63.36
			7.107	6.782	0.1116	1.201
-148511	600	25	152.9	5.846	49.03	97.56
			5.948	4.450	0.2956	2.642
-185471	700	25	197.8	7.426	82.29	107.7
			9.068	8.576	0.5049	1.572
-233999	800	25	243.2	4.862	120.6	117.2
			4.613	4.259	0.4135	1.034
-299519	900	25	320.5	4.702	188.8	126.5
			5.258	5.486	1.141	1.684
-412079	1000	25	430.0	6.282	283.1	140.0
			6.638	6.708	0.3415	1.580

Acknowledgments: We gratefully thank Volker Müller and Sachar Paulus for their former work on the complex multiplication method.

References

AM93. A.O.L. Atkin and F. Morain. Elliptic curves and primality proving. *Mathematics of Computation*, 61:29–67, 1993. 191, 192, 194

Apo90. T.M. Apostol. *Modular Functions and Dirichlet Series in Number Theory.* Springer-Verlag, 1990. 194

BSI00. Geeignete Kryptoalgorithmen gemäß §17(2) SigV, April 2000. Bundesamt für Sicherheit in der Informationstechnik. 191, 193, 193

BSS99. I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography.* Cambridge University Press, 1999. 194

Coh95. H. Cohen. *A Course in Computational Algebraic Number Theory.* Springer-Verlag, 1995. 198, 199

IEEE. IEEE. P1363 Standard Specifications For Public-Key Cryptography. available via WWW from http://grouper.ieee.org/groups/1363/. 191

LiDIA. LiDIA. *A library for computational number theory.* Technical University of Darmstadt. available via WWW from http://www.informatik.tu-darmstadt.de/TI/LiDIA/Welcome.html. 199

LZ94. G.-J. Lay and H.G. Zimmer. Constructing elliptic curves with given group order over large finite fields. In *Proceedings of ANTS I*, LNCS 877, pages 250–263, 1994. 191, 192, 199

MOV91. A. Menezes, T. Okamoto, and S. Vanstone. Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field. In *Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing*, pages 80–89, 1991. 193

MP97. V. Müller and S. Paulus. On the Generation of Cryptographically Strong Elliptic Curves. Technical Report, Technical University of Darmstadt, 1997. 191

Sho95. V. Shoup. A new polynomial factorization algorithm and its implementation. *Journal of Symbolic Computation*, 20:363–397, 1995. 199

Sma99. N.P. Smart. The Discrete Logarithm Problem on Elliptic Curves of Trace One. *Journal of Cryptology*, 12/3:193–196, 1999. 193

vOM99. P.C. van Oorschot and M.J.Wiener. Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology*, 12/1:1–28, 1999. 193

Web02. H. Weber. *Lehrbuch der Algebra III.* Chelsea Publishing Company, 1902. 198

YZ97. N. Yui and D. Zagier. On the singular values of Weber modular functions. *Mathematics of Computation*, 66:1645–1662, 1997. 192, 197

High-Speed Software Multiplication in $\mathbb{F}_{2^m}$

Julio López[1]* and Ricardo Dahab[2]**

[1] Dept. of Computer Science, University of Valle, Colombia
`jlopez@univalle.edu.co`
[2] Institute of Computing, State University of Campinas, Brazil
`rdahab@dcc.unicamp.br`.

Abstract. In this paper we describe an efficient algorithm for multiplication in $\mathbb{F}_{2^m}$, where the field elements of $\mathbb{F}_{2^m}$ are represented in standard polynomial basis. The proposed algorithm can be used in practical software implementations of elliptic curve cryptography. Our timing results, on several platforms, show that the new method is significantly faster than the "shift-and-add" method.

Key words. Multiplication in $\mathbb{F}_{2^m}$, Polynomial Basis, Elliptic Curve Cryptography.

A Introduction

Efficient algorithms for multiplication in $\mathbb{F}_{2^m}$ are required to implement cryptosystems such as the Diffie-Hellman and elliptic curve cryptosystems defined over $\mathbb{F}_{2^m}$. Efficient implementation of the field arithmetic in $\mathbb{F}_{2^m}$ depends greatly on the particular basis used for the finite field. Two common choices of bases for $\mathbb{F}_{2^m}$ are normal and polynomial. Normal bases seem more suitable for hardware implementations (see [1]).

In this paper we describe a technique for multiplication in the finite field $\mathbb{F}_{2^m}$, where the field elements are represented as binary polynomials modulo an irreducible binary polynomial of degree m. The proposed method is about 2-5 times faster than the standard multiplication, and is particularly useful for software implementation of elliptic curve cryptosystems over $\mathbb{F}_{2^m}$. It is based on the observation that Lim/Lee's method [6] (or comb method [8]), designed for exponentiation, can be modified to work in $\mathbb{F}_{2^m}$.

The remainder of this paper is organized as follows. In Section B we describe the finite field $\mathbb{F}_{2^m}$ using a polynomial basis, along with a description of the standard algorithm for multiplication in $\mathbb{F}_{2^m}$. A description of a simple version of Lee/Lim's method and two versions of the proposed method are described in Section C. In Section D, we present timing results on different computational platforms.

* Research supported by a CAPES-Brazil scholarship
** Partially supported by a PRONEX-FINEP research grant no. 107/97

B The Finite Field $\mathbb{F}_{2^m}$

B.1 Polynomial Basis Representation

In this section we describe the finite field $\mathbb{F}_{2^m}$, called a *characteristic two finite field* or a *binary finite field*, in terms of a *polynomial basis representation.* Let $f(x) = x^m + \sum_{i=0}^{m-1} f_i x^i$ (where $f_i \in \{0,1\}$, for $i = 0, \ldots, m-1$) be an irreducible polynomial of degree m over $\mathbb{F}_2$; polynomial $f(x)$ is called the *reduction polynomial.* A *polynomial basis* is specified by a reduction polynomial. In such a representation, the bit string $(a_{m-1} \ldots a_1 a_0)$ is taken to represent the polynomial

$$a_{m-1}x^{m-1} + \ldots + a_1 x^1 + a_0$$

over $\mathbb{F}_2$. Thus, the finite field $\mathbb{F}_{2^m}$ can be represented by the set of all polynomials of degree less than m over $\mathbb{F}_2$. That is,

$$\mathbb{F}_{2^m} = \{(a_{m-1} \ldots a_1 a_0) \mid a_i \in \{0,1\}\}.$$

The field arithmetic is implemented as polynomial arithmetic modulo $f(x)$. In this representation, addition and multiplication of $a = (a_{m-1} \ldots a_1 a_0)$ and $b = (b_{m-1} \ldots b_1 b_0)$ are performed as follows:

– *Addition:* $a + b = (c_{m-1} \ldots c_1 c_0)$, where $c_i = (a_i + b_i) \bmod 2$.
– *Multiplication:* $c = a \cdot b = (c_{m-1} \ldots c_1 c_0)$, where the polynomial $c(x) = \sum_{i=0}^{m-1} c_i x^i$ is the remainder of the division of polynomial $(\sum_{i=0}^{m-1} a_i x^i) \cdot (\sum_{i=0}^{m-1} b_i x^i)$ by $f(x)$. That is, $c = ab \bmod f$.

For efficiency reasons, the reduction polynomial can be selected as a *trinomial* $x^m + x^k + 1$, where $1 \leq k \leq m-1$ or a *pentanomial* $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, where $1 < k_1 < k_2 < k_3 < m-1$. ANSI X9.62 [2] specifies several rules for choosing the reduction polynomial.

In software implementations, we partition the bit representation of a field element $a = (a_{m-1} \ldots a_1 a_0)$ into blocks of the same size. Let w be the word size of a computer (typical values are $w = 8, 16, 32, 64$), and s be the number of words required to pack a into words. That is, $s = \lceil m/w \rceil$. Then, we can write a as an sw-bit number consisting of s words, where each word is of length w. Thus, we can write

$$a = (A_{s-1} \ldots A_1 A_0),$$

where each A_i is of length w and defined as[1]

$$A_i = (a_{iw+w-1} \ldots a_{iw+1} a_{iw}) \in \mathbb{F}_{2^w}.$$

In polynomials terms,

$$a(x) = \sum_{i=0}^{s-1} A_i(x) x^{iw} = \sum_{i=0}^{s-1} \sum_{j=0}^{w-1} a_{iw+j} x^{iw+j}.$$

[1] The leftmost $sw - m$ bits of A_{s-1} are set to 0.

B.2 Recent Methods for Multiplication in $\mathbb{F}_{2^m}$

In recent years, several algorithms for software multiplication in $\mathbb{F}_{2^m}$ have been reported; however, we are interested in techniques that can be used when m is prime.[2] In Schroeppel *et al.* [11] various programming tricks are discussed for implementing the "shift-and-add" method, a basic algorithm for multiplication in $\mathbb{F}_{2^m}$. A slight variant of this method is described by De Win *et al.* [12]. In Koç [5], a word-level Montgomery multiplication algorithm in $\mathbb{F}_{2^m}$ is proposed. This method is significantly faster than the standard method whenever the multiplication of two words of size w, each one representing a polynomial in $\mathbb{F}_{2^w}$ can be performed in few cycles. Since this operation is not available in most general purpose processors, the alternative is to use table lookup. This approach requires, for example, 128 Kbytes for $w = 8$ and 16 Gbytes for $w = 16$, making it less attractive for practical applications. Another well known method for multiplication in $\mathbb{F}_{2^m}$ is that of Karatsuba (see for example [4]).

B.3 The "Shift-and-Add" Method

In this section we describe the basic method for computing $c(x) = a(x) \cdot b(x) \bmod f(x)$ in $\mathbb{F}_{2^m}$. It is analogous to the binary method for exponentiation, with the square and multiplication operations being replaced by the `SHIFT` (multiplication of a field element by x) and field addition operations, respectively. Thus, the "shift-and-add" method processes the bits of polynomial $a(x)$ from left to right, and uses the following equation to perform $c = ab \bmod f$:

$$c(x) = x(\cdots x(xa_{m-1}b(x) + a_{m-2}b(x) \bmod f(x)) + \cdots) + a_0 b(x) \bmod f(x).$$

Assume that $a(x) = \sum_{i=0}^{s-1} A_i x^{wi}$, $b(x) = \sum_{i=0}^{s-1} B_i x^{wi}$, and $f(x) = \sum_{i=0}^{s-1} F_i x^{wi}$. Then the steps of the "shift-and-add" method are given below.

Algorithm 1. The "shift-and-add" method.

```
INPUT: $a = (A_{s-1} \ldots A_0)$, $b = (B_{s-1} \ldots B_0)$, and $f = (F_{s-1} \ldots F_0)$.
OUTPUT: $c = (C_{s-1} \ldots C_0) = a \cdot b \bmod f$.

1. Set $k \leftarrow m - 1 - w(s-1)$,   $c \leftarrow 0$
2. for $i$ from $s-1$ downto 0 do
       for $j$ from $k$ downto 0 do
           Set $c \leftarrow$ SHIFT($c$)
           if $a_{iw+j} = 1$ then $c \leftarrow c \oplus b$
           if $c_m = 1$ then $c \leftarrow c \oplus f$
       Set $k \leftarrow w - 1$
3. return ($c$).
```

This algorithm requires $m-1$ shift operations and m field additions on average, but the number of field additions can be reduced by selecting the reduction

[2] Many standards that include elliptic curves defined over $\mathbb{F}_{2^m}$ recommend for security reasons, the use of binary finite fields with the property that m be prime.

polynomial $f(x)$ as a trinomial or a pentanomial. Observe that in this algorithm, the multiplication step (the computation of $d(x) = a(x) \cdot b(x)$) and the reduction step (the computation of $c(x) = d(x) \bmod f(x)$) are integrated. Since for the proposed algorithm these steps are separated, we include Algorithm 2 for performing the reduction step. Assume that $f(x) = x^m + g(x)$, where the degree of polynomial $g(x)$ is less than $m - w$.

Algorithm 2. Modular reduction.

```
INPUT: $a = (A_{n-1} \dots A_{s-1} \dots A_0)$, and $f = (F_{s-1} \dots F_0)$.
OUTPUT: $c = (C_{s-1} \dots C_0) = a \bmod f$

1. for $i$ from $n-1$ downto $s$ do
       Set $d \leftarrow iw - m$
       Set $t \leftarrow A_i(x)x^d \cdot f(x) = \sum_{j=0}^{w-1} a_{iw+j}x^{d+j} \cdot f(x)$
       // $t = (T_i \dots T_{i-s}0 \dots 0)$, where $T_i = A_i$ //
       for $j$ from $i$ downto $i-s$ do
           Set $A_j \leftarrow A_j \oplus T_j$
2. Set $t \leftarrow \sum_{j=0}^{sw-1-m} a_{m+j}x^j \cdot f(x)$
     // $t = (T_{s-1} \dots T_0)$ //
3. for $j$ from $s-1$ downto 0 do
       Set $A_j \leftarrow A_j \oplus T_j$
4. return ($c \leftarrow (A_{s-1} \dots A_0)$).
```

Algorithm 2 works by zeroing out the most significant word of $a(x)$ in each iteration of step 1. A chosen multiple of the reduction polynomial $f(x)$ is added to $a(x)$ which lowers the degree of $a(x)$ by w. This is possible because the degree of $g(x)$ is less than $m - w$. Finally, the leading $sw - m$ bits of A_{s-1} are canceled in step 3 obtaining a polynomial of degree less than m. The number of XOR operations will depend on the weight of the reduction polynomial $f(x)$. For example, if $f(x)$ is a pentanomial then Algorithm 2 requires at most $8n$ XOR operations.

Remark 1. The use of standard programming tricks such as *separated name variables*, and *loop-unrolled code*, can be used to improve the performance of both Algorithms 1 and 2. See [11] for some suggested programming optimizations.

C Proposed Method

In this section we describe two versions of the new algorithm for multiplication in $\mathbb{F}_{2^m}$. The first version is a straightforward extension of Lim/Lee's method, which does not require extra temporary memory. The second version is based on a window technique. Before we describe the proposed algorithms, we discuss a simple version of Lim/Lee's method for exponentiation, using the terminology of additive groups; this will help us to understand the extension to $\mathbb{F}_{2^m}$.

In order to compute the "multiplication" $a \cdot g$ (the addition of g to itself a times) where a is an integer and g is an element of an additive group, the number a is divided into s words of size w. Then a can be written as

$$a = (A_{s-1} \dots A_1 A_0) = \sum_{i=0}^{s-1} A_i 2^{wi},$$

where each $A_i, 0 \le i < s$, has the binary representation $(a_{iw+w-1} \dots a_{iw+1} a_{iw})_2$. Based on the binary representation $(u_{s-1} \dots u_1 u_0)_2$ of u, $1 \le u < 2^s$, and the group elements $2^{wi} \cdot g, 0 \le i < s-1$, define the vector $P[u]$ of precomputations by the following equation:

$$P[u] = u_{s-1} 2^{w(s-1)} \cdot g + u_{s-2} 2^{w(s-2)} \cdot g + \dots + u_1 2^w \cdot g + u_0 \cdot g.$$

Then the multiplication $a \cdot g = \sum_{i=0}^{s-1} A_i 2^{wi} \cdot g$, can be computed as

$$a \cdot g = \sum_{j=0}^{w-1} 2^j (\sum_{i=0}^{s-1} a_{iw+j} 2^{wi} \cdot g) = \sum_{j=0}^{w-1} 2^j P[I_j], \qquad (1)$$

where $I_j = (a_{(s-1)w+j} \dots a_{w+j} a_j)_2$. A detailed algorithm for computing $a \cdot g$ using the Lim/Lee's precomputation technique is given in Algorithm 3.

Algorithm 3. Lim/Lee's algorithm.

```
INPUT: $a = \sum_{i=0}^{s-1} A_i 2^{wi}, A_i = (a_{iw+w-1} \dots a_{iw})_2, 0 \le i < s$, and $g$.
OUTPUT: $r = a \cdot g$

// Precomputation //
1. for $u$ from 0 downto $2^s - 1$ do
       Set $u \leftarrow (u_{s-1} \dots u_1 u_0)_2$
       Set $P[u] \leftarrow \sum_{i=0}^{s-1} u_i 2^{wi} \cdot g$
// Main Computation //
2. Set $r \leftarrow 0$
3. for $j$ from $w-1$ downto 0 do
       Set $r \leftarrow r + r$
       Set $u \leftarrow (a_{(s-1)w+j} \dots a_{w+j} a_j)_2$
       Set $r \leftarrow r + P[u]$
4. return ($r$).
```

Algorithm 3 performs well in situations where the group element g is known in advance, since the calculation of the precomputation step can be made off-line. A faster version of this algorithm, with more precomputations, is discussed in [6].

Next we explain the extension of Algorithm 3 to the finite field $\mathbb{F}_{2^m}$. Let a and b be two polynomials in $\mathbb{F}_{2^m}$. Assume that a can be represented as $a = (A_{s-1} \dots A_0)$. By replacing 2 by x and $2^w \cdot g$ by $x^w b(x)$ in (1), we obtain the following formal expression for the product $a(x)b(x)$:

$$a(x)b(x) = \sum_{j=0}^{w-1} x^j (\sum_{i=0}^{s-1} a_{iw+j} x^{wi} b(x)).$$

It is easy to verify that indeed the above formula for $a(x)b(x)$ is correct. Then an algorithm, analogue of Algorithm 3, can be derived for computing $ab \bmod f$ when b is a polynomial known in advance. By observing that the operation $x^{wi}b(x)$ is virtually free (it consists of an arrangement of the words representing b), the precomputation of the $2^s - 1$ polynomials: $P[u] = \sum_{i=0}^{s-1} u_i x^{wi}, 1 < u < 2^s, u = (u_{s-1} \dots u_0)_2$, can be made online. This eliminates the need of storing $2^s - 1$ polynomials, and the resulting algorithm is faster than Algorithm 1, even when b is not a fixed polynomial. The details of this method are given in Algorithm 4.

Algorithm 4. Left-to-right comb method

INPUT: $a = (A_{s-1} \dots A_0)$, $b = (B_{s-1} \dots B_0)$, and $f = (F_{s-1} \dots F_0)$.
OUTPUT: $c = (C_{s-1} \dots C_0) = ab \bmod f$

```
1. Set $T_i \leftarrow 0$;  $i = 0, \dots, 2s-1$
2. for $j$ from $w-1$ downto 0 do
       for $i$ from 0 to $s-1$ do
           if the $j$th bit of $A_i$ is 1 then
               for $k$ from 0 to $s-1$ do
                   Set $T_{k+i} \leftarrow T_{k+i} \oplus B_k$
       if  $j \neq 0$ then $T \leftarrow xT$ // shift $T$//
3. Set $c \leftarrow T$ mod $f$ // Use Algorithm 2 //
4. return ($c$).
```

Note that Algorithm 4 requires $w - 1$ SHIFT operations of a polynomial of $2s$-words. By observing that $a(x)b(x)$ can be written as

$$\sum_{j=0}^{w-1} (\sum_{i=0}^{s-1} a_{iw+j} x^{wi} x^j b(x)),$$

a right-to-left version of Algorithm 4 can be derived. The resulting algorithm is an improvement over Algorithm 4 since the word lenght of the shift polynomial $(x^j b(x))$ is less than $2s$.

The idea of window methods [4, pp. 66] for exponentiation can be extended to Algorithm 4 to obtain a more efficient algorithm, provided that extra temporary memory is available. For example, if we define the precomputed vector $P_{16}[u]$ for $0 \le u < 16$, using the equation

$$P_{16}[u](x) = (u_3 x^3 + u_2 x^2 + u_1 x + u_0) b(x),$$

where $u = (u_3 \dots u_0)_2$, then the product $a(x)b(x)$ can be computed as

$$\begin{aligned} a(x)b(x) &= \sum_{i=0}^{s-1}\sum_{j=0}^{w-1} a_{iw+j}x^{iw+j}b(x) \\ &= \sum_{j=0}^{w-1} x^j \sum_{i=0}^{s-1} a_{iw+j}x^{iw}b(x) \\ &= \sum_{j=0}^{w/4-1} x^{4j} \sum_{i=0}^{s-1} (a_{iw+j+3}x^3 + \cdots + a_{iw+j+1}x + a_{iw+j})x^{iw}b(x) \\ &= \sum_{j=0}^{w/4-1} x^{4j} (\sum_{i=0}^{s-1} x^{wi}P_{16}[u_{i,j}](x)), \text{ where } u_{i,j} = (a_{iw+j+3}\ldots a_{iw+j})_2. \end{aligned}$$

Based on the above formula for ab, we derived an algorithm that processes simultaneously four bits of each word of a and trades in each iteration four multiplications by x for one multiplication by x^4. This method is described in Algorithm 5.

Algorithm 5. Left-to-right comb method with windows of width $w = 4$

INPUT: $a = (A_{s-1} \ldots A_0)$, $b = (B_{s-1} \ldots B_0)$, and $f = (F_{s-1} \ldots F_0)$.
OUTPUT: $c = (C_{s-1} \ldots C_0) = ab \bmod f$.

```
1. for $j$ from 0 to 15 do
       Set $P_{16}[j] \leftarrow (j_3x^3 + \cdots + j_0)b(x), j = (j_3j_2j_1j_0)_2$
2. Set $T_i \leftarrow 0$;  $i = 0, \ldots, 2s-1$
3. for $j$ from $w/4-1$ downto 0 do
       for $i$ from 0 to $s-1$ do
           Set $u_{i,j} \leftarrow A_i/2^{4j} \bmod 16$
           for $k$ from 0 to $s-1$ do
               Set $T_{k+i} \leftarrow T_{k+i} \oplus P_{16}[u_{i,j}][k]$
       if $j \neq 0$ then $T \leftarrow x^4T$
4. Set $c \leftarrow T$ mod $f$ // Use Algorithm 2 //
5. return ($c$).
```

Remark 2. When b is known in advance, Algorithm 5 can be modified to work with a larger window size. If we process eight bits at the same time, then we need 256 field elements of precomputations. By observing that $\sum_{i=0}^{7} a_ix^ib(x) = \sum_{j=0}^{3} a_jx^jb(x) + \sum_{j=0}^{3} a_{4+j}x^jx^4b(x)$, we reduce the precomputation to 32 field elements at the expense of doing more XOR operations.

C.1 Performance Comparison

Let us compare the performance of Algorithms 4 and 5. We calculate the number of XOR operations and SHIFT operations required in each algorithm. We assume that the reduction polynomial is a pentanomial, so the total number of XOR operations required by Algorithm 2 is at most $8(2s-1)$. Therefore, Algorithm 4 requires $w-1$ `SHIFT` operations of a $2s$-word polynomial and $sm/2+8(2s-1)$ `XOR` operations on average. Similarly, Algorithm 5 requires 3 `SHIFT` operations of a s-word polynomial for the precomputation step, $w/4-1$ `SHIFT`[3] operations of a $2s$-word polynomial and $s(11+m/4)+8(2s-1)$ `XOR` operations on average. Thus, the time saved in Algorithm 5 is at the expense of using 14 field elements of temporary memory. In Table 1 we compared the number of operations required by Algorithms 1, 4 and 5, for the particular case $m=163$, $w=32$, $s=6$, and the pentanomial $f(x)=x^{163}+x^7+x^6+x^3+1$.

Table 1. Number of operations for Algorithms 1, 4 and 5.

	`XOR`	s-`word SHIFT`
Algorithm 1	81*6+ 81*2 = 648	162
Algorithm 4	81*6 + 42 = 528	62
Algorithm 5	52*6 + 42 = 354	17

D Timings

This section presents timing results for the proposed algorithms and the "shift-and-add" method on the following platforms: a 233 MHz Pentium MMX, a 400 MHz Pentium II, a 450 MHz Sun UltraSparc workstation and a 10 MHz Intel 386 processor (RIM interactive pager [3]). The implementation was written entirely in C, and the compilers used were `gcc` for the workstation Sun and the Pentium MMX, and Microsoft Visual C++ (version 6.0) for the other architectures. All algorithms were implemented with a comparable level of programming optimizations.

Table 2 shows running times to perform a multiplication in $\mathbb{F}_{2^{163}}$.[4] The results in this table illustrates that Algorithm 4 is about 45% to 49% faster than the standard method. We also found that Algorithm 5 required 3.0 to 5.5 times as long as the standard method, however this significant speedup is at the expense of using an extra storage of 336 bytes.

Table 2. Timings (in microseconds) for multiplication in $\mathbb{F}_{2^{163}}$.

	RIM 10 MHz	Pentium 233 MHz	Pentium II 400 MHz	UltraSparc 450 MHz
Algorithm 1	4,848	31.27	16.48	10.97
Algorithm 4	2,500*	17.02	8.40	5.55
Algorithm 5	1,515	10.20	3.00	2.52

(*estimated)

[3] We are assuming that multiplying a polynomial by x^4 is comparable in speed to multiplying a polynomial by x.

[4] Recently, NIST has recommended elliptic curves over $\mathbb{F}_{2^{163}}$ for US federal government use [10].

D.1 Applications

The most important application of this work is in software implementations of elliptic curve cryptography over $\mathbb{F}_{2^m}$. Based on the fast version of the proposed method (Algorithm 5) we implemented an algorithm for the computation of an elliptic scalar multiplication, which is the central operation used by elliptic curve cryptosystems. In Table 3 we show the timing results of our implementation of Montgomery method [7] for computing kP, where k is an integer and P is an arbitrary point on a random curve defined over $\mathbb{F}_{2^{163}}$.

Table 3. Timings (in milliseconds) for computing kP, P an arbitrary point.

Operation	RIM 10 MHz	Pentium 233 MHz	Pentium II 400 MHz	UltraSparc 450 MHz
kP	1,650*	10.96	3.24	2.74

(*estimated)

E Conclusions

In this paper we have shown a technique for speeding up the computation of $c = ab \bmod f$ in $\mathbb{F}_{2^m}$. The software implementation of the proposed method, on different platforms, proved to be significantly faster than the "shift-and-add" method, making it useful for software implementations of elliptic curve cryptography in modern workstations as well in wireless devices such as the RIM pager (a hand-held device with an Intel processor running at 10 MHz [3]).

F Acknowledgments

The authors would like to thank Mike Brown for the help with the implementation and timings, Guido Araújo, Cláudio Lucchesi, Alfred Menezes, Daniel Panario and Routo Terada for many helpful suggestions, and the anonymous referees for their comments.

References

1. G. B. Agnew, R. C. Mullin and S. A. Vanstone, "An implementation of elliptic curve cryptosystems over $\mathbb{F}_{2^{155}}$", *IEEE journal on selected areas in communications*, **11**, pp. 804-813, 1993. 203
2. ANSI X9.62, "The elliptic curve digital signature algorithm (ECDSA)", American Bankers Association, 1999. 204
3. Blackberry, `http://www.blackberry.net` 210, 211
4. I. Blake, G. Seroussi, and N. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, 1999. 205, 208
5. C. K. Koç and T. Acar, "Montgomery multiplication in $GF(2^k)$", *Designs, Codes and Cryptography*, **14**, pp. 57-69, 1998. 205

6. C. H. Lim and P. J. Lee, "More flexible exponentiation with precomputation", In *Advances in Cryptography-CRYPTO'94*, pp. 95-107, Springer-Verlag, 1994. 203, 207
7. J. López and R. Dahab, "Fast multiplication on elliptic curves over $GF(2^n)$ without precomputation", *Cryptographic Hardware and Embedded Systems -CHES'99*, lNCS **1717**, pp. 316-327, 1999. 211
8. A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997. 203
9. R. Mullin, I. Onyszchuk, S. Vanstone and R. Wilson, "Optimal normal bases in $GF(p^n)$", *Discrete Applied Mathematics*, **22**, pp. 149-161, (1988/89).
10. National Institute of Standards and Technology, "Digital signature standard", FIPS Publication 186-2, February 2000. Available at `http://csrc.nist.gov/fips` 210
11. R. Schroeppel, H. Orman, S. O'Malley and O. Spatscheck, "Fast key exchange with elliptic curve systems", University of Arizona, C. S., Tech. report 95-03, 1995. 205, 206
12. E. De Win, A. Bosselaers, S. Vanderberghe, P. De Gersem and J. Vandewalle, "A fast software implementation for arithmetic operations in $GF(2^n)$," *Advances in Cryptology, Proc. Asiacrypt'96*, LNCS **1163**, pp. 65-76, Springer-Verlag, 1996. 205

On Efficient Normal Basis Multiplication

A. Reyhani-Masoleh and M.A. Hasan

Department of Electrical and Computer Engineering
University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada
{arash, ahasan}@vlsi.uwaterloo.ca

Abstract. In cryptographic applications, the use of normal bases to represent elements of the finite field GF(2^m) is quite advantageous, especially for hardware implementation. In this article, we consider an important field operation, namely, multiplication which is used in many cryptographic functions. We present a class of algorithms for normal basis multiplication in GF(2^m). Our proposed multiplication algorithm for composite finite fields requires significantly lower number of bit level operations and hence can reduce the space complexity of cryptographic systems when implemented in hardware.

A Introduction

Many cryptographic functions, such as, key exchange, signing and verification, require significant amount of computations in the finite field GF(2^m). The elements of such a field can be represented in different ways. The choice of the representation plays an important role in determining the complexity of a finite field arithmetic unit and, consequently, that of a cryptographic system. Among the various ways one can represent field elements, the use of normal bases has drawn significant attention, especially for implementing cryptographic functions in hardware.

In a normal basis representation, squaring can be performed simply by a cycle shift of the coordinates of an element, and hence in hardware it is almost free of cost. Such a cost advantage often makes the normal basis a preferred choice of representation. However, a normal basis multiplication is not so simple. In [5], Massey and Omura proposed a normal basis multiplication scheme which can be implemented in bit-parallel fashion using m identical logic blocks whose inputs are cyclically shifted from one another [11]. Although, this normal basis multiplier offers modularity, its space complexity[1] is considerably high.

In the recent past, considerable efforts have been made, for examples, [7], [10], [1], [3], and [8], to reduce the space complexity of the normal basis multiplier. In [7], two special types of normal bases were reported which are known as type-I and type-II optimal normal bases. The use of these optimal normal bases can considerably reduce the complexity of the multiplier [10], [1], [8].

[1] Conventionally, the space complexity of the GF(2^m) multiplier is given in terms of the number of logic gates, namely XOR and AND gates, which corresponds to GF(2) (i.e., bit level) addition and multiplication, respectively.

In this article, we first present an algorithm for multiplication in GF(2^m). This algorithm is quite generic in the sense that it is not restricted to any special type of normal bases. Compared to other generic algorithms for normal basis multiplication in GF(2^m), the proposed one requires fewer bit level multiplications. Although this is achieved at the expense of extra bit level additions, the total number of GF(2) operations is the same as that of the best know generic algorithm. Our algorithm is then applied to the type-I optimal normal basis to further reduce the number of bit level operations. We then present an algorithm for normal basis multiplication in composite finite fields. This algorithm significantly reduces bit level operations, in terms of both addition and multiplication over GF(2). To show the advantage of the proposed algorithms, we compare our results with those of the best known normal basis multipliers.

B Preliminaries

B.1 Normal Basis Representation

It is well known that there exists a normal basis (NB) in the field $GF(2^m)$ over $GF(2)$ for all positive integers m. By finding an element $\beta \in GF(2^m)$ such that $\{\beta, \beta^2, \cdots, \beta^{2^{m-1}}\}$ is a basis of $GF(2^m)$ over $GF(2)$, any element $A \in GF(2^m)$ can be represented as

$$A = \sum_{i=0}^{m-1} a_i \beta^{2^i} = a_0\beta + a_1\beta^2 + \cdots + a_{m-1}\beta^{2^{m-1}}, \tag{1}$$

where $a_i \in GF(2)$, $0 \le i \le m-1$, is the i-th coordinate of A. In short, this normal basis representation of A will be written as $A = (a_0, a_1, \cdots, a_{m-1})$. In vector notation, equation (1) will be written as

$$A = \underline{a} \cdot \underline{\beta}^T = \underline{\beta} \cdot \underline{a}^T, \tag{2}$$

where $\underline{a} = [a_0, a_1, \cdots, a_{m-1}]$, $\underline{\beta} = [\beta, \beta^2, \cdots, \beta^{2^{m-1}}]$, and T denotes vector transposition.

The main advantage of the NB representation is that an element A can be easily squared by a cyclic shift of its coordinates.

B.2 Normal Basis Multiplication

Let A and B be any two elements of $GF(2^m)$ and be represented w.r.t. the NB as $A = \sum_{i=0}^{m-1} a_i\beta^{2^i}$ and $B = \sum_{j=0}^{m-1} b_j\beta^{2^j}$, respectively. Let C denote their product, i.e.,

$$C = A \cdot B = (\underline{a} \cdot \underline{\beta}^T) \cdot (\underline{\beta} \cdot \underline{b}^T) = \underline{a} \cdot \mathbf{M} \cdot \underline{b}^T, \tag{3}$$

where the multiplication matrix $\mathbf{M}$ is defined by

$$\mathbf{M} = \underline{\beta}^T \cdot \underline{\beta} = \left[\beta^{2^i+2^j}\right]_{i,\, j=0}^{m-1} . \tag{4}$$

All entries of $\mathbf{M}$ belong to GF(2^m) and if they are written w.r.t. the NB, then the following is obtained

$$\mathbf{M} = \mathbf{M}_0\beta + \mathbf{M}_1\beta^2 + \cdots + \mathbf{M}_{m-1}\beta^{2^{m-1}}, \tag{5}$$

where $\mathbf{M}_i$'s are $m \times m$ matrices whose entries belong to $GF(2)$. Substituting (5) into (3), the coordinates of C are found as follows

$$c_i = \underline{a}\mathbf{M}_i\underline{b}^T = \underline{a}^{(i)}\mathbf{M}_0\underline{b}^{(i)^T} \quad 0 \le i \le m-1 \tag{6}$$

where $\underline{a}^{(i)}$ is the i-fold left cyclic shift of $\underline{a}$ and the same is for $\underline{b}^{(i)^T}$ [1].

Definition 1. *The numbers of 1's in all M_i's are equal. Let us define this number by*

$$C_N = H(\mathbf{M}_i), \;\; 0 \le i \le m-1, \tag{7}$$

which is known as the complexity of the NB [7]. In (7), $H(\mathbf{M}_i)$ refers to the Hamming weight, i.e., the number of 1's, in $\mathbf{M}_i$.

C A New Multiplication Algorithm

In (4), the multiplication matrix $\mathbf{M}$ is symmetric and its diagonal entries are the elements of the NB. Thus, we can write

$$\mathbf{M} = \mathbf{U} + \mathbf{U}^T + \mathbf{D}, \tag{8}$$

where $\mathbf{D} = \text{diag}(\beta^2, \beta^4, \cdots, \beta^{2^{m-1}}, \beta)$ is a diagonal matrix and $\mathbf{U}$ is an upper triangular matrix with zeros at diagonal entries as given below

$$\mathbf{U} = \begin{bmatrix} 0 & \beta^{1+2^1} & \cdots & \beta^{1+2^{m-2}} & \beta^{1+2^{m-1}} \\ 0 & 0 & \cdots & \beta^{2+2^{m-2}} & \beta^{2+2^{m-1}} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \beta^{2^{m-2}+2^{m-1}} \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix} . \tag{9}$$

In (9), the exponents of β can be represented in the binary form using m bits where each exponent will have exactly two 1's. The *cyclic* distance between the two 1's is in the range $[1, v]$, where $v = \left\lfloor \frac{m}{2} \right\rfloor$. For example, with $m = 4$ each of the following three exponents, *viz.*, $3 = 0011_2$, $6 = 0110_2$, and $9 = 1001_2$, has a cycle distance of one; additionally β^3 as well as β^6 can be obtained from β^9 by squaring operations which are free of cost in the normal basis representation. In this regard, let us define

$$\delta_i \triangleq \beta^{1+2^i} \qquad i = 1, 2, \cdots, v. \tag{10}$$

Lemma 1. *Let A and B be two elements of $GF(2^m)$ and C be their product. Then*

$$C = \begin{cases} \sum_{j=0}^{m-1} \left[a_j b_j \beta^{2^j} + \left(\sum_{i=1}^{v} y_{j,i} \delta_i^{2^j} \right) \right], & \text{for } m \text{ odd} \\ \sum_{j=0}^{m-1} \left[a_j b_j \beta^{2^j} + \left(\sum_{i=1}^{v-1} y_{j,i} \delta_i^{2^j} \right) \right] + \sum_{j=0}^{v-1} y_{j,v} \delta_v^{2^j}, & \text{for } m \text{ even.} \end{cases} \tag{11}$$

where

$$y_{j,i} \triangleq (a_j + a_{((i+j))})(b_j + b_{((i+j))}),\ 1 \le i \le v,\ 0 \le j \le m-1. \tag{12}$$

Let h_i, $1 \le i \le v$, be the number of 1's in the normal basis representation of δ_i, i.e., $h_i = H(\delta_i)$. Let $w_{i,1}, w_{i,2}, \cdots, w_{i,h_i}$ denote the positions of 1's in the normal basis representation of δ_i, i.e.,

$$\delta_i = \sum_{k=1}^{h_i} \beta^{2^{w_{i,k}}},\ 1 \le i \le v, \tag{13}$$

where $0 \le w_{i,1} < w_{i,2} < \cdots < w_{i,h_i} \le m-1$. Also, for even values of m, one can obtain

$$\delta_v = \sum_{k=1}^{\frac{h_v}{2}} (\beta^{2^{w_{v,k}}} + \beta^{2^{w_{v,k}+v}}),\ v = \frac{m}{2} \tag{14}$$

Now, substituting (13) and (14) into (11), we have the following theorem.

Theorem 1. *Let A and B be two elements of $GF(2^m)$ and C be their product. Then*

$$C = \begin{cases} \sum_{j=0}^{m-1} a_j b_j \beta^{2^j} + \sum_{i=1}^{v} \sum_{k=1}^{h_i} \left(\sum_{j=0}^{m-1} y_{((j-w_{i,k})),i} \beta^{2^j} \right), & \text{for } m \text{ odd} \\ \sum_{j=0}^{m-1} a_j b_j \beta^{2^j} + \sum_{i=1}^{v-1} \sum_{k=1}^{h_i} \left(\sum_{j=0}^{m-1} y_{((j-w_{i,k})),i} \beta^{2^j} \right) + D, & \text{for } m \text{ even} \end{cases} \tag{15}$$

where

$$D = \sum_{k=1}^{\frac{h_v}{2}} \sum_{j=0}^{v-1} y_{((j-w_{v,k})),v} (\beta^{2^j} + \beta^{2^j+v}),\ \text{and } v = \frac{m}{2}.$$

Note that for a normal basis, the representation of δ_i is fixed and so is $w_{i,k}$, $1 \le i \le v$, $1 \le k \le h_i$. Based on (15), now we have the following algorithm for low complexity normal basis (LCNB) multiplication.

```
(Low Complexity Normal Basis Multiplication over $GF(2^m)$)
Input: $A,\ B \in GF(2^m),\ w_{i,k},\ 1 \le i \le v,\ 1 \le k \le h_i$
Output: $C = AB$
1.   Generate $y_{j,i} = (a_j + a_{((i+j))})(b_j + b_{((i+j))}),\ 1 \le i < v,\ 0 \le j \le m-1$,
     where $y_{j,i} \in GF(2)$.
2.   Initialize $c_j := a_j b_j,\ 0 \le j \le m-1,\ C := (c_0, c_1, \cdots, c_{m-1})$
3.   For $i = 1$ to $v-1$ {
4.     For $k = 1$ to $h_i$ {
5.       $r_j := y_{((j-w_{i,k})),i},\ 0 \le j \le m-1,\ R := (r_0, r_1, \cdots, r_{m-1})$
6.       $C := C + R$
7.     }
8.   }
9.   If $m$ is odd,
10.    $s := h_v,\ t := m$
11.  else $s := \frac{h_v}{2},\ t := \frac{m}{2}$
12.  Generate $y_{j,v} = (a_j + a_{((v+j))})(b_j + b_{((v+j))}),\ 0 \le j \le t-1$,
13.  If $m$ is even $y_{j+v,v} = y_{j,v},\ 0 \le j \le \frac{m}{2} - 1$
14.  For $k = 1$ to $s$ {
15.    $r_j := y_{((j-w_{v,k})),v},\ 0 \le j \le t-1$
16.    If $m$ is even,
17.      $r_{j+\frac{m}{2}} := r_j,\ 0 \le j \le \frac{m}{2}-1,\ R := (r_0, r_1, \cdots, r_{\frac{m}{2}-1}, r_0, r_1, \cdots, r_{\frac{m}{2}-1})$
18.    $C := C + R$
19.  }
```

Theorem 2. *For the LCNB multiplication algorithm, let $\#Mult_{LCNB}$ and $\#Add_{LCNB}$ denote the numbers of bit level multiplications and additions, respectively. Then*

$$\#Mult_{LCNB} = \frac{m(m+1)}{2}, \tag{16}$$

$$\#Add_{LCNB} = \frac{m}{2}(C_N + 2m - 3). \tag{17}$$

Multipliers	#Mult	#Add	Total bit operations
MO [11]	m^2	$m(C_N - 1)$	$m(C_N + m - 1)$
RR_MO [8]	m^2	$\frac{m}{2}(C_N + m - 2)$	$\frac{m}{2}(C_N + 3m - 2)$
LCNB	$\frac{m(m+1)}{2}$	$\frac{m}{2}(C_N + 2m - 3)$	$\frac{m}{2}(C_N + 3m - 2)$

Table 1. Comparison of normal basis multipliers.

Table 1 compares the number of bit level operations of the LCNB algorithm with those of the Massey-Omura (MO) multiplier of [11] and the reduced redundancy Massey-Omura (RR_MO) multiplier of [8]. The multipliers of [11] and [8] are used for comparison as they appear to be the first and the most recently reported work in this area, and it seems the total number of bit level operations of [8] is the least among the existing normal basis schemes. As it can be seen from the table, the total number of bit level operations of our new LCNB algorithm matches that of [8]. More importantly, the LCNB algorithm has the least number of bit level multiplications. Since the bit level multiplication corresponds to the multiplication in the ground field GF(2), if the algorithm is extended to a ground field of degree more than one, where a multiplication is more expensive than an addition operation, the use of the LCNB algorithm will be advantageous. This is investigated in Section 5 of this article.

In Table 1, the numbers of bit level additions (and consequently, the total operations) are given in terms of C_N. It is well known that $C_N \geq 2m-1$ [7]. If a normal basis has minimum C_N, *i.e.*, $C_N = 2m-1$, then it is called as an optimal normal basis (ONB). There are two types of ONBs, namely, type-I and type-II which are hereafter also referred to as ONB-I and ONB-II, respectively. The ONBs do not exist for all m. The list in [6] shows that only 23% of $m \leq 2000$ have ONBs. For a given m where an ONB exists, the minimum number of bit level additions needed in the LCNB algorithm can be obtained by substituting $C_N = 2m-1$ in (17), *i.e.*, for an ONB we have

$$\#\mathrm{Add}_{\mathrm{LCNB}} = 2m(m-1). \tag{18}$$

Below we show that the number of bit level additions can be further reduced by considering ONB-I.

D Type-I Optimal Normal Basis Multiplication

An ONB-I is generated by roots of an irreducible all-one polynomial (AOP). An AOP of degree m has its all $m+1$ coefficients equal to 1, i.e.,

$$P(x) = x^m + x^{m-1} + \cdots + x + 1. \tag{19}$$

The AOP is irreducible if $m+1$ is prime and 2 is primitive modulo $m+1$ [10]. Thus, the roots of (19) i.e., β^{2^i}, $i = 0, 1, \cdots m-1$, form an ONB-I if and only if $m+1$ is prime and 2 is primitive in modulo $m+1$. Since $m+1$ is prime, i.e., m is even, thus $v = \frac{m}{2}$. Also,

$$\delta_i = \begin{cases} \beta^{2^{k_i}} & i = 1, 2, \cdots, \frac{m}{2}-1 \\ 1 = \sum_{j=0}^{m-1} \beta^{2^j} & i = \frac{m}{2}, \end{cases} \tag{20}$$

where k_i is obtained from

$$(2^i + 1) \mod (m+1) = 2^{k_i} \mod (m+1). \tag{21}$$

Substituting (20) into (11), the product C can be written as

$$C = \left(\sum_{j=0}^{m-1} a_j b_j \beta^{2^j}\right) + \sum_{i=1}^{v-1}\left(\sum_{j=0}^{m-1} y_{j,i}\beta^{2^j}\right)^{2^{k_i}} + \left(\sum_{i=0}^{v-1} y_{i,v}\right), \tag{22}$$

where the right most summation results in 0 or 1, and in the normal basis representation, 0 and 1 correspond to $(0, 0, \cdots, 0)$ and $(1, 1, \cdots, 1)$ respectively.

```
(Low Complexity ONB-I Multiplication over $GF(2^m)$)
Input: $A,\ B \in GF(2^m),\ k_i,\ 1 \le i < v,\ v = \frac{m}{2}$
Output: $C = AB$
1.    Generate $y_{j,i} = (a_j + a_{((i+j))})(b_j + b_{((i+j))}),\ 1 \le i < v,\ 0 \le j \le m-1$,
2.    Generate $y_{j,v} = (a_j + a_{((v+j))})(b_j + b_{((v+j))}),\ 0 \le j \le v-1$,
3.    Initialize $c_j := a_j b_j,\ 0 \le j \le m-1,\ f := y_{0,v},\ f \in GF(2)$
4.    For $i = 1$ to $v-1$ {
5.      $r_j := y_{j,i},\ 0 \le j \le m-1,\ R = (r_0, r_1, \cdots, r_{m-1})$
6.      $R := R^{2^{k_i}}$
7.      $C := C + R$
8.      $f := f + y_{i,v}$
9.    }
10.   If $f$ is 1, $C := C + (1, 1, \cdots, 1, 1)$
11.   }
```

Based on (22), now we can state an algorithm for ONB-I multiplication as follows.

In line 6 of the above algorithm (hereafter referred to as LCONB-I), the operation $R^{2^{k_i}}$ can be accomplished by k_i cyclic shifts. The number of bit level operations of lines 1, 2 and 8 are $2m(v-1)$, $2v$ and $v-1$, respectively. Also, lines 7 and 10 need $m(v-1)$ and m additions. Thus, the total number of additions is

$$\#\text{Add}_{\text{LCONB-I}} = 1.5m^2 - 0.5m - 1, \tag{23}$$

and the number of multiplications is the same as that of the LCNB algorithm given in (16).

For comparison, we consider four other ONB-I multipliers as shown in Table 2. The multiplier of [11] is considered to be the first such work published in the open literature and those of [1], [2], [8] are more recent work and have the best results among the known existing ones. As it can be seen in this table, although the total number of operations of the proposed LCONB-I algorithm is the same as those of the three best multiplication schemes, the LCONB-I algorithm requires the least number of bit level multiplications, which can be advantageous in composite finite fields as discussed below.

Multipliers	#Mult	#Add	Total bit operations
MO [11]	m^2	$2m^2 - 2m$	$3m^2 - 2m$
Hasan *et al.*[1]	”	$m^2 - 1$	$2m^2 - 1$
Koc and Sunar [2]	”	”	”
RR_MO [8]	”	”	”
LCONB-I	$\frac{m(m+1)}{2}$	$1.5m^2 - 0.5m - 1$	”

Table 2. Comparison of bit level operations of ONB-I based multiplication schemes.

E Composite Field Multiplication Algorithm

In this section, we consider multiplications in the finite field GF(2^m) where m is a composite number. These fields are referred to as composite fields. They have been used in the recent past to develop efficient multiplication schemes, however, concerns have been raised regarding the use of such fields in elliptic curve crypto-systems when m has a small factor (around 3-15) [4].

Theorem 3. *[9]* Let $m_1 > 1$, $m_2 > 1$ be relatively prime. Let $N_1 = \{\beta_1^{2^i} \mid 0 \le i \le m_1 - 1\}$ and $N_2 = \{\beta_2^{2^j} \mid 0 \le j \le m_2 - 1\}$ be normal bases for $GF(2^{m_1})$ and $GF(2^{m_2})$, respectively. Then $N = \{\beta_1^{2^i}\beta_2^{2^j} \mid 0 \le i \le m_1 - 1,\ 0 \le j \le m_2 - 1\}$ is a normal basis for $GF(2^{m_1 m_2})$ over $GF(2)$. The complexity of N is $C_N = C_{N_1} C_{N_2}$, where C_{N_1} and C_{N_2} are the complexities of N_1 and N_2 respectively.

Assume that $m = m_1 \cdot m_2$ where m_1 and m_2 are as defined above. Let $A \in GF((2^{m_2})^{m_1})$, then A can be represented w.r.t. the basis

$$N = \{\beta^{2^i} \mid 0 \le i \le m-1\},\ \beta = \beta_1\beta_2,$$

as follows

$$A = \sum_{i=0}^{m-1} a_i \beta^{2^i} = \sum_{i=0}^{m_1 m_2 - 1} a_i \beta_1^{2^{i \bmod m_1}} \beta_2^{2^{i \bmod m_2}} = \sum_{j=0}^{m_1-1} A_j \beta_1^{2^j}, \tag{24}$$

where a_i's are coordinates of A and

$$A_j = \sum_{l=0}^{m_2-1} a_{j+l\cdot m_1} \beta_2^{2^{j+l\cdot m_1 \bmod m_2}}. \tag{25}$$

We assume this kind of representation for any two elements: A and $B \in GF((2^{m_2})^{m_1})$, i.e., $A = \sum_{j=0}^{m_1-1} A_j \beta_1^{2^j}$, $B = \sum_{j=0}^{m_1-1} B_j \beta_1^{2^j}$,where A_j, $B_j \in GF(2^{m_2})$. Without loss of generality, then the product $C = AB$ can be obtained from Lemma 1 as

$$C = \begin{cases} \sum_{j=0}^{m_1-1} \left[A_j B_j \beta_1^{2^j} + \left(\sum_{i=1}^{v_1} Y_{j,i}\gamma_i^{2^j}\right)\right], & \text{for } m_1 \text{ odd} \\ \sum_{j=0}^{m_1-1} \left[A_j B_j \beta_1^{2^j} + \left(\sum_{i=1}^{v_1-1} Y_{j,i}\gamma_i^{2^j}\right)\right] + \sum_{j=0}^{v_1-1} Y_{j,v_1}\gamma_{v_1}^{2^j}, & \text{for } m_1 \text{ even} \end{cases} \tag{26}$$

where $v_1 = \left\lfloor \frac{m_1}{2} \right\rfloor$, $\gamma_i = \beta_1^{1+2^i}$, $1 \le i \le v_1$ and

$$Y_{j,i} \triangleq (A_j + A_{((i+j))})(B_j + B_{((i+j))}),\ 1 \le i \le v_1,\ 0 \le j \le m_1 - 1. \tag{27}$$

In (27), $((i+j)) = i+j \bmod m_1$ and the underlying field operations are performed over the subfield $GF(2^{m_2})$.

Also, using (13), one can write γ_i w.r.t. N_1 as

$$\gamma_i = \sum_{k=1}^{h_i^{(1)}} \beta_1^{2^{w_{i,k}^{(1)}}},\ 1 \le i \le v_1, \tag{28}$$

and similar to (15), the product C can also be obtained as

$$C = \begin{cases} \sum_{j=0}^{m_1-1} A_j B_j \beta_1^{2^j} + \sum_{i=1}^{v_1} \sum_{k=1}^{h_i^{(1)}} \left(\sum_{j=0}^{m_1-1} Y_{((j-w_{i,k}^{(1)})),i}\beta_1^{2^j}\right), & \text{for } m_1 \text{ odd} \\ \sum_{j=0}^{m_1-1} A_j B_j \beta_1^{2^j} + \sum_{i=1}^{v_1-1} \sum_{k=1}^{h_i^{(1)}} \left(\sum_{j=0}^{m_1-1} Y_{((j-w_{i,k}^{(1)})),i}\beta_1^{2^j}\right) + D, & \text{for } m_1 \text{ even} \end{cases} \tag{29}$$

where

$$D = \sum_{k=1}^{\frac{h_{v_1}^{(1)}}{2}} \sum_{j=0}^{v_1-1} Y_{((j-w_{v_1,k}^{(1)})),v_1} (\beta_1^{2^j} + \beta_1^{2^j+v_1}),\ v_1 = \frac{m_1}{2}.$$

Based on (29), we can state the following algorithm for multiplication in GF(2^m) where $m = m_1 \cdot m_2$.

```
(Composite Field Normal Basis Multiplication)
Input: $A,\ B \in GF((2^{m_2})^{m_1}),\ \gamma_i \in GF(2^{m_1}),\ 1 \le i \le v_1$
Output: $C = AB$
1.   $A_j[l'] := A[j + m_1 l],\ B_j[l'] := B[j + m_1 l],\ 0 \le l \le m_2 - 1,\ 0 \le j \le m_1 - 1,$
     where $l' = j + m_1 l \bmod m_1$
2.   Generate $Y_{j,i} := (A_j + A_{((i+j))})(B_j + B_{((i+j))}),\ 1 \le i < v_1,\ 0 \le j \le m_1 - 1,$
      where $Y_{j,i},\ A_j,\ B_j \in GF(2^{m_2})$.
3.   Initialize $C_j := A_j B_j,\ 0 \le j \le m_1 - 1,\ \widetilde{C} := C_0 \mid C_1 \mid \cdots \mid C_{m_1-1}$
4.   For $i = 1$ to $v_1 - 1$ {
5.     For $k = 1$ to $h_i^{(1)}$ {
6.       $R_j := Y_{((j-w_{i,k}^{(1)})),i},\ 0 \le j \le m_1 - 1,\ \widetilde{R} := R_0 \mid R_1 \mid \cdots \mid R_{m_1-1}$
7.       $\widetilde{C} := \widetilde{C} + \widetilde{R}$
8.     }
9.   }
10.  If $m_1$ is odd,
11.    $s := h_{v_1}^{(1)},\ t := m_1$
12.  else $s := \frac{h_{v_1}^{(1)}}{2},\ t := \frac{m_1}{2}$
13.  Generate $Y_{j,v} = (A_j + A_{((v_1+j))})(B_j + B_{((v_1+j))}),\ 0 \le j \le t - 1,$
14.  If $m_1$ is even $Y_{j+v_1,v_1} = Y_{j,v_1},\ 0 \le j \le \frac{m_1}{2} - 1$
15.  For $k = 1$ to $s$ {
16.    $R_j := Y_{((j-w_{v_1,k}^{(1)})),v_1},\ 0 \le j \le t - 1$
17.    If $m_1$ is even,
18.      $R_{j+\frac{m_1}{2}} := R_j,\ 0 \le j \le \frac{m_1}{2} - 1,\ \widetilde{R} := R_0 | \cdots | R_{\frac{m_1}{2}-1} | R_0 | \cdots | R_{\frac{m_1}{2}-1}$
19.    $\widetilde{C} := \widetilde{C} + \widetilde{R}$
20.  }
21.  $C[j + m_1 l] := C_j[l'],\ 0 \le l \le m_2 - 1,\ 0 \le j \le m_1 - 1.$
```

In Algorithm E, $\widetilde{C}$ in line 3 is obtained by concatenating C_i's. $\widetilde{R}$ in line 6 is obtained in a similar way. The total number of operations of the composite field NB (CFNB) multiplication algorithm consists of two parts: multiplications and additions over the subfield $GF(2^{m_2})$. Using Theorem 2, the numbers of multiplications and additions over $GF(2^{m_2})$ are $\frac{m_1(m_1+1)}{2}$ and $\frac{m_1}{2}(C_{N_1} + 2m_1 - 3)$, respectively. Each $GF(2^{m_2})$ addition can be performed by m_2 bit level (*i.e.*, GF(2)) additions. If we use Algorithm C for subfield operations, then at the bit level each $GF(2^{m_2})$ multiplication requires $\frac{m_2(m_2+1)}{2}$ multiplications and $\frac{m_2}{2}(C_{N_2} + 2m_2 - 3)$ additions. Thus, the total numbers of bit level operations are as follows

$$\#\text{Mult}_{\text{CFNB}} = \frac{m(m_1+1)(m_2+1)}{4}, \tag{30}$$

and

$$\begin{aligned}\#\text{Add}_{\text{CFNB}} &= \tfrac{m_1}{2}(C_{N_1}+2m_1-3)\cdot m_2 + \tfrac{m_1(m_1+1)}{2}\cdot\tfrac{m_2}{2}(C_{N_2}+2m_2-3)\\ &= \tfrac{m}{2}\left[C_{N_1}+2m_1-3+\tfrac{m_1+1}{2}\left(C_{N_2}+2m_2-3\right)\right].\end{aligned} \tag{31}$$

Thus, for a given m, we can use $m_1 < m_2$ to reduce the number of addition operations given in (31). Additionally, if m_2+1 is prime and 2 is primitive modulo m_2+1, then there exists an ONB-I over $GF(2^{m_2})$ and Algorithm D can be used for $GF(2^{m_2})$ multiplication. Thus, using (23), the number of additions as given in (31) can be reduced to $\frac{m_1}{2}(C_{N_1}+2m_1-3)m_2+\frac{m_1(m_1+1)}{2}(1.5m_2^2-0.5m_2-1)$.

Example 1. Let $m = 33$, $m_1 = 3$ and $m_2 = 11$. As per Table 3 of [7], there are ONBs for $GF(2^3)$ and $GF(2^{11})$. Thus, $N_1 = \{\beta_1^{2^j} \mid 0 \le j \le 2\}$ and $N_2 = \{\beta_2^{2^l} \mid 0 \le l \le 10\}$ are type-II optimal normal bases of $GF(2^3)$ and $GF(2^{11})$, respectively. Using Theorem 3, $N = \{\beta^{2^i} \mid 0 \le i \le 32\}$, where $\beta = \beta_1\beta_2$ is a normal basis of $GF(2^{33})$ over $GF(2)$. The complexity of N *is* $C_N = C_{N_1}C_{N_2} = (2\cdot 3-1)(2\cdot 11-1) = 105$. Any two field elements A, $B \in GF(2^{33})$ can be written w.r.t. N as

$$A = \sum_{i=0}^{32} a_i\beta^{2^i} = A_0\beta_1 + A_1\beta_1^2 + A_2\beta_1^4,\ B = \sum_{i=0}^{32} b_i\beta^{2^i} = B_0\beta_1 + B_1\beta_1^2 + B_2\beta_1^4$$

where $A_j = \sum_{l=0}^{10} a_{j+3l}\beta_2^{2^{l'}}$, $B_j = \sum_{l=0}^{10} b_{j+3l}\beta_2^{2^{l'}}$, $0 \le j \le 2$, and $l' = j+3l$ mod 11. Let $C = C_0\beta_1 + C_1\beta_1^2 + C_2\beta_1^4$ be the product of A and B. Thus, using (26), we have

$$\begin{aligned}C = {} & A_0B_0\beta_1 && +(A_0+A_1)(B_0+B_1)\beta_1^3\\ & +A_1B_1\beta_1^2 && +(A_1+A_2)(B_1+B_2)\beta_1^6\\ & +A_2B_2\beta_1^4 && +(A_2+A_0)(B_2+B_0)\beta_1^{12}.\end{aligned}$$

Using Table 2 in [3], for the type-II ONB over $GF(2^3)$, we have $\beta_1^3 = \beta_1 + \beta_1^2$. Thus,

$$\begin{aligned}C = {} & ((A_0B_0 + (A_0+A_1)(B_0+B_1) + (A_2+A_0)(B_2+B_0))\beta_1\\ & +((A_1B_1 + (A_1+A_2)(B_1+B_2) + (A_0+A_1)(B_0+B_1))\beta_1^2\\ & +((A_2B_2 + (A_2+A_0)(B_2+B_0) + (A_1+A_2)(B_1+B_2))\beta_1^4\ .\end{aligned} \tag{32}$$

From (32), we see that $\frac{m_1(m_1+1)}{2} = 6$ multiplications and $\frac{m_1}{2}(C_{N_1}+2m_1-3) = 12$ additions over subfield $GF(2^{m_2})$ are needed to generate C_0, C_1 and C_2. Thus, the total numbers of bit level multiplications and additions are 396 and 1452, respectively.

Table 3 compares bit level operations for multiplication over GF(2^{33}) for a number of algorithms. Rows 2, 3 and 4, where $C_N = 65$, use ONB-II which exists for GF(2^{33}) over GF(2). On the other hand, rows 5, 6 and 7, where

$C_N = C_{N_1} \cdot C_{N_2} = 105$, use the two ONB-II's which exist for the subfields GF(2^3) and GF(2^{11}) as discussed in the above example. This comparison shows that the proposed CFNB multiplier has the least number of bit level operations. More interestingly, for composite values of m, the well known optimal normal bases GF(2^m) over GF(2) do not seem to be the best choice when one considers bit level operations, which in turn determines the space complexity for hardware implementation of a normal basis multiplier.

Multipliers	C_N	#Mult	#Add	Total bit operations
MO [11]	65	1089	2112	3201
RR_MO [8]	"	"	1584	2673
LCNB	"	561	2112	2673
MO [11]	105	1089	3432	4521
RR_MO [8]	"	"	2244	3333
LCNB	"	561	2772	3333
CFNB	"	396	1452	1848

Table 3. Comparison of operations for normal basis multipliers over $GF(2^{33})$.

We wind up this section by stating the following theorem which gives the bit level operations for normal basis multiplication over generalized composite fields.

Theorem 4. Let $m = \prod_{i=1}^{n} m_i, 1 < m_1 < m_2 < \cdots < m_n$, *where* $\gcd(m_i, m_j) = 1, i \neq j$. *Then, for a normal basis multiplication over the composite field GF(2^m), the numbers of bit level multiplications and additions are*

$$\#Mult_{CFNB} = \frac{m}{2^n} \prod_{i=1}^{n} (m_i + 1), \tag{33}$$

and

$$\#Add_{CFNB} = \frac{m}{2} \left(C_{N_1} + 2m_1 - 3 + \sum_{j=1}^{n-1} \frac{C_{N_{j+1}} + 2m_{j+1} - 3}{2^j} \prod_{i=1}^{j} (m_i + 1) \right), \tag{34}$$

respectively.

F Concluding Remarks

In this article, efficient algorithms for normal basis multiplication over GF(2^m) have been proposed. It has been shown that when m is composite, the proposed CFNB algorithm requires significantly fewer number of bit level operations compared to other similar algorithms available in the open literature. More interestingly, it has been shown that for composite values of m, the well known optimal normal bases GF(2^m) over GF(2) do not seem to be the best choice when one considers bit level operations, which in turn determines the space complexity for hardware implementation of a normal basis multiplier.

The multiplication algorithms presented in this article are targeted towards efficient hardware implementation. However, they can be modified to suite software implementation on a general purpose processor. This is being considered for discussions in a future article.

References

1. M. A. Hasan, M. Z. Wang, and V. K. Bhargava, "A Modified Massey-Omura Parallel Multiplier for a Class of Finite Fields," *IEEE Transactions on Computers*, vol. 42, no. 10, pp. 1278–1280, Oct. 1993. 213, 213, 215, 219, 219
2. C. K. Koc and B. Sunar, "Low-Complexity Bit-Parallel Canonical and Normal Basis Multipliers for a Class of Finite Fields," *IEEE Transactions on Computers*, vol. 47, no. 3, pp. 353–356, March 1998. 219, 219
3. Chung-Chin Lu, "A Search of Minimal Key Functions for Normal Basis Multipliers," *IEEE Transactions on Computers*, vol. 46, no. 5, pp. 588–592, May 1997. 213, 222
4. S. D. Galbraith and N. P. Smart, "A Cryptographic Application of Weil Descent," in *Proceedings of Cryptography and Coding*, LNCS 1764, pp. 191–200, Springer-Verlag, 1999. 220
5. J. L. Massey and J. K. Omura, "Computational Method and Apparatus for Finite Field Arithmetic," *US Patent No. 4,587,627*, 1986. 213
6. A. J. Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian, *Applications of Finite Fields*, Kluwer Academic Publishers, 1993. 218
7. R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone, and R. M. Wilson, "Optimal normal bases in $GF(p^n)$," *Discrete Applied Mathematics*, vol. 22, pp. 149–161, 1988/1989. 213, 213, 215, 218, 222
8. A. Reyhani-Masoleh and M. A. Hasan, "A Reduced Redundancy Massey-Omura Parallel Multiplier over $GF(2^m)$," in *Proceedings of the* 20^{th} *Biennial Symposium on Communications*, pp. 308–312, Kingston, Ontario, Canada, May 2000. 213, 213, 217, 217, 217, 217, 217, 219, 219, 223, 223
9. J. E. Seguin, "Low complexity normal bases," *Discrete Applied Mathematics*, vol. 28, pp. 309–312, 1990. 220
10. P. K. S. Wah and M. Z. Wang, "Realization and application of the Massey-Omura lock," *presented at the IEEE Int. Zurich Seminar on Digital Communications*, pp. 175–182, 1984. 213, 213, 218
11. C. C. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura and I. S. Reed, "VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$," *IEEE Transactions on Computers*, vol. 34, no. 8, pp. 709–716, Aug. 1985. 213, 217, 217, 217, 219, 219, 223, 223

Symmetrically Private Information Retrieval (Extended Abstract)

Sanjeev Kumar Mishra[1] and Palash Sarkar[2]

[1] Cisco Systems(India) Private Limited
Prestige Waterford, No 9, Brunton Road
Bangalore-560 025. India
sanjeevm@cisco.com
[2] Centre for Applied Cryptographic Research
Department of Combinatorics and Optimization
University of Waterloo, 200 University Avenue West
Waterloo, Ontario, Canada N2L 3G1
(On leave from Indian Statistical Institute, Calcutta)
psarkar@cacr.math.uwaterloo.ca, palash@isical.ac.in

Abstract. In this paper we present a single-round, single-server symmetrically private information retrieval scheme, in which privacy of user follows from intractability of the quadratic residuacity problem and the privacy of the database follows from the XOR assumption for quadratic residues introduced in this paper. The communication complexity of the proposed scheme for retrieving one bit can be made $O(n^{\epsilon})$, for any $\epsilon > 0$, where n is the number of bits in the database. We extend the protocol to a block retrieval scheme which is specially efficient when the number of records in the database is equal to the size of each record.

Keywords: *Private Information retrieval (PIR), Symmetrically Private Information Retrieval (SPIR), Quadratic Residuacity Assumption (QRA), Probabilistic Encryption, Cryptographic Protocols.*

A Introduction

In the age of Internet accessing remote database is common and information is the most sought after and costliest commodity. In such a situation it is very important not only to protect information but also to protect the identity of the information that a user is interested in. Consider the case, when an investor wants to know value of a certain stock, but is reluctant to reveal the identity of that stock, because it may expose his future intentions with regard to that stock. The scheme or protocol which facilitates a user to access database and receive desired information without exposing the identity of information was first introduced by Chor et al. [2] in 1995 under the name of *Private Information Retrieval*. It was also independently studied by Cooper and Birmen [4] in context of implementing an anonymous messaging service for mobile user.

A Private Information Retrieval (PIR) scheme allows a user to retrieve bits from a database (DB) while ensuring that the database does not learn which bits were retrieved. The database is modeled as an array of bits x held by a server, and the user wants to retrieve the bit x_i for some i, without disclosing any information about i to the server. We denote number of bits (records) in database by n. The communication complexity (i.e., the number of bits exchanged between user and DB), of such a protocol is denoted by $C(n)$. Also the exchange of information between the user and the DB may be interactive or non-interactive. In the first case the protocol is single-round while in the second case it is multi-round.

A trivial PIR scheme consists of sending the entire database to the user, resulting in $C(n) = n$. Clearly, such a scheme provides information theoretic privacy. Any PIR scheme with $C(n) < n$ is called non trivial. Chor et al. [2] proved that under the requirement of information theoretic security, and involvement of single database in the protocol, trivial PIR is the only possible solution. A way to overcome this impossibility result is to consider $k > 1$ servers, each holding a copy of the database x. Chor et al. [2] presented a $k > 1$ server scheme with communication complexity $O(n^{\frac{1}{k}})$. This was improved by Ambainis [1] to a $k > 2$ server PIR scheme and having a communication complexity of $O(n^{\frac{1}{(2k-1)}})$.

Another way of getting non-trivial PIR schemes is to lower the privacy requirement from information theoretic privacy to computational privacy. Chor and Gilboa [3] presented multi-server computationally private information retrieval schemes in which the privacy of user is guaranteed against the computationally bounded servers. Kushilevitz and Ostrovsky [7] showed that under the notion of computational privacy one can achieve nontrivial PIR protocol even with a single server. In particular, [7] show that assuming the hardness of quadratic residuacity problem, one can get single database PIR protocol with communication complexity $O(n^{\epsilon})$ for any $\epsilon > 0$.

While protecting the privacy of user, it is equally important that user the should not be allowed to learn more than the amount of information that he/she pays for. This is refered to as database privacy and the corresponding protocol is called a Symmetrically Private Information Retrieval (SPIR) protocol. However, the database is assumed to be honest and carries out its part of the protocol correctly. Gertner et. al. [6] presented general transformations of PIR schemes in multi server setting satisfying certain properties into SPIR schemes, with logarithmic overhead in communication complexity. Kushilevitz and Ostrovsky [7] noted that in a setting of single server, their PIR protocol can be converted into a SPIR protocol employing zero knowledge techniques to verify the validity of query. The problem is that the use of zero knowledge techniques imply that the resulting protocol is no longer a single round protocol. Thus the question of getting single-server, *single-round* nontrivial SPIR protocol was left open.

We provide the first solution to this problem by modifying the basic PIR scheme of [7] to ensure that privacy of the database is maintained. Our scheme introduces two new steps (preprocessing and postprocessing) in the database computation but does not increase the number of rounds. There is a communication overhead in the communication from the user to the database, but in the recursive scheme this is offset by the postprocessing step which effectively decreases the number of bits sent by the database to the user. In fact, for just PIR scheme the preprocessing step is not required and hence the total communication complexity is K times less than that of [7]. The preprocessing step of database computation empower the DB to restrict the user to get information from one column of the matrix formed from the database, while the postprocessing computation prevents the user to get more than one bit from the column selected in preprocessing step. Thus the user is constrained to get exactly a single bit of information from the database for each invocation of protocol. The proof of user privacy is based on the intractibility of the quadratic residuacity problem and the proof of database privacy requires a new assumption which we call the XOR assumption. In the XOR assumption we assume that if $x, y \in Z_N^{+1}$ and $w = x \oplus y$, then from w it is not possible to get any information about the quadratic character of x and y even if the user is computationally powerful enough to determine quadratic residuacity in Z_N^{+1}.

We extend the basic scheme into an efficient (in terms of communication complexity) protocol for SPIR by allowing the database to perform a depth first search on matrices of progressively smaller sizes. As a result we are able to prove the following (see [7] for a similar result on PIR protocol).

Theorem: For every ϵ, there exists a single-server, single-round computational SPIR scheme with communication complexity $O(n^\epsilon)$, where user privacy is based on QRA and database privacy is based on the XOR assumption.

We extend the bit retrieval scheme into a block retrieval scheme which is specially efficient when the number of records is equal to the size of a record.

Remark: Although we will present our schemes based on the Quadratic Residuacity Assumption, it can be based on more general assumptions. Following the approach of Mann [8], we can replace Quadratic Residuacity Predicates with any Homomorphic Trapdoor Predicates.

B Preliminaries

Informally stating, a PIR scheme is a collection of three algorithms, the users query generation algorithm $\mathcal{A}_\mathcal{Q}$, database answer computation algorithm $\mathcal{A}_\mathcal{D}$ and users reconstruction algorithm $\mathcal{A}_\mathcal{R}$, such that it satisfies the following requirements:

Correctness: In every invocation of the scheme the user retrieves the desired bit provided the user's query is correct.

User Privacy: In every invocation of the scheme the server does not gain any information about the index of the bit that the user retrieves.

PIR schemes can be classified into two groups on the basis of privacy they provide to the user.

Information Theoretic Privacy : Given the query to server or database, he cannot gain any information about the index the user is interested in, *regardless of his computational power.*

Computational Privacy : Here the server is assumed to be computationally bounded, say a polynomial size circuit. Thus for computational privacy, the probability distribution of the queries the user sends to database when he is interested in i_{th} bit, and the probability distribution of the queries when he is interested in i'_{th} bit, should be computationally indistinguishable to the server.

A symmetrically private information retrieval (SPIR) scheme is a PIR scheme satisfying an additional requirement, **the privacy of database**. Privacy can again be considered to be information theoretic or computational. Informally, computational data privacy requires, for each execution, there exists an index i, such that the probability distributions on the user's view are computationally indistinguishable for any two databases x, y such that $x_i = y_i$. That is a computationally bounded user does not receive information about more than a single bit of the database, no matter how he forms his query of given length.

For the formal definitions of PIR protocols refer to Chor et. al. [2], Kushilevitz and Ostrovsky [7] and Mann [8]. For the formal definitions of SPIR protocols see Gertner et. al. [6], Crescenzo et. al. [5], Mishra [9]. Also Mishra [9] contains a bibliography on this and related security problems.

C The Basic Scheme

In this section we introduce a basic SPIR scheme for bit retrieval. We add two new and important steps - the preprocessing and the postprocessing steps - to the database computation in the protocol of [7]. The preprocessing step restricts user access to a particular column and the postprocessing step allows the user to get only a single bit from the column. This scheme in itself is not efficient, but it makes the main ideas clear which will be used in the efficient bit retrieval scheme based on a recursive database computation. In the following, by a QNR we will mean a quadratic non residue whose Jacobi symbol is 1. Also by Z_N^{+1} we denote the set of all elements in Z_N^* whose Jacobi symbol is 1.

For clarity, we use DB to denote server or database program which handles the database. We view the n-bit database string x to be an $(s \times t)$ matrix of bits denoted by D. The user is interested in retrieving the i_{th} bit x_i of the database x, which is the $(a, b)_{th}$ entry in matrix D, where, $(a, b) = getIndex(i, t)$. The algorithm $getIndex(,)$ is defined as follows.

```
$getIndex(i, t)$ {
    if $t \mid i$, then $t_1 = \frac{i}{t}$ and $t_2 = t$
    else $t_1 = \lfloor \frac{i}{t} \rfloor + 1$ and $t_2 = i \bmod t$.
    return $(t_1, t_2)$.
}
```

Query Generation (by user) :

1. User generates two $\frac{K}{2}$ bit prime numbers p and q, such that $p \equiv q \equiv 3 \pmod 4$. Here K is the security parameter.

2. User chooses t numbers at random $y_1,\ldots,y_t \in Z_N^*$, such that y_b is a QNR and y_j $(j \neq b)$ is a QR.

3. User chooses s numbers at random $\gamma_1,\ldots,\gamma_s \in Z_N^*$, such that γ_a is a QNR

and γ_j $(j \neq a)$ is a QR.
4. User sends $N, y_1, \ldots, y_t$ and $\gamma_1, \ldots, \gamma_s$ to DB. The factorization of N is kept secret.
Database Computation :
1. ***Preprocessing*** *(Column Control):* For each bit $D(r,\tau)$ of matrix D, a K bit number $\psi(r,\tau)$ is randomly chosen as follows. If $D(r,\tau) = 0$ then $\psi(r,\tau)$ is a QR and if $D(r,\tau) = 1$, then $\psi(r,\tau)$ is a QNR. Denote by $\psi(r,\tau,\kappa)$ the κ_{th} bit of $\psi(r,\tau)$. DB now forms K matrices of size $s \times t$, D_κ :$\kappa = 1, 2, \ldots, K$ as follows: $D_\kappa(r,\tau) = \psi(r,\tau,\kappa)$ Thus for the original matrix D, DB has formed K matrices D_κ of same size. The database can always generate random QR's without knowing the factorization of N. The primes p, q are chosen to be $p, q \equiv 3 \bmod 4$. This allows the DB to also randomly generate QNR's in Z_N^{+1}.
2. For each of the K matrices D_κ, DB computes for every row r of D_κ, a number $z(r,\kappa)$ as follows: $z(r,\kappa) = \nu_r \left(\prod_{l=1}^{t}(y_l)^{D_\kappa(r,l)}\right)$ where ν_r is a randomly chosen QR by DB. Thus, DB computes $s \times K$, numbers $z(r,\kappa)$ where each $z(r,\kappa)$ is a K-bit number. For, $1 \leq c \leq K$, denote by $z(r,\kappa,c)$ the c_{th} bit of $z(r,\kappa)$.
3. ***Post Processing*** *(Row Control):* DB forms K matrices Δ_κ, $1 \leq \kappa \leq K$, such that $\Delta_\kappa(r,c) = z(c,\kappa,r)$. Now, for each of the K matrices Δ_κ $(1 \leq \kappa \leq K)$, the database DB, computes for every row r $(1 \leq r \leq K)$ a number $\zeta(r,\kappa)$ as follows : $\zeta(r,\kappa) = \nu_r \left(\prod_{l=1}^{s}(\gamma_l)^{\Delta_\kappa(r,l)}\right)$ where ν_r is a randomly chosen QR by DB. Thus for each of the K matrices Δ_κ, DB computes K numbers $\zeta(1,\kappa)$,$\zeta(2,\kappa)$,…,$\zeta(s,\kappa)$, where each of these numbers in itself a K-bit number. DB sends these K^2 numbers (a total of K^3 bits) to user.
Bit Reconstruction : User retrieves the desired bit as follows:
1. Observe that $\zeta(r,\kappa)$ is a QR iff $\Delta_\kappa(r,a)$ is 0 (see Lemma 1). Thus determining the quadratic character of the K^2 numbers, $\zeta(r,\kappa)$, gives the user the bits $\Delta_\kappa(1,a), \Delta_\kappa(2,a), \ldots, \Delta_\kappa(K,a)$.
2. From the construction of matrix Δ_κ, we see that, $z(a,\kappa,r) = \Delta_\kappa(r,a)$, and further $z(a,\kappa) = z(a,\kappa,1), \ldots, z(a,\kappa,K)$ for all $1 \leq \kappa \leq K$. Thus for all $1 \leq \kappa \leq K$, user gets $z(a,\kappa)$.
3. The quantity $z(a,\kappa)$ is a QR iff $D_\kappa(a,b)$ is 0 (see Lemma 1). Thus by determining the quadratic characters of K numbers $z(a,\kappa)$ $(1 \leq \kappa \leq K)$, user gets the bits $D_1(a,b), \ldots, D_K(a,b)$. ¿From the construction of matrices D_κ, it is clear that $D_\kappa(a,b) = \psi(a,b,\kappa)$, and further $\psi(a,b) = (a,b,1), \ldots, \psi(a,b,K)$. Using Lemma 1, $\psi(a,b)$ is a QR iff $D(a,b)$ is 0. Thus user gets the bit $D(a,b)$.
Remark: The security parameter K must satisfy $K \geq \max\{K_0,\ \mathit{poly}(\log n)\}$ where n is the number of bits in database, K_0 is the smallest number such that encryption scheme under consideration (encryption by QR's and QNR's here) is secure. The $\mathit{poly}(\log n)$ factor comes because, DB is assumed to be resourceful enough to do a computation of $O(n)$.
Correctness of the protocol follows from the following Lemma which is not difficult to prove.

Lemma 1. *Let $x = [x_1, x_2, \ldots, x_t]$ be a bit array of length t and let $\chi_1, \ldots, \chi_t$ be chosen such that $\chi_1, \ldots, \chi_{i-1}, \chi_{i+1}, \ldots, \chi_t$ are QR's and χ_i is a QNR. Let $y = \prod_{l=1}^{t}(\chi_l)^{x_l}$. Then y is a QR iff $x_i = 0$.*

Privacy of User : Suppose there exists a family of polynomial time circuits C_n that can distinguish between two query distributions Q_i and $Q_{i'}$ (for two indices i and i' of the database) with probability larger than $\frac{1}{n^\ell}$ for some integer ℓ. Then following [7], we can construct another family of polynomial time circuit C'_n from C_n, which will, on input N and $y \in Z_N^{+1}$ compute the quadratic residuacity predicate with probability at least $\frac{1}{2} + \frac{1}{8 \cdot n^\ell}$.

Privacy of database for *Honest but Curious User* is easy to prove and so here we consider the case of a *Dishonest User* A dishonest user can deviate from the protocol to possibly gain any extra information in the following ways:

1. N is a product of more than two primes. It is not clear that the user can gain extra information by using such N. Hence we will assume that N is indeed a product of two primes.

2. Assuming that N is a product of two primes, the numbers that the user sends to DB must be in Z_N^{+1}. This is necessary since the DB can perform this computation and reject a query not confirming to this specification. Hence the only way a dishonest user can cheat is to send more than one QNR's in each query set. We now argue that this provides the user with no information at all, i.e., even if one query set has two QNR's then the user does not get even one bit of the database.

Note that even if the user chooses the QR's and QNR's with some "special" properties, this will not help since in the computations of $z(r, \kappa)$ and $\zeta(r, \kappa)$, the multiplication by a randomly chosen ν_r will destroy these properties. Similar to Lemma 1, we have

Lemma 2. *Let $x_1, \ldots, x_t$ be in $QR \bigcup QNR$ and $b_1, \ldots, b_t$ be a bit string and a number z is computed as: $z = \prod_{l=1}^{t} (x_l)^{b_l}$. Suppose $x_{i_1}, \ldots, x_{i_s}$ are QNR's and rest are QR's, then z is a QR iff $b_{i_1} \oplus \ldots \oplus b_{i_s} = 0$.*

The problem is establishing database security is that we cannot base it on QRA, since the user is capable of determining quadratic residuacity in Z_N^{+1}. Indeed the user is required to determine quadratic residues for the protocol to work. However, we will see later that if the user sends more than one QNR's in his query set then he receives an element z in Z_N, which is the XOR of some randomly chosen elements $x_1, \ldots, x_t$ in Z_N^{+1}. We would like to argue that from z the user gets no information about the individual quadratic characters of any of the x_i's even though he knows the factorization of N. We make this requirement more formal in the following manner.

XOR Assumption : Let $z = x_1 \oplus \ldots \oplus x_t$, where $x_1, \ldots, x_t \in Z_N^{+1}$. Let $X = \{x_1, \ldots, x_t\}$ and A be an arbitrary subset of X. Then we assume that for N sufficiently large $Prob(A \subseteq QR, X - A \subseteq QNR \mid z) \in [\frac{1}{2^t} - \delta(K), \frac{1}{2^t} + \delta(K)]$, where K is the number of bits required to represent N and $\delta(K)$ goes to zero as K increases.

A formal proof of the XOR assumption looks difficult to obtain. Some experiments were conducted to verify the XOR assumption for $t = 2$ and small values of N. We briefly mention the important observations.

1. From simulation it is observed that $\delta(K)$ depends on $\frac{N - 2^{K-1}}{2^{K-1}}$. As this ratio increases from 0 to an upper bound of $\eta(K) > 0$, the value of $\delta(K)$ falls rapidly.

Further the upper bound $\eta(K)$ decreases exponentially with increasing K.
2. The nearer the ratio $\frac{p}{q}$ of two prime factors of N is to 1, the smaller is the value $\delta(K)$.
3. XOR Assumption can be generalized to any probabilistic encryption scheme, there are some supportive evidences which we are unable to present here for the lack of space.

We consider three cases which can occur from the possible nature of query sent by user:
First set contains more than one QNR : Let the first set in the query sent by user contain p many QNR's at positions $b_1, \ldots, b_p$. Then Lemma 2 implies that in the reconstruction phase a number $z(a, \kappa)$, $(1 \leq \kappa \leq K)$ obtained by user is a QR iff $D_\kappa(a, b_1) \oplus \ldots \oplus D_\kappa(a, b_p) = 0$. Thus user is able to reconstruct $\psi(a, b_1) \oplus \ldots \oplus \psi(a, b_p)$.
Second set contains more than one QNR : Let the second set of numbers in the query sent by user contains q many QNR's at positions $a_1, a_2, \ldots, a_q$. Again using Lemma 2 in post processing computation, we see that a number $\zeta(r, \kappa)$ received by user is a QR iff $\Delta_\kappa(r, a_1) \oplus \ldots \oplus \Delta_\kappa(r, a_q) = 0$, $(1 \leq r \leq K)$, and $(1 \leq \kappa \leq K)$. Thus user will be able to reconstruct $z(a_1, \kappa) \oplus z(a_2, \kappa) \oplus \ldots \oplus z(a_m, \kappa)$.
Both sets contain more than one QNR : Let first set contain more than one QNR's at positions $b_1, \ldots, b_p$ and the second set contain QNR's at positions $a_1, a_2, \ldots, a_q$. Then a number $\xi(r, \kappa)$ received by user is a QR iff $\Delta_\kappa(r, a_1) \oplus \ldots \oplus \Delta_\kappa(r, a_q) = 0$, $(1 \leq r \leq K)$, and $(1 \leq \kappa \leq K)$. Thus user will be able to reconstruct $z(a_1, \kappa) \oplus z(a_2, \kappa) \oplus \ldots \oplus z(a_q, \kappa)$ $(1 \leq \kappa \leq K)$.
Thus in all the three cases user will be able to reconstruct only XOR's of more than one numbers, and the XOR assumption says that from the XOR of two or more numbers from the set Z_N^{+1}, it is not possible to know anything about the quadratic characters of the constituent numbers. Hence if user sends more than one QNR's in any set of numbers in his query, he fails to get even one bit of the database x.
Communication complexity : Total communication from user to database DB in this scheme is $(1 + t + s)$ K-bit numbers $(N, y_1, \ldots, y_t, \gamma_1, \ldots, \gamma_s)$. while database returns K^2 K-bit numbers $\zeta_(r, \kappa)$ $(1 \leq r \leq K, 1 \leq \kappa \leq K)$ obtained after the postprocessing to user. Thus the communication complexity is $(1+t+s+K^2) \cdot K$ bit, which can be minimized by choosing $t = s = \sqrt{n}$, and the communication complexity is: $(2\sqrt{n}+1+K^2) \cdot K$. Under similar assumptions on the security parameter Kushilevitz and Ostrovsky [7] obtained the communication complexity $(2 \cdot \sqrt{n}+1) \cdot K$ for their basic PIR scheme. A closer look reveals that, with an extra communication of K^3 bit over the basic PIR scheme presented by Kushilevitz and Ostrovsky [7], we have successfully obtained a SPIR scheme. Even with the weaker assumption on security parameter, i.e.; $K = n^\epsilon$ for some constant $\epsilon > 0$, we get a communication complexity $O(n^{\frac{1}{2}+\epsilon})$, provided $\epsilon < \frac{1}{4}$. Thus we have proved the following theorem :

Theorem 1. *For every $\frac{1}{4} > \epsilon > 0$, there exists a single-server, single-round SPIR protocol with communication complexity $O(n^{\frac{1}{2}+\epsilon})$ where user privacy is based on QRA and the database privacy is based on the XOR assumption.*

D Iterative Bit SPIR Scheme

In this section we develop an improved scheme using the ideas developed in our basic scheme and we manage to bring down the communication complexity. We essentially achieve this by allowing the DB to do a recursive computation (see also [7]). We put stress on the fact that the scheme involves only a single round of communication and security of database as well as user remain preserved.

As before, we view the n-bit database string x to be an $(s \times t)$ matrix of bits denoted by D. The i_{th} bit x_i of the database x is $(a_1, b_1)_{th}$ entry in matrix D, where $(a_1, b_1) = getIndex(i, t)$.

Query Generation :

1. User generates two $\frac{K}{2}$ bit prime number p and q with $p, q \equiv 3 \bmod 4$. Calculate $N = pq$.

2. User calculates t such that $t^{L+1} = n$, where L is the depth of recursion of the database computation. The value of L is chosen such that communication complexity is minimized (as we will see later).

3. User generates a sequence of the pair of indices (a_j, b_j) as follows.

for $j \leftarrow 1$ **to** $L - 1$ $\{(a_{j+1}, b_{j+1}) = getIndex(a_j, t)\}$

The pair (a_j, b_j) correspond to the row and column index of the relevant bit in the matrices in j_{th} round of DB computation. Also define $s_j = \frac{n}{t^j}$ and $t_j = t$ for $1 \leq j \leq L$. (s_j, t_j) are number of rows and columns in each matrix in the j_{th} round of DB computation.

4. User generates an $L \times t$ matrix y, where for $1 \leq \sigma \leq L$, $1 \leq \beta \leq t$: $y(\sigma, \beta)$ is a QNR if $\beta = b_\sigma$, else it is a QR. Clearly each row of y contains exactly one QNR.

5. User randomly chooses s_L numbers $\gamma_1, \ldots, \gamma_{s_L} \in Z_N^*$, such that γ_{a_L} is a QNR and γ_j $(j \neq a_L)$ is a QR.

6. User sends N, the matrix y and the numbers $\gamma_1, \ldots, \gamma_{s_L}$ to the DB. The factorization of N is kept secret.

Database Computation :

Database DB performs a $L + 2$ round of computation in three phases. First phase is the preprocessing round of basic scheme, while the third phase is the post processing round. The middle step is an L round recursive computation.

1. ***Pre Processing*** *(Column Control) :* As in the basic scheme DB, forms K matrices $D_\kappa(\alpha, \beta)$ from the original matrix $D(\alpha, \beta)$. Again the user requires exactly one bit from each of the matrices D_κ's.

2. ***Iterative Computation:*** The database DB performs a L round recursive computation according to following algorithm:

For each D_κ $(1 \leq \kappa \leq K)$ formed in the preprocessing round perform the call $DFSCompute(D_\kappa, s, t, y_1, 1)$.

The algorithm $DFSCompute$ is described below:

```
$DFSCompute(M, s, t, y_l, l)\{$
/* • $y_l$ is the $l_{th}$ row of the matrix of numbers sent by user. Note $y_l[i] = y(l, i)$
     is a $QR$ if $i \neq b_l$ and $y_l[i]$ is a $QNR$ if $i = b_l$.
   • M is an $s \times t$ matrix of bits and we want to retrieve the bit $M[a_l, b_l]$.
```

• l denotes the level of the recursion. */

1. Set for $1 \leq i \leq s$, $z[i] = \nu_i \left(\prod_{j=1}^{t}(y_l[j])^{M[i,j]}\right)$ where ν_i for each i is a QR chosen by DB uniformly at random.

/* Each $z[i]$ is a K-bit string. For $1 \leq j \leq K$, let $z[i,j]$ denote the j_{th} bit of $z[i]$. We require the string $z[a_l]$ */

2. For $1 \leq j \leq K$, form K matrices M_j, where each M_j is an $\frac{s}{t} \times t$ matrix formed from the column vector, $z[*,j] = z[1,j], \ldots, z[s,j]$ by breaking $z[*,j]$ into t-bit blocks and arranging the blocks in a row wise fashion.

/* The string $z[a_l]$ is distributed over the K matrices M_j, i.e., the string $z[a_l]$ is equal to $M_1[a_{l+1}, b_{l+1}], \ldots, M_K[a_{l+1}, b_{l+1}]$, where $(a_{l+1}, b_{l+1}) = getIndex(a_l, t)$. */

```
3. $for(1 \leq j \leq K)\{$
        $if(l < L-1)\ DFSCompute(M_j, \frac{s}{t}, t, y_{l+1}, l+1)$
        $else\ PostCompute(M_j, \frac{s}{t}, t, y_L, \gamma)$
    $\}$
$\}$
```

The routine $PostCompute(\cdot)$ is the postprocessing step and is described below:

3. *Post Processing*(*Row Control*):

$PostCompute(M, s, t, y, \gamma)\{$

/* • As in $DFSCompute$ M is an $s \times t$ matrix of bits and we want to retrieve the bit $M[a,b]$, where the index a and b is hidden in the y and γ.

• $y[j]$ $(1 \leq j \leq t)$ is an array of t numbers (L_{th} row of the matrix y sent by user). $y[j]$ is a QNR if $j = b$ else it is a QR.

• $\gamma[j]$ $(1 \leq j \leq s)$ is an array of s numbers (γ sent by user). $\gamma[j]$ is a QNR if $j = a$ else it is a QR. */

1. Set for $1 \leq i \leq s$, $z[i] = \nu_i \left(\prod_{j=1}^{t}(y[j])^{M[i,j]}\right)$, where ν_i for each i is a QR chosen by DB uniformly at random.

/* Each $z[i]$ is a K-bit string. For $1 \leq j \leq K$, let $z[i,j]$ denote the j_{th} bit of $z[i]$. We require the string $z[a]$ */

2. Set $M'[i,j] = z[j,i]$ for $1 \leq j \leq s$, $1 \leq i \leq K$.

/* M' is an $K \times s$ matrix of bits. */

3. Set for $1 \leq i \leq K$, $\zeta[i] = \nu_i \left(\prod_{j=1}^{s}(\gamma[j])^{M'[i,j]}\right)$ where ν_i for each i is a QR chosen by DB uniformly at random.

4. Output the strings $\zeta[1], \ldots, \zeta[K]$. These are sent to the user.

$\}$

Reconstruction Phase and Correctness :

We show that from the output of $PostCompute(\cdot)$, it is possible to reconstruct the $(a_1, b_1)_{th}$ bit of matrix D i.e., i_{th} bit of database x. This will establish the correctness of protocol and also provide the method using which user can reconstruct the i_{th} bit of database x. (We assume that the user's query is properly formed.)

1. Suppose the call $PostCompute(M, s, t, y, \gamma)$ outputs the strings $\zeta[1], \ldots, \zeta[K]$ and the QNR's of y and γ are the b_{th} and a_{th} position respectively. The value of $\zeta[i]$ is a QR iff $M'[i,a]$ is 0, so it is possible to reconstruct the column vector

$M'[*, a]$ which is equal to the row vector $z[a, *] = z[a]$. Again $z[a]$ is a QR iff $M[a, b]$ is 0. Thus it is possible to find $M[a, b]$. So it is possible to get the bits at level $L - 1$. Now we use backward induction on the depth of recursion.
2. Suppose the call $DFSCompute(M, s, t, y_l, l)$ produces the set of matrices $M_1, \ldots, M_K$. By induction hypothesis it is possible to get the bits $M_i[a_{l+1}, b_{l+1}]$. We show it is possible to get $M[a_l, b_l]$ from these. From the construction of $z[a_l]$ we find that it is equal to $M_1[a_{l+1}, b_{l+1}], \ldots, M_K[a_{l+1}, b_{l+1}]$ and so it is possible to get $z[a_l]$. The quantity $z[a_l]$ is a QR iff $M[a_l, b_l]$ is 0. Thus we get the bit $M[a_l, b_l]$. This proves the induction. Hence the user can get the $(a_1, b_1)_{th}$ bit of all the K matrices D_κ passed in the routines $DFSCompute(M, s, t, y_1, 1)$. Thus user gets the bits $D_1[a_1, b_1], \ldots, D_K[a_1, b_1]$, which on concatenation gives the number $\psi(a_1, b_1)$ by which DB had replaced the $(a_1, b_1)_{th}$ bit of matrix D. Again $\psi(a_1, b_1)$ is a QR iff $D(a_1, b_1)$ is 0. Thus user is able to obtain the desired bit.
Privacy of user and database can be shown similarly as in the case of basic scheme. We omit the details due to lack of space.
Communication Complexity
Communication from user to DB is (1) a K-bit number N, (2) a $L \times t$ query matrix y of K-bit numbers and (3) an array of K-bit numbers of length t. Thus total communication from user to database DB in this scheme is $(1+(L+1)\cdot t)\cdot K$. The DB returns numbers computed at the end of $PostCompute(\cdot)$ routine. We analyze the tree structure formed from the computation process of DB. The Root of the computation tree is the matrix D formed from original database x. Now in preprocessing computation DB obtains K matrices D_κ's $(1 \le \kappa \le K)$ from the matrix D. Each of these K matrices becomes the child of the root. Thus root node, designated at level 0 has K children (all at level 1). The call of routine $DFSCompute(\cdot)$ at l_{th} level of recursion takes a matrix at l level as input and produces K matrices as output. Thus each of the nodes at level $l < L$ has K children. Matrices at level L are input to $PostCompute(\cdot)$ which produces K numbers of K-bit each, which are returned to user. Thus for each of the K^L matrices (leaf nodes of computation tree), user receives K^2 bits. Therefore the total communication from DB to user is K^{L+2} bits. Hence the communication complexity $C(n) = (1+(L+1)\cdot t+K^{(L+1)})\cdot K$ bits, where $t = n^{\frac{1}{L+1}}$. If we choose, $L = \sqrt{\frac{\log n}{\log K}} - 1$, then $C(n) = (1 + (\sqrt{\frac{\log n}{\log K}} + 1) \cdot 2^{\sqrt{\log n \cdot \log K}}) \cdot K$. Compare this with the communication complexity $O(2^{2\cdot\sqrt{\log n \cdot \log K}})$ obtained by Kushilevitz and Ostrovsky [7] for their PIR scheme. *Thus we have a single-round SPIR scheme with the communication complexity even smaller than the PIR scheme of [7].*

Even with the weaker assumption on security parameter, i.e., $K = n^\epsilon$ for some constant $\epsilon > 0$, we get a communication complexity $O(\frac{1}{\sqrt{\epsilon}} \cdot n^{\sqrt{\epsilon}+\epsilon}) = O(n^{\sqrt{\epsilon}+\epsilon})$. which is better than the communication complexity $(n^{2\cdot\sqrt{\epsilon}})$ obtained in [7] under the same assumption. If we take $K = \log^\epsilon n$, then we get the communication complexity of $O(\sqrt{\frac{\log n}{\epsilon \cdot \log\log n}} \cdot \log^\epsilon n \cdot 2^{\sqrt{\epsilon \cdot \log n \cdot \log\log n}})$.

Hence we have proved the following theorem :

Theorem 2. *For every $\epsilon > 0$, there exists a single-server single-round SPIR protocol with communication complexity $O(n^{\epsilon})$, where user privacy is based on QRA and database privacy is based on the XOR assumption.*

E Block Retrieval SPIR Scheme

In previous section we presented scheme for retrieving a bit from a database modeled as a array of bits. But a more realistic view of a database is to assume it partitioned into blocks rather than bits. We view database x as a array of records, each of size m, having n records in total. User wants to retrieve i_{th} record. Number of records n and number of bits in a record m determine L, as $L = \lceil \frac{\log n}{\log m} \rceil - 1$. The value of L determine the recursion depth of database computation.

For the purpose of DB computation database x held by DB is viewed as a stack of m matrices D_{μ} $(1 \leq \mu \leq m)$, where each matrix D_{μ} is an $s \times t$ matrix of bits and user wants to retrieve the bits $D_{\mu}(a_1, b_1)$. Now to retrieve the i_{th} record from database x, user generates a query following the protocol in our bit retrieval scheme, but taking the value of L defined as above. DB applies the query sent by user on all the m matrices in the stack, and send back to user the answer obtained for each of the matrices in the stack. As user can obtain i_{th} bit of each of the matrix, he will get the i_{th} record. Correctness and privacy of user and privacy of database follows from the underlying bit retrieval protocol.
The **Communication Complexity** in this scheme is $C(m, n) = (1 + (L+1) \cdot n^{\frac{1}{L+1}} + m \cdot K^{L+1}) \cdot K$ Therefore, we proved following theorem:

Theorem 3. *There exist a block retrieval SPIR protocol with communication complexity linear in number of bits in the record m and polynomial in security parameter K, i.e., we have $O(m \cdot K^{L+2})$, where m is the number of bits in the record and $L = \lceil \frac{\log n}{\log m} \rceil - 1$, where user privacy is based on QRA and database privacy is based on the XOR assumption.*

Corollary 1. *For $n \leq m$, i.e., number of records not more than number of bits in any record, we get $L = 0$, and communication complexity: $C = (1 + n + m \cdot K) \cdot K < m \cdot (K + K)K$ i.e., $C = O(mK^2)$. For $m < n \leq m^2$), we get $L = 1$ and communication complexity $C = O(m \cdot K^3)$. In general, $n^{\frac{1}{L}} = m$, and $(L+1) < K^(L+1)$, thus communication complexity $C = O(m \cdot K^{L+2})$.*

F Conclusion

In this paper we have presented a single-server, single-round SPIR protocol. The communication complexity of the protocol can be made $O(n^{\epsilon})$ for any $\epsilon > 0$. Further the scheme has been extended to efficient block retrieval protocols. Some of the ideas used in the construction of SPIR protocol is based on the PIR protocol in [7]. In Mishra [9], it is shown that there exists PIR and SPIR schemes having communication complexity $O(K \log n)$ (where K is the security parameter and n is the size of the database) provided there exists probabilistic encryption schemes with certain desirable properties.

References

1. A. Ambainis. *An upper bound on the communication complexity of private information retrieval.* In Proc. of the 24th ICALP. Lecture Notes in Computer Science. vol. 1256. Springer-Verlag, New York, 1997, pp.401-407. 226
2. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. *Private information retrieval.* In Proceedings of 36th Annual Symposium on Foundation of Computer Science. IEEE. Milwaukee, Wisconsin, 23-25 October 1995, pp.41-50. *Journal version in JACM*, vol. 45(6), 1998, pp.965-981. 225, 226, 226, 228
3. B. Chor, and N. Gilboa. *Computationally private information retrieval* . In Proceedings of the 29th Annual Symposium on Theory of Computing, El Paso, Tex., May 4-6, 1997. ACM, New York, pp294-303. 226
4. David A. Cooper, and Kenneth P. Birman. *Preserving privacy in a network of mobile computers.* Proc. IEEE Symposium on Security and Privacy,1995, pp.26-38. 225
5. G. Di Crescenzo, T. Malkin, and R. Ostrovsky. *Single database private information retrieval implies oblivious transfer.* In Proceedings of EUROCRYPT'00, 2000. 228
6. Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. *Protecting data privacy in private information retrieval schemes.* In Proceedings of the 30th Annual Symposium on Theory of Computing, Dallas, Tex., May 23-26, 1998. ACM, New York, pp.151-160. 226, 228
7. E. Kushilevitz, and R. Ostrovsky. *Replication is not needed : single database computationally-private information retrieval* . In Proceedings of 38th Annual Symposium on Foundation of Computer Science. IEEE Computer Society Press, Los Alamitos, Calif., 1997, pp. 364-373. 226, 226, 226, 226, 227, 227, 228, 228, 230, 231, 231, 232, 234, 234, 234, 235
8. E. Mann. *Private access to distributed information.* Master's thesis, Technion, Israel Institute of Technology, Hafia, 1998. 227, 228
9. S.K. Mishra. *On symmetrically private information retrieval.* MTech, Computer Science Dissertation Series, Indian Statistical Institute, Calcutta, 2000. Also available at Cryptology ePrint archive, http://eprint.iacr.org, 2000/041. 228, 228, 235

Two-Pass Authenticated Key Agreement Protocol with Key Confirmation

Boyeon Song and Kwangjo Kim

Information and Communications University (ICU)
58-4 Hwaam-dong, Yusong-gu, Taejon, 305-732, S. Korea
{bysong, kkj}@icu.ac.kr

Abstract. This paper proposes three key agreement protocols that emphasize their security and performance. First, the two-pass authenticated key agreement (AK) protocol is presented in the asymmetric setting, which is based on Diffie-Hellman key agreement working over an elliptic curve group and provides more desirable security attributes than the MTI/A0, two-pass Unified Model and two-pass MQV protocols. Other two protocols are modifications of this protocol: the three-pass authenticated key agreement with key confirmation (AKC) protocol which uses message authentication code (MAC) algorithms for key confirmation, and the two-pass authenticated key agreement protocol with unilateral key confirmation which uses the MAC and the signature.

A Introduction

Key establishment protocol is a process whereby a shared secret key becomes available to participating entities, for subsequent cryptographic use. It is broadly subdivided into key transport and key agreement protocol[18]. In *key transport* protocols, a key is created by one entity and securely transmitted to the other entity. In *key agreement* protocols, both entities contribute information to generate the shared secret key. Key establishment protocol employs symmetric or asymmetric key cryptography. A *symmetric* cryptographic system is a system involving two transformations - one for the initiator and one for the responder - both of which make use of the same secret key[18]. In this system the two entities previously possess common secret information, so the key management problem is a crucial issue. An *asymmetric* cryptographic system is a system involving two related transformations - one defined by a public key (the public transformation), and another defined by a private key (the private transformation) - with the property that it is computationally infeasible to determine the private transformation from the public transformation[18]. In this system the two entities share only public information that has been authenticated[6]. This paper is concerned with two-entity authenticated key agreement protocol working over an elliptic curve group in the asymmetric (public-key) setting.

The goals of any authenticated key establishment protocol is to establish keying data. Ideally, the established key should have precisely the same attributes as a key established face-to-face. However, it is not an easy task to identify the precise security requirements of authenticated key establishment. Several concrete security and performance attributes have been identified as desirable[4].

The fundamental security goals of key establishment protocols are said to be implicit key authentication and explicit key authentication. Let A and B be two honest entities, *i.e.*, legitimate entities who execute the steps of a protocol correctly. Informally speaking, a key agreement protocol is said to provide *implicit key authentication (IKA)* (of B to A) if entity A is assured that no other entity aside from a specifically identified second entity B can possibly learn the value of a particular secret key. A key agreement protocol which provides implicit key authentication to both participating entities is called an *authenticated key agreement (AK)* protocol. A key agreement protocol is said to provide *key confirmation* (of B to A) if entity A is assured that the second entity B actually has possession of a particular secret key. If both implicit key authentication and key confirmation (of B to A) are provided, the key establishment protocol is said to provide *explicit key authentication (EKA)* (of B to A). A key agreement protocol which provides explicit key authentication to both entities is called an *authenticated key agreement with key confirmation (AKC)* protocol[4,14,18].

Desirable security attributes of AK and AKC protocols are known-key security (a protocol should still achieve its goal in the face of an adversary who has learned some other session keys - unique secret keys which each run of a key agreement protocol between A and B should produce.), forward secrecy (if static private keys of one or more entities are compromised, the secrecy of previous session keys is not affected.), key-compromise impersonation (when A's static private key is compromised, it may be desirable that this event does not enable an adversary to impersonate other entities to A.), unknown key-share (entity B cannot be coerced into sharing a key with entity A without B's knowledge, *i.e.*, when B believes the key is shared with some entity $C \neq A$, and A correctly believes the key is shared with B.), *etc*[4,14,18]. Desirable performance attributes of AK and AKC protocols include a minimal number of passes, low communication overhead, low computation overhead and possibility of precomputations[4].

This paper firstly proposes a new two-pass AK protocol in the asymmetric setting, which is based on Diffie-Hellman key agreement working over an elliptic curve group. The protocol provides more desirable security attributes than the MTI/A0, Unified Model and MQV AK protocols. Secondly, an AKC protocol from the AK protocol is derived by using the Message Authentication Code (MAC) algorithms. Finally, a two-pass unilateral AKC protocol is designed to provide mutual IKA and unilateral EKA, and known-key security, forward secrecy, key-compromise impersonation, and unknown key-share attributes discussed in [2,6].

The remaining of this paper is organized as follows. Section 2 reviews AK protocols like the MTI/A0, Unified Model and MQV protocols. Section 3 presents the new AK protocol and its two variants; AKC protocol and two-pass unilateral AKC protocol. Section 4 compares key agreement protocols suggested in Sections 2 and 3. Finally, Section 5 makes concluding remarks.

B AK Protocols

This section describes the MTI/A0, two-pass Unified Model, and two-pass MQV protocols. The security of AK protocols in this paper is based on the *Diffie-Hellman problem*[10] in elliptic curve group: given an elliptic curve E defined over a finite field $\mathbf{F}_q$, a base point $P \in E(\mathbf{F}_q)$ of order n and two points generated by P, xP and yP (where x and y are integer), find xyP. This problem is closely related to the well-known *elliptic curve discrete logarithm problem (ECDLP)* (given $E(\mathbf{F}_q), P, n$ and xP, find x)[7] and there is strong evidence that the two problems are computationally equivalent (*e.g.*, see [8] and [16]).

All protocols in this paper have been described in the setting of the group of points on an elliptic curve defined over a finite field. The following abbreviations are used for clear understanding: IKA denotes implicit key authentication, EKA explicit key authentication, K-KS known-key security, FS forward secrecy, K-CI key-compromise impersonation, and UK-S unknown key-share[4].

We first present the elliptic curve parameters that are common to both entities involved in the protocols (*i.e.*, the *domain parameters*), and the key pairs of each entity. These are the same one used in [14].

Domain Parameter

The domain parameters consist of a suitably chosen elliptic curve E defined over a finite field $\mathbf{F}_q$ of characteristic p, and a *base point* $P \in E(\mathbf{F}_q)$[14].

1. a field size q, where q is a prime power (in practice, either $q = p$, an old prime, or $q = 2^m$);
2. an indication FR (field representation) of the representation used for the elements of $\mathbf{F}_q$;
3. two field elements a and b in $\mathbf{F}_q$ which define the equation of the elliptic curve E over $\mathbf{F}_q$;
4. two field elements x_p and y_p in $\mathbf{F}_q$ which define a finite point $P = (x_p, y_p)$ of prime order in $E(\mathbf{F}_q)$;
5. the order n of the point P; and
6. the cofactor $c = \#E(\mathbf{F}_q)/n$.

A set of domain parameters (q, FR, a, b, P, n, c) can be verified to meet the above requirements (see [14] for more details). This process is called *domain parameter validation.*

Key Pairs

Given a valid set of domain parameters (q, FR, a, b, P, n, c), an entity's *private key* is an integer d selected at random from the interval $[1, n-1]$, and its *public key* is the elliptic curve point $Q = dP$. The key pair is (Q, d). For the protocols described in this paper, each entity has two key pairs: a *static* or *long-term* key pair (which is bound to the entity for a certain period of time), and *ephemeral* or *short-term* key pair (which is generated for each run of the protocol)[14]. A's static key pair and ephemeral key pair are denoted (W_A, w_A) and (R_A, r_A) respectively.

For the remainder of this paper, we will assume that static public keys are exchanged via certificates. $Cert_A$ denotes A's public-key certificate, containing a string of information that uniquely identifies A (such as A's name and address), her static public key W_A, and a certifying authority (CA)'s signature over this information. To avoid a potential unknown key-share attack, the CA should verify that A possesses the private key w_A corresponding to her static public key W_A[14].

Public-Key Validation

Before using an entity's purported public key Q, it is prudent to verify that it possesses the supposed arithmetic properties - namely that Q be a finite point in the subgroup generated by P[14]. This process is called *public-key validation*[12]. A purported public key $Q = (x_Q, y_Q)$ can be validated by verifying that:

1. Q is not equal to $\mathcal{O}$ which denotes a point at infinity;
2. x_Q and y_Q are elements in the field $\mathbf{F}_q$;
3. Q satisfies the defining equation of E; and
4. $nQ = \mathcal{O}$.

The computationally expensive operation in public-key validation is the scalar multiplication in step 4. To reduce this burden, step 4 can be omitted during key validation. It is called *embedded public-key validation*[14]. For ease of explanation, public key validation is omitted from the descriptions of protocols of this section.

B.1 MTI/A0

The MTI/A0 key agreement protocol was proposed by Matsumoto *et al.* in 1986[17]. It was designed to provide implicit key authentication. The similar protocols are KEA[19] and those proposed by Goss[11] and Yacobi[20].

Protocol 1

1. A generates a random integer r_A, $1 \leq r_A \leq n-1$, computes the point $R_A = r_A P$, and sends $(R_A, Cert_A)$ to B.
2. B generates a random integer r_B, $1 \leq r_B \leq n-1$, computes the point $R_B = r_B P$, and sends $(R_B, Cert_B)$ to A.
3. A computes $K = r_A W_B + w_A R_B$.
4. B computes $K = r_B W_A + w_B R_A$.
5. The session key is $k = kdf(K)$ (where kdf is a key derivation function).

Remark

- A and B commonly compute $K = (r_A w_B + r_B w_A)P$.
- It does not provide FS since an adversary who learns w_A and w_B can compute all session keys established by A and B.

B.2 Two-Pass Unified Model

The Unified Model proposed by Ankney *et al.*[1] is an AK protocol in the draft standards ANSI X9.42[21], ANSI X9.63[22], and IEEE P1363[23]. In the protocol, the notation $||$ denotes concatenation.

Protocol 2

1. A generates a random integer r_A, $1 \leq r_A \leq n-1$, computes the point $R_A = r_A P$, and sends $(R_A, Cert_A)$ to B.
2. B generates a random integer r_B, $1 \leq r_B \leq n-1$, computes the point $R_B = r_B P$, and sends $(R_B, Cert_B)$ to A.
3. A computes $Z_s = w_A W_B$ and $Z_e = r_A R_B$
4. B computes $Z_s = w_B W_A$ and $Z_e = r_B R_A$
5. The session key is $k = kdf(Z_s || Z_e)$.

Remark

- A and B commonly compute $K = (w_A w_B P)||(r_A r_B P)$.
- It does not possess the K-CI attribute, since an adversary who learns w_A can impersonate any other entity B to A.

B.3 Two-Pass MQV

The MQV[14] is proposed by Law *et al.*, which is in the draft standards ANSI X9.42[21], ANSI X9.63[22], and IEEE P1363[23].

The following notations are used. f denotes the bitlength of n, the prime order of the base point P; *i.e.*, $f = \lfloor log_2 n \rfloor + 1$. If Q is a finite elliptic curve point, then $\overline{Q}$ is defined as follows. Let x be the x-coordinate of Q, and let $\overline{x}$ be the integer obtained from the binary representation of x. (The value of $\overline{x}$ will depend on the representation chosen for the elements of the field $\mathbf{F}_q$). Then $\overline{Q}$ is defined to be the integer $(\overline{x} \bmod 2^{\lceil f/2 \rceil}) + 2^{\lceil f/2 \rceil}$. Observe that $(\overline{Q} \bmod n) \neq 0$[4].

Protocol 3

1. A generates a random integer r_A, $1 \leq r_A \leq n-1$, computes the point $R_A = r_A P$, and sends $(R_A, Cert_A)$ to B.
2. B generates a random integer r_B, $1 \leq r_B \leq n-1$, computes the point $R_B = r_B P$, and sends $(R_B, Cert_B)$ to A.
3. A computes $s_A = r_A + \overline{R}_A w_A \bmod n$, $K = s_A(R_B + \overline{R}_B W_B)$
4. B computes $s_B = r_B + \overline{R}_B w_B \bmod n$, $K = s_B(R_A + \overline{R}_A W_A)$
5. The session key is $k = kdf(K)$.

Remark

- A and B commonly compute $K = (s_A s_B)P = (r_A r_B + r_A w_B \overline{R}_B + r_B w_A \overline{R}_A + w_A w_B \overline{R}_A \overline{R}_B)P$. The expression for $\overline{R}_A$ uses only half the bits of the x-coordinate of R_A. This was done in order to increase the efficiency of computing K because the scalar multiplication $\overline{R}_A W_A$ can be done in half the time of a full scalar multiplication[14].
- It does not provide UK-S attribute, which has recently been observed by Kaliski[13].

C New Key Agreement Protocols

This section proposes the new two-pass AK, AKC and two-pass unilateral AKC protocols. Domain parameters and static keys are set up and validated as described in Section 2.

C.1 AK Protocol

Protocol 4 provides more desirable security attributes than AK protocols described in Section 2.

Protocol 4

If A and B do not previously possess authentic copies of each other's static public keys, the certificates should be included in the flows[14].

1. A generates a random integer r_A, $1 \leq r_A \leq n-1$, computes the point $R_A = r_A P$, and sends this to B.
2. B generates a random integer r_B, $1 \leq r_B \leq n-1$, computes the point $R_B = r_B P$, and sends this to A.
3. A does an embedded key validation of R_B. If the validation fails, then A terminates the protocol run with failure. Otherwise, A computes $K = cr_A W_B + c(w_A + r_A)R_B$. If $K = \mathcal{O}$, then A terminates the protocol run with failure.
4. B does an embedded key validation of R_A. If the validation fails, then B terminates the protocol run with failure. Otherwise, B computes $K = cr_B W_A + c(w_B + r_B)R_A$. If $K = \mathcal{O}$, then B terminates the protocol run with failure.
5. The session key is $k = kdf(K)$.

Multiplication by c ensures that the shared secret K is a point in the subgroup of order n in $E(\mathbf{F}_q)$ to protect against small subgroup attack as suggested in [15]. The check $K = \mathcal{O}$ ensures that K is a finite point[14].

A and B computes the shared secret $K = c(r_A w_B + r_B w_A + r_A r_B)P$.

Security

Although the security of Protocol 4 has not been formally proven in a model of distributed computing[5], heuristic arguments suggest that Protocol 4 provides mutual IKA[14]; entity A is assured that no other entity aside from B can possibly learn the value of a particular secret key, since the secret key can be computed only by entity who knows B's private keys, r_B and w_B, vice versa. In addition, Protocol 4 also appears to have many desirable security attributes listed in Section 1 as follows.

The protocol provides *known-key security.* Each run of the protocol between two entities A and B should produce a unique session key which depends on r_A and r_B. Although an adversary has learned some other session keys, he can't compute $r_A w_B P$, $r_A w_A P$ and $r_A r_B P$ from them, because he doesn't know ephemeral private keys r_A and r_B. Therefore the protocol still achieve its goal in the face of the adversary.

It also possesses *forward secrecy* provided that EKA of all session keys is supplied. Suppose that static private keys w_A and w_B of two entities are compromised. However, the secrecy of previous session keys established by honest entities is not affected, because it is difficult by the elliptic curve Diffie-Hellman problem that an adversary gains $r_A r_B P$ of the shared secret from $r_A P$ and $r_B P$ which he can know.

It resists *key-compromise impersonation.* Though A's static private key w_A is disclosed, this loss does not enable an adversary to impersonate other entity B to A, since the adversary still faces the elliptic curve Diffie-Hellman style problem of working out $r_A w_B P$ from w_A, r_B, $r_A P$, $r_B P$, $w_A P$ and $w_B P$ to learn the shared secret.

It also prevents *unknown key-share.* According to the assumption of this protocol that CA has verified that A possesses the private key w_A corresponding to her static public key Y_A, an adversary E can't register A's public key Y_A as its own and subsequently deceive B into believing that A's messages are originated from E. In addition, it does not have the *duplicate-signature key selection (DSKS)* property[6]. Therefore B cannot be coerced into sharing a key with entity A without B's knowledge.

Performance

From A's point of view, the dominant computational steps in a run of Protocol 4 are the scalar multiplications $r_A P$, $r_A W_B$ and $(w_A + r_A)R_B$. Refer to an efficient scalar multiplication method using Frobenius expansions suggested by Cheon *et al*[9]. Hence the work required by each entity is 3 (full) scalar multiplications. Since $r_A P$ and $r_A W_B$ can be computed off-line by A, the on-line work required by each entity is only 1 scalar multiplications. (if A and B do previously possess authentic copies of each other's static public key)

Instead of $k = kdf(c(r_A w_B + r_B w_A + r_A r_B)P)$, the following variations can be used as the session key.

(i) $k = kdf(c(r_A w_B + r_B w_A)P || c r_A r_B P)$.
A computes $k = kdf(c r_A W_B + c w_A R_B || c r_A R_B)$, and B does $k = kdf(c r_B W_A + c w_B R_A || c r_B R_A)$.

(ii) $k = kdf(c r_A w_B P || c r_B w_A P || c r_A r_B P)$.
A computes $k = kdf(c r_A W_B || c w_A R_B || c r_A R_B)$, and B does $k = kdf(c w_B R_A ||$
$c r_B W_A || c r_B R_A)$.

The protocol using these session keys requires 4 scalar multiplications and if precomputations are discounted, the total number of on-line scalar multiplications per entity is 2. Therefore the using of these will cause some degradation in performance of the protocol.

C.2 AKC Protocol

This section describes the AKC variant of Protocol 4. Protocol 5 (AKC protocol) is derived from Protocol 4 (AK protocol) by adding the MACs of the flow number, identities, and the ephemeral public keys. Here, MACs are used to provide key confirmation, and $\mathcal{H}_1$ and $\mathcal{H}_2$ are (independent) key derivation functions. Practical instantiations of $\mathcal{H}_1$ and $\mathcal{H}_2$ include $\mathcal{H}_1$=SHA-1$(01, z)$ and $\mathcal{H}_2$=SHA-1$(10, z)$[14].

Protocol 5

1. A generates a random integer r_A, $1 \le r_A \le n-1$, computes the point $R_A = r_A P$, and sends this to B.
2. (a) B does an embedded key validation of R_A. If the validation fails, then B terminates the protocol run with failure.
   (b) Otherwise, B generates a random integer r_B, $1 \le r_B \le n-1$ and computes the point $R_B = r_B P$.
   (c) B computes $K = c r_B W_A + c(w_B + r_B) R_A$. If $K = \mathcal{O}$, then B terminates the protocol run with failure. The shared secret is the point K.
   (d) B uses the x-coordinate z of the point K to compute two shared keys $k = \mathcal{H}_1(z)$ and $k' = \mathcal{H}_2(z)$.
   (e) B computes $\mathrm{MAC}_{k'}(2, ID_B, ID_A, R_B, R_A)$ and sends this together with R_B to A.
3. (a) A does an embedded key validation of R_B. If the validation fails, then A terminates the protocol run with failure.
   (b) Otherwise, A computes $K = c r_A W_B + c(w_A + r_A) R_B$. If $K = \mathcal{O}$, then A terminates the protocol run with failure.
   (c) A uses the x-coordinate z of the point K to compute two shared keys $k = \mathcal{H}_1(z)$ and $k' = \mathcal{H}_2(z)$.
   (d) A computes $\mathrm{MAC}_{k'}(2, ID_B, ID_A, R_B, R_A)$ and verifies that this equals what was sent by B.
   (e) A computes $\mathrm{MAC}_{k'}(3, ID_A, ID_B, R_A, R_B)$ and sends to B.
4. B computes $\mathrm{MAC}_{k'}(3, ID_A, ID_B, R_A, R_B)$ and verifies that this equals what was sent by A.
5. The session key is k.

Security

AKC Protocol 5 is derived from AK Protocol 4 by adding key confirmation to the latter using the fact that key confirmation of k' implies that of k. This is done in exactly the same way AKC Protocol 2 of [5] was derived from AK Protocol 3 of [5]. Protocol 2 of [5] was formally proven to be a secure AKC protocol. Heuristic arguments suggest that Protocol 5 has all the desirable security attributes listed in Section 1[14].

Performance

Since MACs can be computed efficiently, this method of adding key confirmation to an AK protocol does not place a significant computational burden on the key agreement mechanism. However, the number of messages exchanged is increased one more[4].

C.3 Two-Pass Unilateral AKC Protocol

This section describes a new two-pass authenticated key agreement protocol providing unilateral key confirmation. In practice, key agreement protocol is that two entities agree a secret key being used for subsequently cryptographic communication. An entity, initiator, who wants to communicate with a second entity, responder, first sends his information to the responder. And then he receives any information and an encrypted message from the responder, who has computed the secret key with information of the initiator and his own to use in message encryption. He also computes the secret key, and assures the responder possesses the particular secret key through verifying the encrypted message from the responder. Such process is possible by only exchanging two flows.

Protocol 6 is a two-pass key agreement protocol which provides mutual IKA and unilateral EKA. It also provides many desirable security attributes. In protocol, $S_A(M)$ denotes A's signature over M and ID_A denotes A's identity information.

Protocol 6

1. (a) A generates a random integer r_A, $1 \leq r_A \leq n-1$, and computes the point $R_A = r_A P$
   (b) A computes $S_A(R_A, ID_A)$ and sends this with R_A to B.
2. (a) B does an embedded key validation of R_A. If the validation fails, then B terminates the protocol run with failure.
   (b) B verifies the signature $S_A(R_A, ID_A)$ with A's public key. If the verification fails, then B terminates the protocol run with failure.
   (c) Otherwise, B generates a random integer r_B, $1 \leq r_B \leq n-1$, and computes the point $R_B = r_B P$.
   (d) B computes $K = cr_B W_A + c(w_B + r_B)R_A$. If $K = \mathcal{O}$, then B terminates the protocol run with failure.
   (e) B uses the x-coordinate z of the point K to compute two shared keys $k = \mathcal{H}_1(z)$ and $k' = \mathcal{H}_2(z)$.
   (f) B computes $\mathrm{MAC}_{k'}(ID_B, ID_A, R_B, R_A)$ and sends this with R_B to A.
3. (a) A does an embedded key validation of R_B. If the validation fails, then A terminates the protocol run with failure.
   (b) Otherwise, A computes $K = cr_A W_B + c(w_A + r_A)R_B$. If $K = \mathcal{O}$, then A terminates the protocol run with failure.
   (c) A uses the x-coordinate z of the point K to compute two shared keys $k = \mathcal{H}_1(z)$ and $k' = \mathcal{H}_2(z)$.
   (d) A computes $\mathrm{MAC}_{k'}(ID_B, ID_A, R_B, R_A)$ and verifies that this equals what was sent by B.
4. The session key is k.

Security

Entity A sends R_A and $S_A(R_A, ID_A)$ to B, then the signature of A provides entity authentication (the process whereby one entity is assured of the identity of a second entity involved in a protocol, and that the second has actually participated) of A to B. The MAC sent from B to A provides entity authentication of B to A and key confirmation of B to A. Subsequently A and B are on any communication using the shared key. Then an encrypted message sent from A gives an assurance to B that A actually possesses the key (*i.e.*, key confirmation of A to B) during a 'real-time' communication. Protocol 6 also provides K-KS, FS, K-CI and UK-S attributes by the same reasons as Protocol 5.

Performance

This protocol requires additional computations of the signature. However, since $S_A(R_A, ID_A)$ can be precomputed by A, the computations of entity B for verifying the signature only are increased in the on-line work. Most of all, Protocol 6 has the advantage in the number of flow than Protocol 5, because a flow is the dominant burden in performance.

D Comparison

This section compares the security attributes and performance of all the protocols discussed so far. Table 1 presents the shared secret of Protocol 1 to Protocol 6, and A's computations for generating the shared secret. The shared secret of Protocol 1 is composed of $r_A w_B P$ and $r_B w_A P$, Protocol 2 is $r_A r_B P$ and $w_A w_B P$, Protocol 3 is $r_A r_B P$, $r_A w_B \overline{R}_B P$, $r_B w_A \overline{R}_A P$ and $w_A w_B \overline{R}_A \overline{R}_B P$, and Protocols 4, 5 and 6 are $r_A r_B P$, $r_A w_B P$ and $r_B w_A P$, respectively.

Protocol	Shared secret (K)	A's computations
Protocol 1 (MTI/A0)	$(r_A w_B + r_B w_A)P$	$K = r_A W_B + w_A R_B$
Protocol 2 (Unified Model)	$(w_A w_B)P \| (r_A r_B)P$	$K = w_A W_B \| r_A R_B$
Protocol 3 (MQV)	$(r_A + \overline{R}_A w_A)(r_B + \overline{R}_B w_B)P$	$\overline{R}_A = R_A \bmod 2^{\lceil f/2 \rceil} + 2^{\lceil f/2 \rceil}$ $s_A = r_A + \overline{R}_A w_A \bmod n$ $K = s_A(R_B + \overline{R}_B W_B)$
Protocols 4,5,6 (proposed)	$(r_A r_B + r_A w_B + r_B w_A)P$	$K = r_A W_B + (w_A + r_A)R_B$

Table 1. The Shared Secret of AK protocols and A's required computations

Table 2 contains a summary of the services that are believed to be provided by the AK and AKC protocols discussed in Sections 2 and 3. The services are discussed in the context of an entity A who has successfully executed the key agreement protocol over an open network wishing to establish keying data with entity B. The provision of these assurances is considered that both A and B are honest and have always executed the protocol correctly[4].

In Table 3, the number of scalar multiplications required in each protocol is compared. Protocols 1, 2, 4 and 5 commonly require 3 scalar multiplications without precomputations (quantities involving the entity's static and ephemeral keys and the other entity's static keys) and require 1 scalar multiplications with precomputations. Protocol 3 requires 2.5 and 1.5 scalar multiplications, respectively.

Protocol	IKA	EKA	K-KS	FS	K-CI	UK-S
Protocol 1 (AK)	○	×	○	×	○	○
Three-pass MTI/A0 (AKC)	○	○	○	×	○	○
Protocol 2 (AK)	○	×	$\dagger^a$	$\dagger^b$	×	○
Three-pass Unified Model (AKC)	○	○	○	○	×	○
Protocol 3 (AK)	○	×	○	$\dagger^b$	○	×
Three-pass MQV (AKC)	○	○	○	○	○	○
Protocol 4 (proposed AK)	○	×	○	$\dagger^b$	○	○
Protocol 5 (proposed AKC)	○	○	○	○	○	○
Protocol 6 (proposed unilateral AKC)	○	▷	○	○	○	○

○ : the assurance is provided to A no matter whether A initiated the protocol or not.
× : the assurance is not provided to A by the protocol.
▷ : the assurance is provided to A only if A is the protocol's initiator.
$\dagger^a$: Here the technicality hinges on the definition of what contributes another session key. The service of known-key security is certainly provided if the protocol is extended so that explicit authentication of all session keys is supplied[4].
$\dagger^b$: Again the technicality concerns key confirmation. Both protocols provide forward secrecy if explicit authentication is supplied for all session keys. If not supplied, then the service of forward secrecy cannot be guaranteed[4].

Table 2. Security services offered by AK and AKC protocols

Total Number	Without precomputations	With precomputations
Protocol 1	3	1
Protocol 2	3	1
Protocol 3	2.5	1.5
Protocols 4,5	3	1

The number of scalar multiplications per entity of key agreement protocols

As noted in Section 3, MACs can be computed efficiently and hence the AKC variants have essentially the same computational overhead as their AK counterparts[4]. However, they require an extra flow which is dominated in performance. In this sense, Protocol 6 is considerable because it requires two flows like AK Protocol 4. It satisfies more security attributes than Protocol 4 and the same ones as Protocol 5 except merely providing EKA of B to A. However, since EKA of A to B can be provided in subsequently cryptographic communication, it doesn't matter in security to omit the service in key agreement protocol. On the other hand, the computation overhead of the signature is required. As the signature can be precomputed, the computation for verifying the signature only is added to the on-line work. Consequently, Protocol 6 is said to be more efficient than Protocol 5 and more secure than Protocol 4.

As shown in Table 3, Protocol 4 provides more desirable security attributes than other AK protocols suggested so far. If EKA is additionally provided in AK protocols, they possess many desirable security attributes. By the way, the AKC MTI/A0 still does not provide FS and the AKC Unified Model does not K-CI, while the AKC MQV provides UK-S which the AK MQV doesn't exhibit.

AKC Protocol 5 provides all security attributes in Table 2 as well as the AKC MQV. The AKC MQV requires 2.5 scalar multiplication per entity without precomputations and 1.5 scalar multiplication with precomputations like Protocol 3, while Protocol 5 requires 3 and 1 scalar multiplications, respectively. So, in the case that precomputations are discounted, Protocol 5 is more efficient than the MQV protocol in on-line processing.

E Concluding Remarks

Protocol 4 (AK protocol) has been proposed to provide the desirable security attributes which are not provided by the MTI/A0, two-pass Unified Model and two-pass MQV protocols; known-key security, forward secrecy, key-compromise impersonation and unknown key-share attributes.

Protocol 5 (AKC protocol) derived from Protocol 4 provides all the security requirement discussed in [6]; implicit key authentication, explicit key authentication, known-key security, forward secrecy, key-compromise impersonation and unknown key-share attributes. The three-pass MQV protocol (AKC protocol) also possesses the same security attributes as Protocol 5. However, if precomputations (quantities of the entity's static and ephemeral keys and the other entity's static keys) is discounted, Protocol 5 requires less scalar multiplication than the MQV protocol.

Protocol 6, the two-pass unilateral AKC protocol, has been designed as the other variant of Protocol 4. It provides not only mutual implicit key authentication but also unilateral explicit key authentication. Instead, it requires the computation overhead related to the signature and the MAC. However, since the signature can be precomputed and the MAC can be efficiently computed, in practice, the computations increased in on-line work are only quantities for verifying the signature. So, Protocol 6 has the advantage of security than Protocol 4 and of performance than Protocol 5.

The proposed protocols has been shown informally by heuristic arguments to provide the claimed attributes of security. The following three conjectures are required to show that the security of the proposed protocols can be rigorously proved in Bellare-Rogaway model[3] of distributed computing as done in [5].

(i) Protocol 4 is a secure AK protocol provided that ECDHS (Elliptic Curve Diffie-Hellman Scheme) is secure.
(ii) Protocol 5 is a secure AKC protocol provided that ECDHS and MAC are secure, and $\mathcal{H}_1$ and $\mathcal{H}_2$ are independent random oracles.
(iii) Protocol 6 is a secure unilateral AKC protocol provided that ECDHS, MAC and digital signature scheme are secure, and $\mathcal{H}_1$ and $\mathcal{H}_2$ are independent random oracles.

References

1. R. Ankney, D. Hohnson and M. Matyas, "The Unified Model", contribution to X9F1, October 1995. 241

2. J. Baek and K. Kim, "Remarks on the Unknown Key Share Attacks", To appear Trans. of IEICE. 238
3. M. Bellare and P. Rogaway, "Entity Authentication and Key Distributions - the Three Party Case", *Advances in Cryptology – Crypto '93*, LNCS 773, Springer-Verlag, pp232-249, 1994. 248
4. S. Blake-Wilson and A. Menezes, "Authenticated Diffie-Hellman Key Agreement Protocols", Proceedings of the 5th Annual Workshop on Selected Areas in Cryptography (SAC '98), LNCS 1556, Springer-Verlag, pp339-361, 1999. 238, 238, 238, 238, 239, 241, 244, 246, 247, 247, 247
5. S. Blake-Wilson, C. Johnson and A. Menezes, "Key Agreement Protocols and their Security Analysis", Proceedings of the sixth IMA International Conference on Cryptography and Coding, LNCS 1355, Springer-Verlag, pp30-45, 1997. 242, 244, 244, 244, 248
6. S. Blake-Wilson and A. Menezes, "Unknown Key-Share Attacks on the Station-To-Station (STS) Protocol", Technical report CORR 98-42, Univ. of Waterloo, 1998. 237, 238, 243, 248
7. I. F. Blake and G. Seroussi, *Elliptic Curves in Cryptography*, London Mathematical Society Lecture Note Series 265, Cambridge University Press, 1999. 239
8. D. Boneh and R. Lipton, "Algorithms for Black-Box Fields and their Application to Cryptography", *Advances in Cryptology – Crypto '96*, LNCS 1109, Springer-Verlag, pp283-297, 1996. 239
9. J. Cheon, S. Park, C. Park and S. Hahn, "Scalar Multiplication on Elliptic Curves by Frobenius Expansions", ETRI J., Vol.21, No.1, pp27-38, March 1999. 243
10. W. Diffie and M. E. Hellman, "New Directions in Cryptography", IEEE Trans. on Information Theory, 22, pp644-654, 1976. 239
11. K. C. Goss, "Cryptographic Method and Apparatus for Public Key Exchange with Authentication", U.S. patent 4,956,865, September 11, 1990. 240
12. D. Johnson, Contribution to ANSI X9F1 working groups, June 1997. 240
13. B. Kaliski, Contribution to ANSI X9F1 and IEEE P1363 working groups, June 1998. 241
14. L. Law, A. Menezes, M. Qu, J. Solinas and S. Vanstone, "An Efficient Protocol for Authenticated Key Agreement Protocol ", Technical report CORR 98-5, Univ. of Waterloo, Canada, March 1998. 238, 238, 239, 239, 239, 239, 240, 240, 240, 241, 241, 242, 242, 242, 244, 244
15. C. Lim and P. Lee, "A Key Recovery Attack on Discrete Log-based Schemes using a Prime Order Subgroup", *Advances in Cryptology – Crypto '97*, LNCS 1294, Springer-Verlag, pp249-263, 1997. 242
16. U. Maurer and S. Wolf, "Diffie-Hellman Oracles", *Advances in Cryptology – Crypto '96*, LNCS 1109, Springer-Verlag, pp283-297, 1996. 239
17. T. Matsumoto, Y. Takashima and H. Imai, "On Seeking Smart Public-Key Distribution Systems", Trans. of IEICE, Vol.E69, pp99-106, 1986. 240
18. A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997 237, 237, 237, 238, 238
19. National Security Agency, "SKIPJACK and KEA Algorithm Specification", Version 2.0, May 29, 1998. 240
20. Y. Yacobi, "A Key Distribution Paradox", *Advances in Cryptology – Crypto '90*, LNCS 537, Springer-Verlag, pp268-273, 1991. 240
21. ANSI X9.42, *Agreement of Symmetric Algorithm Keys using Diffie-Hellman*, working draft, May 1998. 241, 241
22. ANSI X9.63, *Elliptic Curve Key Agreement and Key Transport Protocols*, working draft, July 1998. 241, 241
23. IEEE P1363, *Standard Specifications for Public-Key Cryptography*, working draft, July 1998. 241, 241

Anonymous Traceability Schemes with Unconditional Security

Reihaneh Safavi-Naini and Yejing Wang

School of IT and CS, University of Wollongong,
Wollongong 2522, Australia
{rei,yw17}@uow.edu.au

Abstract. In this paper we propose a model for anonymous traceability scheme with unconditional security. The model uses two subsystems: an authentication code with arbiter (A^3-code) and an asymmetric traceability with unconditional security. We analyse security of the system against possible attacks and show that security parameters of the system can be described in terms of those of the two subsystems. We give an example construction for the model.

Keywords: Digital fingerprint, traceability schemes, anonymity, unlinkability, authentication codes.

A Introduction

One of the main challenges in the distribution of digital products is preventing illegal copying and reselling. Fingerprinting allows a merchant to insert the identity information of the buyer into the copy and so if an illegal copy is found it is possible to identify the malicious buyer. A *fingerprint* is a sequence of marks that is unique to a buyer. A group of buyers who have bought different copies of the same product containing different fingerprints, can compare their copies, find some of the marks and alter the marks to produce a pirate copy. Collusion secure fingerprinting [1], [2] provides protection against collusion of buyers and ensures that collusion of a group of buyers cannot frame another user, or construct an illegal copy that cannot be traced to any colluder. Boneh and Shaw showed totally c-secure codes, that is codes that allow one traitor to be identified, do not exist and proposed a probabilistic approach to collusion secure fingerprinting. Traitor tracing schemes are also studied in the context of broadcast encryption systems, and in unconditionally secure [10] and computationally secure [6] models. In the traditional model of traitor tracing schemes the merchant is trusted and protection is against colluding buyers. *Asymmetric tracing schemes* provide protection against a dishonest merchant who may frame the buyer by inserting his fingerprint in a second copy of the object. In [7], [8] computationally secure *anonymous tracing schemes* are proposed. Anonymous schemes are asymmetric: that is the merchant is not assumed honest. The merchant only knows the buyer through a pseudo-identity and cannot *link* various purchases made by the same buyer.

In this paper, we propose a model for an unconditionally secure anonymous traceability scheme that provides security against collusion of malicious buyers and cheating of other participants including the merchant. The model employs a coin system similar to the one proposed in [7] to allow the buyer to hide his identity while permitting the merchant to have enough information about the buyer to reveal his identity with the help of a third party. The system uses asymmetric authentication codes [5] and asymmetric traceability schemes [9] as building blocks and its security is directly related to the security of the two underlying schemes. We give an example construction and prove its security.

The paper is organized as follows. Asymmetric authentication codes and traceability schemes are recalled in section B. The model is described in section C and the construction of an unconditionally secure coin system is given in section D. An example for the scheme is shown in section E, and finally, the paper is concluded in section F.

B Preliminaries

B.1 A^3-Codes

A^3-codes are introduced by [3] and further studied in [4] and [5]. In an A^3-code there is a sender, a receiver and an arbiter, none of them assumed trusted. The code allows the sender to send an authenticated message to the receiver over a public channel such that the chance of success of an outsider or one of the insiders in the attack is less than a designed value. An A^3-code is defined by a triple (E_T, E_R, E_A) of matrices. The matrices E_T, E_R and E_A are used by the sender, the receiver, and the arbiter, respectively. Columns of E_T are labelled by $s_1, s_2, \cdots, s_k$, that form S, the set of source states. A source state s will be encoded to a message in the message space $M = \{m_1, m_2, \cdots\}$, $|M| > |S|$, using a row of E_T that consists of k distinct messages $m_1, m_2, \cdots, m_k$, and defines an encoding rule for the sender. We say that a pair (s, m) is valid for encoding rule e, if m appears in the column s of e. Columns of E_R are labelled by $m_1, m_2, \cdots, m_n$. Each row of E_R consists of multiple occurrence of symbols of the set $\{s_1, s_2, \cdots, s_k, -\}$, and defines a verification rule for the receiver. We say that a pair (s, m) is valid for a verification rule v if s appears in the column m in row v. Similarly, columns of E_A are labelled by $m_1, m_2, \cdots, m_n$. Each row of E_A consists of multiple occurrence of symbols from the set $\{s_1, s_2, \cdots, s_k, -\}$, and defines an arbitration rule for the arbiter. We say that a pair (s, m) is valid for an arbitration rule a if s appears in column m of row a.

An A^3-code can protect against the following attacks.

1. Attack O_i: Observing a sequence of i legitimate pairs $(s_1, m_1), (s_2, m_2), \cdots, (s_i, m_i)$, the opponent places another pair $(s, m) \neq (s_1, m_1), \cdots, (s_i, m_i)$ into the channel. He is successful if both the receiver and the arbiter accept (s, m) as a valid pair.

2. Attack R_i: Receiving a sequence of i legitimate pairs $(s_1, m_1), (s_2, m_2), \cdots, (s_i, m_i)$, and using her key (a verification rule), the receiver claims that she has received a pair $(s, m) \neq (s_1, m_1), \cdots, (s_i, m_i)$. She is successful if the arbiter accepts (s, m) as a valid pair under her arbitration rule.
3. Attack A_i: Knowing a sequence of i legitimate pairs $(s_1, m_1), (s_2, m_2), \cdots, (s_i, m_i)$, and using her key (an arbitration rule), the arbiter puts another pair $(s, m) \neq (s_1, m_1), \cdots, (s_i, m_i)$ into the channel. She is successful if the pair (s, m) is valid for the receiver.
4. Attack T: Using his key (an encoding rule) e, the sender sends a fraudulent pair (s, m) which is not valid under e. He succeeds if both the receiver and the arbiter accept the pair.

Denote by $P_{O_i}, P_{R_i}, P_{A_i}$ and P_T the probability in successful attack of O_i, R_i, A_i and T, respectively. To construct a secure and efficient A^3-code, (i) the chance of success in all attacks, $P_{O_i}, P_{R_i}, P_{A_i}$ and P_T must be as small as possible, and (ii) the size of the required key spaces $|E_T|, |E_R|, |E_A|$, and the message space $|M|$ must be as small as possible.

B.2 Asymmetric Tracing

Asymmetric tracing schemes with unconditional security are proposed in [9]. In these schemes the merchant is not trusted and insertion of a fingerprint requires collaboration of the merchant and an arbiter. In the asymmetric tracing schemes proposed in [9] the merchant only knows part of the fingerprint inserted in the object. This ensures that the merchant's chance of success in framing a buyer will be limited. Fingerprinting is performed in two steps: the merchant inserts part of the fingerprint and passes the partially fingerprinted object to the arbiter who then inserts the rest of the fingerprint. The fully fingerprinted object is given to the buyer directly without the merchant being able to access it. In the asymmetric tracing schemes collusion-tolerance is retained as in the symmetric schemes: that is whenever a certain number of buyers collude to produce a pirate copy, the merchant individually can trace it back to one of the traitors. If the buyer does not agree with the merchant's accusation, the arbiter runs an arbitration algorithm on the whole fingerprint and accepts or rejects the merchant's verdict. This ensures that an innocent buyer will never be accused and always one of the colluding buyers can be traced.

C Anonymous Tracing

There are n buyers $\mathcal{B}_1, \mathcal{B}_2, \cdots, \mathcal{B}_n$, a merchant $\mathcal{M}$, a registration center $\mathcal{R}$, a coin issuer (or bank) $\mathcal{I}$, and an arbiter $\mathcal{A}$. Suppose a buyer wants to buy a digital object, for example a software, from the merchant but does not want to reveal his identity. The merchant fingerprints the object for the buyer. To make fingerprints anonymous, the buyer first obtains a pseudo-identity from $\mathcal{R}$, by visiting him and providing identification documents. $\mathcal{R}$ stores the real identity of the buyer and his pseudo-identity in a secure database. Next the buyer presents his pseudo-identity to $\mathcal{I}$ who can verify the correctness of the pseudo-identity and issue coins to the buyer. A coin is verifiable by the merchant and the arbiter who collaboratively insert the fingerprint into the object. Coins have no monetary value and main purpose is to provide revocable anonymity. The coins issued to a buyer are *unlinkable*, that is they do not point to the same buyer.

The system operation has four phases: registration, fingerprinting, tracing and arbitration.

1. *Registration* has two subphases: pseudo-identity generation and coin withdrawal. Registration center generates pseudo-identity secretly, and gives it to the buyer. The buyer shows his pseudo-identity to the coin issuer and receives up to ℓ coins.
2. In *fingerprinting* phase, system works in the same way as an asymmetric traceability scheme. The only difference is that both the merchant and the arbiter, use their knowledge of the coin to check and store a coin that the buyer provided in lieu of his real identity.
3. *Tracing* has two phases, tracing an object to a coin and revealing the identity. Tracing a pirate copy back to one coin is done by the merchant without assistance of any other principal. Revealing an identity needs cooperation of the merchant, the coin issuer, the arbiter and the registration center. This ensures that only the identity of the correctly accused buyer will be revealed.
4. *Arbitration* is called when the buyer disagrees with the merchant's accusation. There are two possibilities: (*i*) Arbitration is after revealing the identity of the buyer in which case the arbiter directly rules between the buyer and the merchant; and (*ii*) Arbitration is before revealing the buyer's identity. In this latter case, the registration center informs the buyer of the accusation and in the case of disagreement acts on his behalf and approaches the arbiter.

The process of tracing and revealing the identity will be as follows.

Suppose the merchant finds a pirate object O. He runs his tracing algorithm to trace the object to one of the colluding buyers and finds the fingerprint ϕ of a traitor. He searches his database for the fingerprint and finds C_ϕ which is the coin matching ϕ. Then, he presents the three tuple (O, ϕ, C_ϕ) to the arbiter who will verify that (*i*) O includes the fingerprint ϕ, and (*ii*) that ϕ matches the coin C_ϕ. This is to ensure that a cheating merchant cannot misuse the system to obtain the pseudo-identity of an innocent buyer.

Next, the merchant presents (O, ϕ, C_ϕ) together with the verification information provided by the arbiter (alternatively the arbiter sends this information directly) to $\mathcal{I}$ and requests the buyer's pseudo-identity. $\mathcal{I}$ verifies the presented information and uses his coin database to find the pseudo-identity. The merchant presents the pseudo-identity to the registration center who has stored the pair (ID, I_{num}) in a secure database during the registration phase. The two possibilities discussed in the *Arbitration* phase above applies to this stage. That is, either the registration centre revealing the identity and allowing the merchant to proceed with the accusation, or the registration centre approaches the buyer and only if there is no objection, reveals the identity information to the merchant. If the buyer does not accept the accusation the registration centre presents the information to the arbiter who will accepts, or rejects, the accusation. The buyer's identity is only revealed if the accusation is accepted by the arbiter. We note that the difference between the two alternatives is mainly operational and does not affect the design of the system.

The process of arbitration will be as follows. If the buyer does not accept the accusation, he presents his coin to the arbiter. Arbiter will obtain the merchant's evidence cosisting of (O, ϕ, C_ϕ) and uses this information to independently find the identity of the buyer by contacting $\mathcal{I}$ and $\mathcal{R}$. If the identity matches user's, then the accusation is held valid. This ensures that the merchant cannot cheat at the stage of either submitting pseudo-identity to $\mathcal{R}$ or after that.

From the above description it is seen that the role of the arbiter in the system is twofold. Firstly to verify the information presented by the merchant before the buyer's pseudo-identity is revealed, and secondly to arbitrate between the merchant and the buyer.

In the following we describe a system with the above four phases using an unconditionally secure asymmetric authentication code and an unconditionally secure asymmetric fingerprinting scheme. The A^3-code is used during the registration and coin withdrawal stage. At the end of this stage, the buyer has a number of coins that he can use for making purchases. The coins are verifiable by the merchant and the arbiter and cannot be forged or tampered with without being caught with a high chance.

D A Coin System

In this section we set up a coin system using an A^3-code. The system provides unconditional security and by choosing an appropriate A^3-code the chance of breaking the security of the system can be made small.

D.1 Construction of Coins

Suppose there is an A^3-code (E_G, E_M, E_A) where E_G denotes the sender's code, E_M denotes the receiver's code and E_A denotes the arbiter's code. The above three codes are public and are used by the registration center, the merchant, and the arbiter respectively. We also use the same notations to refer to the key space in each of the three codes and so for example E_G also refers to the set of keys used by the registration center.

Let f, g, h be three injective functions:

$$f : E_G \longrightarrow \{0,1\}^{n_1}$$
$$g : E_M \longrightarrow \{0,1\}^{n_2}$$
$$h : E_A \longrightarrow \{0,1\}^{n_3}$$

where $n_1 >> \log|E_G|$, $n_2 >> \log|E_M|$, and $n_3 >> \log|E_A|$. We note that f being injective and $2^{n_1} >> |E_G|$ implies that if f is known, the knowledge of

$f(e)$ can uniquely determine e. However without the knowledge of f, choosing a binary string of length of n_1 that is the image of an element of E_G has a very small chance.

We will assume that:
A1: f is secret and is only known by $\mathcal{R}$ and $\mathcal{I}$.
A2: g is secret and is only known by $\mathcal{M}$ and $\mathcal{I}$.
A3: h is secret and is only known by $\mathcal{A}$ and $\mathcal{I}$.

To obtain a coin, a buyer $\mathcal{B}$ must follow the following steps.

1. Identity verification: for this, the buyer needs to prove his identity $ID_{\mathcal{B}}$ to $\mathcal{R}$. $\mathcal{R}$ verifies that $\mathcal{B}$ is actually who he is claiming to be, selects an element $e \in E_G$ and gives $f(e)$ to $\mathcal{B}$ as $\mathcal{B}$'s pseudo-identity. It also securely stores $(e, ID_{\mathcal{B}})$ for the tracing phase.
2. $\mathcal{B}$ presents $f(e)$ to $\mathcal{I}$. $\mathcal{I}$ checks the validity of $f(e)$ by verifying that it is in the image set of f. Next $\mathcal{I}$ selects an element $v \in E_M$ and an element $a \in E_A$ according to the key distribution of A^3-code. That is the three tuple (e, v, a) form a valid key set for the A^3-code.
3. $\mathcal{I}$ produces a coin $C = (s, m, g(v), h(a))$, not used before, where m corresponds to the source state s under the encoding rule e. Finally, $\mathcal{I}$ securely stores $(C, f(e))$. At this stage $\mathcal{I}$ can produce many coins using the same rule and each of them matches the same e. Note that the key pairs (v, a) might not be the same.

Although the merchant does not use $h(a)$, but he stores it as part of the whole coin $(s, m, g(v), h(a))$. This information could be used during the tracing phase. That is if he can show that the whole coin that he has stored is the same he has received from the buyer, then finding the pseudo-identity by $\mathcal{I}$ does not need the input from the arbiter. However arbiter's participation at this stage is necessary.

D.2 Security of Coins

The coins are produced by $\mathcal{I}$, submitted by the buyer to the merchant, who later submits them to the arbiter to complete the fingerprint insertion process. During the tracing phase, the merchant submits an alleged coin, verified by the arbiter, to $\mathcal{I}$.

In the following we analyse security of the coin system against *coin forgery.* That is we consider the chance of success of various participants in constructing a coin which is acceptable during the fingerprinting phase (by the merchant and the arbiter) but becomes ineffective during tracing. We do not consider any attack from $\mathcal{I}$ as he can always issue coins. We recall that the aims of the coin system were (i) to hide identity of an honest buyer, and (ii) to allow revealing the identity of a dishonest buyer who has constructed a pirate copy and this accusation is proved. A successful attack means one or both of the above goals are not achieved.

The following are possible attacks, including collusion attacks, on the coin system. However we do not allow collusion between merchant and the arbiter.

An outsider using his best strategy to construct a coin. We note because $\mathcal{R}$ does not have any key information, will have the same success chance as the outsider.

A buyer may submit a fraudulent coin to the merchant during purchase.

The merchant may submit a false coin (i) to $\mathcal{A}$ during the fingerprinting, (ii) to the $\mathcal{I}$ during the tracing.

An arbiter may collude with a buyer to construct a coin and make a purchase that is not traceable.

We note that since $\mathcal{R}$ has no key information, his collusion with participants with key information (merchant, buyer and arbiter) need not be considered.

The following two possible collusions need to be considered. (i) $\mathcal{M}$ and some buyers $\mathcal{B}_1, \cdots, \mathcal{B}_i$, (ii) $\mathcal{A}$ and some buyers $\mathcal{B}_1, \cdots, \mathcal{B}_i$. Because there is no information related to buyers identities, the knowledge of the collusion of $\mathcal{M}$ and $\mathcal{B}_1, \cdots, \mathcal{B}_i$ is equivalent to $\mathcal{M}$ holding i different coins. So the success chance is no better than the merchant's chance. A similar argument can be used for the collusion of $\mathcal{A}$ and $\mathcal{B}_1, \cdots, \mathcal{B}_i$.

Based on this discussion we define the following attacks. The aim of all the attacks is to construct a *valid coin*, which will be accepted by the merchant and the arbiter, and can not be traced.

1. No key information of the A^3-code is available to the attacker(s).
   We will use P_0 to denote the best chance of success in this attack. This attack might be by an outsider, registration center or a collusion of them.
2. i valid coins, $(s_1, m_1, g(v_1), h(a_1)), \cdots, (s_i, m_i, g(v_i), h(a_i))$, are observed but $v_1, \cdots, v_i$, $a_1, \cdots, a_i$ are not known.
   We will use P_i to denote the best success chance in this attack. This attack might be by a single buyer, a collusion of buyers, a collusion of buyers and an outsider, or a collusion of buyers and the registration center.
3. i valid coins $(s_1, m_1, g(v_1), h(a_1)), \cdots, (s_i, m_i, g(v_i), h(a_i))$ are observed, $v_1, \cdots, v_i$ are known but $a_1, \cdots, a_i$ are not known.
   The best success chance in this attack is denoted by P_M. This attack can be by the merchant, or a collusion of the merchant and buyers, a collusion of the merchant and the registration center, a collusion of the merchant and an outsider.
4. i valid coins $(s_1, m_1, g(v_1), h(a_1)), \cdots, (s_i, m_i, g(v_i), h(a_i))$ are observed. $a_1, \cdots, a_i$ are known but $v_1, \cdots, v_i$ are not known.
   The best success chance in this attack is denoted by P_A. This attack can be by the arbiter, a collusion of the arbiter and buyers, a collusion of the arbiter and the registration center, a collusion of the arbiter and an outsider.

We will analyse the above four kinds of attacks.

For P_0, attacker does not have any privileged information about a coin. However since the A^3-code is public there is a chance that he can construct a valid coin. The following theorem gives the best chance of success of the attacker(s).

Theorem 1. *$P_0 = N(v, a; s, m)/2^{n_2+n_3}$, where $N(v, a; s, m)$ is the number of valid key pairs (v, a) in the underlying A^3-code such that (s, m) is valid for both v and a.*

Proof. To model a coin, the attacker has to produce (i) a valid representation $g(v)$ of a verification key $v \in E_M$, (ii) a valid representation $h(a)$ of an arbitration key $a \in E_A$, and (iii) a pair (s, m) which is valid for v and a. Since g is a secret function, the chance of producing a valid string in its image space $\{0, 1\}^{n_2}$ is $1/2^{n_2}$. Since h is secret, the chance to produce a valid string in its image space $\{0, 1\}^{n_3}$ is $1/2^{n_3}$. Since E_G is public, the attacker can choose a (s, m) that is valid according to E_G. So his success chance is $N(v, a; s, m)/2^{n_2+n_3}$.

For P_i, the attacker has i coins and wants to construct another valid coin.The following theorem gives his success chance.

Theorem 2. *$P_i = \max(P_{O_i}, N(v; s, m)/2^{n_2}, N(a; s, m)/2^{n_3})$, where P_{O_i} is the probability of attack O_i in the underlying A^3-code, $N(v; s, m)$ is the number of keys $v \in E_M$ in A^3-code such that (s, m) is valid for v, $N(a; s, m)$ is the number of keys $a \in E_A$ in A^3-code such that (s, m) is valid for a.*

Proof. Attacker has three ways of attacking the system.

One way is to keep one of the partial strings $(s, m, g(v))$ observed and to produce $h(a)$. His success chance is $N(a; s, m)/2^{n_3}$ because h is a secret function. Similarly, he can keep one of the observed partial strings $(s, m, h(a))$ and produce $g(v)$. His success chance is $N(v; s, m)/2^{n_2}$ since g is a secret function.

Alternatively, the attacker can keep one of the observed partial strings $(g(v), h(a))$ and produce (s, m). The attackers' best chance is when the i coins all have the same strings $g(v), h(a)$; that is, the i coins are,

$$\begin{aligned} C_1 &= (s_1, m_1, g(v), h(a)) \\ C_2 &= (s_2, m_2, g(v), h(a)) \\ &\vdots \\ C_i &= (s_i, m_i, g(v), h(a)) \end{aligned}$$

In this case the attacker's best strategy is to keep $g(v), h(a)$ and replace a pair (s, m) such that (i) $(s, m) \neq (s_1, m_1), (s_2, m_2), \cdots, (s_i, m_i)$, and (ii) (s, m) is valid for v and a. This probability is P_{O_i} in the underlying A^3-code.

For P_M, the attacker knows the verification key and the image under g. After observing strings $h(a_1), h(a_2), \cdots, h(a_i)$ on the coins he has a chance to model a coin. The following theorem gives the probability of this attack.

Theorem 3. *$P_M = \max(P_{R_i}, N(a; s, m)/2^{n_3})$, where P_{R_i} is the probability of attack R_i in the underlying A^3-code, and $N(a; s, m)$ is the number of keys $a \in E_A$ in A^3-code such that (s, m) is valid for a.*

Proof. Because the attacker knows possible values of v and their images $g(v)$, he has two ways of attacking the system: (i) choosing $(s, m, g(v))$ and guessing a valid $h(a)$ or, (ii) keeping one observed $h(a)$ and choosing $(s, m, g(v))$ to match $h(a)$.

After choosing $(s, m, g(v))$, the success probability of guessing a valid $h(a)$ is $N(a; s, m)/2^{n_3}$.

On the other hand, keeping an observed $h(a)$ and choosing $(s, m, g(v))$ to match $h(a)$, will have the best success chance if the i coins all contain the same partial string $h(a)$. That is the i coins are given by,

$$\begin{aligned} C_1 &= (s_1, m_1, g(v_1), h(a)) \\ C_2 &= (s_2, m_2, g(v_2), h(a)) \\ &\vdots \\ C_i &= (s_i, m_i, g(v_i), h(a)) \end{aligned}$$

In this case the attacker's best strategy is to keep $h(a)$ and replace $(s, m, g(v))$ part such that (i) $(s, m, g(v)) \neq (s_1, m_1, g(v_1)), (s_2, m_2, g(v_2)), \cdots, (s_i, m_i, g(v_i))$, and (ii) $(s, m, g(v))$ matches a. This probability is P_{R_i} in A^3-code.

Finally, P_A is similar to P_M. The attacker knows the arbiter's key and its image under h. After observing strings $g(v_1), g(v_2), \cdots, g(v_i)$ on the coins he wants to construct a valid coin. The following theorem gives the success chance of this attack. The proof is similar to theorem 3 and omitted here.

Theorem 4. $P_A = \max(P_{A_i}, N(v; s, m)/2^{n_2})$, *where* P_{A_i} *is the success probability of attack* A_i *in the underlying* A^3*-code,* $N(v; s, m)$ *is the number of keys* $v \in E_M$ *in* A^3*-code such that* (s, m) *is valid for* v.

From theorem 1 - 4 we can see that the probabilities of constructing a valid coin could be kept small by using a properly chosen A^3-code and choosing large enough n_2 and n_3.

D.3 The Number of Coins under One Registration

Suppose the underlying A^3-code in our coin system can protect against ℓ-spoofing attacks, that is using one sender's key, ℓ legitimate pairs can be sent securely. Then each $g(v)$ can be used on at most ℓ coins. This can be controlled by $\mathcal{I}$ when he issues coins. By using such A^3-code, every buyer can withdraw ℓ coins under one registration.

D.4 Anonymity and Unlinkability

The main aim of the coins is to produce anonymity and unlinkability. First we review anonymity provided by the proposed coin system. Observing a coin $C = (s, m, g(v), h(a))$, an outsider or the registration center knows the value of

s and m and nothing else, and so the coin holder's real identity is not revealed to neither the outsider nor the registration center. Also by seeing a coin $C = (s, m, g(v), h(a))$, the merchant or the arbiter will not have any information that reveals the coin holder's real identity. During withdrawal of a coin a buyer only shows his pseudo-identity $f(e)$, and so $\mathcal{I}$ does not know who he actually is. This means by showing a coin, buyers' real identities are not revealed to anyone.

Next we consider the unlinkability of the coin system. During the coin issuing phase, $\mathcal{I}$ randomly chooses $s, m, g(v)$ and $h(a)$, and so it is possible to have different $g(v)$ and $h(a)$ on a single buyer's coins and/or the same $g(v)$ and $h(a)$ on different buyers' coins. So it is not possible to link the purchases that are made by the same buyer.

E An Example

We describe an anonymous traceability scheme using an A^3-code proposed in [5] and an asymmetric traceability scheme given in [9].

Coins

Construction of the coins is based on an A^3-code with multiple use [5]. Let F_q be a field of q elements, and d be an integer such that

$$d < \frac{q}{16} - 1. \tag{1}$$

An element of $F_{q^d} \cong F_q[x]/(p(x))$ with $deg\,(p(x)) = d$ can be thought of as a polynomial $a_0 + a_1x + a_2x^2 + \cdots + a_dx^d$, $a_i \in F_q$. Define a mapping $\phi : F_{q^d} \to F_q$ as

$$\phi(a_0 + a_1x + a_2x^2 + \cdots + a_dx^d) = a_0.$$

The registration center's key is represented as,

$$e = (\tilde{e}_1, \bar{e}_2, \tilde{e}_3, \bar{e}_4), \;\; \tilde{e}_1, \tilde{e}_3 \in F_q^{\ell}, \; \bar{e}_2, \bar{e}_4 \in F_q^{d}, \tag{2}$$

the merchant's key is represented as,

$$v = (\tilde{f}_1, \bar{f}_2, f_3), \;\; \tilde{f}_1 \in F_q^{\ell}, \; \bar{f}_2 \in F_q^{d}, \;\; f_3 \in F_q, \tag{3}$$

and the arbiter's key is represented as,

$$a = (\tilde{e}_1, \bar{e}_2), \;\; \tilde{e}_1 \in F_q^{\ell}, \; \bar{e}_2 \in F_q^{d}. \tag{4}$$

A triple of keys, (e, v, a), is valid if,

$$\tilde{e}_3 = \tilde{f}_1 + \tilde{e}_1 f_3 \quad \text{and} \quad \bar{e}_4 = \bar{f}_2 + \bar{e}_2 f_3.$$

An ℓ-tuple above can be written as

$$\tilde{e}_1 = \begin{pmatrix} e_{11} \\ e_{12} \\ \vdots \\ e_{1\ell} \end{pmatrix}, \; \tilde{e}_3 = \begin{pmatrix} e_{31} \\ e_{32} \\ \vdots \\ e_{3\ell} \end{pmatrix}, \; \tilde{f}_1 = \begin{pmatrix} f_{11} \\ f_{12} \\ \vdots \\ f_{1\ell} \end{pmatrix}.$$

Let $\bar{s} \in F_q^d$, and each message be represented as

$$\mathbf{m} = (\bar{m}_1, m_2, m_3), \quad m_2, m_3 \in F_q,\ \bar{m}_1 \in F_q^d.$$

Using a key e given by (2), ℓ pairs $(\bar{s}_i, \mathbf{m}_i)$, $i = 1, \cdots, \ell$, can be securely constructed. Let

$$(\bar{s}_1, \mathbf{m}_1), (\bar{s}_2, \mathbf{m}_2), \cdots, (\bar{s}_\ell, \mathbf{m}_\ell) \tag{5}$$

be ℓ pairs, where

$$\mathbf{m}_1 = (\bar{m}_{11}, m_{21}, m_{31}), \mathbf{m}_2 = (\bar{m}_{12}, m_{22}, m_{32}), \cdots, \mathbf{m}_\ell = (\bar{m}_{1\ell}, m_{2\ell}, m_{3\ell}). \tag{6}$$

We say ℓ pairs (5) are valid for an e given by (2) if

$$\begin{cases} \bar{m}_i = \bar{s}_i \\ m_{2i} = e_{1i} + \phi(\bar{s}_i \bar{e}_2) \quad 1 \le i \le \ell \\ m_{3i} = e_{3i} + \phi(\bar{s}_i \bar{e}_4) \end{cases} \tag{7}$$

A pair $(\bar{s}, \mathbf{m})$ is valid for a key v given by (3) if

$$m_3 = f_{1i} + \phi(\bar{m}_1 \bar{f}_2) + m_2 f_3, \quad \text{for some } i,\ 1 \le i \le \ell.$$

A pair $(\bar{s}, \mathbf{m})$ is valid for a key a given by (4) if

$$m_2 = e_{1i} + \phi(\bar{m}_1 \bar{e}_2), \quad \text{for some } i,\ 1 \le i \le \ell.$$

For a registration key e one can withdraw ℓ coins in (6) satisfying (7).

Fingerprints

We use the system proposed in [9]. In [9] fingerprints correspond to blocks $B_f = \{(x, f(x)) : x \in F_q\}$ obtained from monic polynomials of degree t. Let $x_1, x_2, \cdots, x_s \in F_q$ be s fixed elements of F_q. $\mathcal{M}$ embeds $q-s$ points $(x, f(x)), x \in F_q \backslash \{x_1, \cdots, x_s\}$, as a partial fingerprint in the object and passes the object to the arbiter. The arbiter verifies that the merchant's codeword is correctly embedded, and then secretly embeds the rest of the s points $(x_1, f(x_1)), \cdots, (x_s, f(x_s))$.

In our example, we will choose t as small as possible to reduce computation. A coin has four parts, $s, \mathbf{m}, g(v), h(a)$, where $s \in F_q^d$, $\mathbf{m} \in F_q^{d+2}$, and $g(v)$ and $h(a)$ are strings of lengths n_2 and n_3, respectively, and $v \in F_q^{d+2}$ and $a \in F_q^{d+1}$. So a coin $(s, \mathbf{m}, g(v), h(a))$ is determined by a $(4d+5)$-tuple over F_q. Hence a mapping Φ such that $\Phi(C) \in F_q^{4d+5}$ for coin C and $\Phi(C_1) \neq \Phi(C_2)$ for two different coins, can be used. After receiving a coin C, $\mathcal{M}$ uses $\Phi(C)$ to generate a monic polynomial $f(x)$ of degree $t = 4d+5$. That is for $\Phi(C) = (a_1, \cdots, a_{4d+5})$, he constructs the polynomial $f(x) = a_1 + \cdots + a_{4d+5} x^{4d+4} + x^{4d+5}$. Since Φ is public, the arbiter is able to check whether the merchant's insertion of fingerprint has been correct.

There can be at most q^{2d+2} buyers in this system, and a buyer can have at most ℓ coins. For attackers, the success probabilities of producing a coin is bounded by $1/q$ [5]. Fingerprints are secure against collusion of size at most,

$$c = \lfloor \frac{-s + \sqrt{s^2 + (4d+5)q}}{4d+5} \rfloor.$$

Condition (1) ensures $c \geq 2$.

F Conclusions

In this paper we proposed a model for unconditionally secure anonymous traceability scheme. Using a coin system based on A^3-codes and an asymmetric traceability scheme ensures that if a pirate copy is found, the merchant can always identify one of the colluders. It also ensures that any innocent buyer will not be framed by either a cheating merchant or a colluding group of buyers. Using a coin system guarantees anonymity of the honest buyers and unlinkability of his different purchases.

References

1. D. Boneh and J. Shaw. Collusion-secure fingerprinting for digital data. In *Advances in Cryptology - CRYPTO'95, Lecture Notes in Computer Science*, volume 963, pages 453–465. Springer-Verlag, Berlin, Heidelberg, New York, 1995. 250
2. D. Boneh and J. Shaw. Collusion-secure fingerprinting for digital data. *IEEE Transactions on Information Theory*, Vol.44 No.5:1897–1905, 1998. 250
3. E. F. Brickell and D. R. Stinson. Authentication codes with multiple arbiters. In *Advances in Cryptology - EUROCRYPT'88, Lecture Notes in Computer Science*, volume 330, pages 51–55. Springer-Verlag, Berlin, Heidelberg, New York, 1988. 251
4. Y. Desmedt and M. Yung. Arbitrated unconditionally secure authentication can be unconditionally protected against arbiter's attack. In *Advances in Cryptology - CRYPTO'90, Lecture Notes in Computer Science*, volume 537, pages 177–188. Springer-Verlag, Berlin, Heidelberg, New York, 1990. 251
5. T. Johansson. Further results on asymmetric authentication schemes. *Information and Computation*, 151:100–133, 1999. 251, 251, 259, 259, 260
6. K. Kurosawa and Y. Desmedt. Optimum traitor tracing and asymmetric schemes. In *Advances in Cryptology - EUROCRYPT'98, Lecture Notes in Computer Science*, volume 1462, pages 502–517. Springer-Verlag, Berlin, Heidelberg, New York, 1998. 250
7. B. Pfitzmann and A-R. Sadeghi. Coin-based anonymous fingerprinting. In *Advances in Cryptology - EUROCRYPT'99, Lecture Notes in Computer Science*, volume 1592, pages 150–164. Springer-Verlag, Berlin, Heidelberg, New York, 1999. 250, 251
8. B. Pfitzmann and M. Waidner. Anonymous fingerprinting. In *Advances in Cryptology - EUROCRYPT'97, Lecture Notes in Computer Science*, volume 1233, pages 88–102. Springer-Verlag, Berlin, Heidelberg, New York, 1997. 250
9. R. Safavi-Naini and Y. Wang. A combinatorial approach to asymmetric traitor tracing. In *Computing and Combinatorics, 6th Annual International Conference, COCOON 2000, Lecture Notes in Computer Science*, volume 1858, pages 416–425. Springer-Verlag, Berlin, Heidelberg, New York, 2000. 251, 252, 252, 259, 260, 260
10. D. Stinson and R. Wei. Combinatorial properties and constructions of traceability schemes and framproof codes. *SIAM Journal on Discrete Mathematics*, 11:41–53, 1998. 250

New Block Cipher DONUT Using Pairwise Perfect Decorrelation[⋆]

Dong Hyeon Cheon[1], Sang Jin Lee[1], Jong In Lim[1], and Sung Jae Lee[2]

[1] Center for Information and Security Technologies(CIST), Korea University, Seoul, Korea, dhcheon@cist.korea.ac.kr, sangjin@tiger.korea.ac.kr, jilim@tiger.korea.ac.kr

[2] Korea Information Security Agency(KISA), sjlee@center.kisa.or.kr

Abstract. Vaudenay[1] proposed a new way of protecting block ciphers against classes of attacks, which was based on the notion of decorrelation. He also suggested two block cipher families COCONUT and PEANUT. Wagner[2] suggested a new differential-style attack called boomerang attack and cryptanalysed COCONUT'98. In this paper we will suggest a new block cipher called DONUT which is made by two pairwise perfect decorrelation modules. DONUT is secure against boomerang attack.

Key words : Block cipher, Decorrelation, Differential Cryptanalysis(DC), Linear Cryptanalysis(LC).

A Introduction

Vaudenay[1] proposed a new way of protecting block ciphers against differential cryptanalysis(DC)[3,4] and linear cryptanalysis(LC)[5], which was based on the notion of decorrelation. This notion is similar to that of universal functions which was introduced by Carter and Wegman[6,7]. Vaudenay also suggested two block cipher families COCONUT(Cipher Organized with Cute Operations and NUT) and PEANUT(Pretty Encryption algorithm with NUT). COCONUT family uses a pairwise perfect decorrelation module and PEANUT family uses a partial decorrelation module.

COCONUT'98 is a product cipher $C_3 \circ C_2 \circ C_1$, where C_1 and C_3 are 4-round Feistel ciphers and C_2 is a pairwise perfect decorrelation module. Wagner[2] suggested a new differential-style attack called boomerang attack and cryptanalysed COCONUT'98. In this paper we will suggest a new block cipher called DONUT(Double Operations with NUT) which is made by two pairwise perfect decorrelation modules. DONUT is secure against boomerang attack.

This paper is organized as follows. In section 2, we recall the basic definitions used in the decorrelation theory and present the previous results of decorrelation

[⋆] This work is supported by Korea Information Security Agency(KISA) grant 2000-S-078.

theory. In section 3, we suggest a frame structure of new cipher and show that this structure is secure against boomerang attack. In section 4, we describe the detailed structure of DONUT, and in section 5, we conclude this paper.

B Preliminaries

In this section, we recall the basic definitions used in the decorrelation theory and briefly present the previous results[1,8].

Definition 1. *Given a random function F from a given set M_1 to a given set M_2 and an integer d, we define the "d-wise distribution matrix" $[F]^d$ of F as a $M_1^d \times M_2^d$-matrix where the (x, y)-entry of $[F]^d$ corresponding to the multipoints $x = (x_1, \cdots, x_d) \in M_1^d$ and $y = (y_1, \cdots, y_d) \in M_2^d$ is defined as the probability that we have $F(x_i) = y_i$ for $i = 1, \cdots, d$.*

Each row of the d-wise distribution matrix corresponds to the distribution of the d-tuple $(F(x_1), \cdots, F(x_d))$ where $(x_1, \cdots, x_d)$ corresponds to the index of the row.

Definition 2. *Given two random functions F and G from a given set M_1 to a given set M_2, an integer d and a distance D over the matrix space $R^{M_1^d \times M_2^d}$, we define the "d-wise decorrelation D-distance between F and G" as being the distance*

$$DecF_D^d(F, G) = D([F]^d, [G]^d).$$

We also define the "d-wise decorrelation D-bias of function F" as being the distance

$$DecF_D^d(F) = D([F]^d, [F^*]^d)$$

where F^* is a uniformly distributed random function from M_1 to M_2. Similarly, for $M_1 = M_2$, if C is a random permutation over M_1 we define the "d-wise decorrelation D-bias of permutation C" as being the distance

$$DecP_D^d(C) = D([C]^d, [C^*]^d)$$

where C^* is a uniformly distributed random permutation over M_1.

In the above definition, C^* is called the Perfect Cipher. If a cipher C has zero d-wise decorrelation bias, we call C a perfectly decorrelated cipher. When message space $M = \{0, 1\}^m$ has a field structure, we can construct pairwise perfectly decorrelated ciphers on M by $C(y) = A \cdot y + B$ where key $K = (A, B)$ is uniformly distributed on $M^\times \times M$ and $M^\times = M - \{0\}$. This pairwise perfect decorrelation module is used for COCONUT family.

COCONUT is a family of ciphers parameterized by $(p(x), m)$, where m is the block size and $p(x)$ is an irreducible polynomial of degree m in $GF(2)[x]$. A COCONUT cipher is a product cipher $C_3 \circ C_2 \circ C_1$, where C_1 and C_3 are any (possibly weak) ciphers, and C_2 is defined as follows:

$$C_2(y) = A \cdot y + B \bmod p(x),$$

where A, B and y are polynomials of degree at most $m-1$ in $GF(2)[x]$. The polynomials A and B are secret and act as round keys. Since the COCONUT family has pairwise perfect decorrelation, the ciphers are secure against the basic differential and basic linear cryptanalysis[1].

COCONUT'98 is a member of COCONUT family parameterized by $(64, x^{64}+x^{11}+x^2+x+1)$ and uses 4-round Feistel structures for C_1 and C_3, respectively. Wagner[2] cryptanalysed COCONUT'98 using boomerang attack, which exploits that high probability differentials exist for both C_1 and C_3.

C Frame Structure

C.1 Frame Structure of New Cipher

Luby-Rackoff[9] showed a method for constructing a pseudorandom permutation from a pseudorandom function. This method is based on composing four Feistel permutations, each of which requires the evaluation of a pseudorandom function, i.e. Luby-Rackoff constructed a $2n$-bit pseudorandom permutation from four n-bit pseudorandom permutations. Naor and Reingold[10] achieved an improvement in the computational complexity by using only two applications of pseudorandom functions on n-bits to compute the value of a $2n$-bit pseudorandom permutation. The central idea is to sandwich the two rounds of Feistel networks involving the pseudorandom functions between two pairwise independent $2n$-bit permutations. The Frame structure of new cipher is motivated by this result. Fig.1 shows the frame structure of new cipher. In Fig. 1, $A_1 \cdot y + B_1$ and $A_2 \cdot y + B_2$ represent pairwise perfect decorrelation modules using A_1, B_1, A_2, B_2 as keys.

C.2 Security of Frame Structure

Let $M = M_0^2$ be a message space of new cipher where $M_0 = \{0,1\}^{\frac{m}{2}}$. Suppose the maximum differential probability of F function is bounded by p_{max}. Aoki and Ohta[11] showed that the differential probability of 2-round Feistel structure is bounded by p_{max}. Also Knudsen[12] showed that the distribution of differences through the decorrelation module $A \cdot y + B$ is very key dependent, i.e. if $a \neq 0$ is an input difference of decorrelation module $A \cdot y + B$ then the output difference is aA. In Fig. 1, since the frame structure of new cipher uses two Feistel permutations as inner 2-round transformation and decorrelation modules $A \cdot y + B$ is very key dependent, the maximum probability of the entire structure is p_{max}. This case occurs when the input difference of F function is zero. For a given nonzero input difference of decorrelation module, the probability that the input difference of F function is zero is $2^{-\frac{m}{2}}$. As a point of view of attacker, he must find the characteristic with probability p_{max}. But this occurs with probability $2^{-\frac{m}{2}}$ and though he can find the characteristic with probability p_{max}, he must find the key A_1 or A_2 in order to attack this cipher with computational complexity 2^m, because he cannot know the difference of inner round functions.

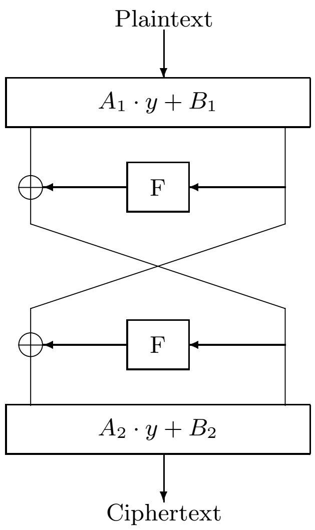

Fig. 1. Frame Structure of New Cipher

In the following, we introduce the boomerang attack and show that DONUT is secure against boomerang attack. Boomerang attack was introduced by D. Wagner[2]. It is a differential attack that attempts to generate a quartet structure at an intermediate value halfway through the cipher. If the best characteristic for half of the rounds of the cipher has probability q, then the boomerang attack can be used in a successful attack needing $O(q^{-4})$ chosen texts. The attacker considers four plaintexts P, P', Q, Q', along with their respective ciphertexts C, C', D, D'. Let $E(\cdot)$ represent the encryption operation, and decompose the cipher into $E = E_1 \circ E_0$, where E_0 represents the first half of the cipher and E_1 represents the last half. We will use two differential characteristics, $\triangle \rightarrow \triangle^*$ for E_0, as well as $\nabla \rightarrow \nabla^*$ for E_1^{-1}.

The attacker wants to cover the pair P, P' with the characteristic for E_0, and to cover the pairs P, Q and P', Q' with the characteristic for E_1^{-1}. Then the pair Q, Q' is perfectly set up to use the characteristic $\triangle^* \rightarrow \triangle$ for E_0^{-1} as follows:

$$\begin{aligned} E_0(Q) \oplus E_0(Q') &= E_0(P) \oplus E_0(P') \oplus E_0(P) \oplus E_0(Q) \oplus E_0(P') \oplus E_0(Q') \\ &= E_0(P) \oplus E_0(P') \oplus E_1^{-1}(C) \oplus E_1^{-1}(D) \oplus E_1^{-1}(C') \oplus E_1^{-1}(D') \\ &= \triangle^* \oplus \nabla^* \oplus \nabla^* \\ &= \triangle^*. \end{aligned}$$

We define a right quartet as one where all four characteristics hold simultaneously. The only remaining issue is how to choose the texts so they have the right differences. We can get this as follows. First, we generate $P' = P \oplus \triangle$ and get the encryptions C, C' of P, P' with two chosen-plaintext queries. Then we generate

D, D' as $D = C \oplus \nabla$ and $D' = C' \oplus \nabla$. Finally we decrypt D, D' to obtain the plaintexts Q, Q' with two adaptive chosen-ciphertext queries.

Let M_0 be the first pairwise perfect decorrelation module $A_1 \cdot y + B_1$ and let M_1 be the second one $A_2 \cdot y + B_2$. Let Ψ_0 be the first 1-round Feistel permutation of DONUT and let Ψ_1 be the last 1-round Feistel permutation of it. Take $E_0 = \Psi_0 \circ M_0$ and $E_1 = M_1 \circ \Psi_1$. Then DONUT can be represented by $E_1 \circ E_0$. Consider the characteristic of Ψ_0. Ψ_0 has characteristics with probability 1 as follows:

$$(c, 0) \overset{\Psi_0}{\rightarrow} (0, c)$$

where c is a nonzero $\frac{m}{2}$-bit value and 0 is a $\frac{m}{2}$-bit zero value. But this occurs only when the output difference of M_0 is of the form $(c, 0)$ and we can get a characteristic $(a, b) \rightarrow (c, 0)$ through M_0 with probability $2^{-\frac{m}{2}}$ because we do not know the key A_1 of M_0. So we cannot apply the boomerang attack to DONUT.

D New Block Cipher DONUT

In this section we describe the detailed structure of the new block cipher called DONUT(Double Operations with NUT).

D.1 Structure

General Structure The new block cipher DONUT transforms a 128-bit plaintext block into a 128-bit ciphertext block. DONUT uses variable key length and consists of 4 rounds. The first round and the fourth round consist of pairwise perfect decorrelation modules $A_1 \cdot y + B_1$ and $A_2 \cdot y + B_2$ where A_1, B_1, A_2, B_2 are 128-bit subkeys. The inner 2-round transformation consists of two Feistel permutations and each round uses six 32-bit subkeys(see Fig. 1).

The Round Function *F* The round function F of DONUT consists of three G functions. A 64-bit input of F is split into two 32-bit words and 3-round transformation is followed with inner function G. Fig. 2 shows the structure of F function.

The inner function *G* The function G is a key-dependent permutation on 32-bit words with two 32-bit subkeys. The function G which we call the SDS function consists of 5 layers as follows:

1. The first key addition layer.
2. The first substitution layer.
3. The diffusion layer.
4. The second key addition layer.
5. The second substitution layer.

Fig. 3 shows the structure of G function.

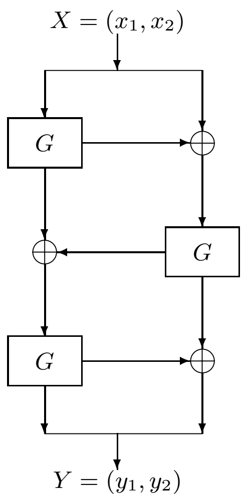

Fig. 2. The structure of F function

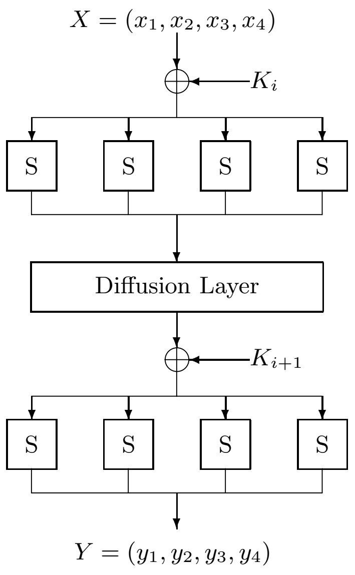

Fig. 3. The structure of G function

The Substitution Layer In the substitution layer we use the same S-box which is 8-bit input/output permutation. The S-box is constructed by the function of the form $a \cdot x^{-1} \oplus b$, where $a = 0\text{x}a5, b = 0\text{x}37 \in GF(2^8)$. The Galois field $GF(2^8)$ is defined by the irreducible polynomial $x^8 + x^4 + x^3 + x^2 + 1(\text{hex} : 0\text{x}11d)$. In the $GF(2^8)$, the x^{-1} has a good resistance against differential and linear attacks. The purpose of using affine transform is preventing from two fixed points such as zero to zero and one to one in the function x^{-1}.

The Diffusion Layer The diffusion layer is performed by the 4×4 circulant-matrix D.

$$D = \begin{pmatrix} 01\ 06\ 07\ 02 \\ 06\ 07\ 02\ 01 \\ 07\ 02\ 01\ 06 \\ 02\ 01\ 06\ 07 \end{pmatrix}$$

Let $X = (x_3, x_2, x_1, x_0) = \sum_{i=0}^{3} x_i 2^{8i}$ be the input of the diffusion layer and $Y = (y_3, y_2, y_1, y_0) = \sum_{i=0}^{3} y_i 2^{8i}$ be the output of the diffusion layer. Then we have the followings:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 01\ 06\ 07\ 02 \\ 06\ 07\ 02\ 01 \\ 07\ 02\ 01\ 06 \\ 02\ 01\ 06\ 07 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 01 \cdot x_0 \oplus 06 \cdot x_1 \oplus 07 \cdot x_2 \oplus 02 \cdot x_3 \\ 06 \cdot x_0 \oplus 07 \cdot x_1 \oplus 02 \cdot x_2 \oplus 01 \cdot x_3 \\ 07 \cdot x_0 \oplus 02 \cdot x_1 \oplus 01 \cdot x_2 \oplus 06 \cdot x_3 \\ 02 \cdot x_0 \oplus 01 \cdot x_1 \oplus 06 \cdot x_2 \oplus 07 \cdot x_3 \end{pmatrix}$$

The Key Scheduling DONUT has flexible key length. Since two decorrelation modules need four 128-bit subkeys and every G function needs two 32-bit(4 bytes) subkeys, we need to generate 28 32-bit subkeys. Our design strategy of key schedule is to avoid finding some round keys from another round keys.

In the following let $G(X, Y, Z)$ be a G-function with input X, the first addition key Y, and the second addition key Z. The notation $<< n$ means n-bit left shift and $<<< n (>>> n)$ means n-bit left(right) rotation. Let $b (> 16)$ be the byte number of key length and $uk[0], uk[1], \cdots, uk[b-1]$ be a user-supplied key. Then the followings are the key schedule of DONUT:

```
– Input : $uk[0], uk[1], \cdots, uk[b-1]$.
– Output : $k[0], k[1] \cdots, k[27]$.
    1. for( $i = 0; i < 112; i++$) $L[i] = uk[i\%b]$;
    2. for( $i = 5; i < 109; i++$) $L[i] = S[L[i]] \oplus S[L[i-5]] \oplus S[L[i+3]]$;
    3. $X[0] = 0\text{x}9e3779b9$;
    4. $Y[0] = 0\text{x}b7e15163$;
    5. for( $i = 0; i < 28; i++$) do the followings:
        (a) $T[i] = L[4i] \mid (L[4i+1] << 8) \mid (L[4i+2] << 16) \mid (L[4i+3] << 24)$;
        (b) $X[i+1] = G(X[i], (T[i] >>> 7), (T[i] <<< 5))$;
        (c) $Y[i+1] = G(Y[i], (T[i] <<< 13), (T[i] >>> 9))$;
        (d) $K[i] = X[i+1] \oplus Y[i+1] \oplus T[i]$;
```

D.2 Efficient Implementation

For the efficient implementation we implement the function G with the T-table. The T-table($T[4][256]$) stores the multiplication of $T[0][i] = 01 \cdot S[i](= S[i])$, $T[1][i] = 06 \cdot S[i]$, $T[2][i] = 07 \cdot S[i]$ and $T[3][i] = 07 \cdot S[i]$, where $0 \le i < 256$. Then the first substitution layer and the diffusion layer are calculated at the same time as the followings:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = D \begin{pmatrix} S[x_0] \\ S[x_1] \\ S[x_2] \\ S[x_3] \end{pmatrix} = \begin{pmatrix} 01\ 06\ 07\ 02 \\ 06\ 07\ 02\ 01 \\ 07\ 02\ 01\ 06 \\ 02\ 01\ 06\ 07 \end{pmatrix} \begin{pmatrix} S[x_0] \\ S[x_1] \\ S[x_2] \\ S[x_3] \end{pmatrix}$$

$$= \begin{pmatrix} 01 \cdot S[x_0] \oplus 06 \cdot S[x_1] \oplus 07 \cdot S[x_2] \oplus 02 \cdot S[x_3] \\ 06 \cdot S[x_0] \oplus 07 \cdot S[x_1] \oplus 02 \cdot S[x_2] \oplus 01 \cdot S[x_3] \\ 07 \cdot S[x_0] \oplus 02 \cdot S[x_1] \oplus 01 \cdot S[x_2] \oplus 06 \cdot S[x_3] \\ 02 \cdot S[x_0] \oplus 01 \cdot S[x_1] \oplus 06 \cdot S[x_2] \oplus 07 \cdot S[x_3] \end{pmatrix}$$

$$= \begin{pmatrix} T[0][x_0] \oplus T[1][x_1] \oplus T[2][x_2] \oplus T[3][x_3] \\ T[1][x_0] \oplus T[2][x_1] \oplus T[3][x_2] \oplus T[0][x_3] \\ T[2][x_0] \oplus T[3][x_1] \oplus T[0][x_2] \oplus T[1][x_3] \\ T[3][x_0] \oplus T[0][x_1] \oplus T[1][x_2] \oplus T[2][x_3] \end{pmatrix}$$

Since $T[0]$ is same as the S-box, we only need 1024 bytes memory to implement the function G.

E Conclusion

In this paper we suggested a new block cipher DONUT(Double Operations with NUT). DONUT is made by two pairwise perfect decorrelation modules in order to avoid boomerang attack. We also showed that DONUT is secure against boomerang attack.

References

1. S. Vaudenay, *Provably Security for Block Ciphers by Decorrelation*, STACS'98, LNCS 1373, Springer-Verlag, 1998. 250, 250, 251, 252
2. D. Wagner, *The boomerang attack*, Fast Software Encryption, Springer-Verlag, 1999. 250, 250, 252, 253
3. E. Biham, A. Shamir, *Differential cryptanalysis of DES-like cryptosystems*, Advances in Cryptology - CRYPTO'90, LNCS 537, Springer-Verlag, 2–21, 1991. 250
4. E. Biham, A. Shamir, *Differential cryptanalysis of DES-like cryptosystems*, Journal of Cryptology, Vol. 4, 3–72, 1991. 250
5. M. Matsui, *Linear cryptanalysis method for DES cipher*, Advances in Cryptology - EUROCRYPT'93, LNCS 765, Springer-Verlag, 386–397, 1994. 250
6. L. Carter, M. Wegman, *Universal classes of hash functions*, Journal of Computer and System Science, Vol. 18, 143–154, 1979. 250

7. M. N. Wegman, J. L. Carter, *New hash functions and their use in authentication and set equality*, Journal of Computer and System Science, Vol. 22, 265–279, 1981. 250
8. S. Vaudenay, *Feistel Ciphers with L_2-Decorrelation*, SAC'98, Springer-Verlag, 1998. 251
9. M. Luby, C. Rackoff, *How to construct pseudorandom permutations from pseudorandom functions*, SIAM Journal of Computing, Vol. 17, 373–386, 1988. 252
10. M. Naor, S. Reingold, *On the Construction of Pseudo-Random Permutations: Luby-Rackoff Revised*, Journal of Cryptology, Vol. 12, 29–66, 1999. 252
11. K. Aoki and K. Ohta, *Stict evaluation for the maximum average of differential probability and the maximem average of linear probability*, IEICE Transcations fundamentals of Elections, Communications and Computer Sciences, No.1, 2–8, 1997. 252
12. L. R. Knudsen, V. Rijmen, *On the Decorrelated Fast Cipher (DFC) and Its Theory*, Fast Software Encryption Workshop 99, 137–151, 1999. 252

Generating RSA Keys on a Handheld Using an Untrusted Server

Dan Boneh, Nagendra Modadugu, and Michael Kim

Stanford University, {dabo,nagendra,mfk}@cs.stanford.edu

Abstract. We show how to e ciently generate RSA keys on a low power handheld device with the help of an untrusted server. Most of the key generation work is o oaded onto the server. However, the server learns no information about the key it helped generate. We experiment with our techniques and show they result in up to a factor of 5 improvement in key generation time. The resulting RSA key looks like an RSA key for paranoids. It can be used for encryption and key exchange, but cannot be used for signatures.

A Introduction

In recent years we have seen an explosion in the number of applications for handheld devices. Many of these applications require the ability to communicate securely with a remote device over an authenticated channel. Example applications include: (1) a wireless purchase using a cell phone, (2) remote secure synchronization with a PDA, (3) using a handheld device as an authentication token [2], and (4) handheld electronic wallets [3]. Many of these handheld applications require the ability to issue digital signatures on behalf of their users.

Currently, the RSA cryptosystem is the most widely used cryptosystem for key exchange and digital signatures: SSL commonly uses RSA-based key exchange, most PKI products use RSA certificates, etc. Unfortunately, RSA on a low power handheld device is somewhat problematic. For example, generating a 1024 bit RSA signature on the PalmPilot takes approximately 30 seconds. Nevertheless, since RSA is so commonly used on servers and desktops it is desirable to improve its performance on handhelds.

In this paper we consider the problem of *generating* RSA keys. Generating a 1024 bit RSA key on the PalmPilot can take as long as 15 minutes. The device locks up while generating the key and is inaccessible to the user. For wireless devices battery life time is a concern. Consider a user who is given a new cellphone application while traveling. The application may need to generate a key before it can function. Generating the key while the user is traveling will lock up the cellphone for some time and may completely drain the batteries.

The obvious solution is to allow the handheld to communicate with a desktop or server and have the server generate the key. The key can then be downloaded onto the handheld. The problem with this approach is that the server learns the user's private key. Consequently, the server must be trusted by the user. This approach limits mobility of the handheld application since users can only generate a key while communicating with their home domain. We would like to enable users to quickly generate an RSA key even when they cannot communicate with a trusted machine.

We study the following question: can we speed up RSA key generation on a handheld with the help of an *untrusted server*? Our goal is to offload most of the key generation work onto the untrusted server. However, once the key is generated the server should have no information about the key it helped generate. This way the handheld can take advantage of the server's processing power without compromising the security of its keys.

Our best results show how to speed up the generation of *unbalanced* RSA keys. We describe these keys and explain how they are used in the next section. Our results come in two flavors. First, we show how to speed up key generation with the help of *two* untrusted servers. We assume that the two servers are unable to share information with each other. For instance, the two untrusted servers may be operated by different organizations. Using two untrusted servers we are able to speed up key generation by a factor of 5. We then show that a *single* untrusted server can be used to speed up key generation by a factor of 2. In Section D we discuss speeding up normal RSA key generation (as opposed to unbalanced keys).

We implemented and experimented with all our algorithms. We used the PalmPilot as an example handheld device since it is easy to program. Clearly our techniques apply to any low power handheld: pagers, cell phones, MP3 players, PDA's, etc. In our implementation, the PalmPilot connects to a desktop machine using the serial port. When a single server is used to help generate the key, the pilot communicates with the desktop using TCP/IP over the serial link. The desktop functions as the helping server. Note that there is no need to protect the serial connection. After all, since the desktop learns no information about the key it helped generate, an attacker snooping the connection will also learn nothing. When two servers are used, the desktop functions as a gateway enabling the pilot to communicate with the two servers. In this case, communication between the pilot and servers is protected by SSL to prevent eavesdropping by the gateway machine, and to prevent one server from listening in on communication intended for the other. Typically, the gateway machine functions as one of the two servers, as shown in Figure 1.

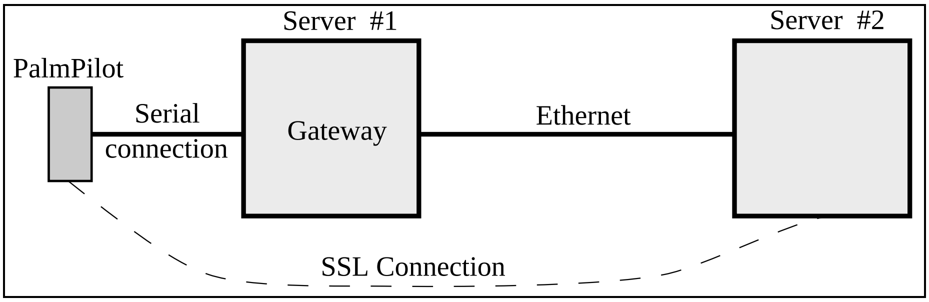

Fig. 1. A two server configuration

A.1 Timing Cryptographic Primitives on the PalmPilot

For completeness we list some running times for cryptographic operations on the PalmPilot. We used the Palm V which uses a 16.6MhZ Dragonball processor. Running times for DES, SHA-1, and RSA were obtained using a port of parts of SSLeay to the PalmPilot started by Ian Goldberg.

Algorithm	Time	Comment
DES encryption	4.9ms/block	
SHA-1	2.7ms/block	
1024 bit RSA key generation	15 minutes on average	
1024 bit RSA sig. generation	27.8 sec.	
1024 bit RSA sig. verify	0.758 sec.	$e = 3$
1024 bit RSA sig. verify	1.860 sec.	$e = 65537$

B Preliminaries

B.1 Overview of RSA Key Generation

As a necessary background we give a brief overview of RSA key generation. Recall that an RSA key is made up of an n-bit modulus $N = pq$ and a pair of integers d, called the private exponent, and e, called the public exponent. Typically, N is the product of two large primes, each $n/2$ bits long. Throughout the paper we focus on generating a 1024 bit key (i.e. $n = 1024$). The algorithm to generate an RSA key is as follows:

Step 1: Repeat the following steps until two primes p, q are found:
- **a. Candidate** Pick a random 512 bit candidate value p.
- **b. Sieve** Using trial division, check that p is not divisible by any small primes (i.e. 2,3,5,7, etc.).
- **c. Test** Run a probabilistic primality test on the candidate. For simplicity one can view the test as checking that $g^{(p-1)/2} \equiv \pm 1 \pmod{p}$, where g is a random value in $1 \ldots p-1$. All primes will pass this test, while a composite will fail with overwhelming probability [10].

Step 2: Compute the product $N = pq$ (the product is 1024 bits long).

Step 3: Pick encryption and decryption exponents e and d where $e \cdot d = 1 \bmod \varphi(N)$ and $\varphi(N) = N - p - q + 1$.

The bulk of the key generation work takes place in step (1). Once the two primes p and q are found, steps (2) and (3) take negligible work. We note that trial division (step 1b) is frequently optimized by using a sieving algorithm. Sieving works as follows: once the candidate p is chosen in step (1a), the sieve is used to quickly find the closest integer to p that is not divisible by any small primes. The candidate p is then updated to be the integer found by the sieve. Throughout the paper we use a sieving algorithm attributed to Phil Zimmerman.

Our goal is to improve the performance of step (1). Within step (1), the exponentiation in step (1c) dominates the running time. Our goal is to offload the primality test to the server without exposing any information about the candidate being tested. Hence, the question is: how can we test that $g^{p-1} \bmod p = 1$ with the help of a server without revealing any information about p? To do so we must show how to carry out the exponentiation while solving two problems: (1) hiding the modulus p, and (2) hiding the exponent $p - 1$.

B.2 Unbalanced RSA Keys

Our best results show how to speed up the generation of unbalanced RSA keys. An unbalanced key uses a modulus N of the form $N = p \cdot R$ where p is a 512 bit prime and R is a 4096 bit *random number*. One can show that with high probability R has a prime factor that is at least 512 bits long (the probability that it does not have such a factor is less than $1/2^{24}$). Consequently, the resulting modulus N is as hard to factor as a standard modulus $N = pq$.

An unbalanced key is used in the same way as standard RSA keys. The public key is $\langle e, N\rangle$ and the private key is $\langle d, N\rangle$. We require that $e \cdot d = 1 \bmod p - 1$. Suppose p is m-bits long. The system can be used to encrypt messages shorter than m bits. As in standard RSA, to encrypt a message M, whose length is much shorter than m bits, the sender first applies a randomized padding mechanism, such as OAEP [4,9]. The padding mechanism results in an $m - 1$ bit integer P (note that $P < p$). The sender then constructs the ciphertext by computing $C = P^e \bmod N$. Note that the ciphertext is as big as N. To decrypt a ciphertext C, the receiver first computes $C_p = C \bmod p$ and then recovers P by computing $P = C_p^d \bmod p$. The plaintext M is then easily extracted from P. Since decryption is done modulo p it is as fast as standard RSA.

The technique described above for using an unbalanced key is similar to Shamir's "RSA for paranoids" [11]. It shows that unbalanced keys can be used for encryption/decryption and key exchange. Unfortunately, unbalanced keys cannot be used for digital signatures. We note that some attacks against RSA for paranoids have been recently proposed [5]. However, these attacks do not apply when one uses proper padding prior to encryption. In particular, when OAEP padding is used [4] the attacks cannot succeed since the security of OAEP (in the random oracle model) only relies on the fact that the function $f : \{0, \ldots, 2^{m-1}\} \to \mathbb{Z}_N$ defined by $f(x) = x^e \bmod N$ is a one-to-one trapdoor one way function.

C Generating an Unbalanced RSA Key with the Help of Untrusted Servers

We show how RSA key generation can be significantly sped up by allowing the PalmPilot to interact with untrusted servers. At the end of the computation the servers should know nothing about the key they helped generate. We begin by showing how *two* untrusted servers can help the Pilot generate RSA keys. The assumption is that these two servers cannot exchange information with each other. To ensure that an attacker cannot eavesdrop on the network and obtain the information being sent to both servers, our full implementation protects the connection between the Pilot and the servers using SSL. Typically, the machine to which the pilot is connected can be used as one of the untrusted servers (Figure 1). We then show how to speed up key generation with the help of a *single* server. In this case there is no need to protect the connection.

C.1 Generating Keys with the Help of Two Servers

Our goal is to generate a modulus of the form $N = pR$ where p is a 512-bit prime and R is a 4096-bit random number. To offload the primality test onto the servers we must hide the modulus p and the exponent $p - 1$. To hide the modulus p we intend to multiply it by a random number R and send the resulting $N = pR$ to the servers. The server will perform computations modulo $N = pR$. If it turns out that p is prime, then sending N to the servers does not expose any information about p or R. If p is not prime we start over. To hide the exponent $p - 1$ used in the primality test we intend to share it among the two servers. Individually, neither one of the servers will learn any information.

Our algorithm for generating an unbalanced RSA modulus $N = pR$ is as follows. The algorithm repeats the following steps until an unbalanced key is generated:

Step 1: Pilot generates a 512 bit candidate p that is not divisible by small primes and a 4096 bit random number R. We require that $p = 3 \bmod 4$.

Step 2: Pilot computes $N = p \cdot R$.

Step 3: Pilot picks random integers s_1 and s_2 in the range $[-p, p]$ such that $s_1 + s_2 = (p - 1)/2$. It also picks a random $g \in \mathbb{Z}_N^*$.

Step 4: Pilot sends $\langle N, g, s_1 \rangle$ to server 1 and $\langle N, g, -s_2 \rangle$ to server 2.

Step 5: Server 1 computes $X_1 = g^{s_1} \bmod N$. Server 2 computes $X_2 = g^{(-s_2)} \bmod N$. Both results X_1 and X_2 are sent back to the pilot.

Step 6: Pilot checks whether $X_1 = \pm X_2 \bmod p$. If equality holds, then $N = pR$ is declared as a potential unbalanced RSA modulus. Otherwise, the algorithm is restarted in Step 1.

Step 7: The Pilot locally runs a probabilistic primality test to verify that p is prime. This is done to ensure that the servers returned correct values. This step adds little time to the entire key generation process.

First, we verify the soundness of the algorithm. In step 6 the Pilot verifies that $g^{s_1} \cdot g^{s_2} = g^{(p-1)/2} = \pm 1 \bmod p$. If the test is satisfied then p is very likely to be prime. Then step 7 ensures that p is in fact prime and that the servers did not respond incorrectly. When generating a 1024 bit RSA key, a single primality test takes little time compared to the search for a 512 bit prime. Hence, Step 7 adds very little to the total running time.

During the search for the prime p, the only computation carried out by the pilot is the probable prime generation and the computation of s_1 and s_2. The time to construct s_1 and s_2 is negligible. On the other hand, generating the probable prime p requires a sieve to ensure that p is not divisible by small factors. As we shall see in Section E the sieve is the bottleneck. This is unusual since in standard RSA modulus generation sieving takes only a small fraction of the entire computation. We use a sieving method attributed to Phil Zimmerman. We note that faster sieves exist, but they result in an insecurity of our algorithm.

Security To analyze the security properties of the algorithm we must argue that the untrusted servers learn no information of value to them. During the search for the RSA modulus many candidates are generated. Since these candidates are independent of each other, any information the servers learn about rejected candidates does not help them in attacking the final chosen RSA modulus. Once the final modulus $N = pR$ is generated in Step 2, each server is sent the value of N and s_i where i is either 1 or 2. The modulus N will become public anyhow (it is part of the public key) and hence reveals no new information. Now, assuming servers 1 and 2 cannot communicate, the value s_1 is simply a random number (from Server 1's point of view). Server 1 could have just as easily picked a random number in the range $[-N, N]$ itself. Hence, s_1 reveals no new information to Server 1 (formally, a simulation argument shows that s_1 reveals at most two bits). The same holds for Server 2. Hence, as long as Server 1 and Server 2 cannot communicate, no useful information is revealed about the factorization of N. We note that if the servers are able to communicate, they can factor N.

Performance The number of iterations until a modulus is found is identical to local generation of an (unbalanced) modulus on the PalmPilot. However, each iteration is much faster than the classic RSA key generation approach of Section B.1. After all, we offloaded the expensive exponentiation to a fast Pentium machine. As we shall see in Section E.1, the total running time is reduced by a factor of 5.

C.2 Generating Keys with the Help of a Single Server

Next, we show how a *single* untrusted server can be used to reduce the time to generate an RSA key on the PalmPilot. Once the key is generated, the server has no information regarding the key it helped generate. Typically, the pilot connects to the helping server directly through the serial or infrared ports.

As before we need to compute $g^{(p-1)/2} \bmod p$ to test whether p is prime. Our technique involves reducing the size of the exponent using the help of the server and hence speeding up exponentiation on the pilot. The algorithm repeats the following steps until an unbalanced modulus is found:

Step 1: Pilot generates a 512 bit candidate p that is not divisible by small primes and a 4096 bit random number R. We require that $p = 3 \bmod 4$.
Step 2: Pilot computes $N = p \cdot R$. It picks a random $g \in \mathbb{Z}_N^*$.
Step 3: Pilot picks a random 160 bit integer r and a random 512 bit integer a. It computes $z = r + a(p-1)/2$.
Step 4: Pilot sends $\langle N, g, z \rangle$ to the server.
Step 5: The server computes $X = g^z \bmod N$ and sends X back to the Pilot.
Step 6: Pilot computes $Y = g^r \bmod p$.

Step 7: Pilot checks if $X = \pm Y \bmod p$. If so then the algorithm is finished and $N = pR$ is declared as a potential unbalanced RSA modulus. Otherwise, the algorithm is restarted in Step 1.

Step 8: The Pilot locally runs a probabilistic primality test to verify that p is prime.

To verify soundness observe that N will make it to step 8 only if $X = \pm Y$ i.e. $g^{r+a(p-1)/2} = \pm g^r \bmod p$. As before, this condition is always satisfied if p is prime. The test will fail with overwhelming probability if p is not prime. Hence, once step 8 is reached the modulus $N = pR$ is very likely to be an unbalanced modulus. The test is Step 8 takes little time compared to the entire search for the 512-bit prime.

Performance Since we are generating an unbalanced modulus the number of iterations until N is found is the same as in local generation of such a modulus on the PalmPilot. Within each iteration the Pilot generates p and R using a sieve and then computes $Y = g^r \bmod p$ (in step 6). However, r is only 160 bits long. This is much shorter than when a key is generated without the help of a server. In that case the Pilot has to compute an exponentiation where the exponent is 512 bits long. Hence, we reduce the exponentiation time by approximately a factor of three. Total key generation time is reduced by a factor of 2, due to the overhead of sieving.

Recall that in Step 6 the Pilot computes $Y = g^r \bmod p$ where r is a 160-bit integer. This step can be further sped up with the help of the server. Let $A = 2^{40}$ and write $r = r_0 + r_1 A + r_2 A^2 + r_3 A^3$ where r_0, r_1, r_2, r_3 are all in the range $[0, A]$. In Step 5 the server could send back the vector $\boldsymbol{R} = \langle g^A, g^{A^2}, g^{A^3} \rangle \bmod N$ in addition to sending X. Let $\boldsymbol{R} = \langle R_1, R_2, R_3 \rangle$. Then in Step 6 the Pilot only has to compute $Y = g^{r_0} \cdot R_1^{r_1} \cdot R_2^{r_2} \cdot R_3^{r_3} \bmod p$. Using Simultaneous Multiple Exponentiation [7, p. 617] Step 6 can now be done in approximately half the time of computing $Y = g^r \bmod p$ on the Pilot directly. This improvement reduces the total exponentiation time on the Pilot by an additional factor of 2 .

Security In the last iteration, when the final p and R are chosen, the server learns the value $z = a(p-1) + r$. Although z is a "random looking" 1024 bit number, it does contain some information about p. In particular, $z \bmod p - 1$ is very small (only 160 bits long). The question is whether z helps an adversary break the resulting key. The best known algorithm for doing so requires $2^{r/2}$ modular exponentiations. Due to our choice of 160 bits for r, the algorithm has security of approximately 2^{80}. This is good enough since a 1024 bits RSA key offers security of 2^{80} anyhow. Nevertheless, the security of the scheme is heuristic since it depends on the assumption that no faster algorithm exists for factoring N given z. More precisely, the scheme depends on the following "$(p-1)$-multiple assumption":

$(p-1)$*-multiple assumption:* Let A_n be the set of integers $N = pq$ where p and q are both n-bit primes. Let m be an integer so that the fastest algorithm for factoring a random element $N \in A_n$ runs in time at least $2^{m/2}$. Then the two distributions: $\langle N, r + a(p-1)/2 \rangle$ and $\langle N, x \rangle$ cannot be distinguished with non-negligible probability by an algorithm whose running time is less than $2^{m/2}$. Here N is randomly chosen in A_n, a is randomly chosen in $[0, p]$, r is randomly chosen in $[0, 2^m]$, and x is randomly chosen in $[0, p^2/2]$.

Based on the $(p-1)$-multiple assumption, the integer z given to the server contains no more statistical information than a random number in the range $[0, p^2]$. Hence, the server learns no new useful information from z.

As before, since the generated key is an unbalanced key it can only be used for encryption/decryption and key exchange. It cannot be used for signatures.

D Generating Standard RSA Keys

One could wonder whether the techniques described in the previous sections can be used to speed up generation of standard RSA keys. We show that at the moment these techniques do not appear to improve the generation time for a 1024 bit key. For shorter keys, e.g. 512 bits keys, we get a small improvement. In what follows we show how to generate a normal RSA key, $N = pq$, with the help of two servers.

We wish to generate an RSA modulus $N = pq$ where p and q are each 512-bits long. As before, we wish to offload the primality test to the servers. To do so we must hide the moduli p and q and the exponents $p-1$ and $q-1$. The basic idea is to *simultaneously* test primality of both p and q. For each pair of candidates p and q the Pilot computes $N = pq$ and sends N to the servers. The servers carry out the exponentiations modulo N. To hide the exponents $p-1$ and $q-1$ we share them among the two servers as in the last section.

The resulting algorithm is similar to that for generating unbalanced keys. In fact, the server-side is identical. The algorithm works as follows. Repeat the following steps until a standard RSA modulus is found:

Step 1: Pilot generates two candidates p, q so that neither one is divisible by small primes. We refer to p and q as probable primes.
Step 2: Pilot computes $N = p \cdot q$ and $\varphi(N) = N - p - q + 1$. Pilot picks a random $g \in \mathbb{Z}_N^*$.
Step 3: Pilot picks random integers φ_1 and φ_2 in the range $[-N, N]$ such that $\varphi_1 + \varphi_2 = \varphi(N)/4$.
Step 4: Pilot sends $\langle N, g, \varphi_1 \rangle$ to server 1 and $\langle N, g, -\varphi_2 \rangle$ to server 2.
Step 5: Server 1 computes $X_1 = g^{\varphi_1} \pmod N$. Server 2 computers $X_2 = g^{-\varphi_2} \pmod N$. Both results X_1 and X_2 are sent back to the pilot.
Step 6: Pilot checks if $X_1 = \pm X_2 \bmod N$. If so, the algorithm is finished and $N = pq$ is declared as a potential RSA modulus. Otherwise, the algorithm is restarted in Step 1.
Step 7: The Pilot locally runs a probabilistic primality test to verify that p and q are prime. This is done to ensure that the servers returned correct values.

First, we verify soundness of the algorithm. In step 6 the Pilot is testing that $X_1 = \pm X_2$, namely that $g^{\varphi_1} = g^{-\varphi_2} \bmod N$. That is, we check that $g^{\varphi_1 + \varphi_2} =$

$g^{\varphi(N)/4} = \pm 1 \bmod N$. Clearly, this condition holds if p and q are both primes. Furthermore, it will fail with overwhelming probability if either p or q are not prime. Hence, Step 7 is reached only if $N = pq$ is extremely likely to be a proper RSA modulus. Step 7 then locally ensures that p and q are primes.

Security To analyze the security properties of the algorithm we must argue that the untrusted servers learn no information of value to them. During the search for the RSA modulus many candidates are generated. Since these candidates are independent of each other, any information the servers learn about rejected candidates does not help them in attacking the final chosen RSA modulus. Once the final modulus $N = pq$ is generated in Step 2, each server is sent the value of N and φ_i where i is either 1 or 2. The modulus N will become public anyhow (it is part of the public key) and hence reveals no new information. Now, assuming servers 1 and 2 cannot communicate, the value φ_1 is simply a random number (from Server 1's point of view). Server 1 could have just as easily picked a random number in the range $[-N, N]$ itself. Hence, φ_1 reveals no new information to Server 1. As long as Server 1 and Server 2 cannot communicate, no useful information is revealed about the factorization of N. If the servers are able to communicate, they can factor N.

Performance Each iteration in our algorithm is much faster than the classic RSA key generation approach of Section B.1 — we offloaded the expensive exponentiation to a fast Pentium machine. Unfortunately, the number of iterations required until an RSA modulus is found is higher. More precisely, suppose in the classic approach one requires k iterations on average until a 512-bit prime is found. Then the total number of iterations to find two primes is $2k$ on average. In contrast, in our approach both p and q must be simultaneously prime. Hence, k^2 iterations are required. We refer to this effect as a *quadratic slowdown*. When generating a 1024 bit modulus the value of k is approximately 14. So even though we are able to speed up each iteration by a factor of 5, there are seven times as many iterations on average. Therefore when generating a standard 1024 bit key these techniques do not improve the running time. When generating a shorter key, e.g. a 512 bit key, the quadratic slowdown penalty is less severe since k is smaller (9 rather than 14). For such short keys we obtain a small improvement in performance.

Similarly, when generating keys with the help of a *single* server, the quadratic slowdown outweighs the reduction in time per iteration. It is an open problem to speed up server aided generation of standard RSA keys.

E Experiments and Implementation Details

The two main components of our implementation were the cryptographic and networking modules. SSLeay provided for the cryptographic code on both the server (Pentium II 400Mhz) and PalmPilot side. In the case of the Pilot, we used SSLeay code that had been previously ported by Ian Goldberg.

Networking We connect the pilot to a Windows NT gateway running RAS (Remote Access Service) through a serial-to-serial interface. The function of the gateway was to provide TCP/IP access to the network. In our single server implementation, we used the gateway as our assisting server while in our dual server implementation, we used the gateway and another local machine.

E.1 Experiments

Tables 1 and 2 show the results we obtained when generating 512, 768 and 1024 bit RSA keys. The network traffic column measures the amount of data (in bytes) exchanged between the Pilot and the servers. We generate keys using three methods:

(1) Local: Key generated locally on the Pilot (no interaction with a server).
(2) One server: Pilot aided by a single untrusted server.
(3) Two servers: Pilot aided by two untrusted servers.

As expected we see that generating unbalanced keys with the aid of one or two servers leads to a performance improvement over generating keys locally on the PalmPilot. The rest of this section discusses these experimental results.

We note that the key factor that determines the time it takes to generate an RSA key is the time *per iteration* (the time to sieve and exponentiate *one* probable prime p). This number is more meaningful than the total running since since the total time has a high variance. More precisely, the number of iterations until a prime is found has high variance. Our tables state the average number of iterations we obtained.

In our experiments, we carried out trial division on a candidate prime using the first 2048 primes (upto approximately 17,000). In all our experiments we observed that the server's responses are instantaneous compared to the Pilot's processing time. Consequently, improving server performance will only marginally affect the overall running time.

Generating a 1024 bit key Table 1 shows detailed timing measurements for generating 1024 bit RSA keys. Our breakdown of timing measurements follows the description in Section B.1. The first column shows the time to pick a probable prime, the second shows the time the Pilot spent waiting for the server to respond, the third shows the time to exponentiate on the PalmPilot (not used in the two-server mode). The last column shows the total network traffic (in bytes).

The first two rows in Table 1 measure the time to generate keys on the Pilot. The first column represents the time to generate an unbalanced key, the second represents the time to generate a normal $N = pq$ key. Since an unbalanced key requires only one prime (the other is a random number) the number of iterations for locally generating an unbalanced key is half that for generating a normal key.

When comparing the time per iteration for local generation and two server generation, we see that using two servers we get an improvement of a factor of 5. Using one server we obtain an improvement of a factor of 2. The average number of iterations is approximately the same in all three methods. Note that the improvements are a result of speeding up (or eliminating) the exponentiation step on the PalmPilot. Observe that when two servers are used the bottleneck is the sieving time — the time to generate a probable prime p.

On average, 406 iterations are needed to generate a normal RSA key ($N = pq$) with the aid of two servers. The large number of iterations is a result of the quadratic slowdown discussed in Section D. Even though each iteration is much faster than the corresponding value for local generation, we end up hurting the total generation time.

Our algorithms require only a few kilobytes of data transfer between the Pilot and the servers. The traffic generated is linear in the number of iterations which explains the large figure for two server normal key generation.

		Sieve time(ms)	Server time(ms)	Exp. time(ms)	total time/ iter.(ms)	average num. iter.	total time	net traf.
Local	unbal.	3,805		21,233	25,038	18.16	7.5min.	
Local	norm.	3,805		21,233	25,038	36.32	15.16min.	
One serv.	unbal.	3,516	955	6,995	11,467	14.5	2.7min.	5,568
Two serv.	unbal.	3,587	1,462		5,156	12.75	1.1min	8,160
Two serv.	norm.	7,850	820	0	8,720	406	59min	311,808

Table 1. Statistics for different key generation methods (1024 bit keys)

Generating various key sizes From Table 2 we see that the total iteration time increases almost linearly with key size for dual server aided generation. Indeed, the dominant component of each iteration is sieving, which takes linear time as a function of the key size. The expected total time for generating the key is the product of the time-per-iteration and the expected-number-of-iterations.

Observe that the improvement over local generation is less significant for shorter keys than for longer keys. The reason is that for smaller keys, the primality test is less of a dominating factor in the running time per iteration (we use the same size sieve for all key sizes). Hence, reducing the exponentiation time has less of an effect on the the total time per iteration.

	512 bits			768 bits			1024 bits		
	num. iter.	time/ iter.(ms)	net traf.	num. iter.	time/ iter.(ms)	net traf.	num. iter.	time/ iter.(ms)	net traf.
Local unbal.	9.15	3,550		10.53	8,215		18.16	25,038	
Local norm.	18.3	3,550		21.1	8,215		36.32	25,038	
One serv. norm.	9.3	4,546	1,785	14.8	7,644	4,262	14.5	11,467	5,568
Two serv. unbal.	9	2,492	2,880	12.55	3,687	6,024	12.75	5,156	8,160
Two serv. norm.	26	4,364	9,984	119.7	6,560	68,947	406	8,720	311808

Table 2. Statistics for different key sizes

F Conclusions

At the present using RSA on a low power handheld is problematic. In this paper we study whether RSA's performance can be improved without a loss of security. In particular, we ask whether an untrusted server can aid in RSA key generation.

We wish to offload most of the work to the server without leaking any of the handheld's secrets.

We showed a significant improvement in the time it takes to generate an *unbalanced* RSA key. With the help of two isolated servers we obtained a speed up of a factor of 5. With the help of a *single* server we obtained a speed up of a factor of 2. For normal RSA keys, $N = pq$, we cannot improve the running time due do the *quadratic slowdown* problem discussed in Section D. It is an open problem to speed up the generation of a normal RSA key using a single server. In all our algorithms the load on the server is minimal; our experiments show that even though the server is doing most of the work, the PalmPilot does not produce candidates fast enough to fully occupy the server.

We thank Certicom for providing us with SSL libraries for the PalmPilot.

References

1. N. Asokan, G. Tsudik and M. Waidner, "Server-Supported Signatures", Journal of Computer Security, Vol. 5, No. 1, pp. 91–108, 1997.
2. D. Balfanz, E. Felten, "Hand-Held Computers Can Be Better Smart Cards", to appear in the 8th USENIX Security Symposium. 271
3. D. Boneh, N. Daswani, "Experimenting with electronic commerce on the PalmPilot", in proc. of Financial-Crypto '99, Lecture Notes in Computer Science, Vol. 1648, Springer-Verlag, pp. 1–16, 1999. 271
4. M. Bellare, P. Rogaway, "Optimal asymmetric encryption – How to encrypt with RSA", in proc. Eurocrypt '94. 274, 274
5. H. Gilbert, D. Gupta, A. M. Odlyzko, and J.-J. Quisquater, "Attacks on Shamir's 'RSA for paranoids," Information Processing Letters 68 (1998), pp. 197–199. 274
6. T. Matsumoto, K. Kato, H. Imai, "Speeding up secret computations with insecure auxiliary devices", In proc. of Crypto '88, Lecture Notes in Computer Science, Vol. 403, Springer-Verlag, pp. 497–506, 1998.
7. A. Menezes, P. van Oorschot and S. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1996. 277
8. P. Nguyen, J. Stern, "The Beguin-Quisquater Server-Aided RSA Protocol from Crypto'95 is not secure", in proc. of AsiaCrypt '98, Lecture Notes in Computer Science, Vol. 1514, Springer-Verlag, pp. 372–380, 1998.
9. Public Key Cryptography Standards (PKCS), No. 1, "RSA Encryption standard", `http://www.rsa.com/rsalabs/pubs/PKCS/`. 274
10. R. Rivest, "Finding four million large random primes", In proc. of Crypto '90, Lecture Notes in Computer Science, Vol. 537, Springer-Verlag, pp. 625–626, 1997. 273
11. A. Shamir, "RSA for paranoids", CryptoBytes, Vol. 1, No. 3, 1995. 274

A Generalized Takagi-Cryptosystem with a Modulus of the Form $p^r q^s$

Seongan Lim[1], Seungjoo Kim[1], Ikkwon Yie[2]*, and Hongsub Lee[1]

[1] KISA (Korea Information Security Agency),
5th FL., Dong-A Tower, 1321-6, Seocho-Dong, Seocho-Gu, Seoul 137-070, Korea
{seongan, skim, hslee}@kisa.or.kr
[2] Department of Mathematics,
Inha University, YongHyun-Dong, Nam-Gu, Incheon, Korea
ikyie@math.inha.ac.kr

Abstract. In this paper, we propose a generalized Takagi-Cryptosystem with a modulus of the form $p^r q^s$. We've studied for the optimal choice for r, s that gives the best efficiency while maintaining a prescribed security level, and we show that the choice of either $p^r q^{r+1}$, $p^{r-1} q^{r+1}$, or $p^{r-2} q^{r+2}$ depending on the value $r + s$ is the optimal. We also present comparison tables for the efficiency of RSA, the multiprime technology, Takagi's scheme, and our proposed scheme.

A Introduction

The RSA system is one of the most practical public key cryptosystems. For security concerns, the common modulus size of the RSA system is at least 1024 bits currently and the size of the modulus must be increased due to the development of the factoring technology [7]. As the size of the modulus is increasing, the required time and storage to implement the RSA system will be a big hurdle to use the RSA system on many occasions.

In order to improve the efficiency of the implementation, many schemes have been proposed. One of the approaches is to give a variation to the form of modulus of the RSA system. The most general form of modulus n is

$$n = p_1^{e_1} p_2^{e_2} \cdots p_u^{e_u}, \quad e_i \geq 1 \text{ for } 1 \leq i \leq u$$

where prime number p_i's are all distinct and about the same size. In the multiprime technology [4], they use the modulus of the form:

$$n = p_1 p_2 \cdots p_u, \quad \text{for } u \geq 3.$$

In the multiprime technology, the encryption process is the same as RSA and the decryption is performed by using CRT(Chinese Remainder Theorem) in a parallel computation mode with u exponentiators. The multiprime technology relieves

* Yie's work was partly supported by Inha Research Fund 2000.

the computational complexity of the original RSA system, and has recently been adopted to a WTLS(Wireless Transport Layer Security) protocol.

In [1], T. Takagi uses the modulus of the form

$$n = p_1^r p_2 \quad \text{for } r \geq 2.$$

The encryption process of Takagi's system is the same as RSA. In the decryption of Takagi's system, he uses his previous method of p-adic expansion [2] to achieve the speedy decryption. Another improvement obtained by Takagi's system is that the size of the private keys has been reduced.

In this paper, we propose a generalization of Takagi's algorithm for general modulus $n = p_1^{e_1} p_2^{e_2} \cdots p_u^{e_u}$, and discuss about the case with optimal efficiency.

A.1 Our Result

Our paper is organized as follows. In Section 2, we describe the proposed encryption and decryption process. In Section 3, we analyse the complexity of the decryption and determine the case with optimal efficiency. Our result says that the optimal efficiency can be obtained when we use two distinct prime factors whose respective exponents are relatively prime and as close as possible. For example, $n = p^r q^{r+1}$ if the sum of the two exponents is odd, $n = p^{r-1} q^{r+1}$ if the sum of exponents is 0 modulo 4, and $n = p^{r-2} q^{r+2}$ if the sum of exponents is 2 modulo 4. In Table 1 and 2 below, a brief comparison of RSA with CRT, the multiprime technology, Takagi's scheme, and the proposed scheme is given. To compare the complexity, we have analysed the number of crucial bit operations needed to implement each of the schemes. We only considered the case when the encrypting exponent e is small and we ignored the complexity concerns involved with e. Also we focused on the case $n = p^r q^{r+1}$.

In Section 4, we discuss about the security of our proposed system with the modulus $n = p^r q^s$ against the known factorization algorithms. Up to this point, the best known factorization methods for large numbers are Elliptic Curve Method(ECM [6]), Number Field Sieve(NFS [5]), and Lattice Factorization Method(LFM [3]). NFS is good for the modulus with smaller number of prime factors, and ECM and LFM are good for the modulus with many prime factors [3].

Table 1. Complexity comparison of RSA, multiprime, Takagi's and ours (non-parallel computing)

	RSA	multiprime	Takagi's	ours
modulus n	pq	$p_1 \cdots p_{2r+1}$	$p^{2r} q$	$p^r q^{r+1}$
modulus size	α	α	α	α
private key size	2α	2α	$\frac{4}{2r+1}\alpha$	$\frac{4}{2r+1}\alpha$
encryption	α^2	α^2	α^2	α^2
decryption	$\frac{\alpha^3}{4}$	$\frac{\alpha^3}{(2r+1)^2}$	$\frac{2\alpha^3}{(2r+1)^3} + \frac{8r^3+6r^2+r-3}{3(2r+1)^2}\alpha^2$	$\frac{2\alpha^3}{(2r+1)^3} + \frac{2r^3+6r^2+7r-3}{3(2r+1)^2}\alpha^2$

Table 2. Complexity comparison of RSA, multiprime, Takagi's and ours (parallel computing)

	modulus size	number of exponentiators	decryption
RSA	α	2	$\frac{\alpha^3}{2}$
multiprime	α	$2r+1$	$\frac{\alpha^3}{(2r+1)^3}$
Takagi's	α	2	$\frac{\alpha^3}{(2r+1)^3} + \frac{8r^3+6r^2+r-3}{3(2r+1)^2}\alpha^2$
ours	α	2	$\frac{\alpha^3}{(2r+1)^3} + \frac{2r^3+9r^2+13r}{6(2r+1)^2}\alpha^2$

A helpful and reliable table for the security margin for coming years and the corresponding size of RSA modulus has been suggested in [7]. First we determine the size of n with the security margin in MIPS-Years according to their table. And then we will decide the optimal number of prime factors of $n = p^r q^s$ to defeat both of ECM and LFM methods. In the multiprime technology, they introduced a technique to determine the optimal number of prime factors with respect to both of the ECM and NFS methods. We apply their method to the case $n = p^r q^s$. Suppose that we have $\alpha = \log n$ and p, q are about the same size, then the optimal number $t = r + s$ to defeat both of ECM and NFS attacks can be obtained by solving the following approximated equation for t :

$$\sqrt{2\frac{\alpha}{t}\log\frac{\alpha}{t}} \approx 1.923\sqrt[3]{\alpha(\log\alpha)^2} - 2\log\alpha + 36.$$

For example, we have Table 3 for the modulus of our proposed scheme that gives the same security level as the RSA modulus of the same size.

Suppose that for a fixed number α and a positive integer k, the proposed scheme has the same security level as RSA with a modulus of size α when we use the modulus $n = p^r q^s$ of size $k\alpha$ with $t_k = r + s$ prime factors. In Section 4, we shall show the following approximated relation for α, k, t_k:

$$\frac{k\alpha}{t_k}\log\frac{k\alpha}{t_k} \approx \frac{\alpha}{t_1}\log\frac{\alpha}{t_1} - 2\log k.$$

>From the above relation, we get the optimal number of factors of the modulus.

For example, current security margin is 2.06×10^{10} MIPS-Years and the corresponding size of the RSA modulus is 1024 bits [7]. For various modulus sizes, we give the optimal choices of modulus which gives the same security level as RSA-1024 and the speed-ratio compared with 1024 bit RSA system in Table 4. This implies that it would be the best to choose the modulus n of 2048 bits of the form of p^3q^4 for our proposed system when RSA-1024 is recommended.

Table 3. Decryption speed comparison of RSA and ours (nonparallel computing)

modulus size	our modulus	performance rate with respect to RSA with CRT
1024 bits	$n = pq^2$	3 times faster than RSA-1024
4096 bits	$n = pq^3$	8 times faster than RSA-4096
8192 bits	$n = p^2q^3$	15 times faster than RSA-8192

Table 4. Speed comparison of RSA-1024 and ours with various modulus sizes

modulus size	1024 bits	2048 bits	4096 bits
form of the modulus	$n = pq^2$	$n = p^3 q^4$	$n = p^7 q^8$
encryption speed compare to RSA-1024	same	4 times slower	8 times slower
decryption speed compare to RSA-1024	3 times faster	10 times faster	6 times faster

B Description of the Proposed Cryptosystem

In this section, we describe the proposed scheme. In order to determine the case of optimal efficiency under the assumption that the sum of exponents is fixed, one should start with the general modulus of the form $n = p_1^{e_1} p_2^{e_2} \cdots p_u^{e_u}$. But for the simplicity, we shall describe only the case $n = p^r q^s$. In Section 3, we shall explain why the best efficiency comes in the two prime cases.

B.1 Key Generation

First we generate keys in the proposed system. For a given relatively prime positive integers r, s, we follow the following directions. When we generate large primes p, q, we apply the same rules as in the RSA system.

1. Randomly choose large primes p, q.
2. Compute $n = p^r q^s$.
3. Compute $L = \text{lcm}(p-1, q-1)$.
4. Randomly choose an odd integer e so that $1 < e < L$ and $\gcd(e, L) = \gcd(e, n) = 1$.
5. Compute $d \equiv e^{-1} \pmod{L}$. i.e., $ed \equiv 1 \pmod{L}$.
6. Publish e, n as the public keys, and keep d, p, q as the secret keys.

It is well-known that if we choose e with $\gcd(e, \phi(n)) = 1$ then the mapping

$$E : Z_n^* \to Z_n^* \quad \text{by} \quad E(m) = m^e \pmod{n} \quad \text{for} \quad m \in Z_n^*$$

becomes a one to one permutation on Z_n^*. For $n = p^r q^s$, the above choice of e in our proposed system gives a one-to-one permutation on Z_n^*. As in Takagi's system [1], the above choice of parameters p, q, e, d allows us to use shorter keys than in the RSA system with the same modulus size.

B.2 Encryption

Now suppose we have set up the keys, $n = p^r q^s$ and e, d, L. In the proposed system, the message space and the ciphertext space are Z_n^*. For a given message m, the ciphertext C is obtained by

$$C = m^e \pmod{n},$$

which is the same as RSA, multiprime technology, and Takagi's system.

B.3 Decryption

In the decryption, we use the same scheme as Takagi's system [2], but we apply the p-adic expansion to the factor p^r and apply q-adic expansion to the factor q^s. Since p, q are distinct primes, by Chinese Remainder Theorem, we have

$$Z_n^* \cong Z_{p^r} \times Z_{q^s}.$$

When we receive a ciphertext C in Z_n^*, C can be splitted into

$$C = (A, B), \quad A \in Z_{p^r},\ B \in Z_{q^s}.$$

Since C is a ciphertext, $C = m^e \pmod n$ for some $m \in Z_n^*$. Similarly, m can be splitted into two parts, $X \in Z_{p^r}$ and $Y \in Z_{q^s}$. It is easy to check that $A = X^e \pmod{p^r}$ and $B = Y^e \pmod{q^s}$. Since $X \in Z_{p^r}$, X can be represented as

$$X = X_0 + pX_1 + p^2X_2 + \cdots + p^{r-1}X_{r-1} \pmod{p^r}$$

for some $X_i \in Z_p$ with $0 \le i \le r-1$. Similarly we have

$$Y = Y_0 + qY_1 + q^2Y_2 + \cdots + q^{s-1}Y_{s-1} \pmod{q^s}$$

for some $Y_i \in Z_q$ with $0 \le i \le s-1$. Because of the similarity, it is enough to give a procedure to find X from a given $A = X^e \pmod{p^r}$ and e, d such that $ed = 1 \pmod{p-1}$. This procedure is given in Takagi's paper [1]. But we present the detail to analyse the complexity of this procedure.

Now suppose $A \in Z_{p^r}$ is written by

$$A = A_0 + pA_1 + p^2A_2 + \cdots + p^{r-1}A_{r-1} \pmod{p^r}.$$

For $1 \le i \le r-1$, we set

$$\begin{aligned} A[i] &= A_0 + pA_1 + \cdots + p^iA_i = (X_0 + pX_1 + \cdots + p^iX_i)^e \pmod{p^{i+1}} \\ F_i &= (X_0 + pX_1 + \cdots + p^{i-1}X_{i-1})^e. \end{aligned}$$

Then we note that $F_r \pmod{p^r} = A$ and $A[r-1] = A$. We also note that

$$\begin{aligned} A[i] &= A_0 + pA_1 + \cdots + p^iA_i \pmod{p^{i+1}} \\ &= (X_0 + pX_1 + \cdots + p^iX_i)^e \pmod{p^{i+1}} \\ &= (X_0 + pX_1 + \cdots + p^{i-1}X_{i-1})^e + eX_0^{e-1}p^iX_i \pmod{p^{i+1}} \\ &= F_i + eX_0^{e-1}p^iX_i \pmod{p^{i+1}}. \end{aligned}$$

It is easy to see that the following values

$$\begin{aligned} X_0 &= A_0^{d \pmod{p-1}} \pmod p, \\ X_i &= \frac{e^{-1}X_0^{1-e}(A[i] - F_i \pmod{p^{i+1}})}{p^i} \pmod p \text{ for } 1 \le i \le r-1, \end{aligned}$$

give X. Note that since e was chosen so $gcd(e,p) = 1$ and X_0 in Z_p^*, the terms e^{-1} and X_0^{1-e} have a unique meaning modulo p. We have reduced the equations into modulo p which is simplified compare to Takagi's algorithm. The overall speed is not affected much, but it is simpler.

Now we estimate the complexity to get X. The complexity to compute X_0 is

$$\log{(d \pmod{p-1})}(\log p)^2 \approx (\log p)^3.$$

For $1 \leq i \leq r-1$, the most time consuming step to get X_i is

$$F_i \pmod{p^{i+1}} = (X_0 + pX_1 + \cdots + p^{i-1}X_{i-1})^e \pmod{p^{i+1}}.$$

Since we use small exponent e, the complexity to obtain X_i is dominated by $(\log p^{i+1})^2$. Thus the complexity to get X is

$$(\log p)^3 + \sum_{i=2}^{r} (\log p^i)^2.$$

By the similar method, we can compute $Y \in Z_{q^s}^*$ so that $B = Y^e \pmod{q^s}$, and the complexity to get Y is

$$(\log q)^3 + \sum_{j=2}^{s} (\log q^j)^2.$$

Now by using CRT, we get the unique message $m \in Z_{p^r q^s}$ that satisfies

$$m = X \pmod{p^r}, \quad \text{and} \quad m = Y \pmod{q^s},$$

so we have recovered the message.

C The Efficiency of the Proposed Cryptosystem

In this section, we shall discuss about the complexity of the proposed scheme and compare the efficiency of RSA with CRT, multiprime technology, Takagi's system and our proposed scheme. Takagi's system is a special case $s = 1$ of our $n = p^r q^s$.

There is no difference in the encryption for all these methods, hence we only consider the complexity involved in the decryption process.

C.1 The Complexity of the Decryption

The complexity of the decryption for the proposed system with $n = p^r q^s$ is dominated by

$$\begin{aligned}(\log p)^3 &+ \sum_{i=2}^{r} (\log p^i)^2 + (\log q)^3 + \sum_{i=2}^{s} (\log q^i)^2 \\ &= (\log p)^3 + \sum_{i=2}^{r} i^2 (\log p)^2 + (\log q)^3 + \sum_{i=2}^{s} i^2 (\log q)^2 \\ &= 2(\log p)^3 + \left[\sum_{i=2}^{r} i^2 + \sum_{i=2}^{s} i^2 \right] (\log p)^2 .\end{aligned}$$

The last equality holds because p, q are of the same size. In this estimate, the dominant term is $2(\log p)^3$. When we use u distinct primes of the same size (i.e., modulus is of the form $n = p_1^{e_1} \cdots p_u^{e_u}$) in the proposed scheme, the main complexity is $u(\log p_1)^3$. Since the sum of the exponents is fixed, we conclude that the best efficiency can be obtained in the cases using two primes, i.e., $n = p^r q^s$. We also note that the term $\sum_{i=2}^{i=r} i^2 + \sum_{i=2}^{i=s} i^2$ has the minimum value when r and s are about the same if $r + s$ is fixed. We also note that r and s need to be relatively prime for security concerns. Hence the modulus of the form

- (case 1) $n = p^r q^{r+1}$ if the sum $r + s$ of exponents is odd
- (case 2) $n = p^{r-1} q^{r+1}$ if the sum $r + s$ of exponents is 0 modulo 4
- (case 3) $n = p^{r-2} q^{r+2}$ if the sum $r + s$ of exponents is 2 modulo 4

gives the best efficiency for the modulus of the form $n = p^r q^s$.

Our proposed scheme is exactly the same as Takagi's system when the sum of exponents is 3, 4, or 6. We note that our system is faster than the Takagi's system in all the steps of the decryption procedure. The difference of complexities of Takagi's system and our proposed system is at least $\frac{6r(r^2-1)}{3(2r+1)^2}(\log n)^2$ in case 1, $\frac{(r-2)(2r+1)}{4r}(\log n)^2$ in case 2, and $\frac{(r-3)(2r^2+3r+1)}{4r^2}$ in case 3. Table 5 and Table 6 illustrate the differences of the crucial complexities of Takagi's and ours.

And the decryption complexity for RSA with CRT is $\frac{(\log n)^3}{4}$. Hence we can say that our proposed scheme is $\frac{t^3}{8}$ times faster than the original RSA system with CRT where $t = 2r + 1$ in case 1, and $t = 2r$ in cases 2 and 3.

When we compute in a parallel environment, the proposed scheme is $\frac{t^3}{4}$ faster than the original RSA with CRT. In the multiprime technology with the modulus

Table 5. Decryption complexity of ours

	ours	
cases	modular form	complexity
1	$n = p^r q^{r+1}$	$\frac{2}{(2r+1)^3}(\log n)^3 + \frac{2r^3+6r^2+7r-3}{3(2r+1)^2}(\log n)^2$
2	$n = p^{r-1} q^{r+1}$	$\frac{2}{8r^3}(\log n)^3 + \frac{2r^3+3r^2+7r-3}{12r^2}(\log n)^2$
3	$n = p^{r-2} q^{r+2}$	$\frac{2}{8r^3}(\log n)^3 + \frac{2r^3+3r^2+25r+6}{12r^2}(\log n)^2$

Table 6. Decryption complexity of Takagi's

	Takagi's	
cases	modular form	complexity
1	$n = p^{2r}q$	$\frac{2}{(2r+1)^3}(\log n)^3 + \frac{8r^3+6r^2+r-3}{3(2r+1)^2}(\log n)^2$
2	$n = p^{2r-1}q$	$\frac{2}{8r^3}(\log n)^3 + \frac{8r^3-6r^2+r-3}{12r^2}(\log n)^2$
3	$n = p^{2r-1}q$	$\frac{2}{8r^3}(\log n)^3 + \frac{8r^3-6r^2+r-3}{12r^2}(\log n)^2$

$n = p_1 p_2 \cdots p_t$, the decryption speed is about $\frac{t^3}{4}$ times faster than RSA system with CRT and this is one of the fastest techniques among RSA-like cryptosystems obtained by varying the number of prime factors of the modulus. If we assume that the computing can be performed in parallel, then our proposed scheme gives about the same decryption speed as the multiprime technology. But for our proposed scheme, we only need two exponentiators for parallel computation while one needs t exponentiators in the multiprime technology using the modulus of a product of t distinct primes. Thus we have

Theorem 1. *For an RSA-like cryptosystem (i.e., the encryption function is defined by e-th exponentiation in Z_n^*) with modulus of the form $n = p_1^{e_1} p_2^{e_2} \cdots p_u^{e_u}$, the fastest decryption speed can be obtained in the case $u = 2$ and e_1, e_2 are about the same as long as we do not consider a parallel computation. In the parallel computing mode, this gives about the same speed as the multiprime technology, but only using two exponentiators.*

D The Security of the Proposed System

The security of our scheme is very similar to Takagi's system [1]. In this paper we only consider the attacks by factoring the modulus. First of all, we assume that r, s are relatively prime. It is because the actual complexity of factorization of $p^r q^s$ is the complexity of the factorization of $p^{\frac{r}{gcd(r,s)}} q^{\frac{s}{gcd(r,s)}}$. In this section we shall consider the case $n = p^r q^{r+1}$ only. The other cases are similar.

D.1 Factorization of $n = p^r q^{r+1}$

First we show that knowing pq from $n = p^r q^{r+1}$ gives the prime factors p and q in polynomial time. Note that if we have pq and $n = p^r q^{r+1}$, then $\frac{n}{p^r q^r} = q$ can be obtained directly. And then r-th root of a positive integer can be obtained in polynomial time, so it gives p in polynomial time. Thus we have

Theorem 2. *Knowing pq from $n = p^r q^{r+1}$ gives the prime factors p and q in polynomial time.*

But there is no known methods of finding pq from $n = p^r q^{r+1}$ without knowing p, q. Currently, the best known factoring algorithms for large numbers are the Number Field Sieve(NFS), Elliptic Curve Method(ECM).

D.2 Proper Choices for r in $n = p^r q^{r+1}$ to Defeat LFM

In [3] Boneh *et.al.* developed an efficient factoring algorithm LFM to factor numbers of the form $n = p^r q$. They compared the complexity of LFM, ECM and NFS. Since the complexity of NFS depends only on the size of the number to be factored, they focused on comparing ECM and LFM to give a guide line in choosing proper r in $n = p^r q$. At the end of the paper [3], they proposed an interesting question the applicability of LFM to the case $n = p^r q^s$ where r, s are about the same size. We show that we can apply LFM to $n = p^r q^{r+1}$, by modifying their method. And we can give a similar bound for r to guarantee $n = p^r q^{r+1}$ is secure against LFM.

Theorem 3. *Let $N = p^r q^{r+1}$ where p, q are of the same size. The factor pq can be recovered from N, r by an algorithm with a running time of :*

$$\exp\left(\frac{3}{2r+1}\log pq\right) O(\gamma),$$

where γ is the time it takes to run LLL on a lattice of dimension $O(r^2)$ with entries of size $(r \log N)$. The algorithm is deterministic, and runs in polynomial space.

We are going to sketch the proof. In Theorem 3.1 of [3], for $n = p^r q$, p can be recovered with a running time

$$\exp\left(\frac{c+1}{r+c}\log p\right) O(\gamma).$$

If we apply this result to the modulus $n = p^r q^{r+1}$, the running time to find p by the LFM algorithm is an exponential time $\exp(\frac{r+2}{2r+1}\log p)O(\gamma)$ algorithm. In fact, in the proof of the theorem 3.1 of [3], they didn't use the primality of p, q. We rewrite n by $n = (pq)^r q$, and Now we apply LFM to find pq and the running time for finding pq from $n = p^r q^{r+1}$ using LFM is

$$\exp\left(\frac{1/2+1}{r+1/2}\log pq\right) O(\gamma).$$

Hence when $r \geq \log(pq)$, then we can factor pq from n in a polynomial time, so it gives p, q in polynomial time by Theorem 4.1.

Hence for the modulus $n = p^r q^{r+1}$, we conclude ;

- When r satisfies $r \geq \log(pq)$, i.e., $r \geq 2\log p$ then $n = p^r q^{r+1}$ can be factored in polynomial time.
- When r satisfies $r < \sqrt{2\log p}$, then ECM is more effective than LFM.

D.3 Optimal Number of Prime Factors of n to Defeat Both of ECM and NFS

The complexity of NFS depends on the size of the modulus, and the complexity of ECM depends on the size of the smallest prime factor. Since we assume that all the prime factors have the same sizes, the complexity of ECM depends on the number of the prime factors of the modulus. Hence for the modulus $n = p^r q^s$, NFS is more effective for factoring n when the number $t = r + s$ of prime factors of n is small, and ECM becomes more effective as t gets larger.

Our object is to choose the optimal number $t = r + s$ to defeat ECM for $n = p^r q^s$. In the multiprime technology, they observed that the optimal number of prime factors with respect to NFS and ECM can be obtained by equating MIPS-Years of NFS and ECM to factor n.

It is well-known that we have the following asymptotic estimates for the running time of NFS, and ECM to factor n when p is one of prime factors of n

$$\text{MIPS-Years for NFS} \approx 1.5 \times 10^{-5} \exp\left(1.923 \sqrt[3]{\log n (\log\log n)^2}\right),$$
$$\text{MIPS-Years for ECM} \approx 3 \times 10^{-15} D^2 \exp\sqrt{2 \log p \log\log p},$$

where D is the number of decimal digits of n.

We first determine the security margin (in MIPS-Years) for current computing technology, and then determine the size of the modulus within the security margin against NFS to factor the modulus. In fact, a table for the proper security margin for coming years and the corresponding RSA modulus size is given in [7]. Hence if we choose proper modulus size according either to the table given by A.K. Lenstra and E.R. Verheul or other appropriate techniques, then the size of the prime factor of the modulus n that guarantees the security against ECM can be determined by solving the following approximated equation :

$$3 \times 10^{-15} D^2 \exp\sqrt{2 \log p \log\log p} \approx 1.5 \times 10^{-5} \exp\left(1.923 \sqrt[3]{\log n (\log\log n)^2}\right).$$

Now we simplify the above approximated equation and we get an approximated equation that solves the optimal number of prime factors of n with respect to the known factoring algorithms.

Theorem 4. *Let $n = p^r q^s$ is of α bits and p, q are of the same size then the solution $t = r + s$ of*

$$\sqrt{2\frac{\alpha}{t} \log \frac{\alpha}{t}} \approx 1.923 \sqrt[3]{\alpha (\log \alpha)^2} - 2 \log \alpha + 36$$

gives the optimal number of prime factors of n with respect to NFS, ECM and LFM as long as $r < \sqrt{2 \log p}$.

When we get a solution t of the above approximated equation, we can decide the form of n as one of $p^r q^{r+1}, p^{r-1} q^{r+1}$ or $p^{r-2} q^{r+2}$ depending on t is odd, 0 modulo 4, or 2 modulo 4, respectively.

For example, we can use $n = pq^2$ for the modulus of 1024 bits ; $n = p^1 q^3$ for modulus 4096 bits ; and $n = p^2 q^3$ for modulus 8192 bits maintaining the same security level as RSA with the same modulus size.

D.4 Choices for $t = r + s$ for $n = p^r q^s$ That Gives the Same Security as the Given RSA Modulus N

Another way to use our scheme is to give variational size of $n = p^r q^s$ according to the recommended RSA modulus N. Usually the secure choice for the RSA modulus is determined by the amount of the work (in MIPS-Years) to factor the modulus.

Suppose W_0 MIPS-Years is needed to factor the modulus N using NFS. And suppose W_0 MIPS-Years is needed to factor $n = p_1^{r_1} q_1^{s_1}$ using ECM, where $r_1 + s_1 = t_1$ and n has the same size as N. Let N be of α bits. Now we shall select a proper number $t_k = r_k + s_k$ of prime factors of $n_k = p^{r_k} q^{s_k}$, where n_k is a modulus of the size $k\alpha$ bits. In order to choose the optimal $t_k = \frac{k\alpha}{\log p}$, we need to solve

$$W_0 \approx 3 \times 10^{-15} D_k^2 \ \exp \sqrt{2 \log p \log \log p},$$

where $D_k^2 = \left(\frac{k\alpha}{\log 10}\right)^2$. But we already have that

$$W_0 \approx 3 \times 10^{-15} \left(\frac{\alpha}{\log 10}\right)^2 \exp \sqrt{2 \frac{\alpha}{t_1} \log \frac{\alpha}{t_1}}.$$

Thus we get an approximated equation for t_k, the number of prime factors of n_k, that gives an equivalent security as the given modulus size α of the RSA system.

Theorem 5. *Suppose that $t_1 = r_1 + s_1$ is the optimal number of prime factors of n of α bits that gives the same security level of the RSA system with modulus size α bits. Then when we expand the size of the modulus by k times, i.e., for modulus n_k of $(k\alpha)$ bits, the optimal number $t_k = r_k + s_k$ for $n_k = p^{r_k} q^{s_k}$ to defeat both of ECM and NFS can be obtained by solving the following approximated equation ;*

$$\frac{k\alpha}{t_k} \log \frac{k\alpha}{t_k} \approx \frac{\alpha}{t_1} \log \frac{\alpha}{t_1} - 2 \log k.$$

As before, when we have a solution t_k of the above equation, then we decide the form of the modulus n_k of the size $k\alpha$ depending on the value of t_k.

We see that t_k gets larger as k increases. Hence the rate of the efficiency improvement of the scheme using $n_k = p^{r_k} q^{s_k}$ compare to the RSA-$(k\alpha)$ is better as k increases. But when we expand the modulus size of RSA by a factor of k, the encryption speed of RSA-$k\alpha$ is k^2 times slower than RSA-α, and the decryption speed of RSA-$k\alpha$ is k^3 times slower than RSA-α, theoretically. Also, as k gets larger, we need more storage for keys, messages and computation environment. Hence we need to choose appropriate k by considering storage available and required encryption speed first and then choose proper r by solving the above equation.

E Conclusion

In this paper we have considered a generalization of Takagi's cryptosystem [1] with a generalized modulus $n = p_1^{e_1} \cdots p_u^{e_u}$. And we propose a public key cryptosystem with modulus of the form $n = p^r q^s$ which is one of the fastest among RSA-like system with general modulus $n = p_1^{e_1} \cdots p_u^{e_u}$. We've shown that the proposed system is faster than Takagi's system which is faster than the RSA system. We also investigated the optimal choices for r, s that gives a prescribed security level. Our proposed system has about the same efficiency as the multiprime technology when we are allowed to use parrallel computing, but needs less number of exponentiators than the multiprime technology. Multiprime scheme has recently been adopted in a WTLS application (RSA's BSAFE-WTLS-C), hence our system can also be adopted to similar applications.

References

1. T. Takagi,: *Fast RSA-type cryptosystem modulo $p^k q$*, Advances in Cryptology-CRYPTO'98 **LNCS 1462**(1998), pp. 318–326. 284, 286, 287, 290, 294
2. T. Takagi,: *Fast RSA-type cryptosystem using n-adic expansion*, Advances in Cryptology-CRYPTO'97 **LNCS 1294** (1997), pp. 372–384. 284, 287
3. D. Boneh, G. Durfee, N. Howgrave-Graham,: *Factoring $N = p^r q$ for large r*, Advances in Cryptology-CRYPTO'99 **LNCS 1666** (1999), pp. 326–337. 284, 284, 291, 291, 291, 291
4. Compaq Nonstop: *Cryptography Using Compaq Multiprime Technology in a parallel Processing Environment*, http://www.compaq.com (2000) 283
5. D. Coppersmith,: *Modifications to the number field sieve*, J. of Cryptology, Vol. 6 (1993), pp. 169–180. 284
6. A.K. Lenstra, H.W. Lenstra, Jr.,: *Algorithms in Number theory*, in Handbook of Theoretical Computer Science (Volume A : Algorithms and Complexity) (1990), pp. 673–715. 284
7. A.K. Lenstra, E.R. Verheul,: *Selecting Cryptographic Key Sizes*, http://www.cryptosavvy.com (1999). 283, 285, 285, 292

Author Index

A
Anand R. S. 93

B
Baier, H. 191
Biham, E. 43, 80
Boneh, D. 271
Buchmann, J. 191

C
Cheon, D. H. 262

D
Dahab, R. 203
Dunkelman, O. 43

F
Filiol, E. 31
Furman, V. 80

H
Hasan, A. 213

J
Jambunathan, K. 9
Jeong, I. R. 104

K
Kim, K. 237
Kim, M. 271
Kim, S. 283

L
Lee, S. J. 262
Lee, D. H. 104
Lee, H. 283
Lim, J, I. 262
Lim, S. 283
Lopez, J. C. 203

M
Mishra, S. K. 225
Modadugu, N. 271
Mu, Y. 155

O
Okeya, K. 178

P
Pandu Rangan, C. 117,130
Pastuszak, J. 143
Pei, D. 1
Pieprzyk, J. 143
Pliam, J. O. 67

Q
Qin, W. 1

R
Reyhani-Masoleh, 213

S
Safavi-Naini, R. 165,250
Sakurai, K. 178
Sarkar, P. 225
Seberry, J. 143
Selcuk, A. A. 52
Song, B. 237
Srinathan, K. 117,130
Susilo, W. 165

T
Tarannikov,Y. V. 19

V
Varadharajan, V. 155
Veni Madhavan, C. 93

W
Wang, H. 165
Wang, Y. 250

Y
Yie, J. 283