Taekyoung Kwon
Mun-Kyu Lee
Daesung Kwon (Eds.)

LNCS 7839

Information Security and Cryptology – ICISC 2012

15th International Conference
Seoul, Korea, November 2012
Revised Selected Papers

Springer

Taekyoung Kwon   Mun-Kyu Lee
Daesung Kwon (Eds.)

Information Security and Cryptology – ICISC 2012

15th International Conference
Seoul, Korea, November 28-30, 2012
Revised Selected Papers

Springer

Volume Editors

Taekyoung Kwon
Sejong University
Department of Computer Engineering
Seoul 143-747, Korea
E-mail: tkwon@sejong.edu

Mun-Kyu Lee
Inha University
School of Computer and Information Engineering
Incheon 402-751, Korea
E-mail: mklee@inha.ac.kr

Daesung Kwon
National Security Research Institute
Daejeon 306-600, Korea
E-mail: ds_kwon@ensec.re.kr

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-37681-8 e-ISBN 978-3-642-37682-5
DOI 10.1007/978-3-642-37682-5
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013935780

CR Subject Classification (1998): E.3, K.6.5, C.2, D.4.6, G.2.1, E.4, J.1

LNCS Sublibrary: SL 4 – Security and Cryptology

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

Printed on acid-free paper

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

Preface

ICISC 2012, the 15th International Conference on Information Security and Cryptology, was held in Seoul, Korea, during November 28–30, 2012. This year the conference was hosted by the KIISC (Korea Institute of Information Security and Cryptology) jointly with the NSRI (National Security Research Institute), in cooperation with the Ministry of Public Administration and Security (MOPAS).

The aim of this conference is to provide an international forum for the latest results of research, development, and applications in the field of information security and cryptology. This year we received 120 submissions from more than 20 countries and were able to accept 32 papers from 13 countries, with the acceptance rate of 26.7%. The review and selection processes were carried out by the Program Committee (PC) members, 88 prominent experts world-wide, via Springer's OCS system. First, each paper was blind reviewed by at least three PC members. Second, to resolve conflicts in the reviewer's decisions, the individual review reports were open to all PC members, and detailed interactive discussions on each paper ensued. For the LNCS post-proceedings, the authors of selected papers had a few weeks to prepare their final versions based on the comments received from the reviewers. We also recommended that authors should revise their papers based on the comments and recommendations they might have received from attendees upon their presentations at the conference.

The conference featured three invited talks: "Machine Learning on Encrypted Data" delivered by Kristin Lauter, Microsoft Research; "Another Look at Affine-Padding RSA Signatures" by David Naccache, Ecole Normale Superieure; and "New Meet-in-the-Middle Attacks in Symmetric Cryptanalysis" by Christian Rechberger, Technical University of Denmark. We thank the invited speakers for their kind acceptance and nice presentations.

We would like to thank all the authors who submitted their papers to ICISC 2012 and all 88 PC members. It was a truly nice experience to work with such talented and hard-working researchers. We also appreciate the external reviewers for assisting the PC members in their particular areas of expertise. Finally, we would like to thank all attendees for their active participation and the Organizing Committee Members, who nicely managed this conference. We look forward to next year's ICISC.

January 2013

Taekyoung Kwon
Mun-Kyu Lee
Daesung Kwon

ICISC 2012

The 15th Annual International Conference
on Information Security

November 28–30, 2012
Konkuk University, Seoul, Korea

Hosted by
Korea Institute of Information Security and Cryptology (KIISC)
National Security Research Institute (NSRI)

Supported by
Ministry of Public Administration and Security (MOPAS)
Electronics and Telecommunications Research Institute (ETRI)
Korea Internet & Security Agency (KISA)
The Korean Federation of Science and Technology Societies (KOFST)

General Chairs

Chang-Seop Park	KIISC and Dankook University, Korea
Seokyoul Kang	NSRI, Korea

Program Co-chairs

Taekyoung Kwon	Sejong University, Korea
Mun-Kyu Lee	Inha University, Korea
Daesung Kwon	NSRI, Korea

Program Committee

Frederik Armknecht	University of Mannheim, Germany
Joonsang Baek	KUSTAR, UAE
Alex Biryukov	University of Luxembourg, Luxembourg
Zhenfu Cao	Shanghai Jiao Tong University, China
Aldar C-F. Chan	Institute for Infocomm Research, Singapore
Ku-Young Chang	ETRI, Korea
Kefei Chen	Shanghai Jiaotong University, China
Jung Hee Cheon	Seoul National University, Korea
Yongwha Chung	Korea University, Korea
Nora Cuppens-Boulahia	TELECOM Bretagne, France

Susan Pancho-Festin	University of the Philippines, Philippines
Omkant Pandey	Microsoft Research India, India / University of Texas, Austin, USA
Raphael C.-W. Phan	Multimedia University, Malaysia
Christian Platzer	Automation Systems Group at the Technical University of Vienna, Austria
Carla Ráfols	Ruhr-Universität Bochum, Germany
C. Pandu Rangan	Indian Institute of Technology Madras, India
Christian Rechberger	DTU, Denmark
Vincent Rijmen	Katholieke Universiteit Leuven, Belgium
Bimal Roy	Indian Statistical Institute, India
Kouichi Sakurai	Kyushu University, Japan
Palash Sarkar	Indian Statistical Institute, India
Nitesh Saxena	University of Alabama, Birmingham, USA
Ji Sun Shin	Sejong University, Korea
Sang Uk Shin	Pukyong National University, Korea
Hong-Yeop Song	Yonsei University, Korea
Rainer Steinwandt	Florida Atlantic University, USA
Hung-Min Sun	National Tsing Hua University, Taiwan
Willy Susilo	University of Wollongong, Australia
Tsuyoshi Takagi	Kyushu University, Japan
Yukiyasu Tsunoo	NEC Corporation, Japan
Marion Videau	University of Lorraine / LORIA, France
Jorge Villar	Universitat Politecnica de Catalunya, Spain
Yongzhuang Wei	Guilin University of Electronic Technology, China
Wenling Wu	SKLOIS, Chinese Academy of Sciences, China
Toshihiro Yamauchi	Okayama University, Japan
Wei-Chuen Yau	Multimedia University, Malaysia
Ching-Hung Yeh	Far East University, Taiwan
Sung-Ming Yen	National Central University, Taiwan
Yongjin Yeom	Kookmin University, Korea
Jeong Hyun Yi	Soongsil University, Korea
Kazuki Yoneyama	NTT, Japan
Myungkeun Yoon	Kookmin University, Korea
Dae Hyun Yum	Myongji University, Korea
Aaram Yun	UNIST, Korea
Fangguo Zhang	Sun Yat-sen University, China

Organizing Chair

Dong Il Seo	ETRI, Korea

Organizing Committee

Heuisu Ryu	Gyeongin National University of Education, Korea
Hokun Moon	KT, Korea
Howon Kim	Pusan National University, Korea
Jason Kim	Korea Internet & Security Agency, Korea
Keecheon Kim	Konkuk University, Korea
Soohyun Oh	Hoseo University, Korea
Tae-Hoon Kim	SYWORKS, Korea
Young-Ho Park	Sejong Cyber University, Korea

Table of Contents

ML Confidential: Machine Learning on Encrypted Data

Thore Graepel[1], Kristin Lauter[1], and Michael Naehrig[1,2]

[1] Microsoft Research
{thoreg,klauter,mnaehrig}@microsoft.com
[2] Eindhoven University of Technology
michael@cryptojedi.org

Abstract. We demonstrate that, by using a recently proposed leveled homomorphic encryption scheme, it is possible to delegate the execution of a machine learning algorithm to a computing service while retaining confidentiality of the training and test data. Since the computational complexity of the homomorphic encryption scheme depends primarily on the number of levels of multiplications to be carried out on the encrypted data, we define a new class of machine learning algorithms in which the algorithm's predictions, viewed as functions of the input data, can be expressed as polynomials of bounded degree. We propose confidential algorithms for binary classification based on polynomial approximations to least-squares solutions obtained by a small number of gradient descent steps. We present experimental validation of the confidential machine learning pipeline and discuss the trade-offs regarding computational complexity, prediction accuracy and cryptographic security.

1 Introduction

Cloud service providers leverage their large investments in data centers to offer services which help smaller companies cut their costs. But one of the barriers to adoption of cloud services is concern over the privacy and confidentiality of the data being handled by the cloud, and the commercial value of that data or the regulations protecting the handling of sensitive data. In this work we propose a cloud service which provides confidential handling of machine learning tasks for various applications. Machine learning (ML) consists of two stages, the training stage and the classification stage, either or both of which can be outsourced to the cloud. In addition, when both stages are outsourced to the cloud, we propose an intermediate probabilistic verification stage to test and validate the learned model which has been computed by the cloud service. In the protocols we describe here, we identify three parties: the Data Owner, the Cloud Service Provider, and the Content Providers.

The Data Owner is the customer for the Cloud Service, and owns or is responsible for the data being processed. Content Providers upload data to the cloud, data which belongs to or is intended for the Data Owner. Content Providers could be remote devices, sensors or monitors which belong to the Data Owner, and which may have been authorized by the Data Owner, for providing data on the Data Owner's behalf. A typical scenario might be a patient who is the Data Owner, and Content Providers which consist of multiple health monitoring devices provisioned to monitor the patient's health and upload data to the Cloud Service. Alternatively, the Data Owner could be some large company with many lab technicians, partners, or contracted Content Providers which upload data to the Cloud Service on behalf of the company, for example in the financial, pharmaceutical, or social media industry. The Cloud Service may be run by a third party, a partner company, or even the company itself, off-premises or in some stand-alone facility.

Our rationale for proposing these protocols is that there are some scenarios where outsourcing *computation* to a Cloud Service makes sense from a practical and rational economic point of view. Namely, when data is collected or uploaded from many diverse sources or parties, an online service can host the collection, storage, and computation of and on this data without requiring interaction with the data owner. This service allows the data owner to access and query their potentially large amount of data at any time from a device with little computational or storage capacity. The Data Owner may subsequently designate privileges to other parties (such as a health care provider) to access the data or to receive alerts or updates concerning some other processed form of the data. When outsourcing computation to a service makes sense, *and* confidentiality of the data is an issue, then our protocols for providing confidential processing of sensitive data are relevant.

One way to preserve confidentiality of data when outsourcing computation is to encrypt the data before uploading it to the cloud. This may limit the utility of the data, but recent advances in cryptography allow searching on encrypted data and performing operations on encrypted data, all *without decrypting* it. An encryption scheme which allows arbitrary operations on ciphertexts is called a Fully Homomorphic Encryption (FHE) scheme. The first FHE scheme was constructed by Gentry [9], and subsequent schemes [20,4,3,10,11] have rapidly become more practical, with improved performance and parameters. Gentry's scheme and several of the subsequent FHE schemes have a so-called Somewhat Homomorphic Encryption (SHE) scheme as an underlying building block, and use a technique called bootstrapping to extend it to an FHE scheme. An SHE scheme performs additions and multiplications on encrypted data, but is limited in the amount of such computations it can perform, because encryption involves the addition of small noise terms into ciphertexts. Operating homomorphically on ciphertexts causes the inherent noise terms to grow, and correct decryption is only possible as long as these noise terms do not exceed a certain bound. Noise growth is much larger in homomorphic multiplications than in additions. This means that an SHE scheme can only evaluate polynomial functions of the data up to a bounded degree before the inherent noise grows too large. Bootstrapping, a very costly procedure, is then necessary to reduce the noise to its initial level, enabling fully homomorphic computation. While noise grows exponentially in SHE schemes, recent improvements have provided homomorphic schemes in which noise grows only polynomial in the number of levels of multiplications performed [3,2]. Such schemes are called Leveled Homomorphic Encryption (LHE) schemes, and they allow evaluation of polynomial functions of a higher, bounded degree without resorting to the bootstrapping component.

Recent schemes are based on computational hardness assumptions for problems related to well known lattice problems such as the Shortest Vector Problem (SVP). Specifically, schemes based on the Ring Learning With Errors (RLWE) assumption operate in polynomial rings, where polynomials can alternatively be viewed as vectors in a lattice. It was shown in [16] how the hardness of the RLWE problem is related to SVP.

In practice, as was observed in [14], many useful computational services only require evaluation of low-degree polynomials, so they can be deployed on encrypted data using only an LHE or SHE scheme. In this paper, we propose a confidential protocol for machine learning tasks, called **ML Confidential**, based on Homomorphic Encryption (HE), and we design confidential machine learning algorithms based on low-degree polynomial versions of classification algorithms. Section 2 describes the general ML Confidential protocol and discusses its security. Section 3 is devoted to explaining basic classification algorithms that can be expressed as low-degree polynomials, including the derivation of division-free, integer (DFI) versions of these algorithms. Section 4 describes the homomorphic encryption scheme we use in our proof-of-concept implementation of the division-free, integer classification algorithms. Our implementation is discussed in Section 5 together with some initial performance numbers and analysis.

Our experiments implement a Linear Means (LM) Classifier and Fisher's Linear Discriminant (FLD) Classifier on a publicly available data set, the Wisconsin Breast Cancer Data set from [8]. Using up to 100 training and test vectors with up to 30 features each for the training and classification stages, our experiments show that (LM) classification can be accomplished in roughly 6 seconds using an unoptimized mathematics software package running on a standard modern laptop. The FLD classifier runs in roughly 20 seconds for vectors with only 10 features. Across all experiments, we observe a slow-down of roughly $6-7$ orders of magnitude for operating on encrypted data at these parameter and data sizes. This compares favorably with other recent benchmarks for HE (see [11]).

Connections between cryptography and machine learning have been considered for a long time (see, e.g., [19]), mostly with the view that they are inverses of one another in the sense that cryptography aims to prevent access to information whereas machine learning attempts to extract information from data. Note that the Confidential ML problem discussed in this paper is also loosely related to doing inference on differentially private data (see [21] and references therein), the difference being that in our case the Cloud Service performing the inference calculations is not even able to interpret the results of its analysis.

2 The ML Confidential Protocol and Security Considerations

This section proposes the ML Confidential protocol based on a homomorphic encryption scheme that provides algorithms $\mathsf{HE.Keygen}, \mathsf{HE.Enc}, \mathsf{HE.Dec}$, and

$\mathsf{HE.Eval}$ for key generation, encryption, decryption, and homomorphic function evaluation. The scheme can be either a symmetric, secret key scheme or an asymmetric, public key scheme. It can be a fully-homomorphic scheme, in which case arbitrary machine learning algorithms can be carried out on the encrypted data by evaluating them with $\mathsf{HE.Eval}$. In a more practical case, it can be a somewhat or leveled homomorphic scheme, where the function $\mathsf{HE.Eval}$ can only evaluate polynomial functions of the input data with a bounded degree comprised of homomorphic addition $\mathsf{HE.Add}$ and multiplication $\mathsf{HE.Mult}$ on the message space. Therefore, in that case, machine learning algorithms are restricted to algorithms that can be expressed as polynomials with bounded degree. In either case, let $\mathsf{ML.Train}$ and $\mathsf{ML.Classify}$ be the training and classification algorithms of the machine learning task which can be homomorphically carried out on encrypted data with the function $\mathsf{HE.Eval}$.

Three types of parties interact in the protocol: the Data Owner, the Cloud Service Provider, and Content Providers. The protocol comprises the following main components.

Key Generation. The Data Owner executes the $\mathsf{HE.Keygen}$ algorithm for either a private key or a public key version of the homomorphic encryption scheme. For the private key version, the Data Owner shares the private encryption key with the Content Providers and they all securely store the key locally. For the public key version, the Data Owner publishes the public key and securely stores the private key locally.

Encryption and Upload of Training Data. Content Providers encrypt confidential, labeled data to upload to the Cloud. For all classes of training vectors, and for all training vectors $\mathbf{x}$ in each class, the Content Providers encrypt $\mathbf{x}$ and send $\mathsf{HE.Enc}(\mathsf{ek}, \mathbf{x})$ to the Cloud Service Provider along with the unencrypted label of the class. Here ek is the encryption key that is known to the Content Providers, i.e. it is equal to the secret key in the symmetric version and to the public key in the asymmetric version of the scheme. Alternatively, the Content Providers may encrypt preprocessed versions of the training set data, e.g. synthetic data such as class sums or class-conditional covariance matrices (i.e. sufficient statistics) for each class of training vectors.

Training. The Cloud Service Provider computes an encrypted Learned Model. Training vectors consisting of encrypted, labeled content, $\mathsf{HE.Enc}(\mathsf{ek}, \mathbf{x})$, are processed by the Cloud Service Provider. This means that the algorithm $\mathsf{HE.Eval}$ of the homomorphic encryption scheme evaluates the machine learning training phase $\mathsf{ML.Train}$ homomorphically on the encrypted training vectors. An encrypted form of the Learned Model is stored by the Cloud Service Provider and can be returned to the Data Owner on request.

Classification. An encryption $\mathsf{HE.Enc}(\mathsf{ek}, \mathbf{x})$ of a test vector $\mathbf{x}$, which usually has not been used in the training stage, is sent to the Cloud Service Provider by the Data Owner or the Content Providers. The Cloud Service Provider evaluates the classification phase $\mathsf{ML.Classify}$ of the machine learning task on the encrypted test vector using the encrypted learned model, and encrypted classifications are returned to the Data Owner. The Data Owner decrypts the results to obtain the classifications.

Verification of the Learned Model. The Data Owner probabilistically tests the Learned Model. The Data Owner encrypts test vectors with known classifications and sends the ciphertexts to the Cloud Service Provider. The Cloud Service Provider classifies the encrypted vectors homomorphically and returns encrypted classification results to the Data Owner. The Data Owner decrypts the results and compares with the known classification labels to assess the test error of the Learned Model in the Cloud.

Security Considerations. The protocol assumes a model in which the Cloud is an *Honest but Curious* party, i.e. the Cloud will follow the stated protocol to provide the desired functionality, and will not deviate nor fail to provide the service or return results, but that it is *Curious* in the sense that it would look at available information. This assumption is reasonable to model a rational, economically motivated Cloud Service Provider: the Cloud is motivated to provide excellent service, and yet would be motivated to take advantage of extra available information. A *Malicious* Cloud is a much stronger adversary, who would potentially mishandle calculations, delete data, refuse to return results, collude with other parties, etc. In most of these malicious behaviors, the Cloud would be likely to get caught, and thus damage its reputation if trying to run a successful business.

The verification step we propose is analogous to a naive version of Proof-of-Storage (PoS) protocols. Verification requires the Data Owner to store a certain number of labeled samples locally in order to be able to test correctness (and determine test errors) of the Cloud's computations. After the training stage, the Data Owner encrypts the test vectors and queries the cloud to provide encrypted classifications of the test vectors, and then the Data Owner decrypts and compares to the correct label. Since we are assuming an Honest but Curious model for the Cloud, the Data Owner only needs to store enough test vectors to determine the test error of the Cloud (or detect any accidental error). We are also implicitly assuming that the Content Providers do not behave maliciously, and correctly encrypt and upload data.

The Cloud must necessarily learn a certain amount of information in order to provide the functionality required. The Cloud computes an encrypted Learned Model from a collection of encrypted and labeled training vectors in Stage 1 and provides encrypted classifications of encrypted test vectors in Stage 2. This includes knowing the number of vectors used in the training phase, and the number of test vectors submitted for classification. In addition, our scheme discloses the number of vectors within each class, and also an upper bound on the entries in the test vectors can be deduced once the parameters for the HE scheme and the number of test vectors are known.

The underlying HE schemes are assumed to be randomized and have semantic security against passive adversaries, a property which ensures that an adversary cannot distinguish an encryption of one message from another. The Cloud handles encrypted data and performs HE operations, and in the public key setting, can encrypt messages of its choice. However, the Cloud does not obtain decryptions of the ciphertexts that it handles.

3 Polynomial Machine Learning

As discussed in Section 2, a homomorphic encryption scheme can be used to implement the ML Confidential protocol to run machine learning algorithms on encrypted training and test data. An FHE scheme theoretically supports arbitrary computations and thus imposes no restrictions on the ML algorithms used in the protocol. However, implementing a scheme that is fully homomorphic and does not require fixing a specific bound on the complexity of the computation to be done is very costly due to the necessity of bootstrapping.

Useful and flexible as it may be, a fully homomorphic scheme is rarely necessary for most applications, see for example [14]. Instead, if the computation is simple and of low complexity, it is possible to use an SHE or LHE scheme. This not only avoids the expensive bootstrapping procedure, but might also result in smaller parameters to instantiate the scheme, leading to a more practical instantiation of homomorphic encryption. Vice versa, fixing an SHE or LHE scheme in advance raises the question of which applications are possible under the restrictions imposed by the homomorphic capability of the scheme. In practice, we can assume that an SHE or LHE scheme with fixed parameters can homomorphically evaluate polynomials of a fixed limited degree D in the encrypted elements of the message space. This means it can homomorphically evaluate and still correctly decrypt a product of D message elements, while a product of $D+1$ elements can not necessarily be decrypted correctly. This section shows that even when using a scheme that is restricted to evaluating polynomials for which the degree bound D is relatively small, it is still possible to perform meaningful machine learning tasks confidentially.

Let us assume that we are given an HE scheme that is able to homomorphically evaluate polynomial functions of encrypted messages of degree at most D, and that we aim at performing a machine learning algorithm on encrypted data. This means that the predictions viewed as functions of the training and test data must be polynomials of limited degree D. Note that, when the classification stage is included, this restriction does not only refer to the actual input-output mapping learned by the algorithm but to the dependency of the predictions on the training and test data. To capture this limitation, we define a class of machine learning algorithms which are represented by polynomial functions of bounded degree.

Definition 1 (Polynomial learning/prediction algorithm). *Let* $A : (\mathbb{R}^n \times \mathcal{Y})^m \times \mathbb{R}^n \to \mathcal{Y}$ *be a learning/prediction algorithm that takes a training sample* $(\mathbb{R}^n \times \mathcal{Y})^m$ *of size* m *and a test input* $\mathbf{x} \in \mathbb{R}^n$ *and returns a prediction* $y \in \mathcal{Y}$. *If the function* A *is at most a polynomial of degree* D *in its arguments, then we call the learning/prediction algorithm* D*-polynomial.*

Straightforward implementation of many machine learning algorithms requires operations which are not necessarily represented by a low-degree polynomial, ruling out certain algorithms, namely:

Comparison. A comparison $x > y$ for $x, y \in \mathbb{R}$ is not D-polynomial, unless the inputs are encrypted bit-wise and a very deep circuit for comparison is implemented. This rules out learning algorithms like the perceptron or the support vector machine because they derive their class labels from thresholding real numbers. It also rules out the k-nearest neighbors classifier, which requires ordering neighbors according to distance, and decision trees, which threshold features at the nodes of the tree.

Division. A division x/y for $x \in \mathbb{R}$ and $y \in \mathbb{R} \setminus \{0\}$ is not D-polynomial. This rules out algorithms that rely on matrix inversion such as exact Fisher's linear discriminant for classification and the standard rule for determining the coefficients in regression.

Other Non-polynomial Functions. Other functions such as trigonometric functions or the exponential function are not D-polynomial, which rules out methods like exact logistic or probit regression and non-linear neural networks which rely on the evaluation of sigmoidal functions, in particular bounded sigmoid functions which are hard to approximate with polynomials.

Given the restrictions imposed by a homomorphic encryption scheme that can only guarantee correct evaluation of polynomial functions of bounded degree, we are still able to design non-trivial machine learning algorithms. Often, it is even possible to sufficiently approximate the above mentioned functions by polynomials of a bounded degree, for example by means of truncated Taylor series. The exponential function can be approximated by a truncation of its Taylor series, so approximate versions of logistic regression can be implemented with HE as was suggested in [14].

The above definition can be applied directly to regression learning algorithms where $\mathcal{Y} = \mathbb{R}^{n'}$, and tells us that exact least-squares linear regression is not D-polynomial due to the required matrix inversion. Note that classification algorithms cannot be D-polynomial by definition because they have discrete outputs $y \in \mathcal{Y}$. However, in this case we can still use the above definition as guidance if we decompose a classification algorithm as $A = g \circ f$, with a mapping $f : (\mathbb{R}^n \times \mathcal{Y})^m \times \mathbb{R}^n \to \mathbb{R}^{n'}$ to a vector of real-valued scores, and a discretization operation $g : \mathbb{R}^{n'} \to \mathcal{Y}$. This decomposition is possible for a large class of algorithms including Linear Discriminant Analysis and Support Vector Machines, and allows the Cloud Service to evaluate the function f under D-polynomial HE, and the Data Provider to evaluate the function g. In the following, we focus on the task of binary classification to deduce examples of D-polynomial machine learning algorithms, but note that tasks like regression and dimensionality reduction could be cast in a similar framework.

3.1 Classification

Let us consider the case of binary classification with inputs in $\mathbb{R}^n$ and binary target outputs from $\mathcal{Y} = \{+1, -1\}$. We consider a linear classifier of the form $A(\mathbf{x}; \mathbf{w}, c) := \mathrm{sign}(f(\mathbf{x}; \mathbf{w}, c))$ with the score function $f(\mathbf{x}; \mathbf{w}, c) := \mathbf{w}^T\mathbf{x} - c$. We assume in the Confidential ML protocol that it is known for two encrypted training examples, whether they are labeled with the same classification (without revealing which one it is). We therefore consider the cardinalities of the positive and negative training sets to be known as well as which ciphertexts encrypt data vectors that belong to the same class. Hence we can carry out operations on these two sets separately. This leads us to consider the simple Linear Means and Fisher's Linear Discriminant classifiers, both of which require only class-conditional statistics to be evaluated.

Linear Means Classifier. The Linear Means (LM) classifier determines $\mathbf{w}$ and c such that $f(\mathbf{x};\mathbf{w},c)=0$ defines a hyper-plane midway on and orthogonal to the line through the two class-conditional means. It can be derived as the Bayes optimal decision boundary in the case that the two class-conditional distributions have identical isotropic Gaussian distributions [5].

Let $I_y := \{i \in \{1,\ldots,m\} | y_i = y\}$ be the index set of training examples with label y and let $m_y := \|I_y\|$. Calculate the class-conditional mean vectors as $\mathbf{m}_y := m_y^{-1}\mathbf{s}_y$ with $\mathbf{s}_y := \sum_{i\in I_y}\mathbf{x}_i$, from which we obtain the weight vector as the difference vector between the two class-conditional means $\mathbf{w}^* := \mathbf{m}_{+1}-\mathbf{m}_{-1}$. The value of the threshold c is calculated using the condition $\mathbf{w}^{*T}\mathbf{x}_0 - c = 0$ for the mid-point, $\mathbf{x}_0 := (\mathbf{m}_{+1}+\mathbf{m}_{-1})/2$, between the two class means, which gives for the threshold: $c^* = (\mathbf{m}_{+1}-\mathbf{m}_{-1})^T(\mathbf{m}_{+1}+\mathbf{m}_{-1})/2$. For a given test example $\mathbf{x}$ the score $f^*(\mathbf{x};\mathbf{w}^*,c^*) := \mathbf{w}^{*T}\mathbf{x} - c^*$ is a quadratic function in the training data and a linear function in the test example and the LM classifier is hence 2-polynomial.

Fisher's Linear Discriminant Classifier. Now let us move on to a more demanding example, Fisher's linear discriminant (FLD) classifier [7]. This algorithm is similar to the Linear Means classifier, but does take into account the class-conditional covariances. It aims at finding a projection that maximizes the separation between classes as the ratio S between the variance σ^2_{inter} between classes and the variance σ^2_{intra} within classes,

$$S := \frac{\sigma^2_{\text{inter}}}{\sigma^2_{\text{intra}}} = \frac{\mathbf{w}^T\mathbf{D}\mathbf{w}}{\mathbf{w}^T\mathbf{C}\mathbf{w}} \tag{1}$$

with $\mathbf{D} := \mathbf{d}\mathbf{d}^T$ and $\mathbf{d} := \mathbf{m}_{+1}-\mathbf{m}_{-1}$ and $\mathbf{C} := \mathbf{C}_{+1}+\mathbf{C}_{-1}$. Here, $\mathbf{C}_y := \frac{1}{m_y}\sum_{i\in I_y}(\mathbf{x}_i-\mathbf{m}_y)(\mathbf{x}_i-\mathbf{m}_y)^T$ is the class-conditional covariance matrix of the data. Taking the gradient w.r.t. $\mathbf{w}$ and setting it to zero shows that $\mathbf{w}^*$ is the solution of a generalized eigenvalue problem $\mathbf{D}\mathbf{w} = \lambda\mathbf{C}\mathbf{w}$. Since $\mathbf{D} = \mathbf{d}\mathbf{d}^T$ has rank one, we can write $\mathbf{D}\mathbf{w} = a\mathbf{d}$ for some $a \in \mathbb{R}$ and hence $\mathbf{C}\mathbf{w}^* \propto \mathbf{d}$. Determining $\mathbf{w}^*$ requires solving a linear system of equations, i.e. it can be determined by calculating the inverse $\mathbf{C}^{-1}$ exactly. This requires division, which is not D-polynomial. In what follows, we refer to this approach as the exact FLD algorithm.

In a second approach, we aim at solving the linear system approximately using a least-squares approach so as to obtain a D-polynomial learning/prediction algorithm. The straight-forward cost function is $E(\mathbf{w}) := \frac{1}{2}||\mathbf{C}\mathbf{w}-\mathbf{d}||^2$, but instead of the standard Euclidean norm, we choose $||\mathbf{v}||^2 := \mathbf{v}^T\mathbf{C}^{-1}\mathbf{v}$ for better conditioning. Then the gradient is $\nabla_{\mathbf{w}} E(\mathbf{w}) = \mathbf{C}\mathbf{w} - \mathbf{d}$ and we can use gradient descent to find the solution $\mathbf{w}^*$. Once $\mathbf{w}$ has been found the threshold can be chosen as $c^* := \mathbf{w}^{*T}(\mathbf{m}_{+1} + \mathbf{m}_{-1})/2$.

The challenge then is to approximately solve a linear system using as few multiplications as possible. For the sake of illustration, let us consider standard gradient descent with a fixed learning rate η. If we define $\mathbf{R} := \mathbf{I} - \eta\mathbf{C}$ and $\mathbf{a} := \eta\mathbf{d}$, we obtain the well-known recursion $\mathbf{w}_{j+1} = \mathbf{R}\mathbf{w}_j + \mathbf{a}$. Defining $\mathbf{w}_0 = \mathbf{0}$, we can express the rth order approximation $\mathbf{w}_r$ of $\mathbf{w}^*$ as

$$\mathbf{w}_r := \left(\sum_{j=0}^{r-1} \mathbf{R}^j \right) \mathbf{a} = \eta \left(\sum_{j=0}^{r-1} (\mathbf{I} - \eta\mathbf{C})^j \right) \mathbf{d} \, . \tag{2}$$

This series converges if the spectral radius of $\mathbf{R}$ is less than one, i.e., if the absolute value of its largest eigenvalue is less than one, which can be ensured by choosing η sufficiently small. Depending on the order of approximation r, we obtain a D-polynomial FLD algorithm with $D = 2(r-1) + 1$. Note that the sufficient statistics for the FLD algorithm are the class-conditional means $\mathbf{m}_y$ and covariance matrices $\mathbf{C}_y$. If it is desired to reduce the required communication overhead at the cost of increasing the Data Provider workload, then instead of transmitting the raw training data to the Cloud Provider, the Data Provider can calculate and transmit the sufficient statistics for the training data instead.

3.2 Division-Free Integer Algorithms for Classification

In all of the above, the data input to a machine learning algorithm has been treated as being comprised of vectors of real numbers. Using standard representations for floating point numbers, one could encrypt approximations to such numbers bitwise and then operate on single bit encryptions, mimicking the unencrypted computations. For the sake of efficiency, it is necessary to deviate from this bitwise encryption paradigm. Instead, we consider messages being integers or polynomials with integer coefficients. In most of the recent, more practical homomorphic encryption schemes, it can be assumed that integers up to a certain size can be embedded into the scheme's message space, and that the homomorphic operations correspond to the same operations on integers, respectively. In such a setting, it is not possible to perform non-polynomial operations, leaving only polynomial functions on integers as the only practical possibility. To encode a real number by an integer, it can first be approximated to a certain precision by a rational number. Multiplying all such approximations through with a fixed denominator and rounding to the nearest integer provides an integer approximation to the original real numbers. We assume from now on that approximations to real numbers are represented by integers and that we homomorphically embed such representations into the message space of the HE scheme.

In particular, this means that we must avoid divisions since there is no corresponding operation for encoded integers. Below, we describe division-free integer (DFI) versions of the LM classifier and the FLD classifier described in Section 3.

The DFI versions of these algorithms are obtained by multiplying with all possible denominators occurring in the computations and adjusting the formulas to exactly compute multiples with the same sign of all magnitudes involved. In detail, computations for both classifiers are as follows.

Linear Means Classifier. For the LM Classifier we compute $m_{-1}\mathbf{s}_{+1}$ and $m_{+1}\mathbf{s}_{-1}$ instead of $\mathbf{m}_{+1}$ and $\mathbf{m}_{-1}$, and replace the weight vector by

$$\tilde{\mathbf{w}}^* := m_{-1}\mathbf{s}_{+1} - m_{+1}\mathbf{s}_{-1} = m_{+1}m_{-1}(\mathbf{m}_{+1} - \mathbf{m}_{-1}) = m_{+1}m_{-1}\mathbf{w}^*. \tag{3}$$

Similarly, the threshold is replaced by $\tilde{c}^* = 2m_{+1}^2 m_{-1}^2 c^*$ using $\tilde{\mathbf{x}}_0 := m_{-1}\mathbf{s}_{+1} + m_{+1}\mathbf{s}_{-1} = 2m_{+1}m_{-1}\mathbf{x}_0$. Given a test vector $\mathbf{x}$, we use the classifier $\tilde{f}^*(\mathbf{x}; \tilde{\mathbf{w}}^*, \tilde{c}^*) := 2m_{+1}m_{-1}\tilde{\mathbf{w}}^{*T}\mathbf{x} - \tilde{c}^*$, which simply computes a multiple of the original LM score function $f^*(\mathbf{x}; \mathbf{w}^*, c^*)$ with the same sign. The algorithm can be made confidential by encoding all real vector coefficients as integers (as described above). Then one encrypts the input vectors coefficient-wise and carries out the linear algebra operations with vectors of ciphertexts using HE.Add and HE.Mult. Note, that the server only returns the result of the score function for each test example, and that the client takes the sign to obtain the class label, because we assume that our HE scheme does not enable comparison.

Fisher's Linear Discriminant Classifier. A similar procedure is done for the approximate version of the FLD classifier using gradient descent. We use the same classifying function $\tilde{f}^*$ as for the LM classifier, but with a different weight vector $\tilde{\mathbf{w}}^*$. As above, to avoid divisions, we compute multiples of the class-conditional covariance matrices as $\tilde{\mathbf{C}}_{+1} = m_{+1}^3\mathbf{C}_{+1}$ and $\tilde{\mathbf{C}}_{-1} = m_{-1}^3\mathbf{C}_{-1}$. In general, we compute $\tilde{\mathbf{C}} = m_{+1}^3\tilde{\mathbf{C}}_{-1} + m_{-1}^3\tilde{\mathbf{C}}_{+1} = m_{+1}^3 m_{-1}^3\mathbf{C}$, but whenever we can use equal size training classes, i.e. $m_{+1} = m_{-1}$, we can reduce the coefficients by a factor m_{+1}^3.

The gradient descent iteration is done with fixed step size η. When $\eta < 1$, we also have to multiply through by its inverse to avoid divisions, which means we need to choose it such that $\eta^{-1} \in \mathbb{Z}$. Taking good care of all denominators that need to be multiplied by, we can deduce that the division free integer gradient descent computes the r-th weight vector $\tilde{\mathbf{w}}_r$, which is $\tilde{\mathbf{w}}_r = (m_{+1}^3 m_{-1}^3 \eta^{-1})^r \mathbf{w}_r$, where $\mathbf{w}_r$ is the result of the r-th iteration described in Section 3.1. In this way, the DFI version computes multiples of the exact same magnitudes as in the standard gradient descent approach described earlier, resulting in the score function being a multiple of the original score function.

3.3 Other Machine Learning Tasks and Generalization Properties

While we focus on binary classification in this paper, it is certainly possible to extend our methodology to other machine learning tasks including regression, dimensionality reduction, and clustering. In particular, the case of multivariate linear regression is quite similar to FLD in that the exact solution requires a matrix inverse, which can be approximated using gradient descent. Also, principal component analysis (PCA) [12], which is probably the most popular method for dimensionality reduction, can be expressed as a least squares problem the solution of which can be approximated by gradient descent. Clustering may well be the most difficult task in this context, but it would appear that spectral clustering solutions [17] could be approximated in a similar way.

Another interesting aspect of polynomial machine learning is its generalization properties. Although in Confidential ML algorithms the hypothesis class (e.g., linear classifiers) remains the same with respect to the exact algorithm, the restrictions imposed by D-polynomial HE require us to produce predictions which are polynomials of limited degree in the input data. As a consequence, the set of hypotheses that can be reached by a D-polynomial learning algorithm is very limited. One would expect that this limited capacity would have a positive effect on the generalization ability. While we do not have any formal results on this, we believe it may be possible to formalize this idea based on the stability bounds on the generalization error in [18], because the approximations required by SHE can be viewed as a specific form of "early stopping".

4 A Homomorphic Encryption Scheme

In this section, we describe a homomorphic public-key encryption scheme based on the Ring Learning With Errors (RLWE) problem [16]. It can be used to realize low degree confidential machine learning algorithms as described in Section 3. It extends the encryption scheme in [16] and resembles the LHE scheme from [2] in the RLWE case, as recently described in [6].

For simplicity and later reference in the description of our experiments, we discuss a special case of the scheme, for more details see [16,2,6]. Ciphertexts consist of polynomials in the ring $R = \mathbb{Z}[x]/(f(x))$, where $f(x) = x^d + 1$ and $d = 2^k$, i.e. integer polynomials of degree at most $d - 1$. Note that f is the $2d$-th cyclotomic polynomial. Computations in R are done by the usual polynomial addition and multiplication with results reduced modulo $f(x)$. We fix an integer modulus $q > 1$ and denote by R_q the set of polynomials in R with coefficients in $(-q/2, q/2]$. For $z \in \mathbb{Z}$ denote by $[z]_q$ the unique integer in $(-q/2, q/2]$ with $[z]_q \equiv z \pmod{q}$. The message space is the set R_t for another integer modulus $t > 1$ ($t < q$). We use the same notation with q replaced by t. Thus, messages to be encrypted under the SHE scheme are polynomials of degree at most $d - 1$ with integer coefficients in $(-t/2, t/2]$. Let $\Delta = \lfloor q/t \rfloor$ be the largest integer less than or equal to q/t. When applied to a polynomial $g \in R$, $\lfloor g \rfloor$ means rounding down coefficient-wise. We also use the notation $\lfloor \cdot \rceil$ for rounding to the nearest integer. As error distribution we take the discrete Gaussian distribution $\chi = D_{\mathbb{Z}^d, \sigma}$ with standard deviation σ over R. The parameters d, q, t and σ need to be chosen in a way to guarantee correctness, i.e. such that decryption works correctly, and security. Section 5 below gives such concrete parameters. Given the above setting (following notation in [6]), we now describe the SHE scheme with algorithms for key generation, encryption, addition, multiplication, and decryption.

SH.Keygen. The key generation algorithm samples $s \leftarrow \chi$ and sets the *secret key* $\mathsf{sk} := s$. It samples a uniformly random ring element $a_1 \leftarrow R_q$ and an error $e \leftarrow \chi$ and computes the *public key* $\mathsf{pk} := (a_0 = [-(a_1 s + e)]_q, a_1)$.

SH.Enc(pk, m). Given the public key $\mathsf{pk} = (a_0, a_1)$ and a message $m \in R_t$, encryption samples $u \leftarrow \chi$, and $f, g \leftarrow \chi$, and computes the ciphertext $\mathsf{ct} = (c_0, c_1) := ([a_0 \cdot u + g + \Delta \cdot m]_q, [a_1 \cdot u + f]_q)$.

Note that a homomorphic multiplication (as described below) increases the length of a ciphertext. Using relinearization techniques, it can be reduced to a two-element ciphertext again (see e.g. [14,6]). For the purpose of this paper, we do not consider relinearization, thus ciphertexts can have more than two elements and we describe decryption and homomorphic operations for general ciphertexts.

SH.Dec($\mathsf{sk}, \mathsf{ct} = (c_0, c_1, \dots, c_k)$). Decryption computes $[\lfloor t \cdot [c_0 + \mathsf{sk} \cdot c_1 + \dots + \mathsf{sk}^k \cdot c_k]_q / q \rceil]_t$.

In general, the homomorphic operations $\mathsf{SH.Add}$ and $\mathsf{SH.Mult}$ get as input two ciphertexts $\mathsf{ct} = (c_0, c_1, \dots, c_k)$ and $\mathsf{ct}' = (c'_0, c'_1, \dots, c'_l)$, where w.l.o.g. $k \geq l$. The output of $\mathsf{SH.Add}$ contains $k+1$ ring elements, whereas the output of $\mathsf{SH.Mult}$ contains $k + l + 1$ ring elements.

SH.Add($\mathsf{pk}, \mathsf{ct}_0, \mathsf{ct}_1$). Let $\mathsf{ct}_1 = (c_0, c_1, \dots, c_k)$ and $\mathsf{ct}_2 = (d_0, d_1, \dots, d_l)$. Homomorphic addition is done by component-wise addition $\mathsf{ct}_{\mathsf{add}} = (c_0 + d_0, c_1 + d_1, \dots, c_l + d_l, c_{l+1}, \dots, c_k)$.

SH.Mult($\mathsf{pk}, \mathsf{ct}_0, \mathsf{ct}_1$). Let $\mathsf{ct}_1 = (c_0, c_1, \dots, c_k)$, $\mathsf{ct}_2 = (d_0, d_1, \dots, d_l)$ and consider the polynomials $\mathsf{ct}_1(X) = c_0 + c_1 X + \dots + c_k X^k$ and $\mathsf{ct}_2(X) = d_0 + d_1 X + \dots + d_l X^l$ over R. The homomorphic multiplication algorithm computes the polynomial product

$$\mathsf{ct}_1(X) \cdot \mathsf{ct}_2(X) = e_0 + e_1 X + \dots + e_{k+l+1} X^{k+l+1} \tag{4}$$

in the polynomial ring $R[X]$ over R. The output ciphertext is $\mathsf{ct}_{\mathsf{mlt}} = (\lfloor t \cdot e_0 / q \rceil, \dots, \lfloor t \cdot e_{k+l+1} / q \rceil)$.

This scheme has been recently described and analysed in [6] and is closely related to the scheme in [4] and [14]. We refer to these papers for correctness and security under the RLWE assumption. However note that the evaluation of the ciphertext polynomial at the secret key (as computed during decryption) can be written as $[\mathsf{ct}(\mathsf{sk})]_q = [\Delta \cdot m + v]_q$, where v is a noise term that grows during homomorphic operations. Only if v is small enough, the ciphertext still decrypts correctly. How quickly v grows with each multiplication and addition determines the capabilities of the SHE scheme. An advantage of the present scheme is that the factor by which v grows is independent of the input ciphertext noise (see [2,6]).

Encoding Real Numbers. In order to do meaningful computations for ML, we would ideally like to do computations on real numbers, i.e. we need to encode real numbers as elements of R_t. Homomorphic operations under HE correspond to polynomial operations in R with coefficients modulo t. To reflect addition and multiplication of given numbers by the corresponding polynomial operations, we resort to the method in [14, Section 4] for encoding integers. We first represent a real number by an integer value. Since any real number can be approximated by rational numbers to arbitrary precision, we can fix a desired precision, multiply through by a fixed denominator, and round to the nearest integer.

An integer value z is encoded as an element $m_z \in R_t$ by using the bits in its binary representation as the coefficients of m_z. This means we use the following encoding function:

$$\text{encode} : \mathbb{Z} \to R_t, \; z = \text{sign}(z)(z_s, z_{s-1}, \ldots, z_1, z_0)_2 \mapsto m_z = \text{sign}(z)(z_0 + z_1 x + \ldots + z_s x^s).$$

To get back a number encoded in a polynomial, we evaluate it at $x = 2$. For the polynomial operations in R_t to reflect integer addition or multiplication, it is important that no reductions modulo t or modulo f occur. A multiplication after which a reduction modulo f is done does not correspond to integer multiplication of the encoded numbers any more. The same holds for reductions modulo t. The value t must therefore be large enough that all coefficients of polynomials representing values in the ML algorithm do not grow out of $(-t/2, t/2]$. Also the initial polynomial degree of encoded integers (i.e. their bit size) must be small enough so that the resulting polynomials after all multiplications still have degree less than d.

5 Proof of Concept and Experimental Results

In this section, we provide experimental results at a small scale to show how confidential machine learning works in principle. Due to the rather high computational cost of HE, we restrict ourselves to binary classification on a standard data set: the Wisconsin Breast Cancer data set with 569 records obtained from [8]. Data vectors in this set have 30 features and whenever we restrict the number of features in our experiments to some $n < 30$, we take the subset of the first n features.

With our experimental data we attempt to demonstrate the following claims: on small data sets, basic Machine Learning algorithms on encrypted data are practical. We give performance numbers for both Linear Means (LM) classifier and Fisher's Linear Discriminant (FLD) classifier, varying both the number of features and the number of vectors used in the training stage to estimate how performance and accuracy scales as these parameters vary. We compare timings for these two classifiers on encrypted and unencrypted data, to show the magnitude of the computational cost for operating on encrypted data. For these experiments, we *fix* the security parameters of the system, to model the real-world setting where a cloud system deploys an implementation based on parameters chosen to optimize for performance. In addition, we demonstrate the difference in accuracy when using the DFI version of the FLD algorithm using gradient descent instead of the exact linear algebra version that includes a matrix inversion (in the case of LM the DFI version is exact and does not require an approximation).

This section is organized as follows: Section 5.1 describes how the security parameters are chosen and how they scale with the operations to be performed. Section 5.2 gives timings for basic HE operations for two different choices of system parameters, (P1) and (P2). Section 5.3 gives performance numbers for the LM classifier on both unencrypted and encrypted data. On encrypted data, with fixed security parameters (P1), we vary the number of training vectors and the number of features. Section 5.4 gives performance numbers for the FLD classifier on both unencrypted and encrypted data. On encrypted data, with fixed security parameters (P2), we vary the number of training vectors and the number of test vectors. In Section 5.5, on unencrypted data, we compare the accuracy of the models computed with the exact and DFI versions of the FLD algorithm with varying number of steps in the gradient descent approximation.

5.1 Choice of Parameters

In this subsection, we discuss the specific parameters chosen for our implementation. It has been recently shown in [13] that the hardness of the RLWE problem is independent of the form of the modulus. This means that security is not compromised by choosing q with a special structure. Using a power of 2 for q dramatically speeds up modular reduction when compared to an implementation where q is prime. Therefore, as in [6] we choose both q and t to be powers of 2, i.e. $\varDelta = \lfloor q/t \rfloor = q/t$ is also a power of 2. We also use the optimization proposed in [6] to choose the secret key $\mathsf{sk} = s$ randomly with binary coefficients in $\{0, 1\}$.

To determine parameters that guarantee a certain level of security, one has to consider the best known algorithms to attack the scheme. Its security is assessed by the logarithm of the running time of such algorithms. A security level of ℓ bits means that the best known attacks take about 2^{ℓ} basic operations. We chose parameters considered secure under the distinguishing attack in [15], using the method described in [14, Section 5.1] and [6, Section 6]. For the exact details of the security evaluation, we refer to [15,14,6]. Security depends on the size of q, σ, and d, and for a given pair q, σ one can determine a lower bound for d.

Additional conditions follow from ensuring correctness of decryption. As long as the inherent noise in ciphertexts is bounded by $\varDelta/2 = q/2t$, decryption works correctly. Since homomorphic computations increase the noise level, this bounds the number of computations from above. In the division free integer algorithms the encrypted numbers tend to grow with the number of operations due to multiplications by denominators. To ensure meaningful results, t needs to be greater than all the coefficients of message polynomials that are held and operated on in encrypted form. The size of the standard deviation for the error terms and the desired number of homomorphic operations bound $\varDelta$ and therefore q from below. For our implementation, we determined these quantities experimentally and then chose the degree d according to the security requirements.

5.2 Timings for Basic HE Operations

We implemented the HE scheme described in Section 4 and the division-free integer ML algorithms under HE in the computer algebra package Magma [1], using internal functions for polynomial arithmetic and modular reductions. Table 1 summarizes timings for the HE operations. The confidential version of the DFI-LM classifier uses the first parameter set (P_1) to encode and encrypt data. Parameters (P_2) were chosen for encoding and encryption for the confidential version of the 3-step FLD method. Due to its higher complexity and the higher value for t it requires a much larger value for q.

Table 1. Timing in seconds for HE operations: key generation, encryption, decryption of 2- or 3-element ciphertexts, homomorphic addition and multiplication

	HE.Keygen	HE.Enc	HE.Dec(2)	HE.Dec(3)	HE.Add	HE.Mult
(P_1) $q = 2^{128}$, $t = 2^{15}$ $\sigma = 16$, $d = 4096$	0.279	0.659	0.055	0.105	0.001	0.208
(P_2) $q = 2^{340}$, $t = 2^{40}$ $\sigma = 8$, $d = 8192$	0.749	1.56	0.227	0.442	0.005	0.853

All timings in this and the remaining subsections and tables were obtained running Magma on an Intel Core i7 running 64-bit Windows 8 at 2.8 GHz with 8GB of memory. Timings are given in seconds (s). No communication costs are included in these experiments since the computations are all done on one machine. Parameters (P_1) have 128 bits of security with distinguishing advantage 2^{-64}. Security for (P_2) is around 80 bits due to small σ compared to q.

5.3 Linear Means Classifier

For the Linear Means Classifier, the exact and the DFI versions of the algorithm coincide, so there is no difference in the quality of the output. We compare in this section the timings for the encrypted and unencrypted DFI versions of the algorithm. The Linear Means Classifier experiments in this section were run with security parameters (P1),

$$q = 2^{128},\ t = 2^{15},\ \sigma = 16,\ f = X^{4096} + 1.$$

The data was preprocessed by shifting the mean to 0 and scaling by the standard deviation. Also, precision of computation was set at 2 digits, which means real numbers are multiplied by 100 and rounded to integers.

Timings on Unencrypted Data for DFI-LM. Each line in the tables reports the number of features used, the number of training vectors used in the training stage to build the classifier, the number of test vectors used to test the model, the time spent in the training stage, the time *per test vector* to classify, and the number of errors in the classification of test vectors.

Remarks. Note that the time for classifying vectors is relatively constant, which is as expected. The number of classsification errors varies, but tends to decrease as the size of the training set increases.

Table 2. DFI-LM Unencrypted Data

# features	# training	# test	train (s)	classify (s)	# errors
2	20	100	2.3500E-6	2.0620E-5	11
5	20	100	3.1500E-6	2.0620E-5	8
10	20	100	3.9000E-6	2.0940E-5	12
20	20	100	5.4500E-6	2.1250E-5	16
30	20	100	6.2500E-6	2.1560E-5	12
2	60	100	3.9000E-6	2.0620E-5	8
5	60	100	6.2500E-6	2.0940E-5	10
10	60	100	7.0500E-6	2.1410E-5	8
20	60	100	1.1700E-5	2.2030E-5	12
30	60	100	1.4850E-5	2.2970E-5	11
2	100	100	6.2500E-6	2.0780E-5	9
5	100	100	8.5999E-6	2.0940E-5	8
10	100	100	1.0900E-5	2.1410E-5	9
20	100	100	1.8000E-5	2.2030E-5	13
30	100	100	2.3450E-5	2.2650E-5	8

Timings on Encrypted Data for DFI-LM. In the tables reporting timings for operations on encrypted data, we also include the total time spent on encrypting and encoding the training vectors for the training stage, and the total time spent on encrypting and encoding the test vectors for the testing stage. The "ee-train" and "ee-test" columns in Table 3 are for the total time including the time to encode and encrypt the training and test vectors, respectively. For the encrypted and unencrypted versions of the the DFI algorithms, there is no need to list the number of classification errors twice, since the algorithm is the same and has the same output on encrypted and unencrypted data.

Remark 5.3

1. The time for classifying a test vector and for encoding and encrypting the test vectors stays relatively constant as the number of training vectors increases, as expected.
2. The time for computing the classifier in the training stage grows roughly linearly with the number of training vectors. This is expected as long as the security parameters are fixed, as is the case here.
3. The time for encoding and encrypting data in the training stage grows roughly linearly with the number of training vectors. Again, this is expected as long as the security parameters are fixed, as we have modeled here.
4. For a fixed training set size, the time for computing the classifier grows approximately linearly with the number of features. Similarly, the time for classifying a test vector, and the time for encoding and encrypting the training and the test vectors each grow approximately linearly with the number of features.
5. The approximate order of magnitude of the slow-down due to operating on encrypted data is 6 or 7 orders of magnitude. This compares favorably with

Table 3. DFI-LM Encrypted Data

# features	# training	# test	train (s)	ee-train (s)	classify (s)	ee-test (s)
2	20	100	0.095	19.953	0.327	133.843
5	20	100	0.156	50.172	0.899	343.250
10	20	100	0.391	101.141	1.938	708.969
20	20	100	0.831	201.578	3.880	1405.875
30	20	100	1.374	303.937	5.961	2122.719
2	60	100	0.127	59.641	0.325	133.703
5	60	100	0.484	148.125	0.879	337.953
10	60	100	0.996	309.078	1.864	688.531
20	60	100	2.504	601.688	3.841	1400.266
30	60	100	3.346	899.953	5.838	2106.453
2	100	100	0.565	98.938	0.417	143.719
5	100	100	0.835	249.359	0.998	351.078
10	100	100	2.629	499.063	1.971	699.531
20	100	100	4.034	999.156	3.989	1403.172
30	100	100	6.221	1504.297	6.038	2110.000

the slow-down for performing an AES encryption operation on encrypted data reported in [11].

6. Note that even with this preliminary unoptimized implementation, both the training stage and the classification of a test vector can be performed on encrypted data in roughly 6 seconds using a Linear Means Classifier on 100 training vectors with 30 attributes.

5.4 Fisher's Linear Discriminant Classifier

The experiments in this section were run with security parameters (P2),

$$q = 2^{340},\ t = 2^{40},\ \sigma = 8,\ f = X^{8192} + 1.$$

The data was preprocessed by shifting the mean to 0 and scaling by the standard deviation. Also, precision of computation was set at 2 digits, which means real numbers are multiplied by 100 and rounded to integers. The DFI version of the FLD algorithm was run using 3 steps in the gradient descent method with step size $\eta = 0.1$.

Timings on Unencrypted Data for 3-Step DFI-FLD. Each line in the table reports the number of features, the number of training vectors used in the training stage to build the classifier, the number of test vectors used to test the model, the time spent in the training stage, the time *per test vector* to classify, and the number of errors in the classification of test vectors.

Timings on Encrypted Data for 3-step DFI-FLD. Each line in the table reports the number of features, the number of training vectors used in the training stage to build the classifier, the number of test vectors used to test the

Table 4. 3-step DFI-FLD Unencrypted Data

# features	# training	# test	train (s)	classify (s)	# errors
2	20	100	2.6600E-4	1.7200E-6	11
5	20	100	7.5000E-4	1.8800E-6	7
10	20	100	2.4690E-3	2.1800E-6	9
20	20	100	9.2030E-3	2.9700E-6	6
30	20	100	0.020390	3.9100E-6	8
2	60	100	4.0700E-4	1.5700E-6	8
5	60	100	1.7030E-3	1.7200E-6	7
10	60	100	6.2190E-3	2.1800E-6	8
20	60	100	0.024125	2.9700E-6	9
30	60	100	0.054250	3.9000E-6	8
2	100	100	5.7800E-4	1.5600E-6	10
5	100	100	2.6410E-3	1.7200E-6	7
10	100	100	0.0100	2.1900E-6	8
20	100	100	0.039282	2.9700E-6	6
30	100	100	0.087594	3.9000E-6	3

model, the time spent in the training stage, the time per test vector to classify, the total time spent on encrypting and encoding the test vectors (in the "ee" column in Table 5), and the number of errors in the classification of test vectors.

Table 5. 3-step DFI-FLD Encrypted Data

# features	# training	# test	train (s)	classify (s)	ee (s)	errors
2	20	20	299.437	3.836	46.562	2
5	20	20	1309.578	10.049	117.033	1
10	20	20	4472.922	20.857	236.514	1
2	60	20	939.156	6.488	51.280	1
5	60	20	3612.953	9.707	117.158	1
10	60	20	12211.719	20.465	235.236	1
2	100	100	1420.781	3.850	619.828	10
5	100	100	6017.688	10.364	1636.265	8
10	100	100	20222.515	21.572	3351.718	35

Remark 5.4

1. The timings in the last line for 10 features and 100 training vectors should be disregarded, because the 35 classification errors indicate that the computation exceeded the allowable bounds for the amount of computation which could be correctly done with these security parameter sizes. This computation most likely resulted in decryption errors and should be redone with larger system parameters.
2. Note that for both the LM and the FLD algorithms, the time spent in computing on encrypted vectors is dominated by the time spent to encrypt

the data. That is due to the fact that each entry to be encrypted requires a multiplication of two polynomials of degree d, where d is either 4096 or 8192 in these experiments. Although Magma does have fast multiplication techniques implemented, this is an aspect of the system where performance can be significantly improved in a robust high-performance implementation.

3. The time spent in the training stage grows roughly quadratically with the number of features. This is expected because the algorithm operates on an $n \times n$ matrix, where n is the number of features.
4. The time spent in the training stage grows roughly linearly with the number of training vectors. Also the time spent on encoding and encryption grows linearly with the number of features.
5. Time spent on classifying test vectors is relatively constant as the number of training vectors increases, as expected.
6. The time spent on classifying test vectors grows roughly linearly with the number of features.
7. As for the confidential LM algorithms, we observe a slow-down of roughly $6-7$ orders of magnitude when executing the 3-step DFI-FLD algorithm on encrypted data under HE.
8. Despite the significant performance penalty for operating on encrypted data, note that classification of test vectors with 10 features is accomplished in 20 seconds with an unoptimized implementation of HE. This time is relatively independent of the number of training vectors used in the training stage for fixed system parameters. However, it is dependent on the amount and size of the data to be processed in the sense that the system parameter sizes must be increased once the bounds on the amount of computation which can be properly handled for a given parameter size are exceeded.

5.5 Comparing the Accuracy of Exact and DFI Versions of Gradient Descent

In Section 5.4 above, we gave performance numbers for the DFI version of the gradient descent method for Fisher's Linear Discriminant Classifier. The gradient descent method for minimizing the cost function is an approximation algorithm, whereas there is an exact algorithm for minimizing the cost function which requires matrix inversion. In this section, we compare the accuracy of the models obtained when using the exact version of FLD versus using the gradient descent approximation method with a varying number of steps. In Table 6 we give the number of classification errors for the exact version of FLD and the gradient descent method with $1-5$ steps. These experiments were performed on unencrypted data, with step size $\eta = 0.1$. Based on these results, three steps seemed to be sufficient in the gradient descent method for these data set sizes, so we used 3 steps in all experiments in Section 5.4 to produce encrypted and unencrypted timings for FLD.

Table 6. # errors in exact and DFI-FLD classification on unencrypted data

# features	# training	# test	exact	1-step	2-step	3-step	4-step	5-step
2	100	100	10	10	10	10	10	11
5	100	100	7	7	7	7	7	7
10	100	100	8	8	9	8	8	8
20	100	100	6	19	5	6	5	6
30	100	100	2	29	6	3	2	1

6 Conclusions and Future Work

With advances in machine learning and cloud computing the enormous value of data for commerce, society, and people's personal lives is becoming more and more evident. In order to realize this value it will be crucial to make data available for analysis while at the same time protect it from unwanted access. In this paper, we pointed out a way to reconcile these two conflicting goals: Confidential Machine Learning. We formalize the problem in terms of a multi-party data machine learning scenario involving a Data Owner, Data Providers, and a Cloud Service Provider and describe the desired functionality and security properties. We showed that it is possible to implement Confidential ML based on a recently proposed Homomorphic Encryption scheme, using polynomial approximations to known ML algorithms.

Homomorphic encryption is a rapidly advancing field and so we expect that more complex ML algorithms applied to larger data sets requiring fewer computational resources may soon be possible. For example, it should soon be possible to use kernel methods to derive low-degree polynomial machine learning algorithms implementing non-linear mappings. Other open problems include the question, which protocols will be useful in practical data analysis scenarios, and how the computational burden can be optimally distributed between cloud and client taking into account the cost of communication. Furthermore, one can imagine even more complex multi-party scenarios in which multiple data-owners (e.g., Amazon, Netflix, Google, Facebook) would like to provide inputs for a single machine learning problem (e.g., product recommendation) without disclosing their data.

References

1. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. J. Symbolic Comput. 24(3-4), 235–265 (1993); Computational algebra and number theory, London (1993)
2. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R. (ed.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Heidelberg (2012)
3. Brakerski, Z., Gentry, C., Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) Innovations in Theoretical Computer Science – ITCS 2012, pp. 309–325. ACM (2012)

4. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg (2011)
5. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, 2nd edn. John Wiley and Sons (2000)
6. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), `http://eprint.iacr.org/`
7. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Annual Eugenics 7(2), 179–188 (1936)
8. Frank, A., Asuncion, A.: UCI machine learning repository (2010), `http://archive.ics.uci.edu/ml`
9. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) STOC, pp. 169–178. ACM (2009)
10. Gentry, C., Halevi, S., Smart, N.P.: Better bootstrapping in fully homomorphic encryption. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 1–16. Springer, Heidelberg (2012)
11. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R. (ed.) CRYPTO 2012. LNCS, vol. 7417, pp. 850–867. Springer, Heidelberg (2012)
12. Jolliffe, I.T.: Principal Component Analysis. Springer, Heidelberg (1986)
13. Langlois, A., Stehlé, D.: Hardness of decision (R)LWE for any modulus. Cryptology ePrint Archive, Report 2012/091 (2012), `http://eprint.iacr.org/`
14. Lauter, K., Naehrig, M., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, pp. 113–124. ACM, New York (2011)
15. Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 319–339. Springer, Heidelberg (2011)
16. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010), `http://eprint.iacr.org/2012/230`
17. Ng, A.Y., Jordan, M.I., Weiss, Y.: On spectral clustering: Analysis and an algorithm. In: Advances in Neural Information Processing Systems 14, pp. 849–856. MIT Press (2002)
18. Poggio, T., Rifkin, R., Mukherjee, S., Niyogi, P.: General conditions for predictivity in learning theory. Nature 428, 419–422 (2004)
19. Rivest, R.L.: Cryptography and machine learning. In: Matsumoto, T., Imai, H., Rivest, R.L. (eds.) ASIACRYPT 1991. LNCS, vol. 739, pp. 427–439. Springer, Heidelberg (1993)
20. Smart, N.P., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 420–443. Springer, Heidelberg (2010)
21. Williams, O., McSherry, F.: Probabilistic inference and differential privacy. In: Lafferty, J., Williams, C.K.I., Shawe-Taylor, J., Zemel, R.S., Culotta, A. (eds.) Advances in Neural Information Processing Systems 23, pp. 2451–2459 (2010)

Another Look at Affine-Padding RSA Signatures

Jean-Sébastien Coron[1], David Naccache[2], and Mehdi Tibouchi[3]

[1] Université du Luxembourg
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg, Luxembourg
jean-sebastien.coron@uni.lu

[2] École normale supérieure, Département d'informatique
45, rue d'Ulm, F-75230, Paris Cedex 05, France
david.naccache@ens.fr

[3] NTT Secure Platform Laboratories – Okamoto Research Laboratory
3-9-11 Midori-cho, Musashino-shi, Tokyo, JP-180-8585, Japan
tibouchi.mehdi@lab.ntt.co.jp

Abstract. Affine-padding RSA signatures consist in signing $\omega \cdot m + \alpha$ instead of the message m for some fixed constants ω, α. A thread of publications progressively reduced the size of m for which affine signatures can be forged in polynomial time. The current bound is $\log m \sim \frac{N}{3}$ where N is the RSA modulus' bit-size. Improving this bound to $\frac{N}{4}$ has been an elusive open problem for the past decade.

In this invited talk we consider a slightly different problem: instead of minimizing m's size we try to minimize its *entropy*. We show that affine-padding signatures on $\frac{N}{4}$ entropy-bit messages can be forged in polynomial time. This problem has no direct cryptographic impact but allows to better understand how malleable the RSA function is. In addition, the techniques presented in this talk might constitute some progress towards a solution to the longstanding $\frac{N}{4}$ forgery open problem.

We also exhibit a sub-exponential time technique (faster than factoring) for creating affine modular relations between strings containing three messages of size $\frac{N}{4}$ and a fourth message of size $\frac{3N}{8}$.

Finally, we show than $\frac{N}{4}$-relations can be obtained in specific scenarios, *e.g.* when one can pad messages with two independent patterns or when the modulus' most significant bits can be chosen by the opponent.

1 Introduction

To prevent forgers from exploiting RSA's multiplicative homomorphism [6], it is a common practice not to sign raw messages m but to first apply to them a *padding function* $\mu(m)$.

This paper considers one of the simplest padding functions called affine padding (or fixed-pattern padding):

$$\sigma = \mu(m)^d \mod n = (\omega \cdot m + \alpha)^d \mod n$$

Here d denotes the RSA private exponent and n the public modulus. Throughout this paper $|x|$ will denote the bit-size of x. Let $N = |n|$.

Following [1], we use the following notations:

$$\mu(m) = \omega \cdot m + \alpha \quad \text{where} \begin{cases} w \text{ is the multiplicative redundancy} \\ \alpha \text{ is the additive redundancy} \end{cases} \tag{1}$$

Since no proof of security is known for RSA signatures using such a μ, those signatures should not be used in practice. Nonetheless, the study of such simple padding formats is useful for understanding how malleable the RSA function is.

In 1985, [2] exhibited forgeries for $\omega = 1$ when $m \sim \sqrt[3]{n^2}$. This attack was extended by [3] in 1997 to any values of ω, α and for $m \sim \sqrt{n}$. Finally, [1] exhibited in 2001 forgeries when $m \sim \sqrt[3]{n}$. This remains the best polynomial-time result to date. Relaxing the polynomial time constraint [4] showed that smaller message sizes can be tackled in complexity lower than that of all currently known factorization algorithms.

It was conjectured that a polynomial time forgery should exist for $m \sim \sqrt[4]{n}$, but this remained an elusive open problem for the past decade.

Our Contribution. This paper does not directly address the $m \sim \sqrt[4]{n}$ conjecture, but presents several new results in that *general direction*. Rather than minimizing $\log m$ we try to minimize m's *entropy*. Using a variant of [1], we show (§3) how to craft $\frac{N}{4}$ entropy signatures instead of $\frac{N}{3}$ ones. This has no specific cryptographic impact but allows to further explore RSA's malleability.

§4 shows how to obtain[1] a relation between four padded messages, three of which are of size $\frac{N}{4}$ and the fourth of size $\frac{3N}{8}$.

Finally, §5, investigates special scenarios in which we obtain $\frac{N}{4}$-relations by allowing the use of two independent padding patterns or by allowing the opponent to select the most significant bits of n.

2 Brier-Clavier-Coron-Naccache's Algorithm

In this section we briefly recall the attack of Brier *et alii* [1] using a slightly different exposition. [1] will serve as a building block in most of the results to come.

The goal is to find four distinct messages $x, y, z, t \in \mathbb{Z}$ each of size $\frac{N}{3}$, such that:

$$(\omega \cdot x + \alpha) \cdot (\omega \cdot y + \alpha) = (\omega \cdot z + \alpha) \cdot (\omega \cdot t + \alpha) \mod n \tag{2}$$

which enables to forge the signature of x using:

$$(\omega \cdot x + \alpha)^d = \frac{(\omega \cdot z + \alpha)^d \cdot (\omega \cdot t + \alpha)^d}{(\omega \cdot y + \alpha)^d} \mod n$$

Denoting $\Delta = \alpha/\omega \mod n$, from (2) it is sufficient to solve the following equation:

$$(\Delta + x)(\Delta + y) = (\Delta + z)(\Delta + t) \mod n$$

[1] in a time equivalent to that needed to factor a $\frac{3N}{8}$-bit number.

Theorem 1 (Brier *et alii*). *Given $n, \Delta \in \mathbb{Z}$ with $\Delta \neq 1$, the equation:*

$$(\Delta + x)(\Delta + y) = (\Delta + z)(\Delta + t) \mod n$$

has a solution $x, y, z, t \in \mathbb{Z}$ computable in polynomial time, with $0 \leq x, y, z, t \leq 8 \cdot \sqrt[3]{n}$ and with $y \neq z$ and $y \neq t$.

Proof. The previous equation gives:

$$\Delta \cdot (x + y - z - t) = z \cdot t - x \cdot y \mod n$$

By developing Δ/n as a continued fraction, we find $U, V \in \mathbb{Z}$ such that $\Delta \cdot U = V \mod n$ where $-\sqrt[3]{n} \leq U < \sqrt[3]{n}$ and $0 < V < 2 \cdot \sqrt[3]{n^2}$ and $\gcd(U, V) = 1$. Therefore it suffices to solve the following system:

$$\begin{cases} x \cdot y - z \cdot t &= V \\ x + y - z - t &= -U \end{cases} \tag{3}$$

A solution can be found using the following lemma:

Lemma 1. *Let $A, B, C \in \mathbb{Z}$ with $\gcd(B, C) = 1$. The system of equations:*

$$\begin{cases} x \cdot y - z \cdot t = A \\ \quad x - z \quad = B \\ \quad y - t \quad = C \end{cases}$$

has a solution given by

$$\begin{cases} t = y - C & = \frac{A - C \cdot (A \cdot C^{-1} \mod B)}{B} - C \\ x = B + z & = B + (A \cdot C^{-1} \mod B) \\ y = \frac{A - C \cdot z}{B} & = \frac{A - C \cdot (A \cdot C^{-1} \mod B)}{B} \\ z = A \cdot C^{-1} \mod B & \end{cases}$$

Proof. Letting $x = B + z$ and $t = y - C$, the first equation can be replaced by:

$$(B + z) \cdot y - z \cdot (y - C) = A$$

which gives:

$$B \cdot y + C \cdot z = A \tag{4}$$

Since $\gcd(B, C) = 1$ we get:

$$z = A \cdot C^{-1} \mod B$$

Moreover from equation (4) we obtain:

$$y = \frac{A - C \cdot z}{B} \tag{5}$$

which is an integer since $A - C \cdot z = 0 \mod B$. This concludes the lemma's proof. □

We return now to the proof of Theorem 1.

Let $A = V$ and choose B such that $\sqrt[3]{n} < B < 2 \cdot \sqrt[3]{n}$ and $\gcd(B, U) = 1$. Let $C = -U - B$ which gives $B + C = -U$; therefore from system (3) it suffices to solve the system:

$$\begin{cases} x \cdot y - z \cdot t = A \\ \quad x - z \quad\; = B \\ \quad y - t \quad\; = C \end{cases}$$

Since $\gcd(B, C) = \gcd(B, -U-B) = \gcd(B, U) = 1$, the previous system can be solved using Lemma 1. Moreover we have $0 \le z < B \le 2 \cdot \sqrt[3]{n}$ and $0 \le x \le 3 \cdot \sqrt[3]{n}$. From $C = -U - B$ we have:

$$1 \le -C \le 3 \cdot \sqrt[3]{n}$$

which gives $0 \le y \le 5 \cdot \sqrt[3]{n}$, and eventually $0 \le t \le 8 \cdot \sqrt[3]{n}$.

Finally since $C \neq 0$ we have $y \neq t$. Moreover if $y = z$ from equation (5) we get $A = (B + C) \cdot z = -U \cdot z = -V$ which gives $V = U \cdot z$;

- if $z \neq 1$ this gives $\gcd(U, V) \neq 1$, a contradiction;
- if $z = 1$ this gives $U = V$ and therefore $\Delta = 1$, a contradiction.

therefore $y \neq z$, which concludes the proof of the theorem. □

3 Minimal Entropy Forgeries

We now consider a slightly different equation. Let $\lambda = 2^{\lfloor \frac{N}{4} \rfloor}$ and consider the equation:

$$(\Delta + x) \cdot (\Delta + \lambda y) = (\Delta + z) \cdot (\Delta + \lambda t) \mod n \tag{6}$$

The following explains how to find in polynomial time four distinct solutions x, y, z, t of size $\sim \frac{N}{4}$. Therefore this gives a relation between four messages $m_1 = x$, $m_2 = \lambda y$, $m_3 = z$ and $m_4 = \lambda t$ of size $\sim \frac{N}{2}$ but an entropy of $\sim \frac{N}{4}$ bits only.

By expanding equation (6) we get:

$$\Delta\,(x - z + \lambda(y - t)) = \lambda(z \cdot t - x \cdot y) \mod n$$

As previously, by developing Δ'/n as a continued fraction with $\Delta' = \Delta/\lambda \mod n$, we can find U, V such that $\Delta \cdot U = \lambda V \mod n$ where this time we take $-\sqrt{n} \le U \le \sqrt{n}$ and $0 \le V \le 2\sqrt{n}$. Then it suffices to solve the system:

$$\begin{cases} x - z + \lambda(y - t) = -U \\ \quad x \cdot y - z \cdot t \quad = V \end{cases}$$

Using Euclidean division we write $U = H\lambda + L$ with $0 \le L < \lambda$; then it suffices to solve the system:

$$\begin{cases} x \cdot y - z \cdot t = V \\ \quad x - z \quad\; = -L \\ \quad y - t \quad\; = -H \end{cases}$$

which can be solved thanks to Lemma 1. Since $V \sim \sqrt{n}$ and the size of both L and H is roughly $\frac{N}{4}$, one obtains four solutions x, y, z and t of size $\frac{N}{4}$ in polynomial time.

However, for the lemma to apply, we must assume that $\gcd(L, H) = 1$; this makes the algorithm heuristic. If we assume that L and H are uniformly distributed, we have:

$$\Pr[\gcd(L, H) = 1] \cong \frac{6}{\pi^2}$$

We illustrate the process in Appendix B using RSA Laboratories' official 1024-bit challenge modulus RSA-309.

Note that one can improve the algorithm's success probability by considering:

$$(L', H') = (L + r\lambda, H - r)$$

instead of (L, H), for small values of $r \in \mathbb{Z}$. Assuming independent probabilities, after ℓ trials the failure probability drops to $(1 - 6/\pi^2)^\ell$, which is negligible even for small values of ℓ (and experiments suggest that in practice failure probability decreases even faster than this rough estimate).

The idea lends itself to many variants. For instance $\frac{5N}{4}$ entropy bit relations can be found by solving (mod n)

$$(\Delta + t\lambda + x)(\Delta + t\lambda + y) = (\Delta + t\lambda + z)(\Delta + t\lambda + w) \Rightarrow \Delta + t\lambda = \frac{xy - wz}{w - x - y + z}$$

Letting $\Delta = \frac{A}{B} \bmod n$ with $|A| = \frac{3N}{4}$ and $|B| = \frac{N}{4}$ we get $\Delta + t\lambda = \frac{A + t\lambda \cdot B}{B}$ and fixing $t = -\lfloor \frac{A}{\lambda B} \rfloor \Rightarrow \Delta + t\lambda = \frac{C}{B}$ for some $|C| = \frac{N}{2}$.

We can hence solve this equation using Lemma 1 by identifying:

$$\begin{cases} C = xy - wz \\ B = w - x - y + z \end{cases}$$

3.1 Message Entropy

We now define more precisely message entropy in the context of affine-padding RSA forgery.

Let Δ be a fixed pattern[2] and n a random variable denoting the RSA modulus. We denote by GenKey the RSA key generation algorithm. Let $\mathcal{F}$ be a forging algorithm making q signature queries for messages $\mathfrak{M} = \{m_1, \ldots, m_q\}$ and producing a forgery m_{q+1}. We regard $\mathfrak{M}$ and m_{q+1} as random variables induced by n (and possibly by the random tape of $\mathcal{F}$). We consider the entropy of individual messages separately and take the maximum entropy over all messages:

[2] *i.e.* the integer Δ appearing in the padding function $\mu(m) = (\Delta + m)$.

$$H_{\Delta} := \max\{H(m_i) \mid (\mathfrak{M}, m_{q+1}) \leftarrow \mathcal{F}(n, e, \Delta), (n, e) \leftarrow \mathsf{GenKey}(1^k)\}$$

We define the forgery's entropy as the maximum over all possible values of the pattern Δ:

$$H := \max\{H_{\Delta} \mid \Delta \in \mathbb{Z}\}$$

We see that in the described algorithm, the message entropy is roughly $\frac{N}{4}$, whereas it was $\frac{N}{3}$ in [1].

Note that in the previous definition we consider the maximum entropy of *all* messages required for the forgery and not only the entropy of the message whose signature is forged. For example [3] is selective, which means that the attacker can forge a signature for a message of his choosing; therefore the forged message can have zero entropy; however the remaining messages in [3] are half the size of n and their entropy is roughly $\frac{N}{2}$.

4 Sub-exponential Strategies

We start by noting that for $\frac{N}{4}$ forgeries of the form

$$(\Delta + x)(\Delta + y) = (\Delta + z)(\Delta + w) \bmod n \quad \Rightarrow \quad \Delta = \frac{z + w - x - y}{xy - wz} \bmod n$$

This means that Δ can be written as a modular ratio of two integers (namely $z + w - x - y$ and $xy - wz$) that are respectively $\frac{N}{4}$ and $\frac{N}{2}$ bits long. As this is expected to occur with probability $\sim 2^{-N/4}$ we infer that for arbitrary Δ values, such forgeries shouldn't exist in general.

Consider a forgery of the form:

$$(\Delta + x)(\Delta + y) = \Delta(\Delta + x + y + z) \bmod n$$

Hence, if x, y, z exist, they are such that:

$$\Delta = \frac{xy}{z} \bmod n$$

Write:

$$\Delta = \frac{A}{B} \bmod n \quad \text{where } |A| = \frac{5N}{8} \text{ and } |B| = \frac{3N}{8}$$

Then

$$\forall u,\ \Delta + u = \frac{A + uB}{B} \bmod n$$

Thus, if we fix:

$$u' = -\lfloor \frac{A}{B} \rfloor$$

we get

$$\varDelta + u' = \frac{A + u'B}{B} = \frac{A'}{B} \bmod n$$

where $|A'| = |B| = \frac{3N}{8}$. We can attempt to factor $A' = x \times y$ into two factors smaller than $\frac{N}{4}$ bits each. If this factorization fails add one to u' and start over again.

The result is a $\frac{N}{4}, \frac{N}{4}, \frac{N}{4}, \frac{3N}{8}$ forgery such as the one given in Appendix C.

Again, the idea lends itself to a number of variants. For instance, relations of the form

$$(\varDelta + tx)(\varDelta + ty) = (\varDelta + tz)(\varDelta + tw) \Rightarrow \varDelta = \frac{t(xy - wz)}{w - x - y + z}$$

can be found by writing $\varDelta = \frac{A}{B} \bmod n$ with $|A| = |B| = \frac{N}{2}$, factoring A to find t and continuing with Lemma 1.

5 Further Research

While computing $\frac{N}{4}$ forgeries remains an open problem, neighboring problems may lead to surprising algorithms. We give here two such variants as departure points for future research.

5.1 The Case of Two Interchangeable Padding Patterns

Let $\varDelta$ and $\varDelta'$ be two independently generated padding patterns and assume that the signer can sign messages using either $\varDelta$ or $\varDelta'$. We have:

$$\begin{gathered}(\varDelta + x)(\varDelta' + y) = (\varDelta + z)(\varDelta' + t) \bmod n \\ \Downarrow \\ \varDelta(y - t) + \varDelta'(x - z) + xy - zt = 0 \bmod n\end{gathered}$$

Find A, B, C of respective sizes $\frac{N}{2}, \frac{N}{4}, \frac{N}{4}$ such that $\varDelta C + \varDelta' B + A = 0 \bmod n$ and solve the system

$$\begin{cases} x \cdot y - z \cdot t = A \\ \quad x - z \quad = B \\ \quad y - t \quad = C \end{cases}$$

as before. Note that this yields an $\frac{N}{4}$ forgery only if $\varDelta$ and $\varDelta'$ are "independent". If $\varDelta = \varDelta' + \alpha$ for a small α then a $\frac{N}{3}$ forgery is found.

5.2 Allowing the Attacker To Influence n

Assume that the attacker can select the most significant half of n (*e.g.* [5] and [7] report that such a practice does not seem to weaken n). Let $\varDelta$ be an arbitrary padding pattern and:

$$(\varDelta + x)(\varDelta + y) = (\varDelta + x')(\varDelta + y') \bmod n$$
$$\Downarrow$$
$$\varDelta(x + y - x' - y') + xy - x'y' = 0 \bmod n$$

where x, y, x', y' are all of size $\frac{N}{4}$. This is solved by writing:

$$\begin{cases} \varDelta \cdot a + b & = 0 \bmod n \\ x + y - x' - y' & = a \\ xy - x'y' & = b \end{cases}$$

Hence, for a given $\varDelta$ we need to find an n for which a and b are of respective sizes $\frac{N}{4}$ and $\frac{N}{2}$. We then find x, y, x', y' exactly as previously but of size $\frac{N}{4}$ (to do so define $x - x' = \alpha$ for an arbitrary α and solve the two equations). Write $\varDelta a + b = kn$ as we can select the most significant bits of n, let $n = n_1 + n_0$ where n_1 (of size 1) is chosen by the attacker and where n_0 (of size $\frac{N}{2}$) is not under the attacker's control.

This boils-down to $\varDelta a + b = k(n_1 + n_0)$. Selecting $n_1 = 2\varDelta$ the attacker gets $\varDelta a + b = k(2\varDelta + n_0)$. Hence $k = 1$, $a = 2$ and $b = n_0$ is a satisfactory choice for which a and b are of respective sizes $\frac{N}{4}$ and $\frac{N}{2}$ (as a matter of fact a is much smaller but this is not an issue).

The attack's Sage code is given in Appendix A.

References

1. Brier, E., Clavier, C., Coron, J.S., Naccache, D.: Cryptanalysis of RSA signatures with fixed-pattern padding. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 433–439. Springer, Heidelberg (2001)
2. De Jonge, W., Chaum, D.: Attacks on some RSA signatures. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 18–27. Springer, Heidelberg (1986)
3. Girault, M., Misarsky, J.-F.: Selective forgery of RSA signatures using redundancy. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 495–507. Springer, Heidelberg (1997)
4. Joux, A., Naccache, D., Thomé, E.: When e-th Roots Become Easier Than Factoring. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 13–28. Springer, Heidelberg (2007)
5. Lenstra, A.K.: Generating RSA moduli with a predetermined portion. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 1–10. Springer, Heidelberg (1998)
6. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public key cryptosystems. CACM 21(2), 120–126 (1978)
7. Shamir, A.: RSA for paranoids. CryptoBytes (The Technical Newsletter of RSA Laboratories) 1(3) (1995)
8. `http://sites.google.com/site/bbuhrow/home`

A Allowing The Attacker to Influence n (Sage code)

```
def hex(x):
  s=x.digits(base=16,digits='0123456789abcdef')
  s.reverse()
  return "".join(s)

def invmod(a,b):
  g,c,d=xgcd(a,b)
  return c

def testattack(n=512):
  P=ZZ.random_element(2^n)
  N1=2*P
  q=random_prime(2^(n//2))
  p=N1//q
  NN=p*q
  print "N=",NN
  print "N/2=",NN//2
  print "P=",P

  a=2
  b=NN-N1

  al=ZZ.random_element(2^(n//4))
  while gcd(al,a)!=1:
    al=ZZ.random_element(2^(n//4))

  xp=ZZ(mod(b*invmod(a-al,al),al))
  y=(b-xp*(a-al))/al
  x=xp+al
  yp=y-(a-al)

  print "x=",x
  print "y=",y
  print "x\'=",xp
  print "y\'=",yp

  print "(delta+x)(delta+y)=(delta+x\')(delta+y\') mod N ",mod((P+x)*(P+y)-(P+xp)*(P+yp),NN)==0
```

B Minimum Entropy Forgery

$\mu(m_1) \cdot \mu(m_2) = \mu(m_3) \cdot \mu(m_4) \mod n_{309}$ with $\omega = 1$ and $\alpha = \Delta = 2^{1023} - 2^{516}$.

n_{309} is RSA Laboratories' unfactored challenge modulus RSA-309.

The entropy of messages m_1, m_2, m_3 and m_4 is $\cong \frac{|n_{309}|}{4}$.

n_{309} = RSA-309

```
= bdd14965 645e9e42 e7f658c6 fc3e4c73 c69dc246 451c714e b182305b 0fd6ed47
  d84bc9a6 10172fb5 6dae2f89 fa40e7c9 521ec3f9 7ea12ff7 c3248181 ceba33b5
  5212378b 579ae662 7bcc0821 30955234 e5b26a3e 425bc125 4326173d 5f4e25a6
  d2e172fe 62d81ced 2c9f362b 982f3065 0881ce46 b7d52f14 885eecf9 03076ca5
```

$\mu(m_1)$ =

```
7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffff0
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001
0a22096c f25655f4 104b2971 bc8b4f4f 817e6f4c d0ca8b25 ac5e8377 819e9d23
```

$\mu(m_2)$ =

```
7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffff0
e1bdb579 4ad9e45a 7b17ee62 bf736d1c 8d897862 ce2c3349 72600b8b 44a8d4fb
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

$\mu(m_3)$ =

```
7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffff0
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
113271f6 527c0815 fbf55d24 4883207b 827b9fb9 dd3ba8a6 2af1b776 d550a12d
```

$\mu(m_4)$ =

```
7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffff1
0c9ca82a 80d6e82a e84d12a7 6415e27e d6d909da c2331285 aca27f4e 632d1556
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

C Fast Sub-Exponential Forgery

$\mu(m'_1) \cdot \mu(m'_2) = \mu(m'_3) \cdot \mu(m'_4) \mod n_{309}$. Factorization for obtaining this relation was done with YAFU [8] using fast MPQS and SIQS implementations for Core2 processors.

$\Delta' =$

```
7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000008 ← note the 8
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

$\mu(m'_1) =$

```
7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000008
78dfd16f afa9c95b 2fecb797 21eae4a7 5217f260 0a9b852a 01dee0cf 315aea20
```

$\mu(m'_2) =$

```
7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000008
78dfd16f afaa4c53 011c40cf ce5ff1d9 f9d2f822 3bf3b3ad c770bdd4 4644e869
```

$\mu(m'_3) =$

```
7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000008
78dfd16f afa9c95b 2fefcd95 a1f55dd7 1f55b73a b29e0570 f72a86d2 940f34d1
```

$\mu(m'_4) =$

```
7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
00000000 00000000 00000000 00000000 98fb10e4 ef8f2456 b2ab14a2 236dadea
6b28b31b 1bfb2493 8f74d85e eceb7450 2d848353 fe9f1dec b40f041b bbb2a1f8
```

On Bruteforce-Like Cryptanalysis: New Meet-in-the-Middle Attacks in Symmetric Cryptanalysis

Christian Rechberger

Technical University of Denmark, Denmark

Abstract. This extended abstract briefly summarizes a talk with the same title and gives literature pointers. In particular, we coin the term bruteforce-like cryptanalysis.

Keywords: block ciphers, hash functions, bicliques, meet-in-the-middle, key recovery, preimages, cryptanalysis.

1 Overview

The basic idea of meet-in-the-middle attacks is to split an invertible transformation into two parts and separate parameters, or chunks, that are involved in only one part. Then these chunks can be searched independently with a match in the middle as a filter indicating a right combination. One of the first applications is the cryptanalysis of DoubleDES $E_{K_2}(E_{K_1}(\cdot))$, which demonstrated that the total security level is not the sum of key lengths [4]. The reason is that given a plaintext/ciphertext pair, an adversary is able to compute the internal middle state of a cipher trying all possible values of K_1 and K_2 independently. The same principle applies at the round level as well. If there is a sequence of rounds in a block cipher that do not depend on a particular key bit, the meet-in-the-middle attack might work.

A basic meet-in-the-middle attack requires only the information-theoretical minimum of plaintext-ciphertext pairs. The limited use of these attacks can be attributed to the requirement for large parts of the cipher to be independent of particular key bits. We also mention that a number of variations of the basic meet-in-the-middle attack theme were used in the literature, including combinations with slide and integral attacks.

Though there has been a great deal of meet-in-the-middle attacks on block ciphers recently, overall they received less attention from cryptanalysts than other standard attack vectors. In fact, it seems this attack vector was overshadowed by the success of statistical attacks like linear and differential attacks.

Recently a number of conceptual improvements to this attack vector have been proposed. A concept called *bicliques* was first introduced for hash cryptanalysis, by Savelieva et al. [8]. It originates from the splice-and-cut framework [1] in hash function cryptanalysis, and more specifically its aspect called initial structure [5,13]. The biclique approach led to the best preimage attacks on the SHA family of hash functions so far, when measured in terms of numbers of rounds, including the attack on 50 rounds of SHA-512, and the first attack on the round-reduced SHA-3 finalist Skein [8]. The concept of bicliques for block ciphers and biclique cryptanalysis for block ciphers was introduced in [2], and a predecessor, the splice-and-cut framework applied to block ciphers is found in [14]. In the following we briefly sketch some of the concepts and ideas in this context.

The Concept of a Biclique. A biclique (a complete bipartite graph) connects 2^d pairs of intermediate states with 2^{2d} keys. This is the main source of computational advantage in the key recovery — by constructing a biclique on 2^d vertices only, one covers quadratically many keys 2^{2d}. d is called the dimension of the biclique.

A biclique is characterized by its length (number of rounds covered) and dimension d. The dimension is related to the cardinality of the biclique elements and is one of the factors that determines the advantage over brute force. We now briefly describe four cryptanalytic techniques that appear in all known applications of the biclique concept.

(1) Bicliques from Independent Related-Key Differentials. Often the easiest way to construct a biclique in a cipher is to consider two related-key differentials holding with probability one – one with forward key modification and one with backward key modification. If those differentials are truncated, this can result in a higher dimensional biclique. A key requirement here is that the characteristics describing the biclique are all independent.

(2) Narrow-Bicliques. A variant of the above mentioned bicliques from independent related-key differentials [7]. The conceptual addition is that instead of probability 1 (truncated) differentials, much smaller probabilities are allowed. the degrees of freedom in the choice of the internal states for each biclique are used to efficiently enumerate enough bicliques of a special property that limit diffusion, and hence the data complexity of the resulting attacks.

(3) Bicliques from Interleaving Related-Key Differentials. This is a more involved approach, and is also based on related-key differentials. However, they can interleave (that is, intersect in active nonlinear components such as S-boxes). The propagation in those differentials can also be of probabilistic nature. This removes the constraint on the biclique length natural for bicliques from independent related-key differentials. This however also makes is very difficult to construct bicliques of higher dimension though. For the only known examples for biclique attacks of this type, $d = 1$ in key recoveries on round-reduced AES in [2]. The construction of such bicliques follows the rebound strategy [10] borrowed from the domain of hash function cryptanalysis.

2 Bruteforce-Like Cryptanalysis

Most published applications of bicliques use them as a means to improve the bruteforce-like cryptanalysis of a certain cipher or hash function, i.e. they allow to include, but also minimize partial brute-force computations [1]. This is based on another technique that was introduced together with the biclique idea:

(4) Precomputation Technique for the Matching Part of a Meet-in-the-Middle Attack. The starting observation is that, in case no matching is possible to merge the independent computations of the two chunks in a meet-in-the-middle attack because too many rounds are covered, a computation looping over all combinations can still give the information required to filter key candidates. Note that this is equivalent to test *all* keys for a particular part of the cipher or hash function. [2] shows that its combination with biclique cryptanalysis allows for larger savings of computations.

We note here that partial brute-force computations have been considered before for cryptanalytically improved preimage search methods for hash functions, e.g. in [1,12]. Even earlier examples that may be considered more implementation- rather than cryptanalytic-centric are [9,11]. In order to distinguish between these earlier implementation-centric optimizations of brute force with those that use more advanced cryptanalytic ideas, we coin the term *bruteforce-like cryptanalysis.*

Bruteforce-like cryptanalysis is not able to conclude that a particular target has a cryptanalytic weakness, as in principle any number of rounds can be "attacked". However it can help to better understand the real security offered against attacks in the absence of other shortcuts. Most recently reported applications of bruteforce-like biclique cryptanalysis have an advantage that is much smaller than a factor of 2. For ciphers like AES with key sizes of 128 bits or more this is merely of academic interest, we argue however that for ciphers with key sizes of 80 bits or less, this is very useful to know, especially when cost savings compared to optimized bruteforce implementations are a factor 2 or more [6].

3 Conclusions

Summarizing, the novel biclique meet-in-the-middle cryptanalysis is a promising cryptanalytic technique for the security evaluation of modern block ciphers and hash functions. It's combination with bruteforce-like cryptanalysis can, by its very nature, not be used to argue that a particular design has a cryptanalytic weakness. However, it seems advisable to include an assessment with respect to it into any new design of a symmetric primitives, as e.g. done for the recently proposed PRINCE [3].

References

1. Aoki, K., Sasaki, Y.: Preimage attacks on one-block MD4, 63-step MD5 and more. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 103–119. Springer, Heidelberg (2009)

[1] Exceptions to this are e.g. the key recoveries on 8-round AES-182 in [2], or on 5 and 7.5 round IDEA [7].

2. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique Cryptanalysis of the Full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 344–371. Springer, Heidelberg (2011)
3. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçın, T.: PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012)
4. Diffie, W., Hellman, M.: Special feature exhaustive cryptanalysis of the NBS Data Encryption Standard. Computer 10, 74–84 (1977)
5. Guo, J., Ling, S., Rechberger, C., Wang, H.: Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 56–75. Springer, Heidelberg (2010)
6. Jia, K., Rechberger, C., Wang, X.: Green Cryptanalysis: Meet-in-the-Middle Key-Recovery for the Full KASUMI Cipher. Cryptology ePrint Archive, Report 2011/466 (2011), `http://eprint.iacr.org/`
7. Khovratovich, D., Leurent, G., Rechberger, C.: Narrow-Bicliques: Cryptanalysis of Full IDEA. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 392–410. Springer, Heidelberg (2012)
8. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 244–263. Springer, Heidelberg (2012)
9. Kumar, S., Paar, C., Pelzl, J., Pfeiffer, G., Schimmler, M.: Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 101–118. Springer, Heidelberg (2006)
10. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 260–276. Springer, Heidelberg (2009)
11. Osvig, D.A.: Efficient Implementation of the Data Encryption Standard. Master thesis (2003)
12. Rechberger, C.: Preimage Search for a Class of Block Cipher based Hash Functions with Less Computation (2008) (unpublished manuscript)
13. Sasaki, Y., Aoki, K.: Finding preimages in full MD5 faster than exhaustive search. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 134–152. Springer, Heidelberg (2009)
14. Wei, L., Rechberger, C., Guo, J., Wu, H., Wang, H., Ling, S.: Improved Meet-in-the-Middle Cryptanalysis of KTANTAN (Poster). In: Parampalli, U., Hawkes, P. (eds.) ACISP 2011. LNCS, vol. 6812, pp. 433–438. Springer, Heidelberg (2011)

Balanced Indexing Method for Efficient Intrusion Detection Systems

BooJoong Kang[1], Hye Seon Kim[1], Ji Su Yang[1], and Eul Gyu Im[2]

[1] Department of Electronics and Computer Engineering,
Hanyang University, Seoul, 133-791, Korea
[2] Division of Computer Science and Engineering,
Hanyang University, Seoul, 133-791, Korea
{deviri,danzun,yjisu,imeg}@hanyang.ac.kr

Abstract. To protect a network from malicious activities, intrusion detection systems can be used. Most of intrusion detection systems examine incoming packets with detection signatures to detect potential malicious packets. Because the portion of malicious packets is usually very small, it is not efficient to examine incoming packets with all signatures. In this paper, we propose a method that reduces the number of signatures to be examined and show the experimental results of our proposed method.

Keywords: Network Security, Pattern Matching, Intrusion Detection System, Indexing.

1 Introduction

Uses of the Internet have increased tremendously in various applications and so has the volume of network traffic. Although most of network traffic is generated for benign purposes such as web browsing, video streaming and peer-to-peer file sharing, some portions of the network traffic are malicious, and malicious traffic might cause degradation of network performance or network based services. Examples of malicious network activities include phishing, sending spam emails, spreading malware and launching distributed denial of service (DDoS) attacks. With the increased volume of general network traffic, intrusion detection systems based on network traffic analysis need to have very high performance to examine a high volume of bypassing packets; moreover, the attacks are getting more sophisticated over time, making the detection of attack traffics on the Internet difficult. It is very import to detect attacks and block the attack traffic as fast as possible to maintain a certain level of network quality of service.

To detect and classify network-based attacks, network packets are examined to determine whether the packets contain malicious contents or not. To inspect network packets, there are two main approaches; packet header inspection and packet payload inspection. Packet header inspection allows fast detection by collecting the information in the packet header such as port numbers and IP addresses. On the other hand, packet payload inspection conducts deeper analysis on the contents of each packet and looks for specific keywords that are related to malicious activities. To maximize the effectiveness, the detection should be accurate as well as fast. Packet header inspection is fairly easy to implement and is able to provide fast detection if the information is used appropriately; however, this method is not suitable for accurate detection schemes because of high false positive rates. Packet payload inspection, on the other hand, does provide highly accurate detection but there are issues to be solved such as storage overheads, computational complexity and delays caused by the payload analysis.

In this paper we propose a method to enhance the performance of network packet inspection. Our proposed method groups a number of signatures together by indexing the signatures in a simple way. The experimental results of our proposed method show the efficiency of signature grouping.

The rest of the paper is organized as follows: Section 2 contains related work. In Section 3, we explain our indexing method for signature grouping. Experimental results with our proposed method are shown in Section 4 and Section 5 summarizes the paper.

2 Related Work

There have been many researches that tried to improve the performance of intrusion detection systems, including string matching algorithms for memory-efficiency, finite automata-based approaches, and signature indexing. In this section, we summarized and explained these researches and their approaches.

2.1 Aho-Corasick Based Approaches

The Aho-Corasick algorithm is one of the most common algorithms for string matching, and the algorithm forms graphs for signatures to be examined. For the 8-bit characters sets, the algorithm requires 256 pointers to keep the information of next states for all possible characters, which eventually causes waste of memory. A memory-efficient implementation of the Aho-Corasick algorithm was presented by N. Tuck et al. [1], to overcome this problem. They used a 256-bit bitmap instead of 256 pointers, and if a state can be a next state, the corresponding bit inside the bitmap is set to one. For the nodes that are close to terminal nodes of the graph, flags are used instead of bitmaps since most of the bits in the bitmap will be set to zero due to very few possible transitions. Their experimental results show that the method reduced the memory utilization down to just 2%, compared to the original approach.

Another approach that aims to improve the performance of the Aho-Corasick algorithm is presented by L. Tan T Sherwood [2]. Their approach is based on a specialized hardware architecture consisted of multiple Rule Modules and Tiles. Their hardware architecture enables Aho-Corasick matching in bit vectors, not in strings. The bit-level matching has only two possible next states, resulting in significant performance improvement.

Many efforts to improve the performance of the Aho-Corasick algorithm, including the above two approaches, are successful to enhance the performance of intrusion detection systems because string matching is one of the expensive processes in the deep packet inspection. However, the number of signature keeps increasing and it is hard to believe that the improvement of string matching algorithm will clear up the signature growth.

2.2 DFA or NFA Based Approaches

Deterministic Finite Automaton (DFA) and Nondeterministic Finite Automaton (NFA) are finite state machine-based models that consist of a number of states and transitions. Since both DFA and NFA can be represented with regular expressions, they are widely used for string matching. Although DFA requires more states than NFA, DFA shows better performance than NFA; therefore making DFA memory-efficient and improving the performance of NFA are the main issues of these two schemes. Memory-efficient DFAs can be achieved in various ways, for example, by introducing cache memory to DFA structure [3], by rewriting the regular expressions to reduce the number of states [4], and by using perfect hash function for faster transition table look-up [5]. While DFA implementation is more suitable in software, NFA implementation shows better performance in hardware environment. J. Bispo et al. [6] suggested a regular expression matching engine with reconfigurable hardware. However, these approaches are also hard to solve the same problem as the string matching approaches are.

2.3 Signature Grouping

Baker et al. proposed a method to avoid redundant comparison of the same characters by partitioning the whole patterns into several smaller groups [7]. The partitioning is done by: (1) maximizing the number of repeated characters within a group, (2) minimizing the number of characters repeated between different groups. Sourdis et al. [8] introduced a packet pre-filtering approach which used an 8-character prefix for each signature. In the pre-filtering step, a subset of signatures is selected by the partial matches, based on the 8-character prefix. If there are matched prefixes during the pre-filtering step, the full-match engine is activated to inspect the candidate set of signatures. In this paper, the authors claimed that the packet pre-filtering significantly reduces the number of signatures matched per packet. Chen et al. proposed a method to decompose the Snort signature patterns into primary patterns [9]. In the Snort pattern set, there are two properties: (1) the repetition which is just the same amongst rules, (2) the composition that each signature is considered as a combination of smaller pattern fragments. The authors decomposed this pattern with the delimiters, such as hyphen (‘-’) and period (‘.’). To distinguish text characters and binary data, any binary data with a ‘|’ in-between, is considered as an individual pattern. Although these signature grouping methods would solve the signature growth problem, the criterion of signature grouping is still questionable since the result of signature grouping is controlled by the criterion.

Among these various approaches to improve intrusion detection systems, there is no silver bullet that can solve the signature growth problem. To improve the performance of intrusion detection systems, various techniques should be explored and multiple techniques should be combined together. Therefore, we argue that our proposed indexing method can contribute to improve the performance of intrusion detection systems.

3 Our Proposed Method

Chen et al. presented the relationship between the number of primary patterns and the number of Snort rules [9]. A primary pattern is a set of strings that appear repeatedly within Snort rule signatures. The result of their experiments shows that the number of primary patterns saturates at some points even though the number of signatures keeps increasing. This implies that classifying signatures with grouping methods can increase the efficiency and speed of packet inspection greatly because the size of primary patterns can be much smaller than that of entire signatures.

We focused on this observation and grouped signatures by applying our indexing method. Indexing is a method which is commonly used to find a certain item from a large set of data. In our proposed method, an index is a substring which is part of a signature. Each index points to a subset of signatures contain the index; therefore the signatures are arranged into several groups according to indices. Before the packet inspection stage, the signatures are pre-processed to create indices and the signatures are grouped together according to the indices.

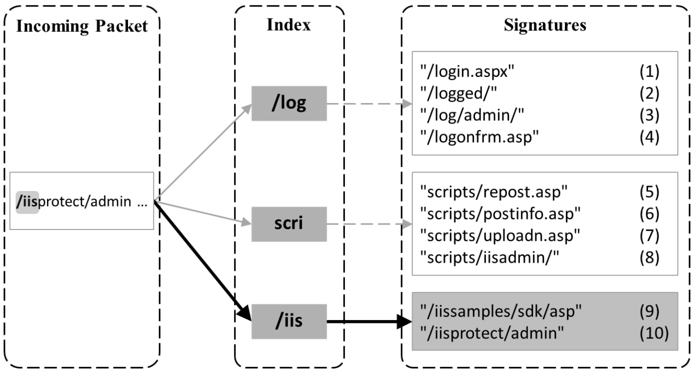


Fig. 1. Indexing

During the packet inspection, if an input packet has turned out to have some indices within its payload, it is highly likely that the packet is related to some types of attacks. In this case, further inspection should be carried on by examining the packet with the signatures that the indices point. On the other hand, if the payload of the packet does not include any indices, the packet is harmless; therefore, the inspection goes on for the next packet. Fig. 1 shows an example of the inspection with the indexing method, with three indices extracted from signatures: '/log', 'scri' and '/iis'.

As the index '/iis' appears in the payload of the incoming packet, only the signatures (9) and (10) will be used for the further inspection. The signatures from (1) to (8) would not be examined for this packet, since the indices '/log' and 'scri' are not matched. In this way, the inspection overhead can be significantly reduced by inspecting only two signatures.

One of the main issues on indexing is how to choose substrings from signatures when building indices. We will address our index selection algorithm, called Balanced Indexing. Sourdis et al. [8] proposed an index selection algorithm, called Prefix Indexing (PI). The indices for PI are selected by extracting the first N-byte string from each signature. This algorithm is efficient in cases that the beginning of signatures shares a number of common strings such as the case shown in Fig. 2.

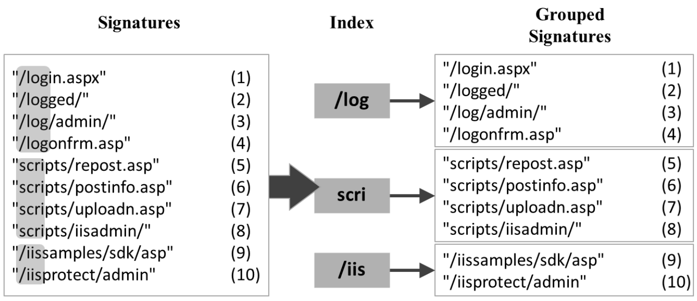


Fig. 2. Prefix Indexing

While PI is a simple way to group signatures with indices, grouping with other substrings that appear in the middle of signatures could be more appropriate in some cases. Based on this intuition, Random Indexing (RI) which selects indices randomly from signatures is proposed [10]. RI extracts an N-byte substring from an arbitrary point of signatures. Fig. 3 shows an example of RI.

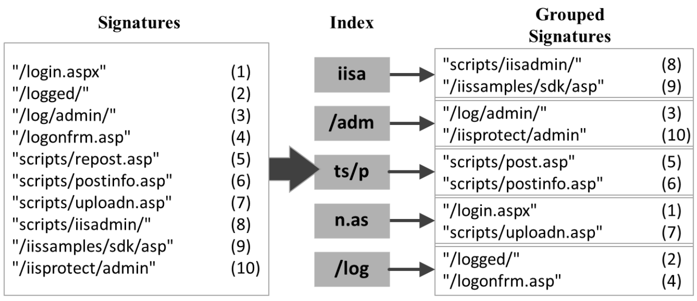


Fig. 3. Random Indexing

RI, however, exhibits poor grouping results than PI in many cases because RI is highly dependent on the order of selected indices. Fig. 4 shows a worse-case example of RI with the same indices in Fig. 3. In this example, 40% of signatures are grouped together; as a result, performance improvement can be reduced, compared with the case of Fig. 3.

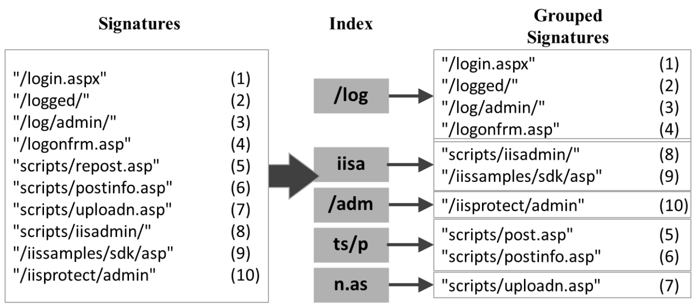


Fig. 4. A Worse-case Example of Random Indexing

To compensate the disadvantages of RI stated above, we propose a method that extracts indices with some restrictions. We applied the maximum and minimum number of signatures per index as restrictions. These restrictions prevent indices from pointing to too many signatures or too few signatures, generating more balanced results than RI. We called this approach as Balanced Indexing (BI). By experimenting various indexing cases, we derived minimum and maximum values as one and three, respectively, but these values can be set differently when applying to other sets of signatures. During the experiments we realized that both minimum and maximum values have limits, depending on the property of the signatures. The limit for the minimum value takes effect when there is a unique substring from a signature, which is not shared with any other signatures. In this case the index that contains the substring will have only one signature. On the other hand, when there is a substring that appears in many other signatures, the maximum number cannot be smaller than a certain value because many signatures share a part of signatures.

Algorithm 1. Balanced Indexing Algorithm

```
Input: All signatures $S$
       Length of an index $L$
       Minimum number of signatures per index $Min$
       Maximum number of signatures per index $Max$
Output: Extracted indices $I$
 1: $I \leftarrow \emptyset$
 2: while $S \neq \emptyset$ do
 3:     $subsigs \leftarrow \emptyset$
 4:     $sig \leftarrow$ a randomly selected signature from $S$
 5:     $index \leftarrow$ a $L$-byte substring of $sig$ (random)
 6:     for $\forall s \in S$ do
 7:         $subsigs \leftarrow subsigs + s$ that includes $index$
 8:     end for
 9:     if $|subsigs| \geq Min$ and $|subsigs| \leq Max$ then
10:         $I \leftarrow I + index$
11:         $S \leftarrow S - subsigs$
12:     end if
13: end while
```

Algorithm 1 shows the pseudo code of the BI algorithm. BI randomly selects a signature and extracts a substring from an arbitrary point of the chosen signature and searches for other signatures that contain the substring. All found signatures are added to the list of the corresponding substring. So far, it is the same as RI. After finding all signatures which contain the substring, the substring will be accepted as an index only when the number of signatures is within the boundary; Max and Min. The accepted index and the signatures are stored in the index list, indices, and the signatures are excluded from further process. BI repeats the above processes until all signatures are exhausted. To avoid infinite loops, Max and Min are selected appropriately.

4 Experiments

In this section, we present experimental results of our indexing method, BI. We used Snort 2.9.0.0 to evaluate the effectiveness of our proposed method. Snort has a number of signature files which are divided into several categories by the types of attacks and a signature consists of multiple strings to be examined with the incoming packet. We experimented with three index selection algorithms; PI, RI and BI.

Table 1 shows the information of Web-IIS signatures which are used in our experiments. Web-IIS contains 143 signatures which are related to attacks against Windows IIS servers. The total length of signatures within the Web-IIS is 2,363 bytes and the average length of signatures is 16.52 bytes. With these signatures, we applied the three indexing methods and analyzed the results. PI always generates the same result, but RI generates different results in every experiment due to random selection of substrings; for this reason we analyzed both the best result and the worst result of RI. BI also generates different result each time; but we only analyzed the best result for BI because the results of the other cases do not show a big difference.

Table 1. Web-IIS Signatures

	Web-IIS
Number of Signatures	143
Total Length of Signatures	2,363
Average Length of Signatures	16.52
Maximum Length of Signatures	65

Indexing results were analyzed with three statistical values: the number of indices (NI), the average number of signatures per index (ANPI) and the maximum number of signatures per index (MNPI). NI refers to the size of the entire indices, which is related to the total length of index to be examined with an input packet; therefore if NI decreases, the packet inspection process can have better performance. ANPI is another important factor of performance as it has a close relationship with the amount of strings to be inspected; the fewer the strings, the less it takes for the deep packet inspection. We found that the appropriate value for ANPI is between 1.5 and 1.8, experimentally. MNPI has effects on the deep packet inspection time, especially for the worst case. If the size of MNPI, let's say M, is significantly large, the worst case time required for the deep packet inspection increases as the packet inspection involves at least M times of comparison to the input packet. To summarize, an indexing result with smaller MNPI, especially which has a close value to ANPI, shows the best performance in the packer inspection.

Table 2 shows the experimental results of the three indexing methods, where the length of each index is set to four bytes. The three methods, PI, RI and BI show similar results in most cases, except the MNPI value. BI has the smallest MNPI, whose value is also close to ANPI; therefore the indexing with BI provides a fairly equally distributed result. Other results with bigger MNPI values mean that their signature groupings were biased to a certain index. If the biased index is matched, a large number of signatures should be fully examined, causing longer deep packet inspection time. We will discuss the performance issues later in this section.

Table 2. Statistical Values of Web-IIS

Methods / Values	PI	RI		BI
		Best	Worst	
Length of Index	4	4	4	4
Number of Indices (NI)	84	80	50	85
Average Number of Signatures per Index (ANPI)	1.70	1.78	2.86	1.68
Maximum Number of Signatures per Index (MNPI)	15	10	41	3

To evaluate the performance of packet examination, we estimated the total length of strings (TLS) to be examined for an input packet since the TLS is one of the critical factors in the deep packet inspection process, regardless of the string matching approach being used. The smaller the TLS to be examined becomes, the shorter the packet examination time would be taken. We estimate the performance of the index selection algorithm, by calculating the TLS to be examined; we describe the estimation process in the following paragraphs.

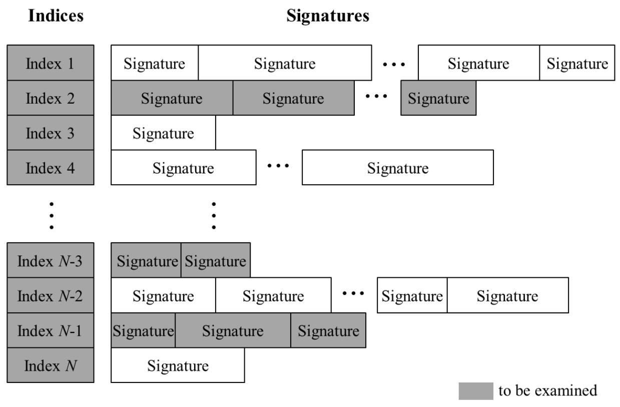


Fig. 5. A Result of Index Selection

The TLS is the total length of indices (TLI) and the total length of signatures of matched indices (TLSI), as shown in Figure 5. TLI is the number of indices times the length of index, as the whole indices should be examined during the inspection. TLSI is the sum of each length of the signatures that are pointed by matched indices. To calculate TLSI, we need to know which index was matched, because only the signatures that are pointed by the matched indices will be examined; however, every time the inspection proceeds, different indices will be matched according to incoming packets. There is no way to foresee the matched indices unless the packet inspection is actually conducted; therefore we formularized Equation 1 to estimate TLSI.

$$\text{TLSI} = \text{Number of matched Indices} \times \text{ANPI} \times \text{Average Length of Signature.} \qquad (1)$$

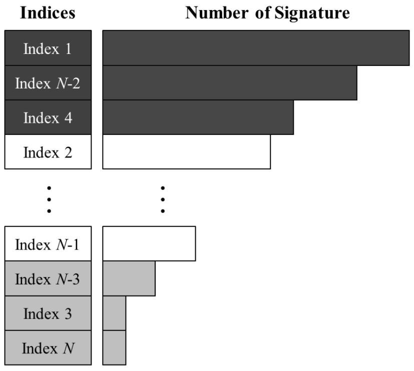


Fig. 6. Index Sorting

As mentioned above, it is hard to estimate the number of matched indices. To overcome this difficulty, we set the number of matched indices as a variable and conducted simulations. The simulations estimate TLS, assuming that x% of indices have been matched to an incoming packet. If we sort the indices by the number of signatures per index, as shown in Figure 6, we can analyze the best case and the worst case. The best case is when x% of indices with the smallest number of signatures were taken and the worst case is the opposite. We also calculated the average case by measuring the average of the best and worst cases. Equation 2 shows how these values are used for the simulation.

$$\text{TLS}(x) = \text{NI} \times \text{Length of Index} + \text{Number of Signatures for } x\% \text{ of Indices} \times \text{Average Length of Signature.} \qquad (2)$$

To compare the performance of three index selection algorithms, we extracted a set of indices from the Web-IIS signatures using each algorithm and estimated TLS for each set. In the case of RI, we chose the best case and the worst case from 100 tries. Figure 7 shows the result of the experiments for the Web-IIS: Figure 7(a) is for the average cases and Figure 7(b) is for the worst cases. The Y axis represents TLS and the X axis represents the percentage of matched indices for an input packet.

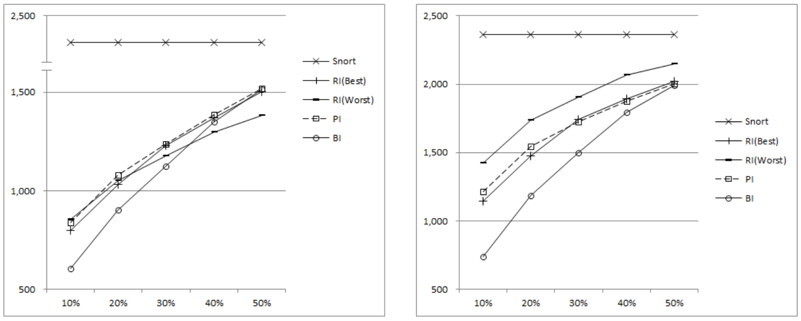


Fig. 7. The Experiment Results of Web-IIS: (a) Average cases, (b) Worst cases

As can be seen in the graphs, all three index selection algorithms show better performance than the original Snort. RI shows unpredictable performance, showing two different aspects: in the best case, the performance of RI is almost equal to PI but, in the worst case, its performance is far worse than PI. On the other hand, BI shows the best performance amongst the three algorithms. When 10% of indices are matched, 604 bytes of strings are to be examined, which is only 26% of strings compared to those that original Snort uses, while 72% of strings examined using PI. In the worst case, the amount of strings to be examined, when 10% of indices are matched, is 736 bytes, which is 31% of the amount of using original Snort. When more than 40% of indices are matched, PI shows the best performance; however, as it is believed that the chance of matching more than 40% of indices is rare, BI would show better performance than other algorithms in most cases. This improvement is a result of having smaller MNPI as we reduced the value of MNPI by selecting appropriate maximum value in BI. It can be said that the rules are well distributed into groups by the BI method, which also implies that the amount of strings to be examined is normalized.

To summarize, BI shows better performance than PI and RI. All three algorithms have similar NI and ANPI values, but BI has the smallest MNPI value, which is an important factor in determining the packet inspection performance. Smaller MNPI value also implies that the signatures are well distributed into groups by our indexing method, BI. The estimated amount of strings to be examined with BI is also less than using the other two algorithms, achieving 13% to 28% improvements on average, and 23% to 39% improvements in the worst case. The actual performance of the packet inspection time, however, would be improved more than the simulation results, as the amount of strings to be examined might increase the packet inspection time exponentially, not linearly.

Our experiments are conducted using only 143 signatures while Snort has more than 5,000 signatures. However, our proposed method is still useful because Snort divides the signatures into several groups using a chain structure for the efficiency. Our proposed method can be applied to those groups in the pre-processing phase.

5 Conclusions

The importance of accurate network intrusion detection is growing over time. The common way of deep packet inspection is examining all signatures in signature-based intrusion detection systems (IDS). This approach, however, is inefficient because the majority of packets are benign. The overall performance of signature-based IDSs can be improved if the number of signatures to be examined decreases. In this paper, we suggested a method named Balanced Indexing (BI) that extracts indices with two parameters, the maximum and minimum number of signatures per index, to reduce the number of signatures to be examined. We verified our assumption with a number of experiments, which showed that BI can achieve 13% to 39% improvements in a view of the estimated amount of strings to be examined.

For the future work, we will design a new detection engine, optimized for BI. The deep packet inspection process, followed by the index matching, examines the packet payload with the candidate Snort rule signatures. For Prefix Indexing (PI), this process is simple as each index is the first few characters of each signature. On the other hand, the index for BI can appear in the middle of a signature; therefore the signature examination should be for both forward and backward directions. We will also consider an index feedback mechanism which uses the detection results to reconstruct indices. With this mechanism, the grouping by indices can be optimized.

Acknowledgment. This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the HNRC(Home Network Research Center) ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency) (NIPA-2012-H0301-12-1002).

References

1. Tuck, N., Sherwood, T., Calder, B., Varghese, G.: Deterministic Memory-Efficient String Matching Algorithms for Intrusion Detection. In: IEEE INFOCOM (2004)
2. Tan, L., Sherwood, T.: A High Throughput String Matching Architecture for Intrusion Detection and Prevention. In: Proceedings of the 32nd Annual International Symposium on Computer Architecture (2005)
3. Song, T., Zhang, W., Wang, D., Xue, Y.: A Memory Efficient Multiple Pattern Matching Architecture for Network Security. In: IEEE INFOCOM (2008)
4. Yu, F., Chen, Z., Diao, Y., Lakshman, T.V., Katz, R.H.: Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection. In: 2nd ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS) (2006)
5. Kastil, J., Korenek, J., Lengal, O.: Methodology for Fast Pattern Matching by Deterministic Finite Automaton with perfect Hashing. In: IEEE 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools (2009)
6. Bispo, J., Sourdis, I., Cardoso, J.M.P., Vassiliadis, S.: Regular Expression Matching for Reconfigurable Packet Inspection. In: IEEE International Conference on Field Programmable Technology (2006)

7. Baker, Z.K., Prasanna, V.K.: A Methodology for Synthesis of Efficient Intrusion Detection System on FPGAs. In: IEEE FCCM (2004)
8. Sourdis, I., Dimopoulos, V., Pnevmatikatos, D., Vassiliadis, S.: Packet pre-filtering for network intrusion detection. In: 2nd ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), pp. 183–192 (2006)
9. Chen, H., Summerville, D.H., Chen, Y.: Two-stage Decomposition of SNORT Rules towards Efficient Hardware Implementation. In: Design of Reliable Communication Networks (DRCN), pp. 359–366 (2009)
10. Kang, B., Kim, H.S., Yang, J.S., Im, E.G.: Rule Indexing for Efficient Intrusion Detection Systems. In: Jung, S., Yung, M. (eds.) WISA 2011. LNCS, vol. 7115, pp. 136–141. Springer, Heidelberg (2012)

Quantitative Questions on Attack–Defense Trees

Barbara Kordy, Sjouke Mauw, and Patrick Schweitzer

University of Luxembourg, SnT
{barbara.kordy,sjouke.mauw,patrick.schweitzer}@uni.lu

Abstract. Attack–defense trees are a novel methodology for graphical security modeling and assessment. The methodology includes intuitive and formal components that can be used for quantitative analysis of attack–defense scenarios. In practice, we use intuitive questions to ask about aspects of scenarios we are interested in. Formally, a computational procedure, using a bottom-up algorithm, is applied to derive the corresponding numerical values. This paper bridges the gap between the intuitive and the formal way of quantitatively assessing attack–defense scenarios. We discuss how to properly specify a question, so that it can be answered unambiguously. Given a well-specified question, we then show how to derive an appropriate attribute domain which constitutes the corresponding formal model.

1 Introduction

Attack–defense trees [15] form a systematic methodology for analysis of attack–defense scenarios. They represent a game between an attacker, whose goal is to attack a system, and a defender who tries to protect the system. The widespread formalism of attack trees is a subclass of attack–defense trees, where only the actions of the attacker are considered. The attack–defense tree methodology combines intuitive and formal components. On the one hand, the intuitive visual attack–defense tree representation is used in practice to answer qualitative and quantitative questions, such as "What are the minimal costs to protect a server?", or "Is the scenario satisfiable?" On the other hand, there exist attack–defense terms and a precise mathematical framework for quantitative analysis using a recursive bottom-up procedure formalized for attack trees in [21] and extended to attack–defense trees in [14].

There exists a significant discrepancy between users focusing on the intuitive components of the model and users working with the formal components. This is due to the fact that intuitive models are user friendly but often ambiguous. In contrast, formal models are rigorous and mathematically sound. This, however, makes them difficult to understand for users without a formal background. This discrepancy between the two worlds is especially visible in the case of quantitative analysis. A proper numerical evaluation can only be performed when all users have a precise and consistent understanding of the considered quantities, which are also called attributes.

Contributions. This work aims to bridge the gap between the intuitive and the formal components of the attack–defense tree methodology for quantitative security analysis. We elaborate which kind of intuitive questions occurring in practical security analysis can be answered with the help of the bottom-up procedure on attack–defense trees. We empirically classify and formally analyze questions that were collected during case studies and literature reviews. For each class we provide detailed guidelines how the questions should be specified, so that they are unambiguous and can be answered correctly. Simultaneously, we discuss templates of the attribute domains corresponding to each class.

Related work. An excellent historical overview on graphical security modeling, was given by Piètre-Cambacédès and Bouissou in [22]. In [26], Schneier introduced the graphical attack tree formalism and proposed how to evaluate, amongst others, attack costs, success probability of an attack, and whether there is a need for special equipment. Baca and Petersen [4] have extended attack trees to countermeasure graphs and quantitatively analyzed an open-source application development. Bistarelli et al. [6], Edge et al. [9] and Roy et al. [24] have augmented attack trees with a notion of defense or mitigation nodes. They all analyze specific types of risk, using particular risk formulas adjusted to their models. Willemson and Jürgenson [29] introduced an order on the leaves of attack trees to be able to optimize the computation of the expected outcome of the attacker. There also exist a number of case studies and experience reports that quantitatively analyze real-life systems. Notable examples are Henniger et al. [11], who have conducted a study using attack trees for vehicular communications systems, Abdulla et al. [1], who analyzed the GSM radio network using attack jungles, and Tanu and Arreymbi [27], who assessed the security of mobile SCADA system for a tank and pump facility. Since all previously mentioned papers focus on specific attributes, they do not address the general problem of the relation between intuitive and formal quantitative analysis.

The formalism of attack–defense trees considered in this work was introduced by Kordy et al. in [14]. Formal aspects of this methodology have been discussed in [13] and [17]. In [5], Bagnato et al. provided guidelines for how to use attack–defense trees in practice. They analyzed a DoS attack scenario on an RFID-based goods management system by evaluating a number of relevant attributes. An extended version of the present paper contains more technical details and illustrative examples [16].

2 Attack–Defense Scenarios Intuitively

An *attack–defense tree* (ADTree) constitutes an intuitive graphical model describing the measures an attacker might take in order to attack a system and the defenses that a defender can employ to protect the system. An ADTree is a node-labeled rooted tree having nodes of two opposite types: *attack nodes* represented with circles and *defense nodes* represented with rectangles. The root node of an ADTree depicts the main goal of one of the players. Each node of an ADTree may have one or more children of the same type which *refine* the node's goal into subgoals. The refinement relation is indicated by solid edges and can be either disjunctive or conjunctive. The goal of a *disjunctively* refined node is achieved when *at least one* of its children's goals is achieved. The goal of a *conjunctively* refined node is achieved when *all* of its children's goals are achieved. To distinguish between the two refinements we indicate the conjunctive refinement with an arc. A node which does not have any children of the same type is called a *non-refined* node. Non-refined nodes represent *basic actions*, i.e., actions which can be easily understood and quantified. Every node in an ADTree may also have one child of the opposite type, representing a *countermeasure*. The countermeasure relation is indicated by dotted edges. Nodes representing countermeasures can again be refined into subgoals and countered by a node of the opposite type.

Example 1. An example of an ADTree is given in Figure 1 (left). The root of the tree represents an attack on a server. Three ways to accomplish this attack are depicted: insider attack, outsider attack (OA) and stealing the server (SS). To achieve his goal, an insider needs to be internally connected (IC) and have the correct user credentials (UC). To not be caught easily, an insider uses a colleague's and not his own credentials. Attack by an outsider can be prevented if a properly configured firewall (FW) is installed.

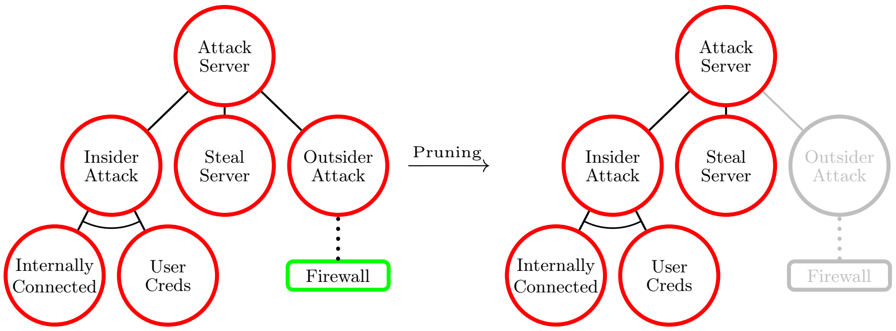


Fig. 1. Left: an ADTree for how to attack a server. Right: pruned "attack on server" scenario for questions of Class 1 owned by the attacker.

Graphical visualization of potential attacks and possible countermeasures constitutes a first step towards a systematic security analysis. The next step is to assign numerical values to ADTree models, i.e., to perform a quantitative analysis. Intuitively speaking, performing a quantitative security analysis means *answering questions related to specific aspects or properties influencing the security of a system or a company.* These questions may be of Boolean type, e.g., "Is the attack satisfiable?", or may concern physical or temporal aspects, e.g., "What are the minimal costs of attacking a system?", or "How long does it take to detect the attack?" In order to facilitate and automate the analysis of vulnerability scenarios using ADTrees, the formal model of ADTerms and their quantitative analysis have been introduced in [15]". We briefly recall necessary definitions in Appendix A.

3 Classification of Questions

In this paper, we provide a pragmatic taxonomy of quantitative questions that can be asked about ADTrees. The presented classification results from case studies, e.g., [5,9,27], as well as from a detailed literature overview concerning quantitative analysis of security. Our study allowed us to identify three main classes of empirical questions, as described below.

Class 1: Questions referring to one player (Section 4). Most of the typical questions for ADTrees have an explicit or implicit reference to one of the players which we call *owner* of the question. Examples of questions referring to one player are "What are the *minimal costs of the attacker*?" (here the owner is the attacker) or "How much does it *cost to protect* the system?" (here the implicitly mentioned owner is the defender). When we ask a question of Class 1, we assume that its owner does not have extensive information concerning his adversary. Thus, we always consider the worst case scenario with respect to the actions of the other player. Most of the questions usually asked for attack trees can be adapted so that they can be answered on ADTrees as well. Thus, questions related to attributes such as *costs* [26,7,27,4,25,21,30,1,24,8,2,28,9], *time* [11,26,28], *detectability* [27,8], *special skills* [21,1,26], *impact* [26,27,11,19,25,21,3,1,23,9,28], *difficulty* [8,10,27,11,21,1,3,28], *penalty* [7,12,28], *profit* [3,12,6,24], etc., all belong to Class 1.

Class 2: Questions where answers for both players can be deduced from each other (Section 5). Exemplary questions belonging to Class 2 are "Is the *scenario satisfiable*?", or "How *probable is it that the scenario will succeed*?". We observe that if one player succeeds with probability p, we also know that the other player succeeds with probability $1-p$. The *satisfiability* attribute is considered, either explicitly or implicitly, in all works concerning attack trees and their extensions. The *probability*[1] attribute has been extensively studied in [26,7,11,19,20,30,1,24,8,9,28].

Class 3: Questions referring to an outside third party (Section 6). Questions belonging to Class 3 relate to a universal property which is influenced by actions of both players. For instance, one could ask about "How much *data traffic is involved in the attack–defense scenario*?". In this case, we do not need to distinguish between traffic resulting from the attacker's and the defender's actions, as both players contribute to the total amount. Attributes corresponding to questions in Class 3 have not been addressed in the attack tree literature, since attack trees focus on a single player. The importance of those questions becomes apparent when actions of two opposite parties are considered.

[1] We would like to point out that the probability attribute can only be evaluated using the bottom-up procedure given by Equation (1) in Appendix A if the ADTree does not contain any dependent actions.

The following three sections set up guidelines for how to correctly specify quantitative questions of all three classes. The guidelines' main purpose is to enable us to find a corresponding attribute domain in order to correctly compute an answer using the bottom-up procedure. Figure 2 depicts the three classes of questions, as well as general templates for the corresponding attribute domains, as introduced in Definition 3 in Appendix A. Symbols $\bullet, \circ, \diamond$ and $\bar{\bullet}$ serve as placeholders for specific operators. Corresponding symbols within a tuple indicate that the functions coincide. For instance, $(D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$ means that $\vee^{\mathrm{p}}_{\alpha} = \wedge^{\mathrm{o}}_{\alpha} = \mathrm{c}^{\mathrm{o}}_{\alpha}$ and that $\wedge^{\mathrm{p}}_{\alpha} = \vee^{\mathrm{o}}_{\alpha} = \mathrm{c}^{\mathrm{p}}_{\alpha}$. We motivate these equalities and give possible instantiations of $\bullet, \circ, \diamond$ and $\bar{\bullet}$ in the following.

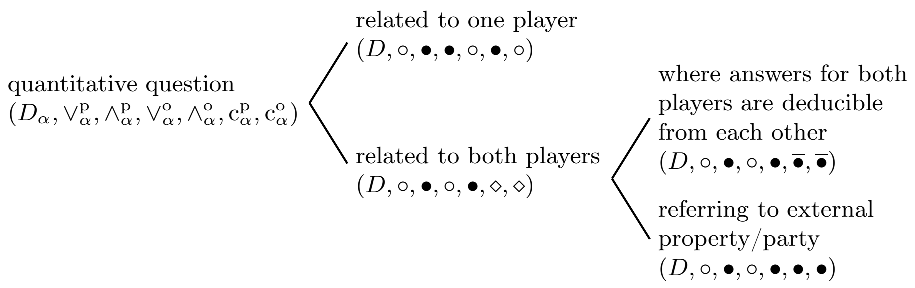

quantitative question
$(D_\alpha, \vee^{\mathrm{p}}_{\alpha}, \wedge^{\mathrm{p}}_{\alpha}, \vee^{\mathrm{o}}_{\alpha}, \wedge^{\mathrm{o}}_{\alpha}, \mathrm{c}^{\mathrm{p}}_{\alpha}, \mathrm{c}^{\mathrm{o}}_{\alpha})$
related to one player
$(D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$
related to both players
$(D, \circ, \bullet, \circ, \bullet, \diamond, \diamond)$
where answers for both players are deducible from each other
$(D, \circ, \bullet, \circ, \bullet, \bar{\bullet}, \bar{\bullet})$
referring to external property/party
$(D, \circ, \bullet, \circ, \bullet, \bullet, \bullet)$

Fig. 2. Classification of questions and attribute domains' templates

4 Questions Referring to One Player

4.1 Defining a Formal Model for Questions of Class 1

Questions belonging to Class 1 refer to exactly one player, which we call the question's *owner*. As we explain below, in the attack–defense tree setting, only two situations occur for a question's owner: either he needs to choose *at least one* option or he needs to execute *all* options. Therefore, two operators suffice to answer questions of Class 1 and the generic attribute domain is of the form $(D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$. Furthermore, if we change a question's owner, the attribute domain changes from $(D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$ into $(D, \bullet, \circ, \circ, \bullet, \circ, \bullet)$.

We illustrate the construction of the formal model for Class 1 using the question "What are the minimal costs of the attacker?", where the owner is the attacker. In the case of Class 1, all values assigned to nodes and subtrees express the property under consideration from the perspective of the question's owner. In the minimal costs example, this means that even subtrees rooted in defense nodes have to be quantified from the attacker's point of view, i.e., a value assigned to the root of a subtree expresses what is the minimal amount of money that the attacker needs to invest in order to be successful in the current subtree.

Subtrees rooted in uncountered attacker's nodes can either be disjunctively or conjunctively refined. In the first case the attacker needs to ensure that he is successful in *at least one* of the refining nodes, in the second case he needs to be successful in *all* refining nodes. The situation for subtrees rooted in uncountered defender's nodes is reciprocal. If a defender's node is disjunctively refined, the attacker needs to successfully counteract *all* possible defenses to ensure that he is successful at the subtree's root node; if the defender's node is conjunctively refined, successfully counteracting *at least one* of the refining nodes already suffices for the attacker to be successful at the subtree's root node.

This reciprocality explains that two different operators suffice to quantify all possible uncountered trees: The operator that we use to combine attribute values for disjunctively refined nodes of one player is the same as the operator we use for conjunctively refined nodes of the other player.

Furthermore, the same two operators can also be used to quantify all remaining subtrees. If a subtree is rooted in a countered attacker's node, the attacker needs to ensure that he is successful at the action represented by the root node *and* that he successfully counteracts the existing defensive measure. Dually, for the attacker to be successful in a subtree rooted in a defender's countered node, it is sufficient to successfully overcome the defensive action *or* to successfully perform the attack represented by the countering node. This implies that we can use the same operator as for conjunctively refined attacker's nodes in the first case and the same operator as for disjunctively refined attacker's nodes in the second case.

4.2 Pruning

For attributes in Class 1, we are only interested in one player, the owner of a question. Therefore for this class, we should disregard subtrees that do not lead to a successful scenario for the owner. We achieve this with the help of the *pruning* procedure illustrated in the following example.

Example 2. Consider the ADTree in Figure 1 (left) and assume that we are interested in calculating the minimal costs of the attacker. In this case, there is no need to consider the subtree rooted in “Outsider Attack”, because it is countered by the defense “Firewall” and thus does not lead to a successful attack. The subtree rooted in “Outsider Attack” therefore should be removed. This simultaneously eliminates having to provide values for the non-refined nodes “Outsider Attack” and ‘Firewall”. The computation of the minimal costs is then executed on the term corresponding to the tree in the right of Figure 1.

To motivate the use of the pruning procedure, let us distinguish two situations. If a non-refined node of the non-owner is countered, its assigned value should not influence the result of the computation. If a non-owner's node is not countered, its value should indicate that the owner does not have a chance to successfully perform this subscenario. Mathematically, it means that the value assigned to the non-refined nodes of the non-owner needs to be neutral with respect to one operator and simultaneously absorbing with respect to the other. Since, in general, such an element may not exist, we use pruning to eliminate one of the described situations, which results in elimination of the absorption condition.

Let us consider a question of Class 1 and its owner. In order to graphically prune an ADTree, we perform the following procedure. Starting from a leaf of the non-owner, we traverse the tree towards the root until we reach the first node v satisfying one of the following conditions.

- v is a node of the owner and part of a proper[2] disjunctive refinement;
- v is a node of the non-owner and part of a proper conjunctive refinement;
- v is a node of the owner that counteracts a refined node of the non-owner;
- v is the root of the ADTree.

The subtree rooted in node v is removed from the ADTree. The procedure is repeated, starting from all leaves of the non-owner. We note that the order in which we perform the procedure does not influence the final result. Also, in some cases the pruning procedure results in the removal of the entire ADTree. This is the case when the owner of the question does not have any way of successfully achieving his goal.

In [16], we show how to prune in a mathematical way and prove that it is equivalent to the presented intuitive way.

4.3 From a Question to an Attribute Domain

In this section we analyze how a question of Class 1 should look like, in order to be able to instantiate the attribute domain template $A = (D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$ with specific value set and operators. To correctly instantiate A, we need a value domain D, two operators (for *all* and *at least one*) and we need to know which of those operators instantiates $\circ$ and which $\bullet$. Thus, a well-specified question of Class 1 contains exactly four parts, as illustrated on the following question:

Modality:	What are the minimal
Notion:	costs
Owner:	of the proponent
Execution:	assuming that all actions are executed one after another?

Each of the four parts has a specific purpose in determining the attribute domain. ***Notion.*** The notion used by the question influences the choice of the value domain. The notions in Class 1, identified during our study, are: *time*, *consequence*, *costs*, *detectability*, *difficulty level*, *elapsed time*, *impact*, *insider required*, *mitigation success*, *outcome*, *penalty*, *profit*, *response time*, *resources*, *severity*, *skill level*, *special equipment needed*, *special skill needed*, *survivability*.
From the notion we determine the value domain, e.g., $\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, etc. The choice of the value domain influences the basic assignments, as well as the operators determined by the modality and the execution style. The selected value domain needs to include all values that we want to use to quantify the owner's actions. It also must contain a neutral element with respect to $\circ$, if own = p, and with respect to $\bullet$, if own = o. This neutral element is assigned to all non-refined nodes of the non-owner, as argued in Section 4.2.

[2] A refinement is called proper if it contains at least two refining nodes.

Modality. The modality of a question clarifies how options are treated. Thus, it determines the characteristic of the *at least one* operator. Different notions are accompanied with different modalities. In the case of costs, interesting modalities are minimal, maximal and average.
Execution. The question also needs to specify an execution style. Its value determines the treatment when all actions need to be executed. Thus, it describes the characteristic of the *all* operator. Exemplary execution styles are: simultaneously/sequentially (for time) or with reuse/without reuse (for resources).
Owner. The owner of a question determines how the modality and the execution are mapped to $\circ$ and $\bullet$. In case the owner is the root player, i.e., the proponent, $\circ$ is instantiated with the *at least one* operator and $\bullet$ with the *all* operator. In case the root player is not the owner, the instantiations are reciprocal.

Given all four parts, we can then construct the appropriate attribute domain. For the notion of continuous time, also called duration, possible combinations of the modality, the execution style and the owner have been determined in Table 1. We instantiate the attribute domain template $(D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$ with the elements of the algebraic structure $(D, \circ, \bullet)$, and use the value indicated in the last column of the table as the basic assignment for all non-refined nodes of the non-owner. The table can be used in the case of other notions as well, as shown in the next example.

Table 1. Determining an instantiation of the structure in Class 1, where e denotes the neutral element with respect to avg

	Notion	Modality	Owner	Execution	Structure $(D, \circ, \bullet)$	Basic assignment for $\overline{\text{own}}$
1	duration	min	p	sequential	$(\mathbb{R}, \min, +)$	$+\infty$
2	duration	avg	p	sequential	$(\mathbb{R}, \text{avg}, +)$	e
3	duration	max	p	sequential	$(\mathbb{R}, \max, +)$	$-\infty$
4	duration	min	o	sequential	$(\mathbb{R}, +, \min)$	0
5	duration	avg	o	sequential	$(\mathbb{R}, +, \text{avg})$	0
6	duration	max	o	sequential	$(\mathbb{R}, +, \max)$	0
7	duration	min	p	parallel	$(\mathbb{R}, \min, \max)$	$+\infty$
8	duration	avg	p	parallel	$(\mathbb{R}, \text{avg}, \max)$	e
9	duration	max	p	parallel	$(\mathbb{R}, \max, \max)$	$-\infty$
10	duration	min	o	parallel	$(\mathbb{R}, \max, \min)$	$-\infty$
11	duration	avg	o	parallel	$(\mathbb{R}, \max, \text{avg})$	$-\infty$
12	duration	max	o	parallel	$(\mathbb{R}, \max, \max)$	$-\infty$

Example 3. The question "What are the minimal costs of the proponent, assuming that reusing tools is infeasible?" can be answered using the attribute domain $A_{\text{co}} = (\mathbb{R}, \min, +, +, \min, +, \min)$. Here the notion is *cost*, which has the same value domain as duration, i.e., $\mathbb{R}$. The modality is *minimum*, the owner is *the proponent* and the execution style is *without reuse*, which corresponds to sequential. Hence, we use the structure $(\mathbb{R}, \min, +)$, as specified in Line 1 of

Table 1. In order to answer the question on the tree in the left of Figure 1, we first prune it, as shown on the right of Figure 1. The only basic actions that are left are “Internally connected”, “User Creds” and “Steal Server”. Suppose their costs are 100€, 200€, and 400€, respectively. We use those values as basic assignment β_{co} and apply the bottom-up computation to the ADTerm $\vee^{\mathrm{p}}(\wedge^{\mathrm{p}}(\mathrm{IC}, \mathrm{UC}), \mathrm{SS})$:

$$\mathrm{co}(\vee^{\mathrm{p}}(\wedge^{\mathrm{p}}(\mathrm{IC}, \mathrm{UC}), \mathrm{SS})) = \vee^{\mathrm{p}}_{\mathrm{co}}\,(\wedge^{\mathrm{p}}_{\mathrm{co}}(\beta_{\mathrm{co}}(\mathrm{IC}), \beta_{\mathrm{co}}(\mathrm{UC}), \beta_{\mathrm{co}}(\mathrm{SS})) = \min\{+(100€, 200€), 400€\} = 300€.$$

We would like to remark that if the structure $(D, \circ, \bullet)$ forms a semi-ring, it is not necessary to prune the ADTree to correctly answer a question Q of Class 1. This is due to the fact that in a semi-ring the neutral element[3] for the first operator is at the same time absorbing for the second operator. Such element can then be assigned to all subtrees which do not yield a successful scenario for the owner of Q, in particular to the uncountered basic actions of the non-owner.

5 Questions Where Answers for Both Players Can Be Deduced from Each other

We illustrate the construction of the attribute domain for Class 2 using the question “What is the success probability of a scenario, assuming that all actions are independent?” In case of questions of Class 2, values assigned to a subtree quantify the considered property from the point of view of the root player of the subtree. This means that, if a subtree rooted in an attack node is assigned the value 0.2, the corresponding *attack is successful* with probability 0.2. If a subtree rooted in a defense node is assigned the value 0.2, the corresponding *defensive measure is successful* with probability 0.2. Thus, in Class 2, conjunctive and disjunctive refinements for the proponent and the opponent have to be treated in the same way: in both cases, they refer to the *at least one* option (here modeled with $\circ$) and the *all* options (modeled with $\bullet$), of the player whose node is currently considered.

Questions in Class 2 have the property that, given a value for one player, we can immediately deduce a corresponding value for the other player. For example, if the attacker succeeds with probability 0.2 the defender succeeds with probability 0.8. This property is modeled using a value domain with a predefined unary negation operation $\overline{}$. Negation allows us to express the operators for both countermeasures using the *all* operator where the second argument is negated, which we represent by $\overline{\bullet}$. Formally, $\overline{\bullet}(x, y) = x \bullet \overline{y}$. Hence attribute domains of Class 2 follow the template $(D, \circ, \bullet, \circ, \bullet, \overline{\bullet}, \overline{\bullet})$.

Below we discuss three aspects that questions in Class 2 need to address.

Notion. Questions of Class 2 refer to notions for which the value domains contain a unary negation operation. This allows us to transform values of one

[3] Such an element is usually called *zero* of the semi-ring. For instance, $+\infty$ is the zero element of the semi-ring $(\mathbb{R}, \min, +)$.

player into values of the other player. Identified notions for Class 2 are: *feasibility*, *needs electricity*, *probability of occurrence*, *probability of success*, *satisfiability*.
Modality. Modality specifies the operator for *at least one* option. For the notions enumerated above, this will either be the logical OR ($\vee$) or the probabilistic addition of independent events $P_{\cup}(A, B) = P(A) + P(B) - P(A)P(B)$, for a given probability distribution P and events A and B.
Execution. Finally, we need to know what is the execution style, so that we can specify the operator for *all* options. In the above notions, this will either be the logical AND ($\wedge$) or the probabilistic multiplication of independent events $P_{\cap}(A, B) = P(A)P(B)$.

Example 4. We calculate the success probability of the scenario given in Figure 1 (left), assuming that all actions are independent. First we set the success probability of all basic actions to $\beta_{\mathrm{pb}} = 0.4$ and then we use the attribute domain $A_{\mathrm{pb}} = ([0,1], P_{\cup}, P_{\cap}, P_{\cup}, P_{\cap}, \overline{P_{\cap}}, \overline{P_{\cap}})$, where $\overline{P_{\cap}}(A, B) = P_{\cap}(A, \overline{B})$ to compute

$$P_{\cup}(P_{\cap}(\beta_{\mathrm{pb}}(\mathrm{IC}), \beta_{\mathrm{pb}}(\mathrm{UC})), \beta(\mathrm{SS}), P_{\cap}(\beta_{\mathrm{pb}}(\mathrm{OA}), 1 - \beta_{\mathrm{pb}}(\mathrm{FW}))) =$$
$$P_{\cup}(P_{\cap}(0.4, 0.4), 0.4, P_{\cap}(0.4, 1 - 0.4)) = P_{\cup}(0.16, 0.4, 0.24) = 0.61696.$$

6 Questions Relating to an Outside Third Party

Suppose an outsider is interested in the overall maximal power consumption of the scenario. As in the previous section, disjunctive refinements of both players should be treated with one operator and conjunctive refinements of both players with another operator. Indeed, for a third party the important information is whether *all* or *at least one* option need to be executed and not who performs the actions. Also countermeasures lose their opposing aspect and their values are aggregated in the same way as conjunctive refinements. Regarding the question, this is plausible since both the countered and the countering action contribute to the overall power consumption. These observations result in the following template for an attribute domain in Class 3: $(D, \circ, \bullet, \circ, \bullet, \bullet, \bullet)$.

We specify relevant parts of the questions in Class 3 on the following example.

Modality:	What is the maximal
Notion:	energy consumption
Execution:	knowing that sharing of power is impossible?

Notion. In Class 3, we use notions that express universal properties covering both players. Found examples are: *combined execution time*, *energy consumption*, *environmental costs*, *environmental damage*, *global costs*, *information flow*, *required network traffic*, *social costs*, *third party costs*.
Modality. The question should also contain enough information to allow us to specify how to deal with *at least one* option. In general, modalities used in Class 3 are the same as those in Class 1, e.g., minimal, maximal and average.
Execution. Finally, we need to know what is the execution style, so that we can define the correct operator for *all* options. The choices for execution style in Class 3 are again the same as in Class 1.

The three parts now straightforwardly define an algebraic structure $(D, \circ, \bullet)$ that we use to construct the attribute domain $(D, \circ, \bullet, \circ, \bullet, \bullet, \bullet)$.

Example 5. Consider the question "What is the maximal energy consumption for the scenario depicted in Figure 1 (left), knowing that sharing of power is impossible?" Both, the proponent's as well as the opponent's actions may require energy. We assume that being "Internally Connected", performing an "Outsider Attack" and running a "Firewall" all consume 20kWh. Obtaining "User Creds" requires 1kWh, whereas "Stealing Server" does not require any energy. These numbers constitute the basic assignment for the considered attribute. From the question we know that, when we have a choice, we should consider the option which consumes the most energy. Furthermore, since sharing of power is impossible, values for actions which require execution of several subactions should be added. Thus, we use the attribute domain $A_{\mathrm{erg}_{\max}} = (\mathbb{R}, \max, +, \max, +, +, +)$ and compute the maximal possible energy consumption in the scenario as

$$\begin{aligned}&\mathrm{erg}_{\max}((\vee^{\mathrm{p}}(\wedge^{\mathrm{p}}(\mathrm{IC}, \mathrm{UC}), \mathrm{SS})) =\\ &\max\{+(20\mathrm{kWh}, 1\mathrm{kWh}), 0\mathrm{kWh}, +(20\mathrm{kWh}, 20\mathrm{kWh})\} = 40\mathrm{kWh}.\end{aligned}$$

Due to similarities for modality and execution style for questions of Class 1 and Class 3, we can make use of Table 1, to choose the structure $(D, \circ, \bullet)$ that determines an attribute domain for a question of Class 3. The table corresponds to the case where the owner is the proponent.

7 Methodological Advancements for Attack Trees

ADTrees extend the well-known formalism of attack trees [26] by incorporating defensive measures to the model. Hence, every attack tree is in particular an ADTree. Underspecified questions are not a new phenomenon of ADTrees, but already occur in the case of pure attack trees. Thus, the formalization of quantitative questions, proposed in this paper, is not only useful in the attack–defense tree methodology but, more importantly, it helps users of the more widely spread formalism of attack trees.

Given a well-specified question on ADTrees and the corresponding attribute domain, we can answer the question on attack trees as well. Formally, attack trees are represented with terms involving only operators $\vee^{\mathrm{p}}$ and $\wedge^{\mathrm{p}}$. If $A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha, \vee^{\mathrm{o}}_\alpha, \wedge^{\mathrm{o}}_\alpha, \mathrm{c}^{\mathrm{p}}_\alpha, \mathrm{c}^{\mathrm{o}}_\alpha)$ is an attribute domain for ADTerms, the corresponding attribute domain for attack trees, as formalized in [21], is $A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha)$. Furthermore, due to the fact that attack trees involve only one player (the attacker), the notions of attacker, proponent, and question's owner coincide in this simplified model. This in turn implies that, in the case of attack trees, the three classes of questions considered in this paper form in fact one class.

8 Prototype Tool

In order to automate the analysis of security scenarios using the attack–defense methodology, we have developed a prototype software tool, called *ADTool*. It is written in Java and is compatible with multiple platforms (Windows, Linux, MAC OS). The ADTool is publicly available [18]. Its main functionalities include easy creation and efficient editing of ADTrees and ADTerms as well as automated evaluation of attributes on ADTrees.

The ADTool combines the features offered by graphical tree representations with mathematical functionalities provided by ADTerms and attributes. The user can choose whether to work with intuitive ADTrees or with formal ADTerms. When one of these models is created or modified, the other one is generated automatically. The possibility of modular display of ADTrees makes the ADTool suitable for dealing with large industrial case studies which may correspond to very complex scenarios and may require large models.

The software supports attribute evaluation on ADTrees, as presented in this paper. A number of predefined attribute domains allow the user to answer questions of Classes 1, 2 and 3. Implemented attributes include: costs, satisfiability, time and skill level, for various owners, modalities and execution styles; scenario's satisfiability and success probability; reachability of the root goal in less than x minutes, where x can be customized by the user; and the maximal energy consumption.

9 Conclusions

A useful feature of the attack–defense tree methodology is that it combines an intuitive representation and algorithms with formal mathematical modeling. In practice we model attack–defense scenarios in a graphical way and we ask intuitive questions about aspects and properties that we are interested in. To formally analyze the scenarios, we employ attack–defense terms and attribute domains. In this paper, we have guided the user in how to properly formulate a quantitative question on an ADTree and how to then construct the corresponding attribute domain. Since attack trees are a subclass of attack–defense trees, our results also advance the practical use of quantitative analysis of attack trees.

We are currently applying the approach presented in this paper to analyze socio-technical weaknesses of real-life scenarios, such as Internet web filtering, which involve trade offs between security and usability. In the future, we also plan to investigate the relation between attribute domains of all three classes and the problem of equivalent representations of a scenario, as formalized in [15].

Acknowledgments. We would like to thank Piotr Kordy for his contribution to the development of the ADTool. The authors were supported by the Fonds National de la Recherche Luxembourg under the grants C08/IS/26 and PHD-09-167.

References

1. Abdulla, P.A., Cederberg, J., Kaati, L.: Analyzing the Security in the GSM Radio Network Using Attack Jungles. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010, Part I. LNCS, vol. 6415, pp. 60–74. Springer, Heidelberg (2010)

2. Amenaza: SecurITree, `http://www.amenaza.com/` (accessed October 5, 2012)
3. Amoroso, E.G.: Fundamentals of Computer Security Technology. Prentice-Hall, Inc., Upper Saddle River (1994), `http://portal.acm.org/citation.cfm?id=179237#`
4. Baca, D., Petersen, K.: Prioritizing Countermeasures through the Countermeasure Method for Software Security (CM-Sec). In: Ali Babar, M., Vierimaa, M., Oivo, M. (eds.) PROFES 2010. LNCS, vol. 6156, pp. 176–190. Springer, Heidelberg (2010)
5. Bagnato, A., Kordy, B., Meland, P.H., Schweitzer, P.: Attribute Decoration of Attack–Defense Trees. International Journal of Secure Software Engineering (IJSSE) 3(2), 1–35 (2012)
6. Bistarelli, S., Dall'Aglio, M., Peretti, P.: Strategic Games on Defense Trees. In: Dimitrakos, T., Martinelli, F., Ryan, P.Y.A., Schneider, S. (eds.) FAST 2006. LNCS, vol. 4691, pp. 1–15. Springer, Heidelberg (2007), `http://www.springerlink.com/content/83115122h9007685/`
7. Buldas, A., Laud, P., Priisalu, J., Saarepera, M., Willemson, J.: Rational Choice of Security Measures Via Multi-parameter Attack Trees. In: López, J. (ed.) CRITIS 2006. LNCS, vol. 4347, pp. 235–248. Springer, Heidelberg (2006)
8. Byres, E.J., Franz, M., Miller, D.: The Use of Attack Trees in Assessing Vulnerabilities in SCADA Systems. In: International Infrastructure Survivability Workshop (IISW 2004). Institute of Electrical and Electronics Engineers, Lisbon (2004)
9. Edge, K.S., Dalton II, G.C., Raines, R.A., Mills, R.F.: Using Attack and Protection Trees to Analyze Threats and Defenses to Homeland Security. In: MILCOM, pp. 1–7. IEEE (2006)
10. Fung, C., Chen, Y.L., Wang, X., Lee, J., Tarquini, R., Anderson, M., Linger, R.: Survivability analysis of distributed systems using attack tree methodology. In: Proceedings of the 2005 IEEE Military Communications Conference, vol. 1, pp. 583–589 (October 2005)
11. Henniger, O., Apvrille, L., Fuchs, A., Roudier, Y., Ruddle, A., Weyl, B.: Security requirements for automotive on-board networks. In: 9th International Conference on Intelligent Transport Systems Telecommunications (ITST 2009), Lille, pp. 641–646 (October 2009)
12. Jürgenson, A., Willemson, J.: Computing Exact Outcomes of Multi-parameter Attack Trees. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part II. LNCS, vol. 5332, pp. 1036–1051. Springer, Heidelberg (2008)
13. Kordy, B., Mauw, S., Melissen, M., Schweitzer, P.: Attack–Defense Trees and Two-Player Binary Zero-Sum Extensive Form Games Are Equivalent. In: Alpcan, T., Buttyán, L., Baras, J.S. (eds.) GameSec 2010. LNCS, vol. 6442, pp. 245–256. Springer, Heidelberg (2010)
14. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Foundations of Attack–Defense Trees. In: Degano, P., Etalle, S., Guttman, J. (eds.) FAST 2010. LNCS, vol. 6561, pp. 80–95. Springer, Heidelberg (2011)
15. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Attack–Defense Trees. Journal of Logic and Computation, 1–33 (2012), `http://logcom.oxfordjournals.org/content/early/2012/06/21/logcom.exs029.short?rss=1`
16. Kordy, B., Mauw, S., Schweitzer, P.: Quantitative Questions on Attack–Defense Trees. arXiv (2012), `http://arxiv.org/abs/1210.8092`
17. Kordy, B., Pouly, M., Schweitzer, P.: Computational Aspects of Attack–Defense Trees. In: Bouvry, P., Kłopotek, M.A., Leprévost, F., Marciniak, M., Mykowiecka, A., Rybiński, H. (eds.) SIIS 2011. LNCS, vol. 7053, pp. 103–116. Springer, Heidelberg (2012)

18. Kordy, P., Schweitzer, P.: The ADTool, `http://satoss.uni.lu/members/piotr/adtool/index.php` (accessed October 12, 2012)
19. Li, X., Liu, R., Feng, Z., He, K.: Threat modeling-oriented attack path evaluating algorithm. Transactions of Tianjin University 15(3), 162–167 (2009), `http://www.springerlink.com/content/v76g872558787214/`
20. Manikas, T.W., Thornton, M.A., Feinstein, D.Y.: Using Multiple-Valued Logic Decision Diagrams to Model System Threat Probabilities. In: 41st IEEE International Symposium on Multiple-Valued Logic (ISMVL 2011), pp. 263–267 (2011)
21. Mauw, S., Oostdijk, M.: Foundations of Attack Trees. In: Won, D., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 186–198. Springer, Heidelberg (2006), `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.97.1056`
22. Piètre-Cambacédès, L., Bouissou, M.: Beyond Attack Trees: Dynamic Security Modeling with Boolean Logic Driven Markov Processes (BDMP). In: European Dependable Computing Conference, pp. 199–208. IEEE Computer Society, Los Alamitos (2010)
23. Roy, A., Kim, D.S., Trivedi, K.S.: Cyber security analysis using attack countermeasure trees. In: Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research (CSIIRW 2010), pp. 28:1–28:4. ACM, New York (2010), `http://doi.acm.org.proxy.bnl.lu/10.1145/1852666.1852698`
24. Roy, A., Kim, D.S., Trivedi, K.S.: Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees. Security and Communication Networks 5(8), 929–943 (2012), `http://dx.doi.org/10.1002/sec.299`
25. Saini, V., Duan, Q., Paruchuri, V.: Threat Modeling Using Attack Trees. J. Computing Small Colleges 23(4), 124–131 (2008), `http://portal.acm.org/citation.cfm?id=1352100`
26. Schneier, B.: Attack Trees. Dr. Dobb's Journal of Software Tools 24(12), 21–29 (1999), `http://www.ddj.com/security/184414879`
27. Tanu, E., Arreymbi, J.: An examination of the security implications of the supervisory control and data acquisition (SCADA) system in a mobile networked environment: An augmented vulnerability tree approach. In: Proceedings of Advances in Computing and Technology (AC&T) The School of Computing and Technology 5th Annual Conference. pp. 228–242. University of East London, School of Computing, Information Technology and Engineering (2010), `http://hdl.handle.net/10552/994`
28. Wang, J., Whitley, J.N., Phan, R.C.W., Parish, D.J.: Unified Parametrizable Attack Tree. International Journal for Information Security Research 1(1), 20–26 (2011), `http://www.infonomics-society.org/IJISR/Unified%20Parametrizable%20Attack%20Tree.pdf`
29. Jürgenson, A., Willemson, J.: Serial Model for Attack Tree Computations. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 118–128. Springer, Heidelberg (2010), `http://research.cyber.ee/~jan/publ/serialattack.pdf`
30. Yager, R.R.: OWA trees and their role in security modeling using attack trees. Inf. Sci. 176(20), 2933–2959 (2006)

A Attack–Defense Scenarios Formally

In this section we recall formal definitions related to our methodology. For more details and explanatory examples we refer the reader to [15]. To formally represent and analyze ADTrees, typed terms over a particular typed signature, called the AD–signature, have been introduced in [14]. To be able to capture ADTrees rooted in an attacker's node as well as those rooted in a defender's node, we distinguish between the *proponent* (denoted by p), which refers to the root player, and the *opponent* (denoted by o), which is the other player. For instance, for the ADTree in Figure 1 (left), the proponent is the attacker and the opponent is the defender. Conversely, if the root of an ADTree is a defense node, the proponent is the defender and the opponent is the attacker.

Furthermore, given a set $\mathcal{S}$, we denote by $\mathcal{S}^*$ the set of all finite strings over $\mathcal{S}$, and by ε the empty string. For $s \in \mathcal{S}$, we denote by s^+ a string composed of a finite number of symbols s.

Definition 1. *The* AD–signature *is a pair* $\Sigma = (\mathcal{S}, \mathcal{F})$*, where*

- $\mathcal{S} = \{\mathrm{p}, \mathrm{o}\}$ *is a set of types, and*
- $\mathcal{F} = \mathbb{B}^{\mathrm{p}} \cup \mathbb{B}^{\mathrm{o}} \cup \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}, \vee^{\mathrm{o}}, \wedge^{\mathrm{o}}, \mathrm{c}^{\mathrm{p}}, \mathrm{c}^{\mathrm{o}}\}$ *is a set of function symbols, such that the sets* $\mathbb{B}^{\mathrm{p}}$*,* $\mathbb{B}^{\mathrm{o}}$ *and* $\{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}, \vee^{\mathrm{p}}, \wedge^{\mathrm{o}}, \wedge^{\mathrm{o}}, \mathrm{c}^{\mathrm{p}}, \mathrm{c}^{\mathrm{o}}\}$ *are pairwise disjoint.*

Every function symbol $F \in \mathcal{F}$ *is equipped with a mapping* rank: $\mathcal{F} \to \mathcal{S}^* \times \mathcal{S}$*, where* $\mathrm{rank}(F)$ *is defined as a pair* $(\mathrm{in}(F), \mathrm{out}(F))$*. The first component of the pair describes the type of the arguments of* F *and the second component describes the type of the values of* F*. We have*

$$\begin{aligned}
&\mathrm{rank}(b) = (\varepsilon, \mathrm{p}), \ \textit{for } b \in \mathbb{B}^{\mathrm{p}}, && \mathrm{rank}(b) = (\varepsilon, \mathrm{o}), \ \textit{for } b \in \mathbb{B}^{\mathrm{o}},\\
&\mathrm{rank}(\vee^{\mathrm{p}}) = (\mathrm{p}^+, \mathrm{p}), && \mathrm{rank}(\vee^{\mathrm{o}}) = (\mathrm{o}^+, \mathrm{o}),\\
&\mathrm{rank}(\wedge^{\mathrm{p}}) = (\mathrm{p}^+, \mathrm{p}), && \mathrm{rank}(\wedge^{\mathrm{o}}) = (\mathrm{o}^+, \mathrm{o}),\\
&\mathrm{rank}(\mathrm{c}^{\mathrm{p}}) = (\mathrm{p}\,\mathrm{o}, \mathrm{p}), && \mathrm{rank}(\mathrm{c}^{\mathrm{o}}) = (\mathrm{o}\,\mathrm{p}, \mathrm{o}).
\end{aligned}$$

Given $F \in \mathcal{F}$ and $s \in \mathcal{S}$, we say that F is of type s, if $\mathrm{out}(F) = s$. The elements of $\mathbb{B}^{\mathrm{p}}$ and $\mathbb{B}^{\mathrm{o}}$ are typed constants, which represent basic actions of the proponent's and opponent's type, respectively. By $\mathbb{B}$ we denote the union $\mathbb{B}^{\mathrm{p}} \cup \mathbb{B}^{\mathrm{o}}$. The functions[4] $\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}, \vee^{\mathrm{o}}$, and $\wedge^{\mathrm{o}}$ represent disjunctive and conjunctive refinement operators for the proponent and the opponent, respectively. We set $\overline{\mathrm{p}} = \mathrm{o}$ and $\overline{\mathrm{o}} = \mathrm{p}$. The binary functions c^s, for $s \in \mathcal{S}$, represent countermeasures and are used to connect components of type s with components of the opposite type $\overline{s}$.

Definition 2. *Typed ground terms over the AD–signature* Σ *are called attack–defense terms (ADTerms). The set of all ADTerms is denoted by* $\mathbb{T}_\Sigma$*.*

For $s \in \{\mathrm{p}, \mathrm{o}\}$, we denote by $\mathbb{T}_\Sigma^s$ the set of all ADTerms with the head symbol of type s. We have $\mathbb{T}_\Sigma = \mathbb{T}_\Sigma^{\mathrm{p}} \cup \mathbb{T}_\Sigma^{\mathrm{o}}$. The elements of $\mathbb{T}_\Sigma^{\mathrm{p}}$ and $\mathbb{T}_\Sigma^{\mathrm{o}}$ are called *ADTerms of the proponent's* and *of the opponent's type*, respectively. The ADTerms of the proponent's type constitute formal representations of ADTrees.

[4] In fact, symbols $\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}, \vee^{\mathrm{o}}$, and $\wedge^{\mathrm{o}}$ represent unranked functions, i.e., they stand for families of functions $(\vee_k^{\mathrm{p}})_{k \in \mathbb{N}}, (\wedge_k^{\mathrm{p}})_{k \in \mathbb{N}}, (\vee_k^{\mathrm{o}})_{k \in \mathbb{N}}, (\wedge_k^{\mathrm{o}})_{k \in \mathbb{N}}$.

Example 6. Consider the ADTree given in Figure 1 (left). The corresponding ADTerm is $t = \vee^{\mathrm{p}}(\wedge^{\mathrm{p}}(\mathrm{IC}, \mathrm{UC}), \mathrm{SS}, \mathrm{c}^{\mathrm{p}}(\mathrm{OA}, \mathrm{FW}))$. The entire ADTerm, and all six subterms $\wedge^{\mathrm{p}}(\mathrm{IC}, \mathrm{UC})$, $\mathrm{c}^{\mathrm{p}}(\mathrm{OA}, \mathrm{FW})$, IC, UC, SS, and OA, are of the proponent's type. Term t also contains a subterm of the opponent's type, namely FW.

In order to facilitate and automate quantitative analysis of vulnerability scenarios, the notion of an attribute for ADTerms has been formalized in [14]. An attribute expresses a particular property, quality, or characteristic of a scenario, such as the minimal costs of an attack or the expected impact of a defensive measure. A specific bottom-up procedure for evaluation of attribute values on ADTerms ensures that the user, for instance a security expert, only needs to quantify the basic actions. From these, the value for the entire scenario is deduced automatically. Attributes are formally modeled using attribute domains.

Definition 3. *The tuple* $A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha, \vee^{\mathrm{o}}_\alpha, \wedge^{\mathrm{o}}_\alpha, \mathrm{c}^{\mathrm{p}}_\alpha, \mathrm{c}^{\mathrm{o}}_\alpha)$, *where* D_α *is a set of values and, for* $s \in \{\mathrm{p}, \mathrm{o}\}$, $\vee^s_\alpha$, $\wedge^s_\alpha$ *are unranked operations on* D_α, *and* c^s *are binary operations on* D_α, *is called an attribute domain for ADTerms.*

Let $A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha, \vee^{\mathrm{o}}_\alpha, \wedge^{\mathrm{o}}_\alpha, \mathrm{c}^{\mathrm{p}}_\alpha, \mathrm{c}^{\mathrm{o}}_\alpha)$ be an attribute domain for ADTerms. The bottom-up computation of attribute values on ADTerms is formalized as follows. First, a value from D_α is assigned to each basic action, with the help of function $\beta_\alpha \colon \mathbb{B} \to D_\alpha$, called a *basic assignment*. Then, a recursively defined function $\alpha \colon \mathbb{T}_\Sigma \to D_\alpha$ assigns a value to every ADTerm t, as follows

$$\alpha(t) = \begin{cases} \beta_\alpha(t), & \text{if } t \in \mathbb{B}, \\ \vee^s_\alpha(\alpha(t_1), \dots, \alpha(t_k)), & \text{if } t = \vee^s(t_1, \dots, t_k), \\ \wedge^s_\alpha(\alpha(t_1), \dots, \alpha(t_k)), & \text{if } t = \wedge^s(t_1, \dots, t_k), \\ \mathrm{c}^s_\alpha(\alpha(t_1), \alpha(t_2)), & \text{if } t = \mathrm{c}^s(t_1, t_2), \end{cases} \tag{1}$$

where $s \in \{\mathrm{p}, \mathrm{o}\}$ and $k > 0$. The example below illustrates the bottom-up procedure for an attribute called *satisfiability.*

Example 7. The question "Is the considered scenario satisfiable?" is formally modeled using the satisfiability attribute. The corresponding attribute domain is $A_{\mathrm{sat}} = (\{0,1\}, \vee, \wedge, \vee, \wedge, \star, \star)$, where $\star(x, y) = x \wedge \neg y$, for all $x, y \in \{0, 1\}$. The basic assignment $\beta_{\mathrm{sat}} \colon \mathbb{B} \to \{0, 1\}$ assigns the value 1 to every basic action which is satisfiable and the value 0 to every basic action which is not satisfiable. Using the recursive evaluation procedure defined by Equation (1), we evaluate the satisfiability attribute on the ADTerm from Example 6. Assuming that all basic actions are satisfied, i.e., that $\beta_{\mathrm{sat}}(X) = 1$ for $X \in \{\mathrm{IC}, \mathrm{UC}, \mathrm{SS}, \mathrm{OA}, \mathrm{FW}\}$, we obtain $\mathrm{sat}(\vee^{\mathrm{p}}(\wedge^{\mathrm{p}}(\mathrm{IC}, \mathrm{UC}), \mathrm{SS}, \mathrm{c}^{\mathrm{p}}(\mathrm{OA}, \mathrm{FW}))) = \vee(\wedge(\beta_{\mathrm{sat}}(\mathrm{IC}), \beta_{\mathrm{sat}}(\mathrm{UC})), \beta_{\mathrm{sat}}(\mathrm{SS}), \star(\beta_{\mathrm{sat}}(\mathrm{OA}), \beta_{\mathrm{sat}}(\mathrm{FW}))) = \vee(\wedge(1,1), 1, \star(1,1)) = \vee(1,1,0) = 1$.

The satisfiability attribute, as introduced in the previous example, allows us to define which player is the *winner* of the considered attack–defense scenario. If the satisfiability value calculated for an ADTerm is equal to 1, the winner of the corresponding scenario is the proponent, otherwise the winner is the opponent. In Example 7, the root attack is satisfied, so the winner is the attacker.

DNS Tunneling for Network Penetration

Daan Raman[1], Bjorn De Sutter[1], Bart Coppens[1], Stijn Volckaert[1], Koen De Bosschere[1], Pieter Danhieux[2], and Erik Van Buggenhout[2]

[1] Computer Systems Lab, Ghent University, Belgium
bjorn.desutter@elis.ugent.be
[2] Ernst & Young, ITRA FSO, Belgium

Abstract. Most networks are connected to the Internet through firewalls to block attacks from the outside and to limit communication initiated from the inside. Because of the limited, supposedly safe functionality of the Domain Name System protocol, its traffic is by and large neglected by firewalls. The resulting possibility for setting up information channels through DNS tunnels is already known, but all existing implementations require help from insiders to set up the tunnels. This paper presents a new Metasploit module for integrated penetration testing of DNS tunnels and uses that module to evaluate the potential of DNS tunnels as communication channels set up through standard, existing exploits and supporting many different command-and-control malware modules.

Keywords: domain name system, tunneling, Metasploit, network penetration.

1 Introduction

Private computer networks are under constant attack. Sometimes attackers try to setup so-called bind connections externally, with which they try to connect to listening sockets in the local network [20]. In other attacks infected local computers or malicious insiders try to set up reverse connections to an external computer [20]. The StuxNet command-and-control (C&C) malware was injected into a local network through a USB key, after which it set up connections to the home network to receive further commands and to leak private data [21]. Botnets are another well known type of C&C malware.

Two common methods to protect against such attacks are firewalls [1] and Intrusion Detection and Prevention Systems (IDPS) [8]. Firewalls permit or block network traffic based on protocols, blacklists and whitelists. IDPS are based on rule sets that describe suspicious network behavior. All network traffic is scanned by an IDPS, and as soon as a sensor picks up suspicious actions, the actions are interrupted and the network administrators are informed.

One commonly used network protocol is DNS, or Domain Name System [15,16]. The main functionality provided by DNS servers is to translate computer names such as www.icisc.org to the IPv4 address 164.125.70.63. This enables people and configuration files to rely on names instead of hard to remember addresses. Most often, a local network has one local DNS server. When some client in the local network needs an address translation, he queries the local DNS server with a so-called lookup request. The server knows which external DNS servers to access for information on external addresses in different domains, and forwards the queries. When the contacted external DNS server is authorized for the requested (domain) name, it returns the response to the query. If that server is not authorized, it returns a link to another DNS server, that is in turn queried by the local server. This iterative process continues until an authoritative server returns the final answer. An important property is that DNS servers can only respond to queries with appropriate responses. They can, in other words, not initiate any communication themselves, and the type of any response they provide must correspond to the type of the lookup.

On the one hand, the DNS protocol thus provides very limited services. For this reason, firewalls or IDPS in practice rarely check or filter DNS traffic originating from the local DNS server. On the other hand, the iterative nature of DNS can be used to hide connections with malicious DNS servers behind connections with the local server and with external, legitimate servers.

This potential for abuse has been exploited by so-called DNS tunnels [2,6,11,12], in which local computers exchange information with malicious DNS servers through a form of steganography. The local computer transmits information by embedding it in the domain names for which it queries the malicious DNS server, and the server transmits information by encoding it in the responses it returns, such as domain name aliases or textual descriptions of properties. To the best of our knowledge, all existing implementations require explicit insider cooperation to set up a DNS tunnel, and support only limited forms of communication.

In this paper, we present a staged attack we developed in the popular Metasploit Framework (MSF) for penetration testing [19]. This attack can leverage almost all known software vulnerabilities for which exploits are present in MSF. Furthermore, it supports the installation of C&C modules in MSF through a DNS tunnel, as well as tunneling those modules' traffic through the tunnel. With this proof-of-concept attack, we demonstrate for the first time that DNS tunnels are a threat even when all legitimate users of a network are benign.

In the remainder of the paper, Section 2 provides background information on DNS, DNS tunneling, and MSF. Section 3 discusses the different stages of our attack. Section 4 reports statistics on the traffic generated with our attack. Finally, Section 5 draws conclusions.

2 Background and Related Work

2.1 Domain Name System Tunneling

DNS servers store information about computer domain names and their addresses in so-called zone files. Type A records in those files specify (32-bit IPv4) address and name translations such as the fact that www.icisc.org corresponds to IPv4 address 164.125.70.63. AAAA records do the same for 128-bit IPv6 addresses. Canonical Name or CNAME records specify aliases, such as the icisc.org alias of www.icisc.org. MX or Mail eXchange records specify which servers to use as mail servers and what their priorities are. TXT records can specify a wide range of properties in the form of strings. Typically, these are used to specify constraints on the behavior of mail servers in the Sender Policy Framework [24]. Finally, NS records specify for which domain a DNS server is authorized.

Clients can issue lookup requests for any type of these records with one DNS packet, which is typically transmitted via the UDP protocol [18]. The response to a lookup can only consist of a packet with a record of the same type.

Besides header information, CNAME and TXT packets contain reasonably long strings corresponding to domain names. The other types of packets contain mainly short IP addresses or are so exotic that using them would be too suspicious. So only CNAME and TXT lookups and responses can be used to achieve a reasonable communication bandwidth through a DNS tunnel. For example, some piece of malware installed on a client computer can leak the password `qwerty123` to the malicious `nameserver.evildomain.com` DNS server by sending a TXT lookup for `passwd.qwerty123.evildomain.com`. Being the authoritative DNS server for `evildomain.com`, this DNS server is the last server in the iterative DNS lookup process to receive the lookup. Instead of treating the lookup as a real DNS lookup, this server extracts the communicated information `passwd.qwerty123` from the query. He then sends back information, such as a reboot command embedded in the TXT response `v=spf1 mx a:reboot.1pm.evildomain.com -all`, of which the legitimate meaning is that only local mail servers and the external host `reboot.1pm.evildomain.com` are allowed to send email from senders with email addresses from that same domain, such as `m.romney@evildomain.com`.

To use DNS servers and the DNS protocol as a covert, stealthy communication tunnel, the software implementing the tunnel should exhibit similar behavior as regular DNS traffic. Over ten periods of time, we recorded 10x500 MB of DNS traffic data on our department's local DNS server, which serves our administration and a wide range of research labs accros multiple campuses. We observed that in those periods, TXT records constituted 1–2% of all traffic, CNAME records constituted 20–30%, and A records constitute 38–48%. Furthermore, around 25% of the traffic constituted AAAA records (most of which in support of the Kerberos authentication protocol), and the remaining 5% was spent on NS records. This is in line with other experiments [25].

This implies that a stealthy DNS tunnel can use some TXT records, which can embed longer strings, but that it should mainly rely on CNAME records. This also implies that most if not all information communicated through a DNS tunnel must be encoded in the form of acceptable domain names. We will discuss the practical implications in Section 3.

2.2 Prior Implementations

DNS Cat by Ron Bowes consists of a server and a client application [6]. The client application needs to be invoked explicitly by a user with the domain name of the malicious DNS server (on which the server application runs). The client application is then attached to another process such as a shell, which is from then on controlled through the tunnel instead of locally. It obtains its standard input from the attacker at the server side through CNAME and TXT response packets sent through the DNS tunnel, and at the same time its standard output is redirected through the tunnel using similar corresponding lookup packets.

DNS Cat can be used in practice to communicate through a DNS tunnel. For example, in hotels with payed internet access, which typically do not filter or block DNS traffic, customers can get online through DNS Cat. The customer can do so because he has full control over a local computer such as his laptop on which he can launch DNS Cat with the appropriate settings. In other words, the customer is a malicious insider. Without help from insiders, however, attacking, e.g., company networks with DNS Cat is not possible.

tcp-over-dns also requires users to start a server-side and a client-side application [2]. Unlike DNS Cat, however, tcp-over-dns enables tunneling any TCP/IP connection through its DNS tunnel. This eases the tunneling of, e.g., browser sessions involving HTTP packets. tcp-over-dns is hence more user-friendly for hotel customers that want (to steal) free Internet access. It is, however, still not useful for attacking company networks without the help from insiders. Iodine [12] (formerly NSTX) and OzymanDNS [11] suffer from the same limitation.

Furthermore, all of these implementations run on the client side as a separate process, which makes them visible to any user or antivirus software.

2.3 DNS Anomaly Detection

Several techniques have been proposed in the past to detect anomalies in DNS traffic, including tunnels and symptoms of other computer network infections.

Some work focuses on relating DNS traffic to real-world events, such as the Tiananmen Square protests [27] or network defects [22]. Detection of network scanning worms has been based on their relatively low number of DNS requests [4,28]. This is obviously not applicable for DNS tunnels, which will operate precisely by executing numerous DNS requests. Fast-flux is a popular and relatively new cyber-criminal technique to hide and protect their critical systems behind an ever-changing network of compromised hosts acting as proxies. The ICANN Security and Stability Advisory Committee gives a clear explanation of the technique [10]. Jose Nazario and Thorsten Holz did some interesting measurements on known fast-flux domains [17]. Fast-flux techniques are mainly orthogonal to the actual launching of (DNS) tunnels and communication through them. The approach by Villamarin-Salomon and Brustoloni focuses on abnormally high or temporally concentrated query rates of dynamic DNS queries [26]. This does not suffice, however, since such patterns also occur for legitimate purposes. Choi et al. check for multiple botnet characteristics based on Dynamic DNS, fixed group activity and a mechanism for detecting migrating C&C servers [7]. They claim that their method works, but the computational demands and processing time seem to prevent it from scaling to large networks. Born and Gustafson propose to use character frequency and N-gram analysis for detecting covert channels in DNS traffic [5]. Their method detects patterns that do not occur in natural languages and are therefore considered anomalous. Master students van

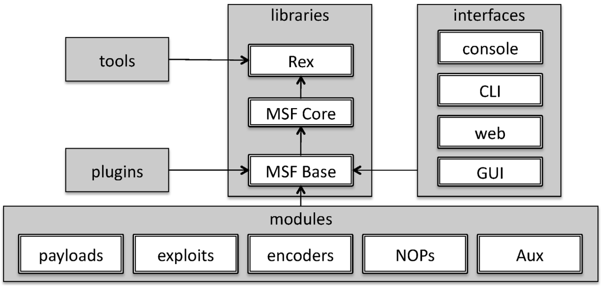


Fig. 1. Metasploit architecture

de Heide and Barendregt provide a preliminary evaluation of the aforementioned techniques without, however, measuring actual implementations [25].

To the best of our knowledge, no existing literature discusses how difficult it is to set up a DNS tunnel without the help of insiders.

2.4 Metasploit

In contrast with existing DNS tunnel implementations, we present an approach that enables outsiders to set up attacks over DNS tunnels without the willing help from insiders. We implemented this approach in MSF [19]. MSF is an open-source framework developed for security experts to ease penetration testing. It consists of a range of tools and modules programmed in Ruby, C and assembler that allow different components of attacks to be reused and combined in different ways. Fig. 1 depicts the MSF architecture. Different (server-side) interfaces are available in the form of a console, a command-line interface (CLI), a web-interface, and a GUI. The most important components for our purpose are the code fragments that will be executed on the client side:

Exploits are small code fragments that can exploit vulnerabilities to take over control of an attacked computer. These fragments can be embedded, e.g., in PDF documents to exploit PDF reader vulnerabilities.

Payloads are the code fragments that attackers want to execute once they have obtained a certain level of control over a machine. There are different types of payloads: *Stagers* try to set up a communication channel between the attacker and the victim as part of a bootstrap process, over which they load stages. *Stages* make up the actual C&C software that gets installed and launched by the stager, such as shell scanning accounts or a spam bot.

Encoders can convert the encoding of a payload without changing its functionality. They consist of packers and unpackers that encrypt and decrypt code to thwart antivirus software, but also of simpler transformations such as removing null bytes from code, to avoid that those bytes are interpreted as the end of a string when trying to exploit buffer overflow vulnerabilities.

NOPs are encoders that can increase the size of payloads to requested sizes by inserting no-operation instructions that do not change the behavior of the payload. This is useful for buffer overflow attacks on fixed sized buffers.

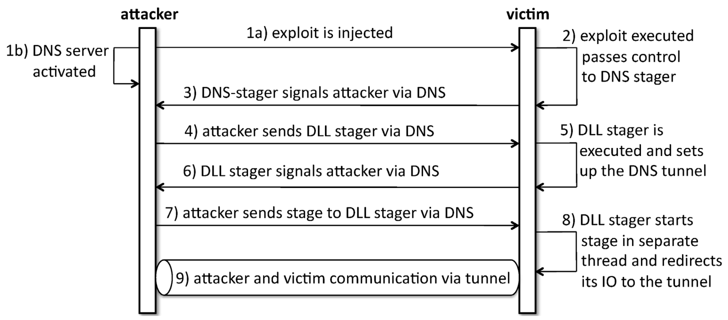

attacker
victim
1a) exploit is injected
1b) DNS server activated
2) exploit executed passes control to DNS stager
3) DNS-stager signals attacker via DNS
4) attacker sends DLL stager via DNS
5) DLL stager is executed and sets up the DNS tunnel
6) DLL stager signals attacker via DNS
7) attacker sends stage to DLL stager via DNS
8) DLL stager starts stage in separate thread and redirects its IO to the tunnel
9) attacker and victim communication via tunnel

Fig. 2. Bootstrap procedure of our attack

In March 2012, MSF support was added to transmit stages through a DNS tunnel with TXT records [3]. However, that implementation cannot bootstrap from existing, small exploits and does not provide a fully functional, bidirectional DNS tunnel to the stage for later communication of commands and data.

3 Our Staged Attack

This section presents a generic, two-stage DNS-tunneling attack that attackers or security researchers can set up by means of existing exploits of software vulnerabilities, such as buffer overflows, and that can serve as a hidden communication channel for any type of C&C malware. The first stager is very small to allow us to combine it with as many as possible existing MSF exploits. It installs a second stager which is much larger. This stager install a generic DNS tunnel and offers an interface for existing MSF C&C stages to the tunnel. Moreover, the software implementing the tunnel and the interface remain resident (but hidden!) on the attacked computer even when the originally exploited software, such as a PDF reader, is terminated. The source code of our MSF components is available at `https://github.com/azerton/metasploit-framework`. As for integration into MSF, it still needs polishing with respect to code guidelines.

As is common for local MSF exploits, we assume that we can inject a small exploit of a software vulnerability into the network under attack. This injection can happen through mail attachments, web sites infected with drive-by malware, phishing web sites, PDF documents on USB sticks, etc. Injecting this code fragment is orthogonal to the remainder of the attack, and is out of scope of this paper. In the remainder of this section, we discuss the bootstrap process that follows the code injection, as depicted in Fig. 2. We focus on the software we developed to run the computer under attack, i.e., the stagers, as the server side consists mostly of standard MSF functionality adapted slightly for our purpose.

3.1 Stage 1: DNS Stager

The DNS stager is the piece of code that will be injected in a process on the victim computer together with the exploit to hijack that process. The most

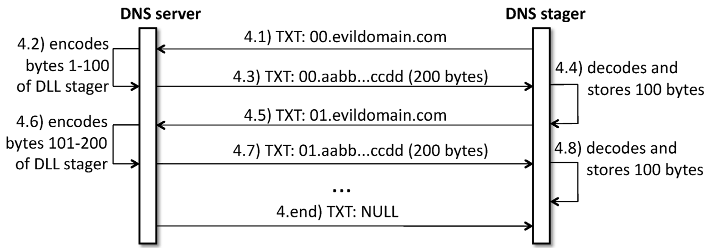

DNS server
DNS stager
4.1) TXT: 00.evildomain.com
4.2) encodes bytes 1-100 of DLL stager
4.3) TXT: 00.aabb...ccdd (200 bytes)
4.4) decodes and stores 100 bytes
4.5) TXT: 01.evildomain.com
4.6) encodes bytes 101-200 of DLL stager
4.7) TXT: 01.aabb...ccdd (200 bytes)
4.8) decodes and stores 100 bytes
...
4.end) TXT: NULL

Fig. 3. Step 4: Downloading of the DLL stager by the DNS stager

commonly used technique to inject code is by means of buffer overflows. At the time of writing, about 68% of all MSF exploits rely on buffer overflows.

Targeting buffer overflows constraints the stager. First, many targeted buffers are small. Hence the stager has to be made as small as possible to fit as many as possible buffer overflow exploits. Secondly, many buffer manipulation functions in software treat certain characters in a special way. For example, string copying ends on a null byte, and FTP-servers treat ampersands and newline characters differently. So some characters should preferably not occur in the stage. Finally, the stager's code has to be as platform-independent as possible, and directly executable at any address at which it is injected. In our case, we opted for position-independent x86 assembly code (PIC) that targets Windows systems.

Starting from Ron Bowes' DNS Cat stager implementation that was only suitable to stage short shell scripts encoded as ASCII text [6], we have written a stager consisting of 168 x86 assembler instructions that occupy 518 bytes. This stager is capable of coordinating steps 3) and 4) in Fig. 2.

Some additional features our stager supports are the transmission of binary data (e.g., the DLL stager's code) through TXT packets by means of NetBIOS coding and decoding, supports for ten times more TXT packets than DNS Cat to enable the transmission of the bigger DLL stager, and allocation of much more heap memory for storing large MSF payloads.

The actual transmission of the DLL stager under coordination of the DNS stager is depicted in Fig. 3. All transmitted packets are numbered to compensate that the UDP protocol does not guarantee delivery, ordering or duplicate protection. While CNAME records are stealthier as discussed above, we use TXT packets for this first stage of the attack because parsing TXT records is easier, and hence can be done with less code than parsing CNAME records. We have chosen to limit the packet data length to 200 bytes in our implementation, but this can easily be adapted. This is much shorter than the theoretical upper limit of 64K bytes per TXT packet because we observed that many software implementations in DNS servers cannot handle atypically long packets, and because IDPSs might consider atypically long DNS packets as suspicious. Switching to even shorter packets will increase the number of packets and hence the amount of time needed to transfer and install the DLL stager. This can be problematic for exploits that result in the process they hijacked being killed. If the process is killed before the DLL stager is up and running, the attack will fail.

Finally, we should note that the NetBIOS [14] coding we use is not the most efficient way to encode binary data in ASCII strings with respect to communication bandwidth. It converts each 4-bit nibble into an ASCII character by adding 0x41 to it. So it uses only 16 of the more than 26 lowercase + 26 uppercase + 10 digits + punctuation possible values. There are two reasons for using this encoding. NetBIOS decoding is simple enough to be supported in the very small stager of 168 instructions. Secondly, domain names are case-insensitive. For efficiency reasons, e.g., to save space in caches, DNS server software typically use normalized lower-case names. Hence upper-case characters risk not surviving the passage through DNS servers not controlled by the attacker.

3.2 Stage 2: DLL Stager

The DLL stager differs significantly from the DNS stager with respect to the way in which it is programmed, launched and executed. Whereas the first, DNS stager was programmed in PIC assembly to be injected and executed directly in the hijacked process, the second stager is too complex to be programmed in assembler. And whereas the DNS stager only has to provide a tunnel for itself, i.e., a tunnel that only supports loading the DLL stager, the DLL stager has to set up a more generic tunnel for use by the stages, keep that tunnel alive, and offer the stages an interface to it. For the latter two reasons, the DLL server needs to stay alive even after the originally hijacked process is killed. But it should stay alive in a way that is not easily detected, and hence not in a new, separate process. In other words, the DLL stager has to be injected into and operate in another, longer running process on the victim computer. The most convenient way to inject code into a running process and execute it, is by means of reflective DLLs [9,23]. Reflective DLL injection is a library injection technique in which the concept of reflective programming is employed to perform the loading of a library from memory into a host process. As such the library is responsible for loading itself by implementing a minimal Portable Executable (PE) file loader [13]. We implemented the second stager as such a reflective DLL, hence the name DLL stager.

When the 168 instruction DNS stager has downloaded the DLL stager via DNS, this DLL stager is launched (step 5 in Fig. 2). This stager is responsible for downloading one or more MSF stages as shown in Fig. 4. It will do so in a very similar way as the DNS stager, but there are some significant differences. First, CNAME packets will be used instead of TXT records to avoid detection. This implies that also in the responses, a considerable amount of space is spent on repeating domain names. Secondly, besides order numbers, the communicated domain names also includes session numbers, which allows an attacker to set up multiple concurrent sessions from within the same network. Furthermore, both the DNS server and the DLL stager add a third random number between 0 and 100 to the transmitted domain names to prevent the DNS servers from caching requests [6]. This enables the DLL stager to load multiple different stages if wanted. What remains of the 63 characters that are available in a CNAME packet is filled with useful information, such as the `GETPAYLOAD` and the NetBIOS string

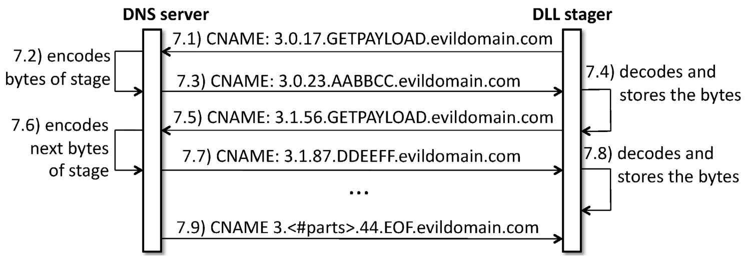

DNS server
DLL stager
7.1) CNAME: 3.0.17.GETPAYLOAD.evildomain.com
7.2) encodes bytes of stage
7.3) CNAME: 3.0.23.AABBCC.evildomain.com
7.4) decodes and stores the bytes
7.5) CNAME: 3.1.56.GETPAYLOAD.evildomain.com
7.6) encodes next bytes of stage
7.7) CNAME: 3.1.87.DDEEFF.evildomain.com
7.8) decodes and stores the bytes
...
7.9) CNAME 3.<#parts>.44.EOF.evildomain.com

Fig. 4. Step 7: Downloading of the stage by the DLL stager

`AABBCC` that represents stage code in Fig. 4. On the server side, MSF encoders automatically ensure that the transmitted stages (which over time may evolve independently from the DLL stager in the MSF) are in the least vulnerable format as chosen by and from the perspective of the attacker. This facilitates the integration and maintenance cost of our DNS and DLL stagers in the MSF.

When the DLL stager has downloaded a stage (step 7 in Fig. 2), it injects it as a new thread in the same application in which the DLL stager was injected itself, as chosen by the DNS stager. This injection is step 8 in Fig. 2. From that point on, the DLL stager becomes the bridge between the stage and the DNS tunnel. Internally, that phase of the DLL stager is designed as two cooperating components, as depicted in Fig. 5.

The DNS tunnel client is responsible for setting up and keeping alive the DNS tunnel. It implements the encoding and decoding of all information transmitted and received through CNAME packets, as was done for loading the stage(s). The component uses session numbers and packet order numbers to overcome reliability issues of the UDP protocol and to support multiple sessions. Furthermore, this component implements a form of polling through the DNS tunnel. In the DNS protocol, DNS servers can only respond to queries from clients. This means that a malicious DNS server cannot initiate any communication with a C&C client. Many such clients are designed to wait for commands from a server, however, without explicitly asking for such commands. In other words, stages in Fig. 5 might simply be sleeping and waiting to be woken up by a command. To allow the server to send commands that wake up sleeping clients, the DNS tunnel client sends lookups to the server of an initiated session on a regular basis. Whenever the server wants to send a command, it does so in a response to the polling lookup, which the DLL stager then forwards to the stage.

To facilitate the communication between client stages and servers without having to adapt the stages to the fact that they use a DNS tunnel, a second DLL stager component offers a TCP abstraction of the tunnel client. As a result, stages only have to communicate with socket pairs, which is a fairly standard method. To implement this component without having to implement all base functionality ourselves, we relied on the standard `winsock2` library.

As a whole, this stager consists of two parts: the reflective loader present only to install the stager, and the stager itself. Like the DNS stager, the reflective loader consist of manually engineered PIC (in this case written in heavily

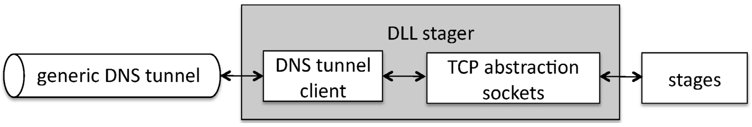


Fig. 5. Components of the DLL stager

constrained C code). Whenever this code wants to perform an API call, it first computes the address of the callee itself. These features are necessary because the reflective loader is invoked without its binary code getting relocated by the standard OS loader [13]. By avoiding the use of the standard OS loader, we also avoid the need to embed full PE headers in the binary and it allows us to put the DLL on the standard heap instead of on separate pages. Avoiding the standard loader, full headers and separate pages make the DLL stager much stealthier for antivirus software, which typically attaches itself to the standard loader to monitor the binaries being loaded.

In a first attempt, we implemented the second part, i.e., the DLL stager components, in C++. While this was very productive from a software-engineering perspective, the compiled DLL stager proved to be too big. This posed no technical problems, but it did increase the number of TXT packets that had to be transmitted to download the DLL stager, which increases the change of being detected. We therefore reimplemented the DLL stager in C, which resulted in an acceptable total binary size of 61KB.

4 Evaluation

To evaluate our implementation, we have set up our own subdomain DNS server with the free `www.afraid.org` service for static and dynamic DNS domain and subdomain hosting. This DNS server was the authoritative server for the azertontunnel.chickenkiller.com domain, a relatively long name that has to be included in each DNS lookup. We use the Google DNS server at 8.8.8.8 as a primary DNS server. This service is free and requires no authentication, which is perfect for an automated attack. In an alternative experiment, we installed a server and client locally, such that all information is sent directly from victim to server and back, without being delayed by external DNS severs. We refer to the first and second experiments as the global and local experiments respectively. In both experiments, the stage consisted of a small shell that runs C&C commands to obtain system information, of which the output is transmitted to the attacker through the DNS tunnel.

4.1 Throughput

First, we measured the maximal throughput we obtained with the described staged attacks, i.e., the amount of useful data an attacker can transmit. This excludes, e.g., the overhead bytes to embed the purposely long domain name `azertontunnel.chickenkiller.com` in the packets and the packet headers. To

Table 1. Observed throughputs

phase	DNS stager	DLL stager	shell
record type	TXT	CNAME	CNAME
data	DLL stager	shell	system info & polling
data size	61KB	0.23KB	40KB
local throughput	8.14 KB/s	0.80 KB/s	3.34 KB/s
global throughput	0.68 KB/s	0.68 KB/s	2.18 KB/s

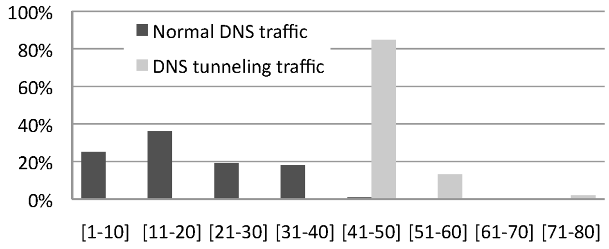


Fig. 6. Histogram of CNAME packet lengths

measure the throughout of the C&C shell, we ran an automated script on the attacker's side. In the evaluated implementation, the TXT records were limited to 200 bytes of useful information per DNS packet, and the CNAME records to 16 bytes, which we estimated to result in non-suspicious packets. Table 1 presents the throughput results.

The most important observation is that TXT records used by the DNS stager proved to give higher throughput only on the local network. It is unclear why the DNS stager in the global experiment is slowed down to the same level as the DLL stager. We suspect that it has to do with prioritization in DNS servers. As delays in A records and in CNAME records are typically more noticeable to clients, DNS servers might be handling those with higher priority. Further research is needed to clarify this. Even when we don't understand the cause of this behavior completely, we can conclude from this experiment that for global attacks, the DNS stager might as well use CNAME records. This will not slow the attack down, while at the same time making the attack more stealthy. As discussed in Section 3, it does increase the risk of attacks not succeeding, however.

Furthermore, we observe a shell C&C throughput of 2.18 KB/s, which is plenty for many real-world attacks, such as for stealing passwords, credit card numbers, and PINs by means of keyloggers.

In our experiments, the stage itself obtains a higher throughput than the DLL stager obtains while loading the stage. The reason is that the DLL stager's code that handles the stage's data handles this data more efficiently than the code in the DLL stager that downloads the stage itself.

4.2 Packet Sizes

With interactive C&C sessions, rather than automated attack scripts, we measured the DNS packet sizes to evaluate their stealthiness. The packet sizes,

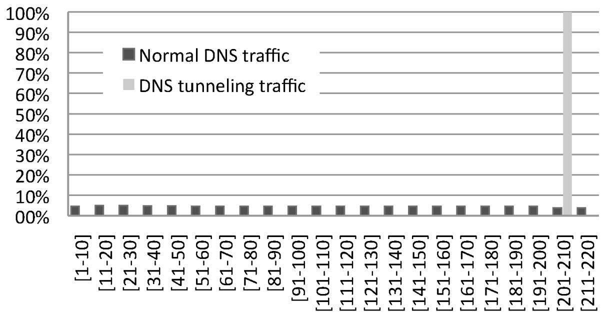


Fig. 7. Histogram of TXT packet lengths

including the overhead of domain names but not the fixed size headers, are depicted in Figs. 6 and 7.

It is clear that even with a limitation of 200 and 16 bytes per TXT and CNAME record, the packet lengths of tunneled traffic are easily distinguished from normal traffic. So IDPS could, in theory, easily detect and block our tunnel.

To prevent this, an attacker can in practice rely on CNAME packets only, which will in practice not lower his throughput as discussed above, limit the number of useful bytes per packet and use a shorter domain name than our purposely long `azertontunnel.chickenkiller.com`. He might then reach a lower throughput than 2.18 KB/s, but it will still be enough to obtain the most valuable, privacy-sensitive information.

5 Conclusions and Future Work

With the presented proof-of-concept Metasploit prototype, we have demonstrated that it is possible to set up fully functional DNS tunnels to private networks starting from small local exploits such as buffer overflows, i.e., without the willing help from insiders, and to use those tunnels for command-and-control attacks. This provides a strong incentive for firewalls and intrusion detection systems to start monitoring the often neglected DNS traffic. Our current implementation is probably not stealthy enough to avoid detection by adapted protection systems, but there seems to be plenty of room (i.e., bandwidth) to make it stealthier, so more research in this direction is needed in the future.

References

1. Amon, C., Shinder, T.W., Carasik-Henmi, A.: The Best Damn Firewall Book Period, 2nd edn. Syngress Publishing (2007)
2. AnalogBit: tcp-over-dns, `http://analogbit.com/software/tcp-over-dns`
3. Beardsley, T.: Weekly Metasploit Update: DNS payloads, Exploit-DB, and More. Rapid7 Blog Post (March 2012), `https://community.rapid7.com/community/metasploit/blog/2012/03/28/metasploit-update`

4. Binsalleeh, H., Youssef, A.: An implementation for a worm detection and mitigation system. In: Proc. 24th Biennial Symposium on Communications, pp. 54–57 (June 2008)
5. Born, K., Gustafson, D.: Detecting DNS tunnels using character frequency analysis. In: Proceedings of the 9th Annual Security Conference (April 2010)
6. Bowes, R.: DNS Cat, `http://www.skullsecurity.org/wiki/index.php/Dnscat`
7. Choi, H., Lee, H., Lee, H., Kim, H.: Botnet detection by monitoring group activities in DNS traffic. In: Proc. 7th IEEE Int. Conf. on Computer and Information Technology, pp. 715–720 (2007)
8. Di Pietro, R., Mancini, L.V.: Intrusion Detection Systems, 1st edn. Springer Publishing Company, Incorporated (2008)
9. Fewer, S.: Reflective DLL injection. Technical Report, Harmony Security (2008)
10. ICANN Security and Stability Advisory Committee: SSAC advisory on fast flux hosting and DNS (2008)
11. Kaminsky, D.: OzymanDNS, `http://en.cship.org/wiki/OzymanDNS`
12. Kryo: iodine, `http://code.kryo.se/iodine`
13. Levine, J.: Linkers & Loaders. Morgan Kaufmann Publishers (2000)
14. Microsoft Corporation: ASCII and hex representation of NetBIOS names, `http://support.microsoft.com/kb/194203`
15. Mockapetris, P.: RFC 1034 Domain Names - Concepts and Facilities. The Internet Engineering Task Force, Network Working Group (November 1987)
16. Mockapetris, P.: RFC 1035 Domain Names - Implementation and Specification. The Internet Engineering Task Force, Network Working Group (November 1987)
17. Nazario, J., Holz, T.: As the net churns: Fast-flux botnet observations. In: Proc. 3rd International Conference on Malicious and Unwanted Software, pp. 24–31 (October 2008)
18. Postel, J.: RFC 768 User Datagram Protocol. The Internet Engineering Task Force (August 1980)
19. Rapid7: Metasploit framework, `http://www.metasploit.com`
20. Rapid7: Metasploit pro user guide, `http://community.rapid7.com/docs/DOC-1501`
21. Rebane, J.C.: The Stuxnet Computer Worm and Industrial Control System Security. Nova Science Publishers, Inc., Commack (2011)
22. Shin, H.J.: A DNS anomaly detection and analysis system. NANOG 40 (June 2007)
23. "skape", Turkulainen, J.: Remote library injection. Technical Report, nologin (2004)
24. The SPF Council: Sender policy framework, `http://www.openspf.org/`
25. van der Heide, H., Barendregt, N.: DNS anomaly detection. Technical Report, Universiteit van Amsterdam (2011)
26. Villamarin-Salomon, R., Brustoloni, J.: Identifying botnets using anomaly detection techniques applied to DNS traffic. In: Proc. 5th IEEE Consumer Communications and Networking Conference, pp. 476–481 (January 2008)
27. Whang, Z., Tseng, S.S.: Anomaly detection of domain name system (DNS) query traffic at top level domain servers. Scientific Research and Essays 6(18), 3858–3872 (2011)
28. Whyte, D., Kranakis, E., van Oorschot, P.: DNS-based detection of scanning worms in an enterprise network. In: Proc. of the 12th Annual Network and Distributed System Security Symposium, pp. 181–195 (2005)

MeadDroid: Detecting Monetary Theft Attacks in Android by DVM Monitoring

Lingguang Lei[1,2,*], Yuewu Wang[1], Jiwu Jing[1], Zhongwen Zhang[1,2], and Xingjie Yu[1,2]

[1] State Key Laboratory of Information Security,
Institute of Information Engineering, CAS, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
{lglei,ywwang,jing,zwzhang,xjyu}@is.ac.cn

Abstract. Monetary theft attacks are one of the most popular attack forms towards Android system in recent years. In this paper, we present MeadDroid, a lightweight real-time detection system atop Android, to hold back this type of attacks. An FSM of monetary theft attacks is constructed, based on the analysis of real-world attacks. Employing an FSM-based detection approach, with the information obtained from dynamically monitoring the API calls and tracking the processing flow of UI (User Interface) inputs, MeadDroid can detect monetary theft attacks effectively and incurs only a small performance overhead. In addition, realized as an extension of Dalvik VM, MeadDroid is transparent to the user, and thus can provide a good user experience. Based on a prototype system, experiments are conducted with 195 popular Android applications. 11 applications with monetary theft attacks are found and the detection accuracy is almost 100% through comparing the results with the charge bill of the phone number used in the experiments. The performance overhead on a CPU-bound micro-benchmark is 8.97%. Experimental results demonstrate that MeadDroid has good performance in terms of effectiveness and efficiency.

Keywords: Monetary Theft Attack, DVM, Android, API Calls Monitoring, FSM.

1 Introduction

With the prevalence of Android applications [1] and loose management of Android Market, Android has become the preferred target of malicious codes attacks [4]. Among all these attacks, monetary theft attacks are one of the most popular forms [3]. The target of monetary theft attacks is subtracting extra fees from users' accounts stealthily. In a typical monetary theft attack, the malicious codes usually subscribe to or consume many uncalled-for paid services without notifying the users. Thus, a large amount of monetary losses to users are caused furtively, and the adversaries can get

[*] This work was supported by National Natural Science Foundation of China (Grant No. 70890084/ G021102, 61003274 and 61003273) and Strategic Priority Research Program of Chinese Academy of Sciences under Grant XDA06010702.

great monetary benefits from the service fees. Because of the generous returns, monetary theft attacks have attracted more and more attackers. For example, on April 8, 2012, China's national television CCTV has reported monetary theft attacks of handset with a special news program [2]. Especially, a malicious code mentioned in the report, called 'ShiRenYu', subtracts 50 Million Yuan per year from the users' accounts stealthily. Several other classic monetary theft attacks in Android are shown in [15].

Current research mainly focuses on two fields as follows: i) Android security model analysis, such as the permission-based security model and code signature mechanism analysis [6-11]; ii) privacy protection [12, 13]. However, the current Android permission and code signature mechanisms cannot hold back monetary theft attacks effectively. The privacy protection technology is also not suitable for monetary theft attack defense. There is also some work on detecting malicious applications on Android OS [19-24]. However, these methods are usually time-consuming and are not suitable to be deployed on the resource-constrained mobile platforms. In order to hold back monetary theft attacks timely and effectively, a real-time and lightweight attack detection method is needed.

In this paper, a lightweight monetary theft attack detection system called MeadDroid (Monetary Theft Attack Detection System) is presented. The realization of MeadDroid faces three challenges.

Firstly, MeadDroid should be able to detect monetary theft attacks effectively and with low overhead. We achieve this by two findings that monetary theft attacks share some common behavior patterns which can be described as an FSM (Finite State Machine), and these behavior patterns can be detected through DVM (Dalvik Virtual Machine) monitoring. An FSM based attack detection approach is introduced in MeadDroid. As a behavior-based detection system, MeadDroid is more efficient than the signature-based systems. A typical monetary theft attack is usually launched through SMS (Short Message Service)-related operations, but with abnormal behavior patterns. In Android, SMS-related operations are conducted by calling Java APIs, and Java APIs are all interpreted and executed in DVM [18]. Thus, we can detect the malicious API calls entirely by DVM monitoring. Meanwhile, as SMS-related APIs are only a small portion of APIs, dynamically monitoring in MeadDroid incurs low overhead.

Secondly, monetary theft attacks are always launched stealthily. MeadDroid must be able to detect this stealth effectively. The stealth of a monetary theft attack can be obtained by two features: the SMS contents being not inputted by the user or the SMS sending operations being not initiated manually through UI operations. Two tagging technologies are introduced in MeadDroid to detect these two features respectively.

Finally, the attack detection process should be transparent to the user, in order for good user experience. Being implemented as an extension of DVM, MeadDroid cannot be felt by the user, and can be compatible with Android applications well.

The main contributions of this paper are summarized as follows.

1. An FSM is designed according to the malicious behavior patterns extracted from real-world attacks, and is used as the foundation in monetary theft attack detection.

2. A lightweight real-time dynamic monetary theft attack detection system, MeadDroid, is implemented. Realized as an extension of DVM, MeadDroid can detect the attack behaviors effectively, while the overhead to Android system incurred by the detection system is negligible.
3. Experiments are conducted to evaluate the effectiveness and efficiency of the system with 195 Android Applications. Experimental results demonstrate that the performance of MeadDroid in effectiveness and efficiency is very good.

The rest of this paper is organized as follows: Section 2 describes the overall architecture of MeadDroid in detail, Section 3 describes the implementation of MeadDroid, Section 4 evaluates the performance of MeadDroid with a series of experiments, Section 5 describes the related work, and Section 6 concludes this paper with a brief summary and an outline of future work.

2 The Design of MeadDroid

We seek to design a lightweight detection system that can detect monetary theft attacks effectively. So that it can be deployed on the resource constrained mobile platforms in real-time. In this Section, we discuss the design of MeadDroid.

In our discussion, we assume that malwares always launch attacks from the application level, rather than embed the malicious codes in the DVM or Linux Kernel. Meanwhile, we assume that the SMS-related operations are all achieved by calling Java APIs provided in Android, for realization of an SMS stack in Android is complicated. Therefore, the attack behaviors can be detected entirely in DVM on the hypothesis that the kernel and DVM are secure.

2.1 The Principles of MeadDroid Design

Three principles are considered during the design of MeadDroid:

1. *Lightweight*. Smartphone platforms are usually resource-constrained. Thus, the overhead of MeadDroid should be minimal to make it feasible for real-world deployment. Furthermore, MeadDroid is an extension of DVM, in which all applications are interpreted and executed. High overhead will affect all of the applications.
2. *Completeness & accuracy*. The traces of monetary theft attacks are complex. The number of APIs involved in monetary theft attacks is 25, and the SMS-related operations can be completed by calling APIs directly or using Intents indirectly. Therefore, the monitoring system must cover all available traces of the attacks, so that, MeadDroid can get enough information to identify the malicious behaviors of the attacks accurately.
3. *Good User Experience*. MeadDroid should preferably be transparent to App level of Android. So, the user operations on the Android with MeadDroid deployment will be the same as that on normal Android. Meanwhile, the Apps need not any modification to adapt to the MeadDroid, and thus the feasibility of deployment of MeadDroid can be enhanced significantly.

2.2 The Architecture of MeadDroid

In order to satisfy the three design principles, MeadDroid is implemented as an extension of DVM and composed of three function modules: App behaviors monitoring, malicious behaviors detecting and responding modules. The technical idea of MeadDroid is monitoring the operations in an application dynamically and detecting the malicious behaviors based on FSM detection. The App behaviors monitoring module can obtain enough dynamic information during the execution of an application, and provide it to the malicious behaviors detecting module. Then, the malicious behaviors included in an application can be detected by the detecting module based on FSM detection. If an attack is identified, necessary corresponding actions can be taken by responding module to hold back the attack. The dynamic information of an application execution includes the API calls, and the processing flow of UI inputs. Both the monitoring of API calls and tracking of UI inputs can be achieved in DVM. So, the detection of MeadDroid is Completeness & accuracy. The FSM based detection ensures that MeadDroid is lightweight. Finally, the DVM level implementation makes MeadDroid transparent to App level.

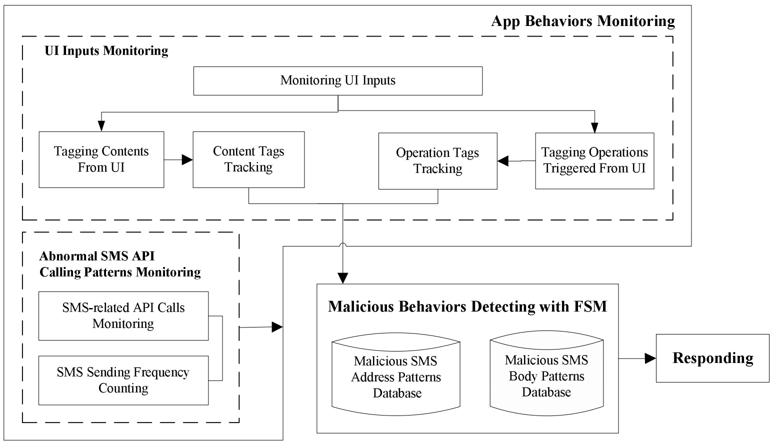

App Behaviors Monitoring
UI Inputs Monitoring
Monitoring UI Inputs
Tagging Contents From UI
Content Tags Tracking
Operation Tags Tracking
Tagging Operations Triggered From UI
Abnormal SMS API Calling Patterns Monitoring
SMS-related API Calls Monitoring
SMS Sending Frequency Counting
Malicious Behaviors Detecting with FSM
Malicious SMS Address Patterns Database
Malicious SMS Body Patterns Database
Responding

Fig. 1. The architecture of MeadDroid

The architecture of MeadDroid is shown in **Fig. 1**. The core of the architecture is the Malicious Behaviors Detecting module. This module is implemented based on an FSM of monetary theft attacks. The details of the FSM will be described later. With the information provided by the two monitoring sub modules and two databases on SMS-related patterns, FSM will be transited from one state to one state. If a malicious state is reached, the Responding module located at the right of **Fig. 1** will be initiated to hold back the attack. Because the previous comparison information can be reused fully, the efficiency of FSM based detection is higher than the signature based detection.

UI inputs monitoring sub module is used to judge the SMS-related operations are initiated by the user or not and the contents of the SMS messages are inputted by the user or not. A three-level operation tagging and tracking technology is employed in

MeadDroid to track the operations triggered by Keyboard & Touch-screen operations. Meanwhile, a data tagging and tracking technology is introduced to track the contents inputted through UI. The implementation of these two tagging technologies will be described in Section 3.3. This sub module can provide enough accurate information to the malicious behaviors detecting module to distinguish the origin of an SMS sending operation.

Abnormal SMS API calling patterns monitoring sub module is used to monitor the SMS-related API calls in an Android application. The API calling patterns, such as the frequency and the sequence of the API calls, are the important foundation of the attack detection. All API calls can be monitored by this module. However, in order to keep MeadDroid lightweight, only the information on SMS-related API calls is provided to the core detecting module.

Based on the data provided by the above 2 monitoring modules, we can detect stealthy SMS-related API calls in an application. Then, the malicious SMS Body and Address databases are used to detect whether the SMS messages processed in the APIs are abnormal. The content and address of an SMS message can be gotten by parsing the parameters of the SMS-related APIs, the details will be described in Section 3.2. Through comparing the content and address of an SMS message with the Malicious SMS databases, a questionable SMS message can be identified.

2.3 FSM of Monetary Theft Attacks

FSM is the core of the MeadDroid. In this Section, the details of the FSM design will be described.

Two malicious operations are involved in a monetary theft attack. One is SMS sending, the other is SMS hijacking.

As for SMS sending, one can send an SMS message by calling the APIs or by sending a specific Intent to start corresponding activity to complete the message sending. The later can be detected by monitoring the APIs named '*StartActivity()*' and '*startActivityForResult()*'. Therefore, all of the SMS sending operations can be detected with API calls monitoring. Both the normal and malicious SMS sending operations are implemented by APIs calling. In order to distinguish the malicious SMS sending operations from normal SMS sending operations, the behavior patterns of the SMS sending operations should be extracted, which include UI tags, SMS sending frequency, SMS body and address. A malicious SMS sending pattern is a path from initial state to the malicious state in FSM.

As for messages hijacking, one can hijack the incoming messages in two ways: hijacks messages with '*BroadcastReceiver*', and hijacks messages with '*ContentObserver*' and '*ContentResolver*'. In the first way, one can register a '*BroadcastReceiver*' in the Android application to receive SMS messages. By setting the priority of the intentfilter of '*BroadcastReceiver*' to the highest, this application can receive the SMS messages firstly. When receiving the messages, the '*onReceive()*' method of the '*BroadcastReceiver*' will be invoked. Then the attacker can check the bodies and addresses of the messages, and then discard the specific messages by calling '*abortBroadcast ()*'.

In the second way, the attacker can register a '*ContentObserver*' to monitor the changing of the SMS database in Android. When the SMS database is changing, such as a new incoming message is inserted, the '*onchange()*' method of the '*ContentObserver*' will be invoked. In the '*onchange()*' method, the attacker can delete the SMS messages from the database with the '*ContentResolver.delete()*' method or mark them as read-done with the '*update()*' method, before they are pushed to the user.

According to the processes of malicious operations involved in monetary theft attacks, the FSM is designed, as illustrated in **Fig. 2**.

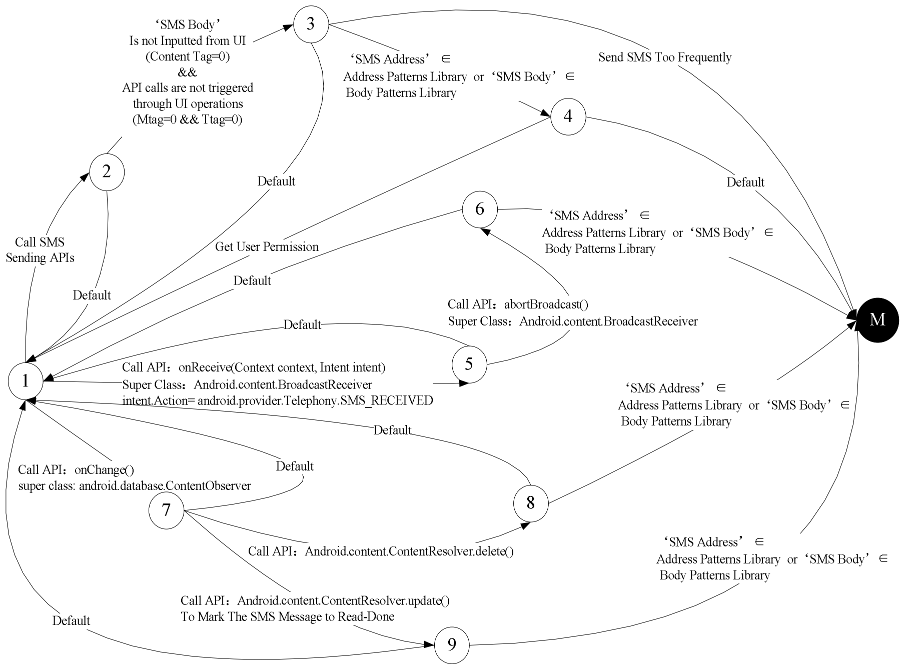

'SMS Body' Is not Inputted from UI (Content Tag=0) && API calls are not triggered through UI operations (Mtag=0 && Ttag=0)
'SMS Address' ∈ Address Patterns Library or 'SMS Body' ∈ Body Patterns Library
Send SMS Too Frequently
Default
Call SMS Sending APIs
Get User Permission
Call API：abortBroadcast() Super Class：Android.content.BroadcastReceiver
Call API：onReceive(Context context, Intent intent) Super Class：Android.content.BroadcastReceiver intent.Action= android.provider.Telephony.SMS_RECEIVED
Call API：onChange() super class: android.database.ContentObserver
Call API：Android.content.ContentResolver.delete()
Call API：Android.content.ContentResolver.update() To Mark The SMS Message to Read-Done
1
2
3
4
5
6
7
8
9
M

Fig. 2. FSM of Monetary Theft Attacks

The initial state of the FSM is 1. When an SMS sending API call is detected, the state of the FSM will be transited to 2. If the SMS body isn't inputted from UI (which means the Tag of the SMS content is 0), and the calling of SMS sending APIs isn't triggered by UI actions (which means the Mtag of the SMS sending API and the Ttag of the thread sending SMS are both 0, the meanings of Mtag and Ttag are described in Section 3.3), the state will be transited to 3 further. Otherwise, the SMS sending API call will be permitted and the state will be backed to 1. If the state 3 is reached, the SMS sending frequency is checked, and the SMS body and address are parsed out. If the SMS sending frequency is too high, the state will be transited to M, i.e. the process is infected with monetary theft attack codes. If the SMS body or address is included in the pattern databases, the state will be transited to 4. If the above two conditions are not satisfied, the state will be also backed to 1. When the state 4 is reached, an alarm will be popped to the user. If the permission is gotten from the user, the SMS sending API call is permitted as that in state 2. Otherwise, the state will be transited to M.

If the method '*onReceive()*' or '*onChange()*' is called, as shown in **Fig. 2**, the state of the FSM will be transited to 5 or 7. If the method '*abortBroadcast()*' is called in state 5, or the method '*delete()*' or '*update()*' is called, an SMS receiving hijack may happen. With the SMS body and address parsing, the SMS receiving hijack can be proved further. When an SMS receiving hijack happens, the state will be transited to M and an attack is detected. If no abnormal actions happen, all states in the SMS receiving hijack detection path will back to initial state 1 by default.

3 System Implementation

The architecture of MeadDroid is illustrated in **Fig. 1**, and implementing this architecture needs to address several system challenges, including: a) monitoring points selecting, b) API calls detecting and parameters parsing, c) UI inputs tagging and tracking. Solutions to these challenges should guarantee that MeadDroid is lightweight enough to be deployed on the mobile platforms. The remainder of this section describes our solutions to these challenges.

3.1 Monitoring Points Selecting

As described in Section 2.2, there are two ways to call a Java API in Android, by calling directly from Java codes and by calling from native codes via JNI. In both ways, the methods are pushed into the interpreted stack before executed. Thus, the pushing operation of the interpreted stack is a suitable point to monitor the Java API calls of an application. In fact, the pushing operation of the interpreted stack is implemented in two points. For the API calls in Java codes, the pushing-operation codes are included in the hardware-related part of Dalvik, i.e. the file 'Android\dalvik\vm\mterp\out\InterpC-portstd.c'. As for the API calls from native codes, the pushing operation is complete in the common part of Dalvik, i.e. the file 'Android\dalvik\vm\Interp\Stack.c'. Thus, the monitoring codes are added to these two points to cover all pushing operations of the interpreted stack completely.

In order to keep lightweight, only the API calls are monitored instead of all Java codes in the application. At the same time, only the API calls related to SMS operations are checked with the FSM introduced in Section 2.3.

3.2 API Calls Detecting and Parameters Parsing

There is a hierarchical relationship among the SMS-related APIs. For example, when the method '*sendTextMessage*()' is called, the method '*sendText*()' will be called certainly to complete the SMS sending operation. According to the hierarchical relationship among these SMS-related API calls, only the low-level API calls are processed in MeadDroid. Thus, the overhead of MeadDroid can be reduced further, while without hurting the detection accuracy.

Every time a method is called, a method frame is pushed into the interpreted stack. The method name and parameters are all included in the method frame. By parsing the method frame, the parameters of the method are gotten. As for parameters of the class data type, only the references are stored in the interpreted stack and the real instances are stored in the DVM heap. In this case, the reference tables are used to access the instances to get details of the parameters. However, this process is time-consuming. Therefore, in our system, only the necessary parameters of the SMS-related APIs are parsed to keep lightweight.

3.3 UI Inputs Tagging and Tracking

An event handle function will be triggered in a real UI operation. Thus, in a normal SMS sending operations initiated by the user, we can detect that obtaining of the SMS body or calling of the SMS sending APIs are triggered in the event handle function.

In MeadDroid, we firstly detect the UI operations by monitoring Linux Input Subsystem; then we introduce two tagging and tracking technologies to track the contents inputted and API calls triggered in the event handle function.

UI Operations Detecting

Detecting of the UI operations composed of following two steps:

1. Monitor the writing operations to '/dev/input/' in Linux kernel. The UI input event is first written into '/dev/input/' by Linux kernel. On the hypothesis that the kernel is secure, this operation cannot be forged by the attackers.
2. Detect the event handle functions triggered by the input events. The processing flow from the input events recorded by the Linux kernel to the handle functions triggered involves multiple processes, and thoroughly tracking this flow is time-consuming. Through hundreds of times experiment, we found that the time span between the above two operations is less than 5ms, while the time span between either two UI operations is more than 15ms. Therefore, it is possible and convenient to use the time information in the event to determine whether an event handle function is triggered by the specified input action. And this approach is employed in our MeadDroid system.

Data Tagging and Tracking

For tracking of SMS content, we use a data tagging approach. After inputted by the user, the SMS content is stored into a data structure. We add a special tag to the data structure, and propagate it through normal SMS sending processes in Android. Finally, when the inputted content is processed by SMS sending APIs as a parameter, we check the tag to determine whether it is originated from the user manipulation. This tagging technology just needs to track one kind of data, and a small fragment of the processing flow of the data. Therefore, comparing with TaintDroid, our tagging technology is more lightweight.

Operations Tagging and Tracking

For tracking the operations triggered in the event handle functions, we introduce a three level tagging technology. In a normal SMS sending process, the SMS sending operation will be triggered through one of the following three ways: 1) directly calling the SMS sending APIs in the handle functions; 2) creating a new thread to launch the SMS sending operations; 3) sending a message to tell an existing thread to call the SMS sending APIs. As shown in **Fig. 3**, we introduce three tags to mark for the above three scenarios, which are 'Mtag', 'Ttag' and 'Msgtag'. We add an 'Mtag' tag to each method in the interpreted stack to indicated whether it is triggered by the event handle functions. The 'Mtag' of a method is set to 1, if the method is an event handle function or 'Mtag' of the calling method is 1. In the interpreted stack, all methods in the position between pushing and popping of one method are called by the method. We add a 'Ttag' to each thread, threads created in one method will have the 'Ttag' being set to 1, if the 'Mtag' of the method is 1, otherwise the 'Ttag' will be set to 0. We add an 'Msgtag' to the message, and all messages sent in the method with 'Mtag' 1 or in the thread with 'Ttag' 1, will have the 'Msgtag' being set to 1. Then, we will check the message's 'Msgtag' when the message handle function is called, and set the function's 'Mtag' to 1 if the message's 'Msgtag' tag is 1. With this tagging approach, we can track all the operations triggered by the event handle functions, either with 'Mtag' being set to 1, or with 'Ttag' being set to 1. On the contrary, if an action, including the SMS sending action, is launched stealthily, both the 'Mtag' of the method and the 'Ttag' of the thread calling the method will have value 0.

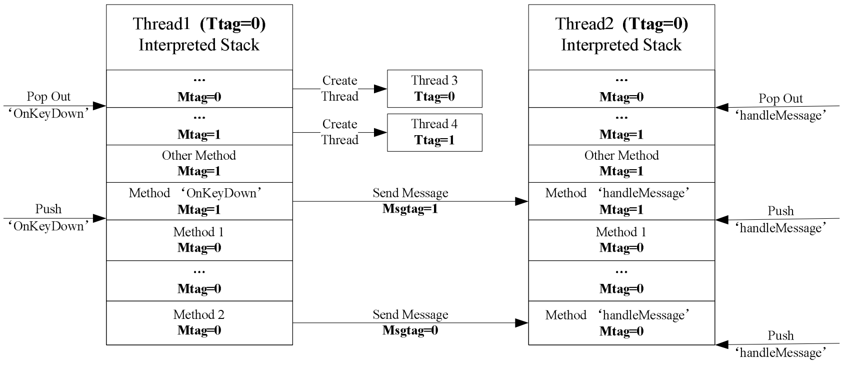


Fig. 3. Tagging and Tracking the Operations Triggered in the Event Handle Functions

With above technologies, we can determine whether an SMS-related operation is initiated by the user, on the hypothesis that the kernel is secure.

4 Evaluation

Experiments are conducted to evaluate the effectiveness and efficiency of the system, on a Google Nexus S smart-phone running Android OS Version 2.3.4.

The experimental results demonstrate that MeadDroid can effectively detect monetary theft attacks in Android with reasonable overhead.

4.1 Experiment Sample Set Generation

We downloaded the most popular free applications of 12 categories from the AppChina [25] in August 2012 to generate the experiment sample set. In order to launch the SMS-related operations in an application, SMS-related permissions must be applied for in an application and declared in its manifest.xml. To make the sample set more effective, we check the manifest.xml of each application, and screen out the applications without SMS-related permissions. 1200 applications are downloaded, and 195 (16.25%) of them have applied for the SMS-related permissions, which are included in the sample set.

4.2 Effectiveness Evaluation

Conducting experiments for 195 applications manually is time-consuming. Thus, a script-based tool is constructed to conduct the experiment process automatically. The atomic tool is composed of three parts: i) installing the applications with "adb install" command; ii) running the applications automatically with Android monkey event generator; iii) uninstalling the applications with "adb uninstall" command. The total time to execute these applications is about two weeks.

11 applications with attack behaviors are found in the experiments. 7 of them just initiate SMS sending operations, and send an SMS message every time they are launched. 3 of them contain SMS sending and hijacking behaviors. Especially, one application starts a background service to send SMS messages to '12114' periodically after it is fired. All the SMS bodies of these SMS operations are not inputted from UI, and all SMS operations are initiated stealthily. 1 application is detected to contain SMS hijacking behaviors only. This application checks all of the incoming SMS messages and aborts the messages from '10****'.

In order to evaluate the detection accuracy, we modify MeadDroid to just log the malicious behaviors detected rather than hold back them. Then, we compare the detection results with the charge bill from communication service provider. The comparing results show that the malicious SMS sending operations detected are consistent with the charge bill of the phone number used in the experiments, which means MeadDroid achieves almost 100% detection accuracy and low false positive rates in protecting the applications from monetary theft attacks.

Like most dynamic detection technology, MeadDroid may not cover the entire attack paths completely. However, our system can detect and hold back the attack behaviors effectively, whenever these behaviors are executed really.

4.3 Efficiency Evaluation

Efficiency is another factor that should be paid enough attention, because MeadDroid is a solution running on resource-limited platforms. Experiments are conducted to evaluate the efficiency of MeadDroid.

MeadDroid is implemented as an extension of Dalvik VM. Therefore, we evaluate the efficiency by comparing it with normal Android Dalvik VM. Java Microbenchmark is a classic metric used to evaluate the performance of Java VM. CaffeineMark 3.0 for Android [26] is adopted to generate the scores of Java Microbenchmark. CaffeineMark uses an internal scoring metric only useful for relative comparisons. The experimental results are shown in **Fig. 4**.

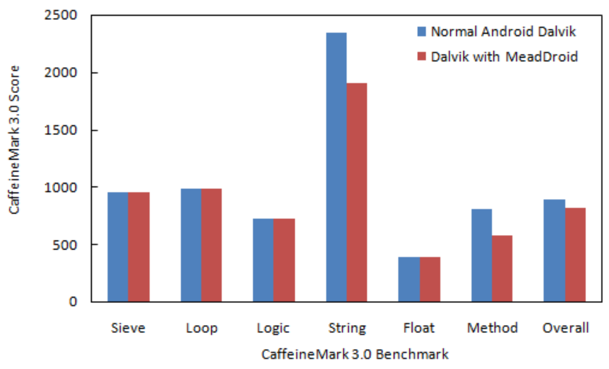


Fig. 4. Microbenchmark of the overhead on Normal Dalvik and Dalvik with MeadDroid

The results are consistent with design expectations. The overhead incurred by MeadDroid is almost zero for the benchmarks dominated by arithmetic and logic operations. The string benchmark experiences more overhead, which is due to the memory comparisons that occur in the method name and parameters detection. Because API calls monitoring is the most important task in MeadDroid, the method benchmark experiences the greatest overhead.

The "overall" benchmark indicates cumulative score across all individual benchmarks. CaffeineMark documentation indicates that the score of "overall" benchmark roughly corresponds to the number of Java instructions executed per second. Here, the normal Android system has an average score of 891, while the score of MeadDroid is 811. MeadDroid has an 8.97% overhead with respect to the normal system, which is a reasonable efficiency.

5 Related Work

Malicious detection on Android has attracted considerable attention since 2008.

Some researchers provide permission based methods to detect the malicious behaviors in the Android applications [20, 28]. These methods deduced the behaviors of an application according to the permissions requested by the application. In [20], Enck et al. proposes a lightweight malicious detection system Kirin. It reads application permission requirements during installation and checks them against a black list of malicious patterns that is in the form of permission combinations. However, this installation inspection can be bypassed easily. Moreover, accuracy of this method is not very perfect, because many applications often request uncalled-for permissions and the permission mechanism can be bypassed in some case.

Some other work [5-8] analyzes the applications by statically analyzing Android source codes, Dalvik bytecodes or Java bytecodes converted from Dalvik bytecodes. These methods usually target for overall security analysis of Android applications, and can't handle the codes being confused or packed and malicious attacks based on time or event-triggered. The detection accuracy of these methods is largely affected when using Java reflection in the codes, and they are usually large time and resource consuming. Therefore, these static analysis approaches are not adapted to real-time monetary theft attack detection.

A few dynamic methods [12, 13] are proposed for malicious detection. TaintDroid [13] uses system-wide dynamic taint tracking to identify privacy leaks in Android applications. [12] is a further work based on TaintDroid, and also targets for privacy disclosure detection. The proposed system, MeadDroid, is also a dynamic detection system, but it is different from TaintDroid in the following aspects: 1) MeadDroid targets for monetary theft attacks and can hold back this type of attacks effectively in real-time, while TaintDroid targets for privacy protection and only tracks the private data in system without taking any measure to hold back privacy disclosure. 2) MeadDroid is a behavior-based detection system, and it detects the attacks through dynamically monitoring the behaviors of an application execution. TaintDroid is a data tracking system, which only focuses on the private data. 3) MeadDroid is a light-weight scheme and can be deployed easily on smart-phone platforms.

There are also some methods based on monitoring events occurring on Linux-kernel level [19, 22, 23]. These methods are often resource expensive and are not suitable for resource-constrained environment.

6 Conclusion

Monetary theft attacks are one of the most popular attacks in Android in the recent period. To address this, we present MeadDroid, an efficient and effective system, to detect monetary theft attacks. A key design goal of MeadDroid is efficiency. MeadDroid achieves the high efficiency by dynamically monitoring the API calls of an application and comparing them with a monetary-theft-attack FSM. We also use our MeadDroid implementation to study the behaviors of 195 third-party applications with SMS-related permissions. The efficiency and effectiveness of MeadDroid are validated with experiments, and the experimental results demonstrate that MeadDroid possesses a good performance at efficiency and effectiveness.

Many other SMS-based attack forms are emerging, such as privacy information stealing. In our future work, we will extend MeadDroid to other SMS-related attack detection. In addition, an attack may be implemented on native codes level completely and this attack cannot be detected by our current scheme. Although the native level attacks are difficult and very rare currently, a Linux kernel based scheme with the same technology idea has been put on our agenda.

References

1. http://searchenginewatch.com/article/2155122/Android-Market-vs.-App-Store-Prices-Why-Android-Users-Pay-Double-Study
2. http://news.cntv.cn/china/20120408/110689.shtml
3. Porter Felt, A., Finifter, M., Chin, E., Hanna, S., Wagner, D.: A Survey of Mobile Malware In The Wild. In: Proceedings of the 1st Workshop on Security and Privacy in Smartphones and Mobile Devices, CCS-SPSM 2011 (2011)
4. http://techcrunch.com/2011/11/20/mcafee-nearly-all-new-mobile-malware-in-q3-targeted-at-Android-phones-up-37-percent/
5. Chin, E., Felt, A.P., Greenwood, K., Wagner, D.: Analyzing Inter-Application Communication in Android. In: Proceedings of the 9th Annual Symposium on Network and Distributed System Security, MobiSys 2011 (2011)
6. Fuchs, A., Chaudhuri, A., Foster, J.: SCanDroid: Automated Security Certification of Android Applications., http://www.cs.umd.edu/avik/projects/scAndroidascaa
7. Enck, W., Octeau, D., McDaniel, P., Chaudhuri, S.: A Study of Android Application Security. In: Proceedings of the 20th USENIX Security Symposium, USENIX Security 2011 (2011)
8. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android Permissions Demystified. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011 (October 2011)
9. Shin, W., Kiyomoto, S., Fukushima, K., Tanaka, T.: Towards formal analysis of the permission-based security model for Android. In: Proceedings of the 2009 Fifth International Conferenceon Wireless and Mobile Communications, ICWMC 2009, pp. 87–92. IEEE Computer Society, Washington, DC (2009)
10. Shin, W., Kiyomoto, S., Fukushima, K., Tanaka, T.: A formal model to analyze the permission authorization and enforcement in the Android framework. In: Proceedings of the 2010 IEEE Second International Conference on Social Computing, SOCIALCOM 2010, pp. 944–951. IEEE Computer Society, Washington, DC (2010)
11. Shin, W., Kwak, S., Kiyomoto, S., Fukushima, K., Tanaka, T.: A small but non-negligible flaw in the Android permission scheme. In: Proceedings of the 2010 IEEE International Symposium on Policies for Distributed Systems and Networks, POLICY 2010, pp. 107–110. IEEE Computer Society, Washington, DC (2010)
12. Hornyack, P., Han, S., Jung, J., Schechter, S., Wetherall, D.: These Aren't the Droids You're Looking For: Retrofitting Android to Protect Data from Imperious Applications. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011 (2011)
13. Enck, W., Gilbert, P., Chun, B.-G., Cox, L.P., Jung, J., Mc- Daniel, P., Sheth, A.N.: TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In: Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010 (February 2010)
14. Android Permissions, http://Android-permissions.org/

15. `http://www.f-secure.com/en/web/labs_global/mobile-security`
16. Trojan:Android/RogueSPPush, `http://www.cs.ncsu.edu/faculty/jiang/RogueSPPush/`
17. Trojan:Android/ Zsone.a, `http://www.f-secure.com/v-descs/trojan_Android_zsone_a.shtml`
18. Android dalvik, `http://source.Android.com/tech/dalvik/index.html`
19. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: Behavior-Based Malware Detection System for Android. Proceedings of the 1st Workshop on Security and Privacy in Smartphones and Mobile Devices, CCS SPSM 2011 (2011)
20. Enck, W., Ongtang, M., McDaniel, P.: On Lightweight Mobile Phone Application Certification. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS 2009 (2009)
21. Schmidt, A.-D., Schmidt, H.-G., Clausen, J., Yuksel, K.A., Kiraz, O., Camtepe, A., Albayrak, S.: Enhancing security of linux-based Android devices. In: Proceedings of 15th International Linux Kongress, Lehmann (October 2008)
22. Schmidt, A.-D., Bye, R., Schmidt, H.-G., Clausen, J., Kiraz, O., Yxksel, K., Camtepe, S., Sahin, A.: Static analysis of executables for collaborative malware detection on Android. In: ICC 2009 Communication and Information Systems Security Symposium, Dresden, Germany (June 2009)
23. Blasing, T., Schmidt, A.-D., Batyuk, L., Camtepe, S.A., Albayrak, S.: An Android application sandbox system for suspicious software detection. In: 5th International Conference on Malicious and Unwanted Software, MALWARE 2010, Nancy, France (2010)
24. Zhou, Y., Wang, Z., Zhou, W., Jiang, X.: Hey, You, Get Off My Market: Detecting Malicious Apps in Alternative Android Markets. In: Proceedings of the 16th Network and Distributed System Security Symposium, NDSS 2012 (February 2012)
25. AppChina, `http://www.appchina.com/`
26. Pendragon Software Corporation. CaffeineMark 3.0
27. `http://www.benchmarkhq.ru/cm30/`
28. Di Cerbo, F., Girardello, A., Michahelles, F., Voronkova, S.: Detection of malicious applications on Android os. In: Sako, H., Franke, K.Y., Saitoh, S. (eds.) IWCF 2010. LNCS, vol. 6540, pp. 138–149. Springer, Heidelberg (2011)
29. `http://developer.android.com/reference/android/content/Intent.html`

iBinHunt: Binary Hunting with Inter-procedural Control Flow

Jiang Ming[1], Meng Pan[2], and Debin Gao[3]

[1] College of Info Sciences and Tech, Penn State University
jum310@ist.psu.edu
[2] D'Crypt Pte Ltd.
pinpinmao@gmail.com
[3] School of Info Systems, Singapore Management University
dbgao@smu.edu.sg

Abstract. Techniques have been proposed to find the semantic differences between two binary programs when the source code is not available. Analyzing control flow, and in particular, intra-procedural control flow, has become an attractive technique in the latest binary diffing tools since it is more resistant to syntactic, but non-semantic, differences. However, this makes such techniques vulnerable to simple function obfuscation techniques (e.g., function inlining) attackers any malware writers could use. In this paper, we first show function obfuscation as an attack to such binary diffing techniques, and then propose iBinHunt which uses *deep taint* and automatic input generation to find semantic differences in *inter-procedural* control flows. Evaluation on comparing various versions of a `http` server and `gzip` shows that iBinHunt not only is capable of comparing inter-procedural control flows of two programs, but offers substantially better accuracy and efficiency in binary diffing.

Keywords: binary diffing, semantic difference, taint analysis.

1 Introduction

Binary diffing tools for finding semantic differences between two programs have many security applications, e.g., automatically finding security vulnerabilities in a binary program given its patched version [17], large-scale malware indexing with function-call graphs [20], automatically adapting trained anomaly detectors to software patches [24], profile reuse in application development [33], etc. However, binary diffing is difficult due to different register allocation, semantically equivalent instruction replacement, and other program obfuscation techniques which make semantically equivalent programs syntactically different [17].

One of the latest solutions in binary diffing for finding semantic differences is to find similarity/difference in *control flow structure* rather than binary instructions [14,12,17,20]. Such tools have the advantage of being resistant to semantically equivalent instruction replacements and other program obfuscation techniques, and therefore are more suitable in security analysis in which programs (potentially malware) are usually intentionally produced to make analysis difficult. An interesting aspect we analyze in this paper is whether such analysis should be based on *inter-procedural* control flow or *intra-procedural* control flow.

Most previous work [14,12,17,20] focus on the intra-procedural control flow.[1] There is a good reason for this choice as the control flow comparison usually involves maximum common subgraph isomorphism, an NP-complete problem [18]. Working with all basic blocks in an inter-procedural control flow graph (ICFG) would require manipulation of graphs with thousands or tens of thousands of nodes, where finding a graph isomorphism becomes impractical. Working with basic blocks in an intra-procedural control flow graph (CFG), instead, is practical as the number of nodes does not usually go beyond hundreds. However, comparing the control flow structure of basic blocks in each function is vulnerable to function obfuscation techniques (e.g., function inlining) that could be used in producing the binary programs under analysis. This is a serious problem as applying some function obfuscation, e.g., function inlining, is extremely easy.

In this paper, we first demonstrate the attack of function obfuscation on binary diffing tools that compare intra-procedural control flow. We then propose a new binary diffing technique called *iBinHunt* that is resistant to such an attack. iBinHunt discards all function boundary information and compares the inter-procedural control flow of binary programs. It uses *deep taint*, a novel dynamic taint analysis technique that assigns different taint tags to various parts of the program input and traces the propagation of these taint tags to reduce the number of candidates of basic block matching. With deep taint, the set of matching candidates of each basic block changes from the set of all basic blocks in a program (in the order of thousands or tens of thousands) to just a few basic blocks on a particular execution trace with the same taint tag. To increase the coverage of execution traces on basic blocks, iBinHunt automatically generates program inputs that traverse different execute paths for the deep taint analysis.

We implemented iBinHunt and used it to compare various versions of a `http` server and `gzip`. Results show that iBinHunt finds semantic differences by analyzing the inter-procedural control flows with better accuracy, and is capable of comparing binary programs with relatively large differences, an improvement over previous techniques which are only shown to work on programs with small changes. We also show that iBinHunt is more efficient and faster in finding basic block matchings than previous techniques by a factor of two.

2 Existing Binary Diffing Tools and Function Obfuscation

We focus on binary diffing tools for finding semantic differences instead of syntactic differences. Semantic differences refer to differences in functionality (i.e., input-output behavior), whereas syntactic differences refer to those in instructions [17]. Therefore we do not consider binary diffing tools that base its analysis on the binary instructions (bsdiff, bspatch, xdelta, JDiff, etc.), because they are

[1] Some of them zoom in to do intra-procedural control flow analysis first, and subsequently zoom out for inter-procedural control flow analysis where each procedure is represented as a simple node with details ignored.

more vulnerable to different register allocation, basic block re-ordering, functionally equivalent instruction(s), and other instruction obfuscation techniques.

2.1 Existing Binary Diffing Tools Based on Control-Flow Structure

To find semantic differences between two binaries, some latest binary diffing techniques [14,12,27,17] base their comparison on intra-procedural control-flow structure. BinDiff [14] and its extension [12] use some heuristics (e.g., graphs with the same number of basic blocks, edges, and caller nodes) to test if two graphs or basic blocks are similar. BinHunt [17] compares basic blocks by symbolic execution and theorem proving, and then compares intra-procedural control-flow graphs to find the matchings between basic blocks. Call graphs are then compared to find matchings between functions. DarunGrim2 [21,11] relies heavily on function boundary information due to its simplicity. For basic blocks in every function, DarunGrim2 first generates a fingerprint to abstract the instruction sequences and then uses that as a key to a hash table, from which fingerprint matching is performed to find differences in the two functions. Intra-procedural control-flow graphs have also been used frequently in malware clustering and classification [20,2,5,22] because it's more resilient to instruction-level obfuscations. SMIT [20] searches for the most similar malware samples by finding a nearest-neighbor in malware's function-call graph database. Kruegel et al. [22] present an approach based on the analysis of a worm's intra-procedural control-flow graph to identify structural similarities between different worm mutations.

2.2 Function Obfuscation

Binary diffing tools based on control-flow structure are more resistant to different register allocation, basic block re-ordering, functionally equivalent instruction(s), and other instruction-level obfuscation techniques. However, most of them rely heavily on function boundary information from the binary, i.e., they analyze the *intra-procedural* control-flow structure of each function. We believe that this is mainly due to efficiency of the graph comparison techniques used. The graph comparison problem (and the subgraph isomorphism problem) is NP-complete. Existing algorithms for subgraph isomorphism are efficient only in processing small graphs [23,28,17]. Appendix A shows the number of basic blocks in different functions in a typical server program binary, which suggests that graph isomorphism is practical when analyzing intra-procedural control flows.

However, function boundary information is not reliable due to well-studied *function transformation obfuscation* techniques [9], which include

- **Inlining functions:** a function call to f is replaced with the body of f while f itself is removed;
- **Outlining functions:** a new function f is created by extracting a sequence of statements into f and replacing them with a function call to f;
- **Cloning functions:** copies of the same function are created (with different names) to make them appear as different functions;

– **Interleaving functions:** various function bodies are merged into one function f, while calls to these functions are replaced by calls to f.

Here we focus on function inlining and outlining because they have a large impact on graph isomorphism as discussed in Appendix B.

3 Diffing Binary Programs with Inter-Procedural Control-Flow Graphs

In Section 2.2, we discuss the function obfuscation attacks which existing binary diffing tools based on intra-procedural control-flow analysis cannot deal with. A natural solution to such attacks is to find repetitions of code sequences and combine them into one subroutine (to combat function inlining and cloning), and to flatten the hierarchical structure created by functions and to simply treat function calls as execution jumps (to combat function outlining and interleaving). After this there is only one graph left for each binary program containing all basic blocks and the corresponding control flows, and this graph is essentially the inter-procedural control-flow graph (ICFG).

However, such a simple solution has disadvantages in both accuracy and efficiency. Each basic block in one binary program will have a large number of candidates of basic block matchings in the other binary program. Even if all these candidates are examined, there could be multiple ones that are semantically similar that originally come from non-matching functions. However, since function information is ignored, all these basic blocks are good candidates and may make the result inaccurate. We also need to work on graph isomorphism of two graphs with large number of nodes. We tried this with BinHunt [17], one of the latest and most sophisticated binary diffing tools with graph isomorphism, and found that after working for 6 hours on basic block comparison with a desktop computer with a Core2 Duo CPU of 3.0 GHz and RAM of 4 GB on a server program `thttpd`, only 7% of the possible mappings had been compared.

3.1 Overview of iBinHunt

iBinHunt reduces the number of candidates of basic block matchings with a novel technique called *deep taint*. Taint analysis is to dynamically trace data from untrustworthy sources to monitor basic blocks in a program that process such data [26,7,35,31,16,13]. We monitor the execution of the two binary programs under a common input and use taint analysis to record all basic blocks involved in the processing of the input. This reduces the number of candidates of basic block matching from all basic blocks in the binary to those tainted.

iBinHunt goes one step further to assign different taint tags to various parts of the input, a method we call *deep taint*. Deep taint differentiates various parts of the input by assigning them different taint tags, and monitors propagation of different taint tags to basic blocks on a dynamic trace. Only basic blocks from two binary programs that are marked with the same taint tags are considered matching candidates. This further reduces the number of candidates of basic block matchings by a factor of up to 74% in our experiments.

Deep taint and the taint tags help reduce the number of matching candidates. However, only a small number of basic blocks are on the trace of the processing of a single input, and we need to find the matching of a large number of basic blocks (if not all) to make the graph isomorphism efficient. iBinHunt increases the coverage of execution traces on tainted basic blocks by automatically generating inputs that result in different execution traces in the binary program, a technique inspired by recent advances in white-box fuzz testing [19]. We first record the execution trace of a seeding input, and then symbolically replay the recorded trace and collect constraints of the input that lead to the recorded trace. The collected constraints are then negated and solved with a constraint solver to generate a new input, which will result in a different execution trace due to the negated constraint. A large number of inputs can be generated in this way, making more and more basic blocks tainted with different taint tags.

Next, we present the details of deep taint and automatic input generation.

3.2 Deep Taint for Basic Block Comparison

Previous taint analysis treats taint sources as *streams*, e.g., byte streams from keyboard, effectively tainting all input bytes with a single taint tag. Basic blocks processing different parts of such input will therefore be tainted with the same taint tag. In iBinHunt, we differentiate these basic blocks if they process different parts of the input. For example, basic blocks that process the `version` field of an `http` request should never match with basic blocks that process the `host` field of the same `http` request. Differentiating these basic blocks will reduce the number of candidates of basic block matchings.

Table 1 shows an example of the different taint tags assigned to various parts of an `http` request. Each unique taint tag corresponds to a particular bit in a binary number that allows *disjunction* manipulation. Deep taint works on the *protocol level* with a finer granularity such that various protocol fields are monitored with different taint tags. The process of locating different fields of the program input can be automated with a protocol analyzer [10,34,4].

Table 1. Program input and its taint tags

Input	`Get`	`index.html`	`HTTP/1.1`	`.`
Field	Method	URL	Version	Host
Taint tags	0001	0010	0100	1000

Multiple taint tags for a basic block. By monitoring the dynamic execution of an input, we can see the propagation of different taint tags to basic blocks in the program. Note that a basic block may appear multiple times on a dynamic execution trace due to loops. Such a basic block may record the same taint tag in the execution (when it processes the same part of the input in a loop) or different taint tags (when it processes different parts of the input).

Figure 1 shows an example of this in our experiment with `thttpd-2.25`. The highlighted instructions in the source code is located inside a `for` loop, which executes multiple times in the processing of an input and records multiple taint tags. The dynamic execution trace we obtained recorded four different taint tags for a basic block `BB_10088`, which corresponds to the highlighted instructions in the source code. We take the disjunction of these tags to obtain the final taint representation for the corresponding basic block.

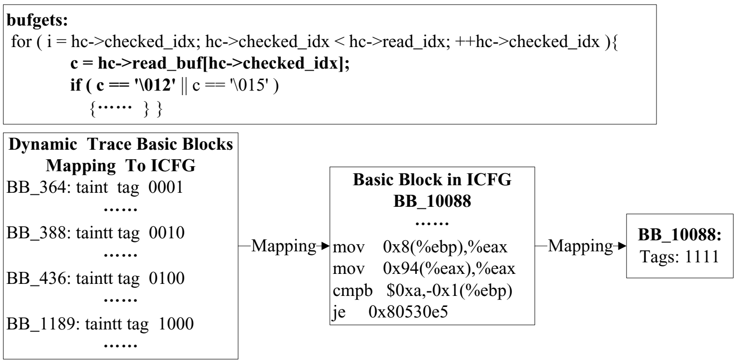

bufgets:
for (i = hc->checked_idx; hc->checked_idx < hc->read_idx; ++hc->checked_idx){
c = hc->read_buf[hc->checked_idx];
if (c == '\012' || c == '\015')
{…… } }
Dynamic Trace Basic Blocks Mapping To ICFG
BB_364: taint tag 0001
BB_388: taintt tag 0010
BB_436: taintt tag 0100
BB_1189: taintt tag 1000
Mapping
Basic Block in ICFG BB_10088
mov 0x8(%ebp),%eax
mov 0x94(%eax),%eax
cmpb $0xa,-0x1(%ebp)
je 0x80530e5
Mapping
BB_10088:
Tags: 1111

Fig. 1. Multiple taint tags

Basic block comparison. As mentioned in Section 3.1, basic blocks from the two binary programs that have the same taint representation will be candidates for matching. We compare these candidate basic blocks by applying the same algorithm as BinHunt [17], in which symbolic execution is used to represent the outputs of a basic block in terms of its symbolic inputs, and a theorem prover is used to test if the outputs from the two basic blocks are semantically equivalent. Although this basic block comparison might take relatively long time to converge (due to the use of a theorem prover), the number of comparisons is limited to the small number of blocks with the same taint representation, and therefore iBinHunt is more efficient (see Section 4 for our evaluation results).

Basic block matching. There are two other groups of blocks we need to consider for finding matched blocks. One group consists of blocks that are not semantically equivalent but have the same taint representation. They could very likely represent the differences between the two programs that iBinHunt is trying to locate. Another group consists of blocks that are not tainted but are on the dynamic execution trace. These blocks are not tainted due to various reasons, including limitations of taint analysis to avoid taint explosion [6,29], not directly processing program inputs (e.g., signal processing), etc. However, they are also very likely to match with one another as they are on the dynamic trace of processing the same input. Appendix C shows an example of these two groups of blocks in `thttpd-2.19` and `thttpd-2.25`.

One way of dealing with these two groups of blocks is to define a matching strength for basic block comparison, and consider two blocks matching when the matching strength exceeds certain threshold; an approach used in BinHunt [17]. We do not use this approach because 1) iBinHunt emphasizes using control-flow structural information rather than comparing binary instructions in basic blocks, and 2) the setting of such a threshold is difficult and different settings may lead to different results. Instead, we apply a more stringent requirement that basic blocks b_1 and b_2 are considered matched to one another if b_1 and b_2 have the same taint representation (possibly both non-tainted) and

– b_1 and b_2 are semantically equivalent (evaluated by symbolic execution and theorem proving as explained above); or
– a predecessor of b_1 and a predecessor of b_2 match; or
– a successor of b_1 and a successor of b_2 match.

We want to see how far we can go with such a stringent definition of matching. Note that it is possible that some matching blocks are not found unless a relaxed definition is used, which can be easily applied in iBinHunt for practical usage.

3.3 Automatic Generation of Program Inputs

Although deep taint reduces the number of matching candidates in basic block comparison, it only helps finding the matchings for basic blocks on the corresponding execution trace. Therefore, deep taint applied to more program inputs is needed. However, random inputs are not the most desired because they may result in the same execution paths. We need to find inputs that traverse different paths in the binary program, which is a similar requirement to those in program testing where test cases are needed to cover more program execution paths.

White-box exploration on binary files has been used in many previous work [3,25,19]. We apply the same idea to generate execution traces in an iterative process that incrementally explores new execution paths. In each iteration, we first monitor and record an execution trace. We then use a constraint collector [30] to run symbolic execution on the recorded trace and gather the constraints on inputs on every branching conditions. These constraints capture how the input was processed in the corresponding dynamic execution. We then negate one of these constraints collected to obtain the input constraints that would result in a different execution path, and solve these constraints with the theorem prover to obtain a corresponding real input for deep taint in the next iteration.

There are typically many branching locations on an execution trace. We pick one that may result in the largest number of new basic blocks explored by counting all the uncovered basic blocks of the corresponding sub-tree.

4 Implementation and Evaluation

4.1 Implementation of iBinHunt

Figure 2 shows the architecture of iBinHunt. In the rest of this subsection, we briefly describe how each component of iBinHunt was implemented.

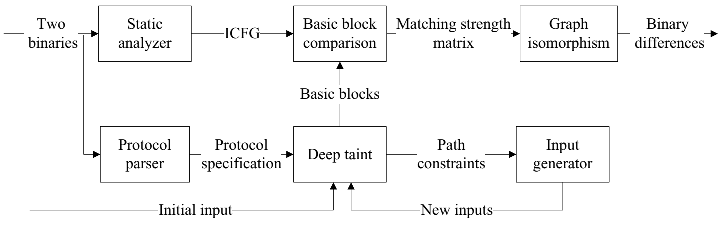


Fig. 2. Architecture of iBinHunt

Static analyzer. iBinHunt uses the same static analyzer as in BinHunt [17]. It first disassembles the two binary programs to obtain the x86 instructions, and then converts the x86 instructions into an intermediate representation (IR) for further analysis. The IR we use is the same as in BinHunt and BitBlaze [1,30], which consists of roughly a dozen different statements. Control flow is analyzed on the IR of the two binary programs to obtain the inter-procedural control-flow graph (ICFG), where nodes correspond to basic blocks in the program and edges correspond to transitions among the basic blocks.

Protocol analyzer. We assume that the protocol specifications are known, and therefore a protocol analyzer is not needed. In case the protocol specification is not known, any automatic protocol analyzer [10,34,4] can be used.

Deep taint. Deep taint was based on TEMU [36] and QEMU[2]. TEMU uses a shadow memory to store the taint status. We modify the shadow memory and add a small data structure for each taint byte to store its corresponding taint tag. Currently deep taint supports up to 64 different tags.

Basic block comparison. The dynamic traces from deep taint are first mapped to the ICFG. This mapping is simple as the `eip` value recorded in deep taint and the program counter value in ICFG differ by the length of the corresponding instruction. Once this mapping is obtained, comparison of two basic blocks from the two binary programs is carried out if they have the same taint representation and are on dynamic traces recorded given the same program input.

We use the same basic block comparison technique as in BinHunt [17], i.e., symbolic execution is first used to represent outputs of the basic blocks with their input symbols, and a theorem prover (STP [15]) is then used to check if the outputs from the two basic block are semantically equivalent. Note that the basic block comparison performed here is slightly different from BinHunt in that here the comparison is context aware, i.e., the permutation of outputs of the equivalent basic blocks is the permutation of inputs of the successor blocks. This is because the basic blocks to be compared here are on a particular execution path, where there is always a unique predecessor and a unique successor.

[2] QEMU, `www.qemu.org`

Graph isomorphism. iBinHunt also uses the same (customized) backtracking technique to find the maximum isomorphic subgraph as in BinHunt [17].

Difference from BinHunt. Although some components of iBinHunt are very similar to those in BinHunt as explained above, there is a major difference between the two, namely iBinHunt uses a dynamic component of deep taint while BinHunt bases purely on static analysis of the binary programs.

Input generator. Path constraints are collected as in `appreplay` [30]. We use STP [15] to find a new input that satisfies the negated constraints.

4.2 Evaluation

We applied iBinHunt to find semantic differences in several versions of `thttpd` and `gzip`. We chose to work on `thttpd` and `gzip` for two main reasons. First, they were commonly used programs for which we could find various older versions that are substantially different from the latest one, an evaluation criteria we have for iBinHunt. Second, both `thttpd` and `gzip` had known vulnerabilities in their earlier versions, which is a typical application scenario of iBinHunt.

To evaluate iBinHunt in its resistance to function obfuscation, we simply use iBinHunt to analyze the inter-procedural control-flow graphs instead of enumerating different obfuscation techniques. As discussed in Section 3, iBinHunt removes repetitions and flattens function structures, which will result in the same ICFG no matter what function obfuscation techniques are used.

There are two main aspects on which we want to evaluate. First, we want to see how many basic blocks can be matched, how many matchings are identified by deep taint, and how long it takes to find these matchings. Second, we want to take a closer look at the differences found, and confirm these differences by comparing them to the ground truth (program source code).

Table 2 and Table 3 show the simple statistics of the various versions of `thttpd` and `gzip`, respectively. Note that in some cases, the differences account to nearly 40% of the source code, which we consider very big changes between the two versions. Due to the space limitation, we do not detail all these changes, most of which are due to bug fixing and new features added.

Table 2. Different versions of thttpd (number of lines changed / total number of lines)

`thttpd-`	`2.20`	`2.20c`	`2.21`	`2.25`
`2.19`	252/6029	254/5843	1483/6641	2908/7271

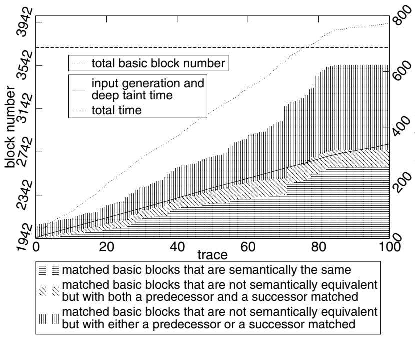
We performed our experiments on two machines, one with a Core2 Duo CPU of 2.6 GHz and RAM of 4 GB (for deep tainting) and another with a Core2 Duo CPU of 3.0 GHz and RAM of 4 GB (for all other components).

Figure 3 and Figure 4 show the results of `thttpd` and `gzip`, respectively. Each graph shows six different types of information.

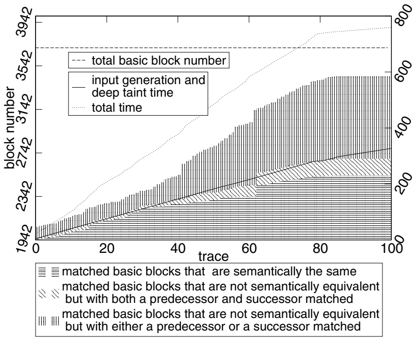
Table 3. Different versions of gzip (number of lines changed / total number of lines)

gzip-	1.3.12	1.3.13	1.40
1.2.4	1317/4959	1351/4929	1446/4841

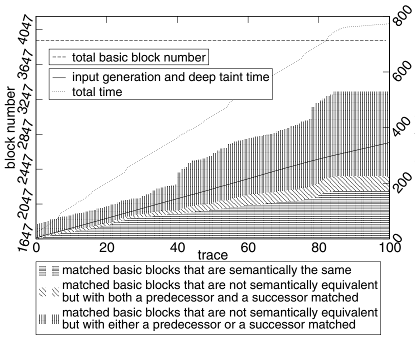
– Shaded areas: the three shaded areas show the number of matched blocks according to our definition of matching in Section 3.2. The horizontal shaded area corresponds to matched basic blocks that are semantically the same; the 135-degree shaded area corresponds to matched ones that are not semantically equivalent but have both a predecessor and a successor matched; and the vertical shaded area corresponds to those that are not semantically equivalent but have either a predecessor or a successor matched.
– Lines: the lower slanted line indicates the time taken for input generation and deep taint; the upper slanted line indicates the total time spent; and the horizontal line shows the total number of basic blocks in the binary program;



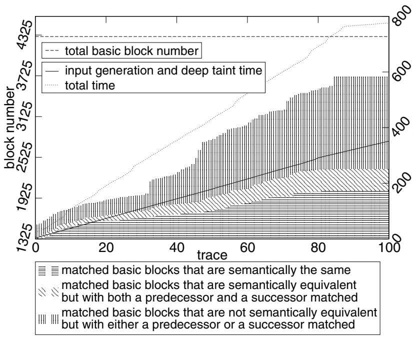
(a) thttpd-2.19 vs. thttpd-2.20



(b) thttpd-2.19 vs. thttpd-2.20c

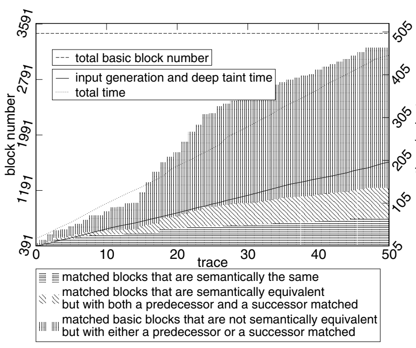


(c) thttpd-2.19 vs. thttpd-2.21

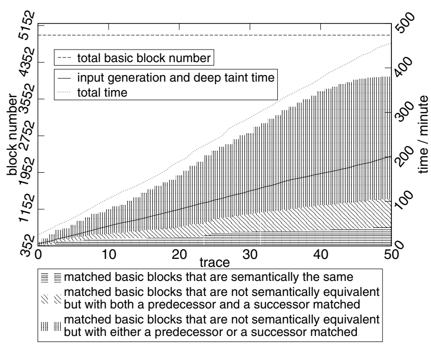


(d) thttpd-2.19 vs. thttpd-2.25

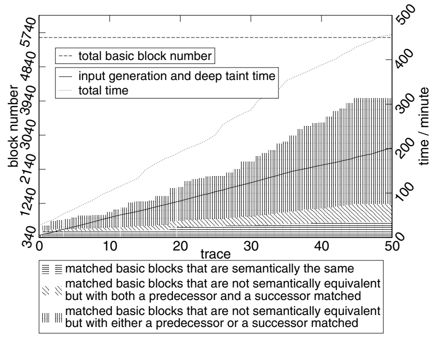
Fig. 3. Evaluation on different versions of thttpd



(a) gzip-1.2.4 vs. gzip-1.3.12



(b) gzip-1.2.4 vs. gzip-1.3.13



(c) gzip-1.2.4 vs. gzip-1.40

Fig. 4. Evaluation on different versions of gzip

Matching basic blocks. Although we use a relatively stringent definition of matching (see Section 3.2), iBinHunt manages to find most of the matching blocks. For example, Figure 4 shows that about 90% of the basic blocks are matched in comparing `gzip-1.2.4` and `gzip-1.3.12`, which have over 25% of the lines of code changed. We also study the matchings found, and confirm that they are correct. Most differences are reflected in these matchings, too, with some differences not found; see Section 4.3 for more discussions.

Effectiveness of deep taint. Among successfully matched basic blocks, we count the number of them that actually contain the same taint representation (the rest are not tainted). Results (see Table 4 and Table 5) show that more than 34% and 67% of the matched basic blocks in `thttpd` and `gzip`, respectively, contain the same taint representation. This shows that 1) deep taint is effective in helping to identify basic block matchings since a large number of these matchings do contain the same taint representation; 2) even though many basic blocks are not tainted by our limited number of program inputs, their neighbors are tainted in most cases and the tainted neighbors help matchings to be identified.

Table 4. Matched basic blocks with the same taint representation (thttpd)

thttpd-	2.20	2.20c	2.21	2.25
2.19	34.8%	38.2%	39.9%	37.4%

Table 5. Matched basic blocks with the same taint representation (gzip)

gzip-	1.3.12	1.3.13	1.40
1.2.4	67.9%	72.2%	72.6%

Accuracy. iBinHunt has better accuracy in basic block matching because deep taint reduces the number of matching candidates. Typically, the number of candidate matchings is 8% and 5% of total basic block pairs in our experiments with `thttpd` and `gzip`. Refer to Appendix D for another example of accuracy improvement of iBinHunt.

Handling binary programs with big differences. The results clearly show that iBinHunt is good in handling binary programs with big differences, a property previous tools for finding semantic differences [17] do not have. These can be seen from the percentage of basic block matched (all shaded areas), which does not decay significantly when dealing with binary programs with larger differences.

Time taken in the analysis. From Figure 3 and Figure 4, we see that when more traces are used, more basic blocks are matched until a steady state is reached. 85 and 50 inputs were needed before the number of matched basic blocks stops increasing for `thttpd` and `gzip`, respectively. These 85 or 50 input generations and deep taint analysis are incremental and cannot be parallelized. However, the basic block comparison can be easily parallelized to shorten the time needed. Also note that our implementation is an un-optimized one and there are rooms for improvements. That said, we still see more than a factor of 2 improvement when compared to BinHunt [17] (see Table 6 and Table 7). The starting percentage corresponds to basic blocks that are syntactically the same.

Table 6. Progress made in comparing thttpd-2.19 and thttpd-2.25

	Percentage of basic blocks matched			Time
	Starting	Ending	Progress made	spent
BinHunt	31%	38%	7%	6 hours
iBinHunt	31%	47%	18%	6 hours

Note that results in Table 6 and Table 7 are obtained without parallelizing basic block comparison for a fair comparison. Parallelizing the comparison could speed up the process a lot to make iBinHunt practical in analyzing real programs.

Table 7. Progress made in comparing gzip-1.24 and gzip-1.40

	Percentage of basic blocks matched			Time
	Starting	Ending	Progress made	spent
BinHunt	11%	16%	5%	3 hours
iBinHunt	11%	25%	14%	3 hours

4.3 Discussions

Although we focus on analyzing the inter-procedural control-flow graph in demonstrating the advantages of iBinHunt in this paper, iBinHunt is also resistant to other types of program obfuscations, e.g., control flow flattening [32,8], that existing binary diffing tools cannot handle. This is mainly due to the deep taint analysis we employ, which is a dynamic analysis approach.

The power of iBinHunt is limited by the non-perfect basic block coverage. This is mainly due to limitations of white box exploration technique [19], e.g., path explosion and imperfect symbolic execution to system calls.

Since iBinHunt uses deep taint, it also suffers from some limitations of taint analysis in general, e.g., control dependence, pointer indirection, and implicit information flow evasions [6,29].

We performed our evaluation and analysis by comparing iBinHunt with another state-of-the-art binary diffing tool BinHunt [17]. We could have made compassion with other binary diffing tools, e.g., BinDiff. However, due to the many heuristics BinDiff and other binary diffing tools use, it is hard to have a fair comparison with iBinHunt, in which such heuristics are not used. We leave it as future work to compare with other binary diffing tools.

5 Conclusion

In this paper, we first introduce function obfuscation attacks in existing binary diffing tools that analyze intra-procedural control flow of programs. We propose a novel binary diffing tool called iBinHunt which, instead, analyzes the inter-procedural control flow. iBinHunt makes use of a novel technique called deep taint which assigns different taint tags to various parts of the program input and traces the propagation of these taint tags in program execution. iBinHunt automatically generates program inputs to improve basic block coverage. Evaluations on comparing various versions of `thttpd` and `gzip` show that iBinHunt offers better accuracy and efficiency than existing binary diffing tools.

References

1. BitBlaze: Binary analysis for computer security, `http://bitblaze.cs.berkeley.edu/`
2. Briones, I., Gomez, A.: Graphs, entropy and grid computing: Automatic comparison of malware. In: Proceedings of the 2004 Virus Bulletin Conference (2004)

3. Brumley, D., Hartwig, C., Kang, M.G., Liang, Z., Newsome, J., Poosankam, P., Song, D., Yin, H.: Bitscope: Automatically dissecting malicious binaries. Technical Report, CMU-CS-07-133, School of Computer Science, Carnegie Mellon University (March 2007)
4. Caballero, J., Poosankam, P., Kreibich, C., Song, D.: Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In: Proceedings of the 16th ACM Conference on Computer and Communication Security, Chicago, IL (November 2009)
5. Carrera, E., Erdelyi, G.: Digital genome mapping al advanced binary malware analysis. In: Proceedings of the 2004 Virus Bulletin Conference (2004)
6. Cavallaro, L., Saxena, P., Sekar, R.: On the limits of information flow techniques for malware analysis and containment. In: Zamboni, D. (ed.) DIMVA 2008. LNCS, vol. 5137, pp. 143–163. Springer, Heidelberg (2008)
7. Chow, J., Pfaff, B., Garfinkel, T., Christopher, K., Rosenblum, M.: Understanding data lifetime via whole system simulation. In: 13th USENIX Security Symposium (2004)
8. Chow, S., Gu, Y., Johnson, H., Zakharov, V.: An approach to the obfuscation of control-flow of sequential computer programs. In: Davida, G.I., Frankel, Y. (eds.) ISC 2001. LNCS, vol. 2200, pp. 144–155. Springer, Heidelberg (2001)
9. Collberg, C., Thomborson, C., Low, D.: A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Sciences, The University of Auckland (July 1997)
10. Cui, W.: Discoverer: Automatic protocol reverse engineering from network traces. In: Proceedings of the 16th USENIX Security Symposium (2007)
11. DarunGrim, J.O.: A binary diffing tool, `http://www.darungrim.org`
12. Dullien, T., Rolles, R.: Graph-based comparison of executable objects. In: Proceedings of SSTIC 2005 (2005)
13. Egele, M., Kruegel, C., Kirda, E., Yin, H., Song, D.: Dynamic spyware analysis. In: Proceedings of the 2007 Usenix Annual Conference (2007)
14. Flake, H.: Structural comparison of executable objects. In: Proceedings of the GI International Conference on Detection of Intrusions & Malware, and Vulnerability Assessment 2004 (2004)
15. Ganesh, V., Dill, D.L.: A decision procedure for bit-vectors and arrays. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 519–531. Springer, Heidelberg (2007)
16. Ganesh, V., Leek, T., Rinard, M.: Taint-based directed whitebox fuzzing. In: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering (2009)
17. Gao, D., Reiter, M.K., Song, D.: BinHunt: Automatically finding semantic differences in binary programs. In: Chen, L., Ryan, M.D., Wang, G. (eds.) ICICS 2008. LNCS, vol. 5308, pp. 238–255. Springer, Heidelberg (2008)
18. Garey, M.R., Johnso, D.S.: Computers and intractability: A guide to the theory of np-completeness (1979)
19. Godefroid, P., Levin, M.Y., Molnar, D.: Automated whitebox fuzz testing. In: Network and Distributed System Security Symposium, NDSS 2008 (2008)
20. Hu, X., Chiueh, T., Shin, K.: Large-scale malware indexing using function-call graphs. In: Proceedings of the 16th ACM Conference on Computer and Communications Security (2009)
21. Jeongwook, O.: Fight against 1-day exploits: Diffing binaries vs anti-diffing binaries. In: Black Hat (2009)

22. Kruegel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G.: Polymorphic worm detection using structural information of executables. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 207–226. Springer, Heidelberg (2006)
23. Levi, G.: A note on the derivation of maximal common subgraphs of two directed or undirected graphs. Calcolo 9 (1972)
24. Li, P., Gao, D., Reiter, M.K.: Automatically adapting a trained anomaly detector to software patches. In: Kirda, E., Jha, S., Balzarotti, D. (eds.) RAID 2009. LNCS, vol. 5758, pp. 142–160. Springer, Heidelberg (2009)
25. Molnar, D., Li, X.C., Wagner, D.: Dynamic test generation to find integer bugs in x86 binary linux programs. In: Proceedings of USENIX Security Symposium (2009)
26. Newsome, J., Song, D.: Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: Network and Distributed System Security Symposium, NDSS 2005 (2005)
27. Tenable Network Security Inc. PatchDiff. A patch analysis plugin for ida, `http://cgi.tenablesecurity.com/tenable/patchdiff.php`
28. Raymond, J., Willett, P.: Maximum common subgraph isomorphism algorithms for the matching of chemical structures. Journal of Computer-Aided Molecular Design 16 (2002)
29. Schwartz, E.J., Avgerinos, T., Brumley, D.: All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In: Proceedings of the 2010 IEEE Symposium on Security and Privacy (2010)
30. Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M.G., Liang, Z., Newsome, J., Poosankam, P., Saxena, P.: BitBlaze: A new approach to computer security via binary analysis. In: Sekar, R., Pujari, A.K. (eds.) ICISS 2008. LNCS, vol. 5352, pp. 1–25. Springer, Heidelberg (2008)
31. Suh, G.E., Lee, J.W., Zhang, D., Devadas, S.: Secure program execution via dynamic information flow tracking. In: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (2004)
32. Wang, C., Davidson, J., Hill, J., Knight, J.: Protection of software-based survivability mechanisms. In: Proceedings of International Conference of Dependable Systems and Networks (2001)
33. Wang, Z., Pierce, K., McFarling, S.: Bmat – a binary matching tool for stale profile propagation. Journal of Instruction-Level Parallelism 2 (2000)
34. Wondracek, G., Comparetti, P.M., Kruegel, C., Kirda, E.: Automatic network protocol analysis. In: Proceedings of the 15th Annual Network and Distributed System Security Symposium, NDSS 2008 (2008)
35. Yin, H., Song, D.: Panorama: capturing system-wide information flow for malware detection and analysis. In: ACM Conference on Computer and Communications Security, CCS 2007 (2007)
36. Yin, H., Song, D.: Temu: Binary code analysis via whole-system layered annotative execution. Technical Report, EECS Department, University of California, Berkeley (January 2010)

A Size of Different Functions in Thttpd

Figure 5 shows the cumulative histogram of functions with different number of basic blocks in `thttpd`, an `http` server. It can be seen that 96% of the 459 non-empty functions have fewer than 30 basic blocks. Only 7 functions have more than 50 basic blocks. This makes the graph comparison simple, as in most cases we only need to deal with graphs of fewer than 30 nodes. Graph isomorphism is therefore practical in analyzing programs like `thttpd`.

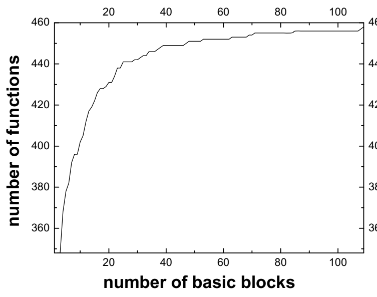


Fig. 5. Number of basic blocks in different functions (cumulative histogram)

B Function Inlining and Outlining

Figure 6 shows the basic idea of function inlining and outlining transformations. Such simple attacks are effective in confusing existing binary diffing tools because inlining and outlining can arbitrarily increase or decrease the size of any functions. The intra-procedural control-flow graph may contain unreliable information, resulting in a small maximum common subgraph (as in many binary diffing tools, e.g., [14,12,27,17]) or complete failure when the whole program contains only a single function (as in some malware analysis tools, e.g., [20]).

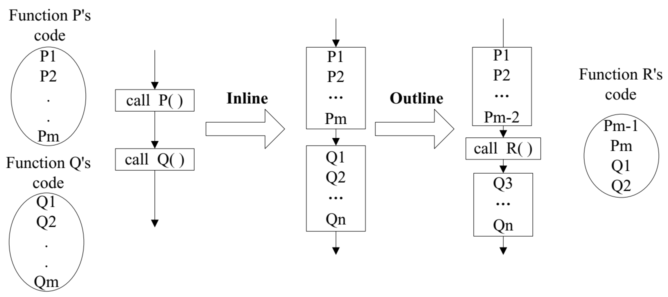
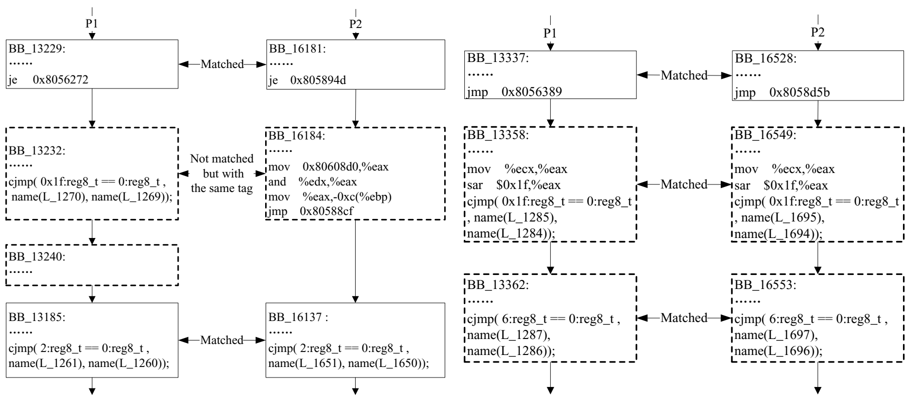


Fig. 6. Inlining and outlining transformations

C Example of Potential Matching Blocks

Figure 7 shows an example of these two groups of blocks in `thttpd-2.19` and `thttpd-2.25`. In Figure 7(a), `BB_13232` and `BB_16184` are not semantically equivalent, but they have the same taint representation (0011). They both originally come from function `find_hash()` corresponding to a difference in the hash algorithm used in the two versions of `thttpd`. In Figure 7(b), the four dashed blocks are not tainted. A closer look into the corresponding source shows that these blocks are part of the function `tmr_create()`, which does some simple time routine and therefore are not tainted.



```
BB_13229:
......
je    0x8056272

BB_16181:
......
je    0x805894d

BB_13232:
......
cjmp( 0x1f:reg8_t == 0:reg8_t ,
 name(L_1270), name(L_1269));

BB_16184:
......
mov    0x80608d0,%eax
and    %edx,%eax
mov    %eax,-0xc(%ebp)
jmp    0x80588cf

BB_13240:
......

BB_13185:
......
cjmp( 2:reg8_t == 0:reg8_t ,
name(L_1261), name(L_1260));

BB_16137 :
......
cjmp( 2:reg8_t == 0:reg8_t ,
name(L_1651), name(L_1650));
```

```
BB_13337:
......
jmp    0x8056389

BB_16528:
......
jmp    0x8058d5b

BB_13358:
......
mov    %ecx,%eax
sar    $0x1f,%eax
cjmp( 0x1f:reg8_t == 0:reg8_t
, name(L_1285),
name(L_1284));

BB_16549:
......
mov    %ecx,%eax
sar    $0x1f,%eax
cjmp( 0x1f:reg8_t == 0:reg8_t
, name(L_1695),
name(L_1694));

BB_13362:
......
cjmp( 6:reg8_t == 0:reg8_t ,
name(L_1287),
name(L_1286));

BB_16553:
......
cjmp( 6:reg8_t == 0:reg8_t ,
name(L_1697),
name(L_1696));
```

(a) Blocks with same taint representation (b) Blocks not tainted

Fig. 7. Potential matching blocks

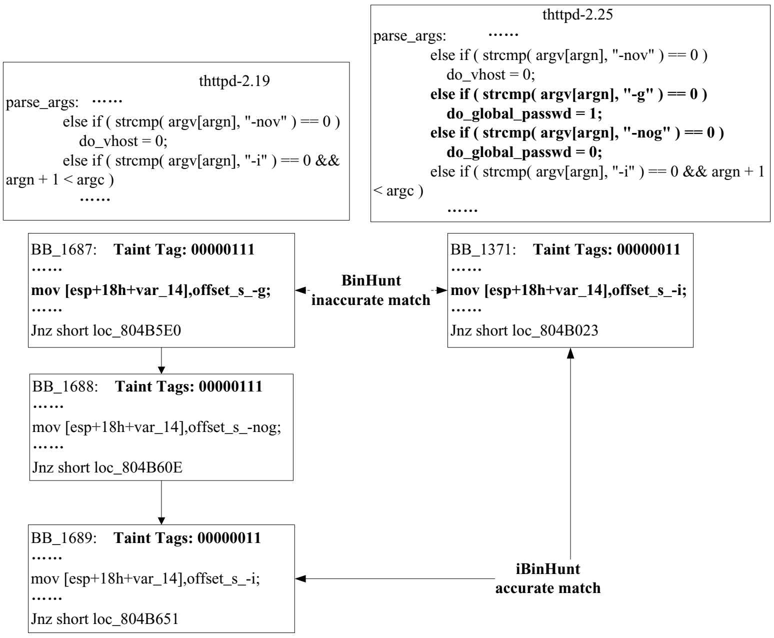
D Improved Accuracy of iBinHunt

Figure 8 shows an example in which iBinHunt outputs basic block matching with improved accuracy.

In this example, `BB_1371` from `thttpd-2.25` should match with `BB_1689` in `thttpd-2.19`, both of which deal with the "-i" argument. However, `BB_1687` in `thttpd-2.19` also contains the same (type of) instructions, which confuses the binary diffing tool in the matching. We tried BinHunt [17] and found that BinHunt, in fact, finds the wrong matching in this case.

On the other hand, iBinHunt easily avoids such errors because the different taint representation `BB_1687` has, and therefore `BB_1687` is not even on the list of matching candidates of `BB_1371`.

Besides confirming that the differences found by iBinHunt correspond to semantic differences in the source code, we also verified that these differences include many patches to vulnerabilities in the earlier version. Therefore, iBinHunt



```
thttpd-2.19
parse_args: ……
        else if ( strcmp( argv[argn], "-nov" ) == 0 )
            do_vhost = 0;
        else if ( strcmp( argv[argn], "-i" ) == 0 &&
argn + 1 < argc )
            ……
```

```
thttpd-2.25
parse_args:     ……
        else if ( strcmp( argv[argn], "-nov" ) == 0 )
            do_vhost = 0;
        else if ( strcmp( argv[argn], "-g" ) == 0 )
            do_global_passwd = 1;
        else if ( strcmp( argv[argn], "-nog" ) == 0 )
            do_global_passwd = 0;
        else if ( strcmp( argv[argn], "-i" ) == 0 && argn + 1
< argc )
            ……
```

```
BB_1687:   Taint Tag: 00000111
……
mov [esp+18h+var_14],offset_s_-g;
……
Jnz short loc_804B5E0
```

```
BB_1688:   Taint Tags: 00000111
……
mov [esp+18h+var_14],offset_s_-nog;
……
Jnz short loc_804B60E
```

```
BB_1689:   Taint Tags: 00000011
……
mov [esp+18h+var_14],offset_s_-i;
……
Jnz short loc_804B651
```

```
BB_1371:   Taint Tags: 00000011
……
mov [esp+18h+var_14],offset_s_-i;
……
Jnz short loc_804B023
```

Fig. 8. Accuracy improvement

can be used to automatically find vulnerabilities by comparing different versions of a program for automatic vulnerability discovery, which is an important security application.

Sometimes It's Better to Be STUCK! SAML Transportation Unit for Cryptographic Keys[*]

Christopher Meyer, Florian Feldmann, and Jörg Schwenk

Horst Görtz Institute for IT-Security, Ruhr-University Bochum
{christopher.meyer,florian.feldmann,joerg.schwenk}@rub.de

Abstract. Over the last decade the Security Assertion Markup Language (SAML) framework evolved to a versatile standard for exchanging security statements about subjects. Most notably, SAML facilitates the *authentication* of users, and is thus deployed in both Webservice (SOAP, WS-Security) and REST-based (SAML SSO webbrowser profile, SAML Bearer token in OAuth) services.

This paper recommends an extension to the SAML framework which provides an easy way to transport cryptographic key material bound to assertions issued by particular subjects. The proposal fits into existing solutions and is fully compliant with the Security Assertion Markup Language, XML Digital Signature and XML Encryption standards.

Keywords: SAML, XML, Key Transportation, Key Distribution, SAML Extension.

1 Introduction

SAML. In the world of Single Sign-On (SSO), and authentication of users in general, the Security Assertion Markup Language (SAML) [1] evolved to be a successful standard. Companies like Google[1] and Salesforce[2] rely on its flexibility and benefits. SAML's ability to map security statements about subjects to XML provides an easy and human readable solution for demands concerning authentication and authorization of data exchange.

AKE. Multiple real-world applications depend on an *authenticated key exchange (AKE)*, which usually consists of a key agreement protocol combined with a corresponding authentication protocol. It is necessary to combine identity management and federation with key exchange capabilites between the participants in a secure way.

[*] This work was partially funded by the *Sec*^{2} project of the German Federal Ministry of Education and Research (BMBF, FKZ: 01BY1030).

[1] `http://www.google.com`

[2] `http://www.salesforce.com`

Contribution. This paper describes how to perform authenticated key transport within the SAML framework. More precisely, it provides the following contributions:

– It is shown how to embed key information into SAML Assertions, in a fully standard compliant way. Thus key management can easily be integrated in any SSO/IDM system.
– A proof of concept implementation of the proposed solution is available within the *Sec*^{2} project[3] which aims at adressing the issue of user encrypted cloud storage by performing en- and decryption exclusively at client side (and by using hardware enabled key stores). For this to work it is necessary to exchange key material (the solution will be introduced in detail in section 6).

2 Related Work

The idea of combining SAML and key management/distribution capabilities is not new and has already been subject of several other publications such as the *SAML V2.0 Kerberos Web Browser SSO Profile Version1.0* [2] specification. The aforementioned standard aims at a seamless integration of Kerberos into the browser world in combination with SAML usage. Thus, Kerberos already provides a complete solution for symmetric key management and distribution. Due to its great success and wide spread distribution, Kerberos can be seen as the de facto standard for key distribution in the symmetric world. Technologies like Microsoft ActiveDirectory[4] rely on the security of the Kerberos protocol. Additionally, many vendors such as e.g., Oracle[5] or SAP[6] offer Kerberos support in their products - mostly for authentication purposes. However, the main drawback of Kerberos - from this paper's point of view - is that it is limited to be used with symmetric keys only.

For use with asymmetric keys, there is an existing standard for key management, the *XML Key Management Specification (XKMS 2.0)* [3]. The main focus of XKMS is to define a protocol for distribution and registration of public keys. The goal is to provide a WebService for the management of public key material so that other WebServices can obtain public keys for encryption and verification of digital signatures. This WebService protocol can be compared to the well known public key server functionality introduced by *PGP* [4]. Since this standard is solely based on asymmetric public keys it is also not applicable for this proposal, as this paper aims for a technology independent solution regarding both, symmetric and asymmetric key material.

Binding keys to identities is not only a major goal of this proposal, but also the *X.509* standard [5] and *PGP* [4] address this topic. Keys should be undoubtedly

[3] http://www.sec2.org
[4] http://www.microsoft.com/en-us/server-cloud/windows-server/active-directory.aspx
[5] http://www.oracle.com
[6] http://www.sap.com

connected to the corresponding entities. But one has to keep in mind that these bindings are static and non flexible. In contrast to this kind of key binding, STUCK is flexible since it binds keys to Assertions (which are themselves bound to static identities, but the keys are only implicitly bound to these identities via the Assertion). As Assertions are in general only short-lived, this can be turned into an advantage. Binding keys to Assertions and not directly to certificates offers much more flexibility and introduces a new kind of abstraction layer.

Another standard to mention is *WS-Trust* [6]. Though similar ideas of this paper could also be realized by using the WS-Trust specification, this proposal is based on SAML due to its wider usage and acceptance at major companies.

3 Motivation

With emerging new capabilities of servers and clients transporting keys or key material over the internet, in a secure and reliable way, will become more and more important in the following years. For example, the proposal of the *web crypto API*[7] or the suggestions made by the *Web Cryptography Working Group*[8] will provide clients and servers with cryptographic capabilities. In these scenarios it is often mandatory to securely exchange keys between multiple parties. Therefor standardized means for secure key transportation are necessary. Regarding this SAML recommends itself, due to its flexibility and wide-spread deployment.

3.1 Advantages of the Proposal

This proposal offers the option to bind key material to an `Assertion`. Key transportation, whether encrypted or unencrypted (as in case of asymmetric public keys), can now easily be done in the same communication process and same protocol as SAML. An additional step for key management or distribution can be ommitted.

Further on, a clean standardized way may ease and facillitate identity federation beyond company borders. Not only identities could be shared, but key material, too. It could even be possible to offer key establishment facilities as an additional benefit on an identity provider's side.

The practical need for key transportation, management and distribution can be seen in the previous work that has been done, as for example the already mentioned *SAML V2.0 Kerberos Web Browser SSO Profile* [2]. But proposing a standard for only a single usage scenario is not sufficient, since it does not care about the needs of solutions not relying on this specific scenario (in this case Kerberos). What is really needed is a very flexible approach decoupled from SAML profiles and existing key establishment and distribution systems. The proposed solution in this paper is open for every possible usage scenario.

[7] `http://html5.creation.net/webcrypto-api/`
[8] `http://www.w3.org/2012/webcrypto/`

Binding keys to SAML `Assertion`s as explained in the prior sections eliminates the necessity for additional transport media encryption since the confidential parts of the key structure can be protected at application level - either by securing the key material itself using `EncryptedKey` elements, or obfuscating the whole `KeyInfo` structure by using `EncryptedAttribute`.

Also, one has to be aware that transport encryption is not equal to identity binding since the transferred data is neither bound to an identity, nor protected after the transport has been performed. Identities and keys obtained before or after transportation (e.g., through malware or careless data processing) can be used independent of corresponding identities. Channel binding approaches may solve this issue, but add an additional server side requirement: Servers need to support both, standard conformity concerning XML processing and means like SSL/TLS for transport layer encryption. It should be noted that XML Encryption and XML Signature are partly necessary for SAML to work properly, thus it remains a valid assumption that those two standards are already available at server side.

4 Technological Foundations

The following section will introduce the major technologies utilized by the proposal. Readers familiar with XML, XML Signature, XML Encryption, SAML and key management capabilities of these standards may skip this section.

XML. The eXtensible Markup Language (XML) [7] represents a human readable and machine processable language for data structuring. Data can be organized in a tree-based manner and tagged with attributes. As a major benefit, XML offers the option to be automatically validated against XML Schema Definition (XSD) files to guarantee conformance with particular data structuring rules. Both XML and optional XSD files are highly flexible and adjustable to fit nearly every scenario regarding data structuring.

XML Signature. For applying the concept of digital signatures to XML documents the XML-Signature Syntax and Processing Standard (XML Signature) [8] was created. By using XML Signature it is possible to sign parts of XML documents or even the whole document.

An XML Signature is introduced by adding a `<ds:Signature>` element into an XML document. In most cases this element consists of three main subelements: The `<ds:SignedInfo>` element specifies the necessary setup for signature creation and verification such as an optional canonicalization - a document restructuring option -, the signature algorithm and the references - the signed document parts which can be referenced e.g., via ID or XPath. Within the references also the digest method and possibly transformation methods as well as the digest value can be specified. The `<ds:SignatureValue>` element contains the actual signature associated with the referenced document parts. Information about the public key, which can be used to validate the signature, can be stored in the `<ds:KeyInfo>` element.

XML Encryption. With the capabilities provided by the XML Encryption Syntax and Processing Standard (XML Encryption) [9] the security goal of confidentiality can be achieved. XML Encryption includes the features of popular encryption solutions such as DES [10] or AES [11] into the XML world. Parts of XML documents can be encrypted and decrypted by using XML Encryption.

The `<enc:EncryptedData>` element introduces an encrypted part or subpart of the XML document. Obviously, this element is not added to the XML structure as a signature would be, but it replaces the encrypted cleartext. The three main components of an encrypted data block are the `<enc:EncryptionMethod>`, which specifies the cipher algorithm used for encryption and decryption, the `<ds:KeyInfo>` and the `<enc:CipherData>`. The latter contains the encrypted data itself, while `<ds:KeyInfo>` holds information about the key which has to be used for decryption of the ciphertext, this may also be an encrypted key.

Mostly, *hybrid encryption* [12] schemes are used i.e., the symmetric key is encrypted with the recipients public key. This aims at combining the speed advantages of symmetric encryption schemes with the absence of shared secrets offered by asymmetric schemes.

SAML. The Security Assertion Markup Language (SAML) standard is based on XML and defines a framework for delivery of issuer and security statements. Authentication and authorization statements can be modeled around subjects. Therefore the standard defines a `<saml:Assertion>` element which can nest security statements and additional information. Moreover SAML can be bound to underlying transport media and ships with some predefined usage protocols for example a protocol implementing the popular Single Sign On use case. The SAML pre-defined protocols offer XSD files exactly defining the message structure. Message integrity and validness are achieved by using optional digital signatures via the XML Signature standard, whereas confidentiality can be achieved by using optional encryption, according to the XML Encryption standard.

5 Get STUCK - The SAML Transportation Unit for Cryptographic Keys

After having listed and explained the existing structures and technologies in the previous section, the following sections explain how these structures can be utilized to enable secure key transportation in a fully SAML 2.0 compatible way.

5.1 Goals of the Contribution

The main focus while designing the STUCK solution was to provide a standardized way how to transport keys securely without breaking existing technologies. Therefore the proposal has to come with terms of XML and especially SAML compatibility, as well as major security goals when exchanging confidential key material.

SAML 2.0 Compatibility. The extension proposal focuses on a solution without modifications to the existing XML Schema definitions of SAML. The solution must not break existing implementations and has to be fully compatible to the SAML 2.0 specification (*Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0* [1]).

The SAML Standard provides flexible extension points within the `Assertion` element. As mentioned in chapter 4, these `Assertions` are one of the core features of SAML and used for security statements about subjects. An `Assertion` can be digitally signed so that integrity protection can be guaranteed. The proposed extension puts the key information inside this element to utilize this integrity protection. Together with the subject information and a digital signature over the element the key information is inextricably bound to an entity identified by the information of the `Subject` element.

Due to the fact that a common extension point is used, the additional key information neither breaks the SAML Schema (XSD files), nor influences `Assertion` or signature processing. Existing implementations do not need to be adjusted. The application logic behind has simply to deal with additional key information inside of an `Assertion`.

Security Goals. The proposed solution addresses multiple design goals valuable when dealing with key transportation mechanisms.

– **Confidentiality** - provided by utilizing XML Encryption on the key material (`EncryptedKey` elements are used)
– **Integrity** - provided by utilizing XML Signature on the transfered `Assertion`
– **Authentication** - provided by utilizing XML Signature on the transfered `Assertion` which contains `Subject` and `Issuer` elements (the information should be equal to the one of the Issuer's certificates)

5.2 Identification of Extension Point

In order to combine a SAML `Assertion` with cryptographic key information, the necessary extension point has to be identified.

Within an `Assertion` there can be any amount of `AttributeStatement` elements with an `unbounded` number of `Attribute` elements as child nodes. An `Attribute` element requires the presence of an XML attribute of type `Name` identifying the content and a sequence of zero to `unbounded` `AttributeValue` elements. An `AttributeValue` can hold content of type `anyType`, which weakens the strict schema definition and allows any well-formed XML data at this place. This is the extension point used by STUCK to integrate key information into an `Assertion`.

For clarity reasons, figure 1 provides a schematic illustration of a SAML `Assertion` containing key information.

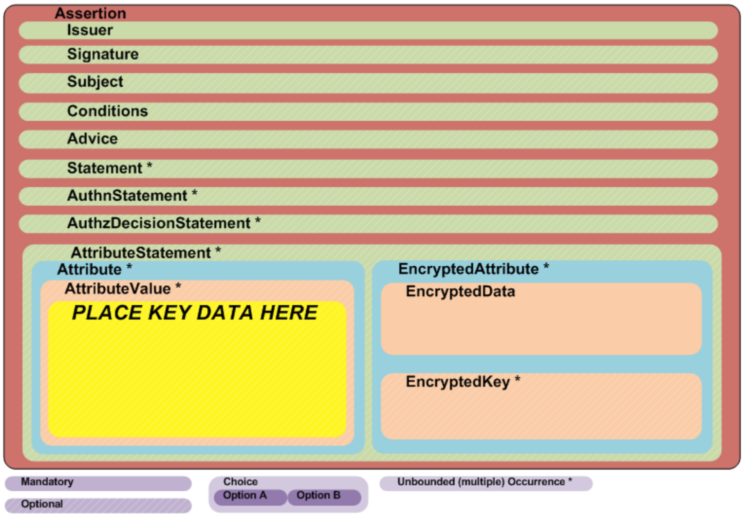


Fig. 1. Schematic illustration of a SAML `Assertion` with highlighted extension point

5.3 XML Key Data Structure

Additional to the identified extension point for including key data into a SAML `Assertion`, a suitable XML structure for holding cryptographic keys is required. For this purpose, XML Signature already offers versatile structures for keys and certificates. Supplemented by XML Encryption and its capabilities to define encrypted keys, all necessary structures for key distribution, management and transport are present yet. No additional structures have to be defined.

In the following the existing structures are briefly discussed. We mainly focus on a single element of the XML Signature Standard, the `ds:KeyInfo` element.

ds:KeyInfo. The `ds:KeyInfo` element, taken from the XML Signature Standard (here denoted as namespace `ds`), can be used to carry data somehow relevant for cryptographic keys. This includes several predefined data structures for storing information regarding e.g., key data for RSA, DSA, PGP or SPKI, as well as key related meta data like e.g., X.509 certificate data, key names, retrieval methods for externally located keys or general management information.

The following child elements from `ds:KeyInfo` are important for the STUCK proposal:

- **ds:KeyName**
  This element may contain a key identifer string which identifies key material.
- **ds:KeyValue**
  Originally defined to contain public keys used for signature verification, this element may also contain symmetric key material or any data structure defined in a namespace differnet from `ds`. The following child nodes are allowed in the schema:

- **ds:DSAKeyValue**
  Defines how to store DSA [13] public keys.
- **ds:RSAKeyValue**
  Defines how to store RSA [14] public keys.
- **any ##other**
  The `ds:KeyValue` element offers the option to include additional elements from arbitrary namespaces (other than the one refered to by `ds`). This allows to extend this element by including an `enc:EncryptedKey` element from the XML Encryption Standard (see below).

enc:EncryptedKey. This approach aims for a flexible solution able to carry all kinds of keys or key material, but the number of predefined key data structures in the `ds:KeyInfo` element is limited. This can be remedied by utilizing the extension point found in `ds:KeyInfo`: The element can easily be extended to allow key data usually unsuitable for these predefined data structures by adding elements from a differing namespace which provide a data structure for the desired keys. In this approach, an element from the XML Encryption Standard (here denoted by namespace `enc`), `enc:EncryptedKey`, is used.

This element offers support for transportation and storage of encrypted key material. Since it is obviously not advantageous to transport critical keys (such as private or secret keys) in an unencrypted manner, this element remains essential for a complete key distribution solution (such as Kerberos [15]).

5.4 Putting the Pieces Together - Extended SAML Assertion

After having identified the required XML structures and their respective extension points, the STUCK approach combines these into a single solution for secure key transportation in the SAML context.

The first step in the STUCK approach is to insert the key or key material which is to be transported into a `ds:KeyInfo` element. For this purpose, the previously identified extension point within `ds:KeyInfo` can be used to include an `enc:EncryptedKey` element which can hold any type of key or key material.

Note also that in case where a key of a predefined type for `ds:KeyInfo` should be transported (e.g., DSA or RSA keys as stated above), `enc:EncryptedKey` can be used instead of the predefined structures to utilize its inherent encryption features. Thus, the confidentiality of the key material itself is provided by XML Encryption.

The next step in the STUCK approach is to insert the `ds:KeyInfo` element including the `enc:EncryptedKey` element into a SAML `Assertion`. This is done by utilizing the previously defined extension point within a SAML `Assertion`, i.e., the `ds:KeyInfo` structure holding the key or key material is inserted into an `AttributeValue` element within the `Assertion`.

Thus, the associated key material is explicitly secured by the same means that protect the `Assertion` itself. This means integrity and authenticity of the key or key material within this extension point are implictly protected by the (optional) digital signature that protects the `Assertion`.

If further confidentiality beyond the content of an `Attribute` is necessary (as for example to obfuscate the structures behind an `Attribute`, so that not only the key material itself will be confidential, but also the accompanying additional information like e.g., key name or management data) the whole `ds:KeyInfo` element can be secured by applying encryption using the `enc:EncryptedData` element from the XML Encryption Standard before embedding it into the `Assertion`.

As an alternative, the application of `EncryptedAttribute` as child of `AttributeStatement` can be used instead of `Attribute`. This approach, however, does not have any benefits over the usage of XML Encryption to secure the `ds:KeyInfo` structure and is not considered any further in this paper.

An example SAML `Assertion` including key information according to our contribution is depicted in figure 7 in the appendix. The `Assertion` is extended with an `AttributeStatement` which holds an `Attribute` with `Name="desired Key"`. This `Attribute` contains an encrypted key as `AttributeValue`. The whole content (including the key element) is protected by a `Signature` refering to `URI="#referToMe"` which targets the `Assertion` itself. In addition the `Cipher Data` element following the `KeyInfo` element may contain data encrypted with the transferred key (e.g., key confirmation/information data etc.)

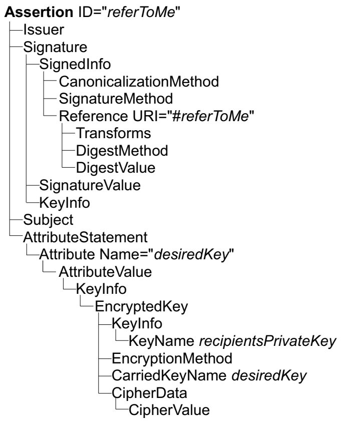


Fig. 2. Proof of concept SAML Assertion

5.5 Usage in the SAML Assertion Query and Request Protocol

The modified `Assertion` can be used with any of the predefined SAML protocols. Figure 3, gives a simplified example scenario on how a Key Requestor (KR) is able to obtain key material from a Key Server (KS) using only SAML compliant messages:

– KR sends a SAML Attribute Query to KS, authenticated with an XML signature. This request contains a reference to the requested key.

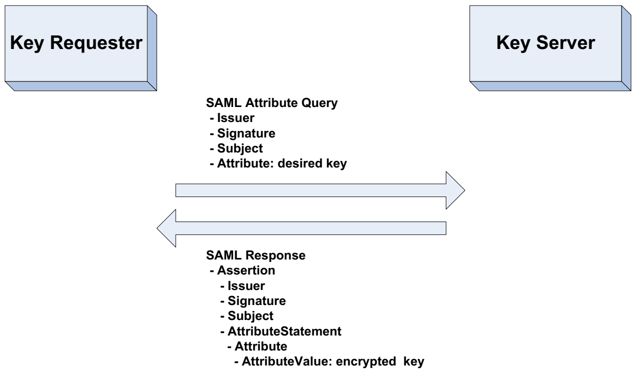


Fig. 3. Example scenario on the usage of the proposal

– After validating signature and request at Key Server side, KS may decide to deliver the requested key, in an encrypted form, to KR via the corresponding SAML Response, including an Assertion. For this purpose, the encrypted key is included in a SAML Attribute Statement within this Assertion to provide maximum compliance with the SAML standard. The requested key is encrypted with e.g., KR's public key preserving confidentiality.

The detailed messages (c.f., Figures 5 and 6), as well as a detailed explanation are listed in the appendix. We will come back to this scenario in more detail in the Case Study (c.f., section 6) when the solution is embedded to a real world application.

6 Case Study

A reference implementation of STUCK is implemented within a research project where key transport capabilities in conjunction with SAML are required.

6.1 *Sec*^{2} Research Project

The *Sec*^{2} research project[9] provides a hardware supported solution for secure mobile storage on public clouds. Therefore the user is able to define confidential parts of data which will then be encrypted before they are stored in the cloud. An underlying middleware handles the en-/decryption process transparently *before* the data leaves the device. For reasons of convenience the key management and distribution should be kept as automated as possible. Part of the solution is a publicly available trusted key server as depicted in figure 4.

This key server is used for key distribution to clients. The key distribution follows the principle of *hybrid encryption* - the key server wraps (encrypts) a

[9] `http://www.sec2.org`

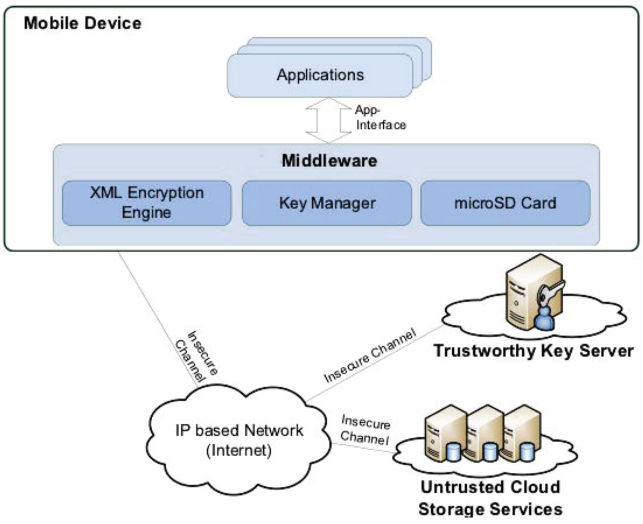


Fig. 4. *Sec*^{2} system architecture

symmetric secret key with a mobile device's (client) individual public key. The asymmetric keypair at the client side is bound to an entity (device owner) and delivered together with a special microSD Card that has to be installed at the mobile device. The microSD Card can be considered as a Smart Card that stores key material in an unextractable way. All cryptographic operations that utilize the key material have to be performed on the card. The corresponding public key is deposited at the key server. The client is able to unwrap (decrypt) the delivered key because she is in possession of the corresponding private key.

Since the whole communication between client and key server is SAML based the proposal of this paper is applicable and used for key transportation from the key server to its clients. Additionally, another major goal is to give up transport security and render its usage optional. The (wrapped) keys should be bound to SAML `Assertion`s to provide integrity and authentication at the same time. And all that has to be in line with the SAML specification(s). So to recap, the following requirements are given:

– (encrypted) keys have to be delivered from a key server to the client
– all critical parts (such as the encrypted key or authentication information) have to be authenticated and their integrity must be ensured
– deviations from the SAML standard have to be avoided

The solution within the project combines all requirements and integrates key transport mechanisms seamlessly into SAML without Schema validations or specification of extensions. The solution uses the approach introduced in section 5. In the following an example communication procedure is outlined:

1. The middleware fetches data from a public cloud storage and determines necessary key(s) for decryption - *Key X*.
2. After a lookup the middleware is informed by the microSD Card that it is not in possession of *Key X*.
3. A SAML attribute query including authorization data and identifier of the desired key is built (c.f., figure 5) and sent to the key server.
4. The key server validates the signature and checks for the corresponding access rights of the requesting client.
5. If all preconditions are met, the client's public key together with the key identifier of the desired key is passed to a Hardware Security Module (HSM) attached to the key server - The client's public key was deposited previously at the key server during client registration. The HSM is not directly accessible by the client and can only be contacted in case of sufficient access rights - only the key server can access the HSM.
6. The HSM wraps the key identified by the key identifier with the passed public key and returns the encrypted key to the key server.
7. The wrapped key is included in a signed SAML response (c.f., figure 6) and returned to the client.
8. The client verifies the SAML response, validates the digital signature, extracts the wrapped key and passes it to the microSD Card.
9. The microSD Card unwraps the desired key by utilizing the private key and stores it unextractable after successful unwrapping.
10. The middleware is now in posession of the necessary key and can proceed as if the key had been present at the beginning.

For further information you are invited to visit the project homepage and have a look at the papers and information material. Criticism, tips and feature requests are very welcome!

7 Conclusion

The proposed solution for identity bound key material and key information offers major enhancements to the Security Assertion Markup Language. Additional means for key transport can be skipped and instead directly mapped to the SAML level. A reference implementation is integrated within the *Sec*^{2} research project and will be soon available as open source. The proposal offers key management and distribution capabilities without schema violation, thus no adjustments to existing standards have to be made.

References

1. Cantor, S., Kemp, J., Philpott, R., Maler, E.: Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. Technical report (March 2005)
2. Hardjono, Klingenstein, Howlett, Scavo: SAML V2.0 Kerberos Web Browser SSO Profile Version 1.0. Technical Report (March 2010)

3. Hallam-Baker, P., Mysore, S.H.: XML Key Management Specification (XKMS 2.0). W3C Recommendation, W3C (June 2005)
4. Garfinkel, S.: PGP: Pretty Good Privacy. O'Reilly Media (November 1994)
5. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard) (May 2008)
6. Lawrence, K., Kaler, C.: WS-trust specification. Technical Report (March 2007)
7. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible Markup Language (XML) 1.0, 5th edn. World Wide Web Consortium, Recommendation REC-xml-20081126 (November 2008)
8. Eastlake, D., Reagle, J., Solo, D.: XML-Signature Syntax and Processing. XML Signature Working Group (2002)
9. Imamura, T., Dillaway, B., Simon, E.: XML Encryption Syntax and Processing. Technical Report, W3C XML Encryption Working Group (December 2002)
10. US Department of Commerce: Data Encryption Standard (DES) (December 1993)
11. National Institute for Science, Technology (NIST): Advanced Encryption Standard (FIPS PUB 197) (November 2001)
12. Wikipedia: Hybrid cryptosystem — Wikipedia, The Free Encyclopedia (2011) (Online; accessed March 12, 2012)
13. National Institute of Standards and Technology (NIST): NIST FIPS PUB 186 – Digital Signature Standard (May 1994)
14. Rivest, R., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM 21, 120–126 (1978)
15. Miller, S.P., Neuman, B.C., Schiller, J.I., Saltzer, J.H.: Kerberos Authentication and Authorization System. In: Project Athena Technical Plan (1988)

Appendix

A Using STUCK within protocols

STUCK can easily be integrated into existing SAML protocols. To demonstrate the usage please have a look at the corresponding messages in figures 5, 6 while reading.

As mentioned, a requester simply queries for a key by passing the key name as `Attribute` name (or as `AttributeValue` of a predefined `Attribute` for key queries `<saml:Attribute Name="requestKey"> <saml:AttributeValue> desiredKey</saml:AttributeValue></saml:Attribute>`). The KS will return the desired key (in an encrypted from) as `AttributeValue` carried by an `Assertion` inside a `Response`.

```
<samlp:AttributeQuery ID="myQueryID"
 Version="2.0" IssueInstant="2012-07-11T17:05:40Z"
 xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol">
  <saml:Issuer> ... </saml:Issuer>
  <ds:Signature> ... </ds:Signature>
  <saml:Subject> ... </saml:Subject>
  <saml:Attribute Name="requestKey">
    <saml:AttributeValue>desiredKey</saml:AttributeValue>
  </saml:Attribute>
</samlp:AttributeQuery>
```

Fig. 5. Proof of concept SAML AttributeQuery

```
<samlp:Response ID="myResponseID"
 Version="2.0" IssueInstant="2012-07-11T17:10:40Z"
 xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol">
  <saml:Assertion ID="referToMe"
   Version="2.0" IssueInstant="2012-07-11T17:10:40Z"
   xmlns:ds="http://.../xmldsig#">
    <saml:Issuer> ... </saml:Issuer>
    <ds:Signature> ... </ds:Signature>
    <saml:Subject> ... </saml:Subject>
    <saml:AttributeStatement>
      <saml:Attribute Name="desiredKey">
        <saml:AttributeValue> ... </saml:AttributeValue>
      </saml:Attribute>
    </saml:AttributeStatement>
  </saml:Assertion>
</samlp:Response>
```

Fig. 6. Proof of concept SAML Response

```
<saml:Assertion ID="referToMe"
 Version="2.0" IssueInstant="2012-03-01T12:59:48Z"
 xmlns:ds="http://.../xmldsig#" xmlns:enc="http://.../xmlenc#">
  <saml:Issuer> ... </saml:Issuer>
  <ds:Signature>
    <ds:SignedInfo>
      <ds:CanonicalizationMethod
       Algorithm="http://.../xml-exc-c14n#" />
      <ds:SignatureMethod
       Algorithm="http://.../xmldsig#rsa-sha1" />
      <ds:Reference URI="#referToMe">
        <ds:Transforms>  ... </ds:Transforms>
        <ds:DigestMethod Algorithm="http://.../xmldsig#sha1" />
        <ds:DigestValue> ... </ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue> ... </ds:SignatureValue>
    <ds:KeyInfo> ... </ds:KeyInfo>
  </ds:Signature>
  <saml:Subject> ... </saml:Subject>
  <saml:AttributeStatement>
    <saml:Attribute Name="desiredKey">
      <saml:AttributeValue>
        <ds:KeyInfo>
          <enc:EncryptedKey>
            <ds:KeyInfo>
              <ds:KeyName>recipientsPrivateKey</ds:KeyName>
            </ds:KeyInfo>
            <enc:EncryptionMethod Algorithm=".../xmlenc#rsa-1_5" />
            <enc:CarriedKeyName>desiredKey</enc:CarriedKeyName>
            <enc:CipherData>
              <enc:CipherValue> ... </enc:CipherValue>
            </enc:CipherData>
          </enc:EncryptedKey>
        </ds:KeyInfo>
      </saml:AttributeValue>
    </saml:Attribute>
  </saml:AttributeStatement>
</saml:Assertion>
```

Fig. 7. Proof of concept SAML Assertion

```
<complexType name="EncryptedElementType">
  <sequence>
    <element ref="xenc:EncryptedData"/>
    <element ref="xenc:EncryptedKey"
     minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>

<element name="Assertion" type="saml:AssertionType"/>
<complexType name="AssertionType">
  <sequence>
    ...
    <choice minOccurs="0" maxOccurs="unbounded">
      ...
      <element ref="saml:AttributeStatement"/>
    </choice>
  </sequence>
    ...
</complexType>

<element name="AttributeStatement" type="saml:AttributeStatementType"/>
<complexType name="AttributeStatementType">
  <complexContent>
    <extension base="saml:StatementAbstractType">
      <choice maxOccurs="unbounded">
        <element ref="saml:Attribute"/>
        ...
      </choice>
    </extension>
  </complexContent>
</complexType>

<element name="Attribute" type="saml:AttributeType"/>
<complexType name="AttributeType">
  <sequence>
    <element ref="saml:AttributeValue"
     minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Name" type="string" use="required"/>
  ...
</complexType>

<element name="AttributeValue" type="anyType" nillable="true"/>
<element name="EncryptedAttribute" type="saml:EncryptedElementType"/>
```

Fig. 8. (Stripped) XSD of a SAML Assertion - Source: *OASIS* (`http://docs.oasis-open.org/security/saml/v2.0/saml-schema-assertion-2.0.xsd`)

Improved Impossible Differential Attacks on Large-Block Rijndael[⋆]

Qingju Wang[1,2], Dawu Gu[1], Vincent Rijmen[2], Ya Liu[1], Jiazhe Chen[3], and Andrey Bogdanov[4]

[1] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, 200240, China

[2] KU Leuven, ESAT/COSIC and iMinds, Belgium

{qingju.wang,vincent.rijmen}@esat.kuleuven.be, dwgu@sjtu.edu.cn

[3] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, School of Mathematics, Shandong University, Jinan, 250100, China

[4] Technical University of Denmark, Department of Mathematics, Denmark

a.bogdanov@mat.dtu.dk

Abstract. In this paper, we present more powerful 6-round impossible differentials for large-block Rijndael-224 and Rijndael-256 than the ones used by Zhang et al. in ISC 2008. Using those, we can improve the previous impossible differential cryptanalysis of both 9-round Rijndael-224 and Rijndael-256. The improvement can lead to 10-round attack on Rijndael-256 as well. With $2^{198.1}$ chosen plaintexts, an attack is demonstrated on 9-round Rijndael-224 with $2^{195.2}$ encryptions and $2^{140.4}$ bytes memory. Increasing the data complexity to 2^{216} plaintexts, the time complexity can be reduced to 2^{130} encryptions and the memory requirements to $2^{93.6}$ bytes. For 9-round Rijndael-256, we provide an attack requiring $2^{229.3}$ chosen plaintexts, 2^{194} encryptions, and $2^{139.6}$ bytes memory. Alternatively, with $2^{245.3}$ plaintexts, an attack with a reduced time of $2^{127.1}$ encryptions and a memory complexity of $2^{90.9}$ bytes can be mounted. With $2^{244.2}$ chosen plaintexts, we can attack 10-round Rijndael-256 with $2^{253.9}$ encryptions and $2^{186.8}$ bytes of memory.

Keywords: block cipher, impossible differential attack, Rijndael, large block.

1 Introduction

Rijndael [11] is a block cipher designed by Joan Daemen and Vincent Rijmen built upon a Substitution Permutation Network (SPN). A subset of Rijndael variants has been standardized as Advanced Encryption Standard (AES) by

[⋆] This work was supported by the National Natural Science Foundation of China (No. 61073150), and in part by the Research Council K.U.Leuven: GOA TENSE, the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II.

the U.S. National Institute of Standards and Technology (NIST) [14] in 2002. Rijndael follows the design principles of Square [9]. In its full version, both the block and the key sizes can range from 128 to 256 bits in steps of 32 bits. For AES, the block size of Rijndael is restricted to 128 bits. This paper deals with non-AES Rijndael variants, that is, *large-block Rijndael-b*, with $b > 128$ indicating the block size and key size in bits.

AES is probably the most well-studied block cipher, having received about 15 years of extensive public scrutiny by now. Square attacks, impossible differential attacks, boomerang attacks, rectangle attacks and meet-in-the-middle attacks in both the single-key and related-key settings are just several prominent examples of cryptanalysis techniques applied to AES [1,4–7,12,17,20,22–24,26–28].

The large-block Rijndael is arguably less analyzed, being a highly relevant cipher though. Among others, an important motivation for the study of large-block Rijndael is the deployment of Rijndael-like permutations in the design of hash functions, Whirlwind [2] and SHA-3 finalist Grøstl [16] constituting some especially interesting instances. We mention here several multiset and integral cryptanalytic results [13,15,18,21], as well as impossible differential cryptanalysis [19,25]. In terms of the impossible differential cryptanalysis – the major object of our study in this paper – the best attack has been proposed by Zhang et al. [25] which cryptanalyzes 9-round Rijndael-224 and Rijndael-256 with 2^{209} and $2^{208.8}$ encryptions, respectively.

Impossible differential cryptanalysis, which was proposed by [3,8], is a widely used cryptanalytic technique. The attack starts with finding a certain input difference that can never result in a certain output difference, which makes up an *impossible differential*. Usually, impossible differentials have truncated input and output differences. By adding rounds before and/or after the impossible differential, one can collect pairs with certain plaintext and ciphertext differences. If there exists a pair that meets the input and output values of the impossible differential under some subkey bits, these bits must be wrong. In this way, we discard as many wrong keys as possible and exhaustively search the rest of the keys. The early abort technique is usually used during the key recovery phase, that is, one does not guess all the subkey bits at once, but guesses some subkey bits instead to discard some pairs that do not satisfy certain conditions step by step. In this case, we can discard the unwished pairs as soon as possible to reduce the time complexity.

Our Contributions. In this paper, we present more powerful 6-round impossible differentials for Rijndael-224 and Rijndael-256. Using these impossible differentials, we can improve the existing impossible differential cryptanalyses of both Rijndael-224 and Rijndael-256. In addition, the improvement can result in a 10-round attack on Rijndael-256.

Our impossible differentials for both Rijndael-224 and Rijndael-256 have more active bytes in the output difference and, therefore, the number of subkey bytes needed to be guessed during the key recovery phase can range with more options, while the probability for a pair of plaintexts to pass the test of sieving wrong pairs is higher compared to [25].

Table 1. Summary of Attacks on Rijndael-224 and Rijndael-256

Cipher	Number of Round	Complexity		Attack type	Source
		Time (EN)	Data(CP)		
Rijndael-224	7	2^{141}	$2^{130.5}$	Multiset	[18]
	7	2^{167}	2^{138}	Imp. Diff.	[19]
	7	$2^{113.4}$	$2^{93.2}$	Imp. Diff.	[25]
	9	$2^{196.5}$	$2^{196.5}$	Integral	[21]
	9	2^{209}	$2^{212.3}$	Imp. Diff.	[25]
	9	$2^{195.2}$	$2^{198.1}$	Imp. Diff.	sect. 4
	9	2^{162}	2^{208}	Imp. Diff.	sect. 4
	9	2^{130}	2^{216}	Imp. Diff.	sect. 4
Rijndael-256	7	$2^{128} - 2^{119}$	$2^{128} - 2^{119}$	Part. Sum	[13]
	7	2^{141}	$2^{130.5}$	Multiset	[18]
	7	2^{44}	6×2^{32}	Integral	[15]
	7	2^{182}	2^{153}	Imp. Diff.	[19]
	7	$2^{113.2}$	2^{93}	Imp. Diff.	[25]
	8	$2^{128} - 2^{119}$	$2^{128} - 2^{119}$	Integral	[15]
	9	2^{204}	$2^{128} - 2^{119}$	Integral	[15]
	9	$2^{174.5}$	$2^{132.5}$	Integral	[21]
	9	$2^{208.8}$	$2^{244.3}$	Imp. Diff.	[25]
	9	2^{194}	$2^{229.3}$	Imp. Diff.	subsect. 3.2
	9	$2^{159.1}$	$2^{237.3}$	Imp. Diff.	subsect. 3.3
	9	$2^{127.1}$	$2^{245.3}$	Imp. Diff.	subsect. 3.3
	10	$2^{253.9}$	$2^{244.2}$	Imp. Diff.	subsect. 3.4

CP: Chosen Plaintext; EN: Number of round encryptions

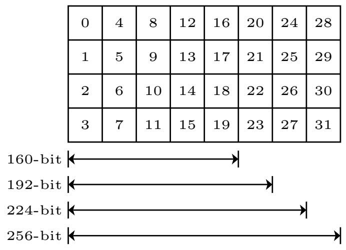
For 9-round Rijndael-256, utilizing the new impossible differential and depending on the number of subkey bytes needed to be guessed in key recovery phase, three improved attacks can be obtained. If we guess the same number of subkey bytes as [25], an attack can be mounted with reduced data complexity of $2^{229.3}$ Chosen Ciphertexts (CP), time complexity 2^{194} encryptions and memory complexity $2^{139.6}$ bytes respectively. In addition, if the number of subkey bytes need to guess is less than [25], given $2^{237.3}$ CP, we can attack 9-round Rijndael-256 with $2^{159.1}$ encryptions and $2^{115.3}$ bytes of memory. If the data complexity are increased to $2^{245.3}$ CP, the time and memory complexity can be significantly reduced to $2^{127.1}$ encryptions and $2^{90.9}$ bytes. Moreover, based on the same impossible differential, considering $2^{244.2}$ CP, we can even attack 10-round Rijndael-256 with $2^{253.9}$ encryptions and $2^{186.8}$ bytes of memory accesses. As for Rijndael-224, similarly three attacks can also be mounted on 9-round with lower complexity. With $2^{198.1}$ CP, an attack is demonstrated on 9-round Rijndael-224 with $2^{195.2}$ encryptions and $2^{140.4}$ bytes memory. Take 2^{208} CP, we can attack 9-round Rijndael-224 with 2^{162} encryptions and 2^{117} bytes memory. Increasing the data complexity to 2^{216} chosen plaintexts, the time complexity can be greatly reduced to 2^{130} encryptions and the memory requirements to $2^{93.6}$ bytes.

To the best of our knowledge, these results are the best impossible differential attacks on Rijndael-224 and Rijndael-256. We summarize our results for Rijndael-224 and Rijndael-256, as well as the major previous results in Table 1.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of Rijndael and introduces the notations used in this paper. In Section 3 we first derive a new 6-round impossible differential, and then present three improved impossible differential attacks on 9-round Rijndael-256. The attack can also be extended to 10-round Rijndael-256. Then in Section 4, after a new 6-round impossible differential distinguisher is presented, we mount three improved attacks on 9-round Rijndael-224. Finally, we conclude this paper in Section 5.

2 Description of Rijndael and Notations

Rijndael has N_r rounds, which can be 10, 12, or 14 depending on the key size. In Rijndael, both the text block and the key sizes can range for 128 up to 256 bits in steps of 32 bits. The 128-bit block version of Rijndael, with the key size 128, 192 or 256, is officially known as AES [14]. The plaintext, ciphertext, subkey, and all the intermediate data are represented by a $4 \times N_b$ `state matrix` of bytes, where N_b is the number of 32-bit words in the block. The byte indexing for the `state matrix` is shown in the left part of Figure 1. The key schedule



N_b	C_0	C_1	C_2	C_3
5	0	1	2	3
6	0	1	2	3
7	0	1	2	4
8	0	1	3	4

Fig. 1. Byte Index of the State Matrix and the Shift Offsets for Each Block Length N_b

derives $(N_r + 1)$ b-bit `RoundKey(RK)` from the master key, denoted from RK_0 to RK_{N_r}. The Expanded Key is a linear array of 4-byte words and is denoted by $W[N_b * (N_r + 1)]$. The first N_k words $W[0]\|W[1]\| \cdots \|W[N_k - 1]$ are directly initialised by the N_k words of the master key, while the remaining key words, $W[i]$ for $i \in \{N_k, \cdots, N_k * (N_r + 1) - 1\}$ are generated by the following algorithm:

```
if $(i \bmod N) = 0$ then $W[i] = W[i - N_k] \oplus f(W[i-1]) \oplus Rcon[i/N_k]$
else if $((N_k > 6)$ and $(i \bmod N = 4))$ then $W[i] = W[i - N_k] \oplus g(W[i-1])$
    else $W[i] = W[i - N_k] \oplus W[i-1]$
```

where $f, g : \{0,1\}^{32} \to \{0,1\}^{32}$ are nonlinear permutations, $Rcon$ denotes fixed constants depending on its input. `Roundkey` RK_i is given by the Round Key buffer words $W[N_b * i]$ to $W[N_b * (i + 1)]$.

The round function, which is repeated $(N_r - 1)$ times, involves four operations: `SubBytes(SB)`, `ShiftRows(SR)`, `MixColumns(MC)` and `AddRoundKey(ARK)`. The

`SubBytes` operation consists of the parallel application of a fixed 8-bit to 8-bit Sbox to each byte of the state. `ShiftRows` is a byte transposition that left shifts the rows of the state over different offsets. The shift offsets C_i of row i which depend on the block length N_b, are specified in the right part of Figure 1 for each block length of Rijndael. `MixColumns` is an (4×4) Maximum Distance Separable (MDS) matrix multiplication over $GF(2^8)$ for each column of the state. Obviously the branch number of this MDS matrix is five. `AddRoundKey` consists of the exclusive-or combination of the `RoundKey` with the intermediate state.

These $(N_r - 1)$ rounds are surrounded by an whitening layer consisting of `AddRoundKey` only, and the last round with `MixColumns` operation omitted. We also assume that this is the same case for the reduced Rijndael we are focusing on throughout this paper. Here we only give a brief description of Rijndael, for more detailed specification of the cipher, we refer to [10, 11].

We will also use the technique that the operations of `MixColumns` and `AddRoundKey` can be interchanged under some conditions [11]. Here we introduce some notations as well for later use in the following.

X_i	: the state of the i-th round;
ΔX_i	: the difference for state of the i-th round;
X_i^I	: the input state of the i-th round;
RK_i	: the subkey of the i-th round;
RK_i^*	: the value of the subkey of the i-th round after the inverse of the `MixColumns` operation;
X_i^{SB}	: the intermediate state after the `SubBytes` operation in the i-th round;
X_i^{SR}	: the intermediate state after the `ShiftRows` operation in the i-th round;
X_i^{MC}	: the intermediate state after the `MixColumns` operation in the i-th round;
X_i^W	: the intermediate state after the `AddRoundKey` operation with RK_i^* in the i-th round;
X_i^O	: the intermediate state after the `AddRoundKey` operation in the i-th round;
?	: an indeterminate difference.

Obviously, $X_i^I = X_{i-1}^O$ hold. Note that the operation of `AddRoundKey` will be represented as ARK^* throughout this paper when the Roundkey RK^* is used.

3 Improved Impossible Differential Attacks on Rijndael-256

In this section, we first give a new 6-round impossible differential for Rijndael-256 in Section 3.1. Based on this impossible differential and depending on the number of the subkey bytes need to guess during the key recovery phase, three improved 9-round impossible differential attacks compared to [25] will be presented in Subsection 3.2 and 3.3 respectively. Using the same impossible differential, we can extend it to an attack of 10-round Rijndael-256 in Subsection 3.4.

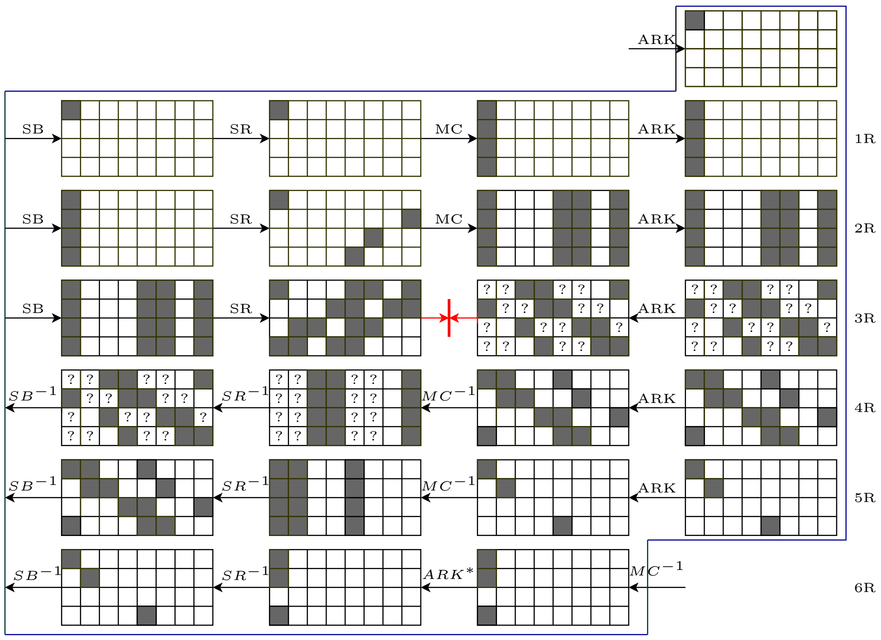


Fig. 2. The New 6-Round Impossible Differential of Rijndael-256

3.1 New 6-Round Impossible Differential on Rijndael-256

Assume we start with round 1 (denoted as 1R in Figure 2) and the input difference ΔX_1 has one active byte whereas the other bytes are zero, one illustration with the first byte active is depicted in Figure 2. Then request 2.5 rounds encryption from the `SB` operation in round 1 to the `SR` operation in round 3 to get the difference ΔX_3^{SR}. Consider the output difference with three nonzero bytes in the first column of the state, one option with the active bytes at (0,1,3) is shown in Figure 2, the other option has the active bytes at (0,2,3). Decrypt 3.5 rounds (as depicted from the operation `ARK`* in round 6 to the operation `ARK` in round 3) in order to get the difference ΔX_3^{MC}. Note there is no `AddRoundKey` operation in round 6 because the order of `MixColumns` and `AddRoundKey` operations can be interchanged as mentioned before in Section 2. For the third column of the state ΔX_3^{SR}, the number of nonzero bytes is one, while it is at most three for the nonzero bytes of ΔX_3^{MC} (it is indeterminate at byte 9). Since the branch number of the MDS matrix is five, there is a contradiction before and after the `MixColumns` operation. By similar reasoning, there is also a contradiction in the seventh column in the state before and after the `MixColumns` operation in round 3. Therefore, we make up a 6-round impossible differential for Rijndael-256.

There exist more active bytes in the output of the impossible differential compared to [25] (they has one byte), therefore we have more options in guessing the subkey bytes to meet the output of the impossible differential while adding extra rounds after the impossible differential distinguisher. There are three active bytes in one column after MC^{-1} at the bottom of the impossible differential, thus the number of the subkey bytes we need to guess in order to calculate the output after MC^{-1} can range from two to four, therefore three attacks can be mounted using this impossible differential.

3.2 9-Round Attack on Rijndael-256 with Lower Data Complexity

In this subsection we present the attacks on 9-round Rijndael-256 utilizing the 6-round impossible differential in Subsection 3.1. In our attack, we guess the same number (i.e. four) of subkey bytes of RK_8^* as [25] in the *key recovery phase*. As a result, 16 bytes of subkey RK_9 will have to be guessed to partially decrypt round 9 in order to calculate X_8^W (as shown in Figure 3), which will also provide a 128-bit condition for ciphertexts in the *data collection phase*. The number of active bytes at the end of the new impossible differential distinguisher will filter out more wrong pairs during the *key recovery phase*. Therefore, an improved attack with significantly reduced data complexity compared to [25] will be result in. The detailed procedures of the attack will be described as follows.

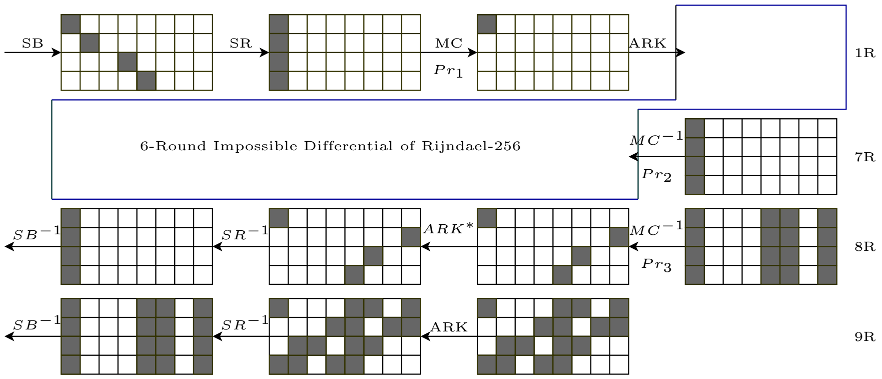


Fig. 3. Improved 9-Round Attack on Rijndael-256 with Lower Data Complexity

Data Collection. We first choose 2^n structures of plaintexts. In each structure the plaintexts range over all 32-bit values at bytes (0,5,14,19), while the other bytes can take certain fixed values. Each structure includes about $(2^{32})^2/2 = 2^{63}$ pairs of plaintexts, therefore $2^n \cdot 2^{63} = 2^{n+63}$ pairs of plaintexts will be prepared. Encrypt these pairs and keep the one whose ciphertext difference are zero at bytes (1,2,4,5,8,9,11,12,14,21,23,24,26,27,30,31). The probability of such ciphertexts is about $2^{-8 \cdot 16} = 2^{-128}$, thus the expected number of the remaining pairs after this phase is about $2^{n+63-128} = 2^{n-65}$.

The sieving of the ciphertexts can be done by birthday attack. As a result, the time complexity of this phase is about 2^{n+32}. In addition, we need $2^{n-65} \cdot 5 \cdot 32 = 2^{n-57.7}$ bytes memory to store these pairs.

Key Recovery. In order to check if the pairs generated in *data collection phase* satisfy the impossible differential in Figure 3, we need to guess certain bytes of subkey $(RK_9, RK_0, , RK_8^*)$ during the *key recovery phase.* The details are described in the following:

Step 1. For all the pairs of plaintext obtained in the data collection phase, we guess the 32-bit subkey $(RK_{9,0}, RK_{9,29}, RK_{9,22}, RK_{10,19})$ and partially decrypt round 9 to compute the first column of ΔX_8^W. Check if the differences at byte (1,2,3) are zero. If it is not the case, discard the pair. The probability of this event is 2^{-24}. After this step the expected number of remaining pairs is about $2^{n-65-24} = 2^{n-89}$.

Step 2. For every guess of the 32-bit subkey $(RK_{9,16}, RK_{9,13}, RK_{9,6}, RK_{9,3})$, we partially decrypt round 9 to compute the fifth column of ΔX_8^W. Check if the differences at byte (0,1,2) are zero. If it is not the case, discard the pair. The probability of this event is 2^{-24}. After this step the expected number of remaining pairs is about $2^{n-89-24} = 2^{n-113}$.

Step 3. For every guess of the 32-bit subkey $(RK_{9,20}, RK_{9,17}, RK_{9,10}, RK_{9,7})$, we partially decrypt round 9 to compute the sixth column of ΔX_8^W. Check if the differences at byte (0,1,3) are zero. If it is not the case, discard the pair. The probability of this event is 2^{-24}. After this step the expected number of remaining pairs is about $2^{n-113-24} = 2^{n-137}$.

Step 4. For every guess of the 32-bit subkey $(RK_{9,28}, RK_{9,25}, RK_{9,18}, RK_{9,15})$, we partially decrypt round 9 to compute the eight column of ΔX_8^W. Check if the differences at byte (0,2,3) are zero. If it is not the case, discard the pair. The probability of this event is 2^{-24}. After this step the expected number of remaining pairs is about $2^{n-137-24} = 2^{n-161}$.

Step 5. We need to guess the 32-bit of subkey $(RK_{0,0}, RK_{0,5}, RK_{0,14}, RK_{0,19})$ for all the remaining pairs, and partially encrypt round 1 to get the first column of ΔX_1^{MC}. Check if the difference at byte (1,2,3) are zero. If it is not the case, discard the pair. The probability of this event is about $4 \cdot (2^8 - 1)/2^{32} \approx 2^{-22}$. Thus after this step the remained pairs is about $2^{n-161-22} = 2^{n-183}$.

Step 6. For every guess of the 16-bit subkey $(RK_{8,0}^*, RK_{8,29}^*, RK_{8,22}^*, RK_{8,19}^*)$, partially decrypt round 8 to compute the first column of ΔX_7^W. Check if the differences at the third byte is zero. If it is correct, delete all the 32-bit subkey guesses of RK_8^* since such a differential is impossible, each subkey guess that proposes such a differential is a wrong key. After analyzing all the 2^{n-183} remaining pairs, if there still remains value of RK_8^*, output the 192-bit subkey guess of (RK_0, RK_8^*, RK_9) as the correct key. Our experiments provide the evidence that the probability of the pairs pass this step is about $Pr_2 = 2 \cdot 2^{-8} = 2^{-7}$.

The process steps of the *key recovery phase* above are described in Table 2, whereas the second column lists the bytes need to be guessed in the corresponding round for each step. The third column stands for the number of remained pairs after sieving in each step, and the time complexity of each step will be measured in the fourth column in Table 2. Note that when evaluating the time complexity of the recovery, it is measured by one round encryption. Similar tables will be adopted to describe the steps of the *key recovery phase* throughout this paper.

Table 2. Key Recovery Processes of the Attack on Rijndael-256 with lower Data Complexity

Step	Guessed Bytes	#Pairs Kept	Time Complexity
1	$RK_9 : 0, 29, 22, 19$	$2^{n-65-24} = 2^{n-89}$	$2^{32} \cdot 2^{n-65} \cdot 2/8 = 2^{n-35}$
2	$RK_9 : 16, 13, 6, 3$	$2^{n-89-24} = 2^{n-113}$	$2^{64} \cdot 2^{n-89} \cdot 2/8 = 2^{n-27}$
3	$RK_9 : 20, 17, 10, 7$	$2^{n-113-24} = 2^{n-137}$	$2^{96} \cdot 2^{n-113} \cdot 2/8 = 2^{n-19}$
4	$RK_9 : 28, 25, 18, 15$	$2^{n-137-24} = 2^{n-161}$	$2^{128} \cdot 2^{n-137} \cdot 2/8 = 2^{n-11}$
5	$RK_0 : 0, 5, 14, 19$	$2^{n-161-22} = 2^{n-183}$	$2^{160} \cdot 2^{n-161} \cdot 2/8 = 2^{n-3}$
6	$RK_8^* : 0, 29, 22, 19$	-	$2^{192} \cdot 2 \cdot [1 + (1 - 2^{-7}) + (1 - 2^{-7})^2 + \cdots + (1 - 2^{-7})^{2^{n-183}}]/8$

Analysis of the Attack. Take $n = 197.3$, after analyzing all the remaining pairs, there will be about $2^{192} \cdot (1-2^{-7})^{2^{n-183}} = 2^{-36.2}$ wrong subkeys of RK_0 left, we can get rid of the wrong subkeys by $2^{187.8}$ trail encryptions. Therefore the data complexity will be $2^{n+32} = 2^{229.3}$, the time complexity will be $2^{197.2}/9 \approx 2^{194}$ 9-round encryptions, the memory required is about $2^{139.6}$ bytes.

3.3 9-Round Attack on Rijndael-256 with Lower Time Complexity

We will use the same new 6-round impossible differential as the previous section, which helps to get rid of more pairs. As mentioned in Subsection 3.1, the number of subkey bytes need to be guessed in round 8 can be reduced compared to [25], i.e. two or three bytes of RK_8^*. Here we take two bytes for example. As a result, it will be the same case for round 9, which means fewer columns need to be decrypted. Meanwhile, 192 bits are zero for the ciphertexts, which provides an stronger condition of ciphertexts for sieving wrong pairs compared to 128 bits in [25]. As a result an improved attack with the time complexity greatly reduced can be mounted on 9-round Rijndael-256. Because of the similarity of the attack with the one in Subsection 3.2, only a brief description of this attack will be demonstrated as follows:

In the *data collection phase*, we take the same structures as in Subsect 3.2, thus 2^{n+63} pairs of plaintexts will be generated. there exists the 192-bit condition for ciphertext to discard wrong pairs, thus the expected number of the remaining pairs is $2^{n+63-192} = 2^{n-129}$ at the end of this phase.

In the *key recovery phase*, we only guess 8 bytes of RK_9, 4 bytes of RK_0 and 2 bytes of RK_8^* to check if the impossible differential will be satisfied for the remaining pairs. When filtering out wrong pairs, we obtain the probabilities that the pairs pass the tests in round 8, round 1 and round 7 are $Pr_3 = (2^{-24})^2 = 2^{-48}$, $Pr_1 = 4 \cdot (2^8 - 1)/2^{32} \approx 2^{-22}$ and $Pr_2 = 2 \cdot 2^{-8} \approx 2^{-7}$ respectively. The expected number of remaining pairs after this phase is about 2^{n-199}. The steps and the time complexity evaluation of this phase are given in Table 3. Take $n = 213.3$, the data complexity is $2^{n+32} = 2^{245.3}$ CP, the time complexity will be $2^{130.3}/9 \approx 2^{127.1}$ 9-round encryptions, the memory required is about $2^{90.9}$ bytes.

Table 3. Key Recovery Processes of the Improved Attack on Rijndael-256 with lower Time Complexity

Step	Guessed Bytes	#Pairs Kept	Time Complexity
1	$RK_9 : 0, 29, 22, 19$	$2^{n-129-24} = 2^{n-153}$	$2^{32} \cdot 2 \cdot 2^{n-129}/8 = 2^{n-99}$
2	$RK_9 : 28, 25, 18, 15$	$2^{n-153-24} = 2^{n-177}$	$2^{64} \cdot 2 \cdot 2^{n-153}/8 = 2^{n-91}$
3	$RK_0 : 0, 5, 14, 19$	$2^{n-177-22} = 2^{n-199}$	$2^{96} \cdot 2 \cdot 2^{n-177}/8 = 2^{n-83}$
4	$RK_8^* : 0, 29$	-	$2^{112} \cdot 2 \cdot [1 + (1 - 2^{-7}) + (1 - 2^{-7})^2 + \cdots + (1 - 2^{-7})^{2^{n-199}}]/16$

Moreover, as mentioned at the beginning of this subsection, it is also possible to guess three bytes of the subkey RK_8^* to calculate ΔX_7^W in order to check if the impossible differential can be satisfied. As a result 12 bytes of RK_9 have to be guessed to partially decrypt round 9 in the key recovery phase. In this case, the data complexity is about $2^{237.3}$ CP, the time complexity is about $2^{159.1}$ 9-round encryption, and the memory is about $2^{115.3}$ bytes.

3.4 10-Round Impossible Differential Attack on Rijndael-256

Based on the same impossible differential as in the previous subsection, we will extend two rounds backwards and forwards respectively, an attack on 10-round Rijndael-256 will be led with complexity less than exhaustive search. We adopt the 9-round attack with lower time complexity in Subsection 3.3 to act as our internal 9-round attack, on which we make some modification. In addition, we will take the key schedule into consideration. The brief attack will be given out as follows.

In the *data collection phase*, take 2^n structures of plaintexts, in which the plaintexts range over 128-bit values at bytes (0,3,4,5,9,12,14,16∼19,21,23,26, 30,31), while the other bytes can take certain fixed values. Each structure includes about $(2^{128})^2/2 = 2^{255}$ pairs of plaintexts, therefore $2^n \cdot 2^{255} = 2^{n+255}$ pairs of plaintexts are obtained. Encrypt these pairs and keep the one whose ciphertext difference are zero at bytes (1∼14,16,17,20,21,23,24,26,27,30,31). The probability of such ciphertexts is about $2^{-8 \cdot 24} = 2^{-192}$, thus the expected number of the remaining pairs after this phase is about $2^{n+255-192} = 2^{n+63}$.

In the *key recovery phase*, as in the 9-round attack in Subsection 3.3, 8 subkey bytes of RK_{10} should be guessed. Because of the extra round backward extension, 16 bytes of RK_0 and 4 bytes of RK_1 will also be guessed respectively.

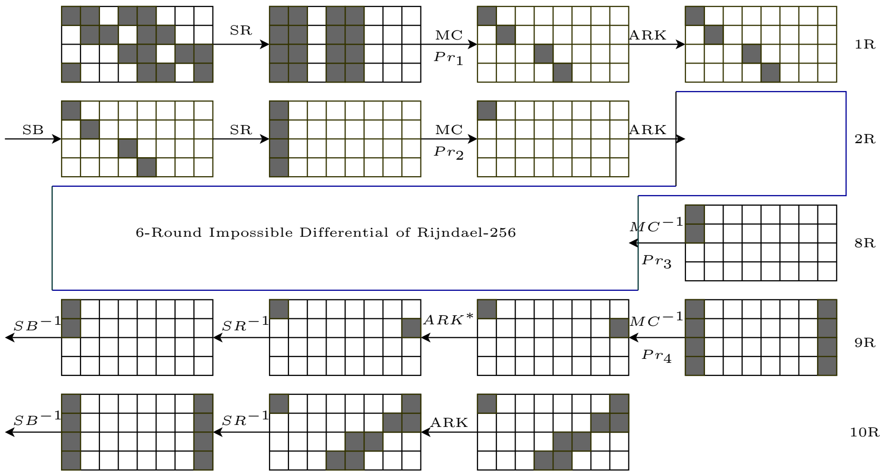


Fig. 4. 10-Round Impossible Differential Attack on Rijndael-256

At the end of this phase, the number of remaining pairs is 2^{n-103}. The process steps of this phase are described in Table 4. By the key schedule, we can calculate $RK_{0,29}$ from $RK_{0,0}$ and $RK_{1,0}$. $RK_{0,5}$ and $RK_{1,5}$ determine $RK_{1,1}$, then $RK_{1,1}$ together with $RK_{0,30}$ determine $RK_{0,1}$. Therefore, in order to recover the key, there are 14 bytes of RK_0 left to guess. We can take $n = 116.2$, from the *data collection phase* we know that the data complexity of the attack is $2^n \cdot 2^{128} = 2^{244.2}$ Chosen Ciphertext (CP). In the *key recovery phase*, after analyzing the remaining $2^{n-103} = 2^{13.2}$ pairs, the expected number of wrong subkeys is $2^{240} \cdot (1-2^{-7})^{2^{n-103}} \approx 2^{133.5}$. With about $2^{112} \cdot 2^{133.5} = 2^{245.5}$ trail encryptions, the correct key will be recovered. The time complexity is about $2^{257.2}/10 \approx 2^{253.9}$ 10-round encryptions. The memory required to store the pairs is about $2^{186.8}$ bytes.

Table 4. Key Recovery Processes of the Attack on 10-Round Rijndael-256

Step	Guessed Bytes	#Pairs Kept	Time Complexity
1	$RK_{10} : 0, 29, 22, 19$	$2^{n+63-24} = 2^{n+39}$	$2^{32} \cdot 2^{n+63} \cdot 2/8 = 2^{n+93}$
2	$RK_{10} : 28, 25, 18, 15$	$2^{n+39-24} = 2^{n+15}$	$2^{64} \cdot 2^{n+39} \cdot 2/8 = 2^{n+101}$
3	$RK_0 : 0, 5, 14, 19$	$2^{n+15-24} = 2^{n-9}$	$2^{96} \cdot 2^{n+15} \cdot 2/8 = 2^{n+109}$
4	$RK_0 : 4, 9, 18, 23$	$2^{n-9-24} = 2^{n-33}$	$2^{128} \cdot 2^{n-9} \cdot 2/8 = 2^{n+117}$
5	$RK_0 : 12, 17, 26, 31$	$2^{n-33-24} = 2^{n-57}$	$2^{160} \cdot 2^{n-33} \cdot 2/8 = 2^{n+125}$
6	$RK_0 : 16, 21, 30, 3$	$2^{n-57-24} = 2^{n-81}$	$2^{192} \cdot 2^{n-57} \cdot 2/8 = 2^{n+133}$
7	$RK_1 : 0, 5, 14, 19$	$2^{n-81-22} = 2^{n-103}$	$2^{224} \cdot 2^{n-81} \cdot 2/8 = 2^{n+141}$
8	$RK_9^* : 0, 29$	-	$2^{240} \cdot 2 \cdot [1 + (1-2^{-7}) + (1-2^{-7})^2 + \cdots + (1-2^{-7})^{2^{n-103}}]/16$

4 Improved Impossible Differential Attacks on Rijndael-224

In this section, we first give a new 6-round impossible differential of Rijndeal-224 (see Figure 5 in Appendix A). Utilizing the new 6-round impossible differential, we extend one round at the top and two rounds at the bottom to mount 9-round impossible differential attacks on Rijndael-224. As we can see there exist three active bytes at the bottom of the distinguisher, as a result the number of the subkey bytes need to guess in round 8 during the key recovery stage can range from two to four. Therefore three 9-round attacks on Rijndael-224 can be obtained respectively. First assume there are four bytes of subkey RK_8^* need to guess in order to check if the impossible differential distinguisher is satisfied during the key recovery phase, as depicted in Figure 6.

In the *data collection phase*, choose structures of 2^{32} plaintexts, in which the plaintexts take all possible 32-bit values at bytes (0,5,10,19) while the others take certain fixed values. Take $2^{166.1}$ structures, about $2^{229.1}$ pairs of plaintexts will be generated. Filter out the pairs whose ciphertext difference are not zero at byte (1∼5,8,10,13,16,19,23,26). Because of this 96-bit condition for ciphertexts, the expected number of remaining pairs is $2^{133.1}$ at the end of this phase.

In the process of *key recovery phase*, we need to guess 16 bytes of subkey RK_9, 4 bytes of subkey RK_8^* and 4 bytes of RK_0 to check if the 6-round of impossible differential is satisfied. While guessing the 4 bytes of RK_8^*, the probability that a pair can pass the test is about $Pr_2 = 2^{-8}$. The rest of the steps are similar to Subsection 3.2. At the end of this phase, there exist about $2^{133.1-96-22} = 2^{15.1}$ pairs.

After analyzing the remaining $2^{15.1}$ pairs, we can get rid of $2^{192} \cdot (1-2^{-8})^{2^{15.1}} \approx 2^{-6.3}$ wrong pairs. With about $2^{185.7}$ encryption trails the key can be recovered. The data complexity of this attack is about $2^{198.1}$ CP, the time complexity is about $2^{198.4}/9 \approx 2^{195.2}$ encryptions, and the memory we need for storing pairs is about $2^{133.1} \cdot 5 \cdot 32 = 2^{140.4}$ bytes.

As mentioned above, we can also guess three bytes of subkey RK_8^*, given 2^{208} CP, a 9-round attack can be mounted with the time complexity and memory about 2^{162} encryptions and 2^{117} bytes respectively. Moreover in the case that two bytes of RK_8^* are guessed, the data, time and memory complexity can be 2^{216} CP, 2^{130} encryptions and $2^{93.6}$ bytes respectively.

5 Conclusion

More powerful 6-round impossible differentials for both Rijndael-224 and Rijndael-256 are presented in this paper. Based on those, we significantly improve impossible differential attacks on both Rijndael-224 and Rijndael-256. The improvement can also result in a 10-round attack on Rijndael-256.

Acknowledgments. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

References

1. Bahrak, B., Aref, M.R.: A Novel Impossible Differential Cryptanalysis of AES. In: Proceedings of WEWoRC (2007)
2. Barreto, P.S.L.M., Nikov, V., Nikova, S., Rijmen, V., Tischhauser, E.: Whirlwind: a new cryptographic hash function. Des. Codes Cryptography 56(2-3), 141–162 (2010)
3. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 12–23. Springer, Heidelberg (1999)
4. Biham, E., Dunkelman, O., Keller, N.: Related-Key Impossible Differential Attacks on 8-Round AES-192. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 21–33. Springer, Heidelberg (2006)
5. Biryukov, A.: The Boomerang Attack on 5 and 6-Round Reduced AES. In: Dobbertin, H., Rijmen, V., Sowa, A. (eds.) AES 2005. LNCS, vol. 3373, pp. 11–15. Springer, Heidelberg (2005)
6. Biryukov, A., Khovratovich, D.: Related-Key Cryptanalysis of the Full AES-192 and AES-256. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 1–18. Springer, Heidelberg (2009)
7. Biryukov, A., Khovratovich, D., Nikolić, I.: Distinguisher and Related-Key Attack on the Full AES-256. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 231–249. Springer, Heidelberg (2009)
8. Borst, J., Knudsen, L.R., Rijmen, V.: Two attacks on reduced idea. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 1–13. Springer, Heidelberg (1997)
9. Daemen, J., Knudsen, L.R., Rijmen, V.: The Block Cipher Square. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997)
10. Daemen, J., Rijmen, V.: AES Proposal: Rijndael. In: 1st AES Conference, California, USA (1998)
11. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer (2002)
12. Demirci, H., Selçuk, A.A.: A Meet-in-the-Middle Attack on 8-Round AES. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 116–126. Springer, Heidelberg (2008)
13. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., Whiting, D.: Improved Cryptanalysis of Rijndael. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 213–230. Springer, Heidelberg (2001)
14. FIPS 197: Advanced Encryption Standard. Federal Information Processing Standards Publication 197, U.S. Department of Commerce/N.I.S.T (2001)
15. Galice, S., Minier, M.: Improving Integral Attacks Against Rijndael-256 Up to 9 Rounds. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 1–15. Springer, Heidelberg (2008)
16. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl - a SHA-3 candidate. Submission to NIST (2008), http://www.groestl.info
17. Gilbert, H., Minier, M.: A Collision Attack on 7 Rounds of Rijndael. In: AES Candidate Conference, pp. 230–241 (2000)
18. Nakahara Jr., J., de Freitas, D.S., Phan, R.C.-W.: New Multiset Attacks on Rijndael with Large Blocks. In: Dawson, E., Vaudenay, S. (eds.) Mycrypt 2005. LNCS, vol. 3715, pp. 277–295. Springer, Heidelberg (2005)

19. Nakahara Jr., J., Pavão, I.C.: Impossible-Differential Attacks on Large-Block Rijndael. In: Garay, J.A., Lenstra, A.K., Mambo, M., Peralta, R. (eds.) ISC 2007. LNCS, vol. 4779, pp. 104–117. Springer, Heidelberg (2007)
20. Kim, J., Hong, S., Preneel, B.: Related-Key Rectangle Attacks on Reduced AES-192 and AES-256. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 225–241. Springer, Heidelberg (2007)
21. Li, Y., Wu, W.: Improved Integral Attacks on Rijndael. Journal of Information Science and Engineering 27(6), 2031–2045 (2011)
22. Lu, J., Dunkelman, O., Keller, N., Kim, J.: New Impossible Differential Attacks on AES. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 279–293. Springer, Heidelberg (2008)
23. Lucks, S.: Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Keys. In: AES Candidate Conference, pp. 215–229 (2000)
24. Phan, R.C.W.: Impossible differential cryptanalysis of 7-round Advanced Encryption Standard (AES). Inf. Process. Lett. 91(1), 33–38 (2004)
25. Zhang, L., Wu, W., Park, J.H., Koo, B.W., Yeom, Y.: Improved Impossible Differential Attacks on Large-Block Rijndael. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 298–315. Springer, Heidelberg (2008)
26. Zhang, W., Wu, W., Feng, D.: New Results on Impossible Differential Cryptanalysis of Reduced AES. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 239–250. Springer, Heidelberg (2007)
27. Zhang, W., Wu, W., Zhang, L., Feng, D.: Improved Related-Key Impossible Differential Attacks on Reduced-Round AES-192. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 15–27. Springer, Heidelberg (2007)
28. Zhang, W., Zhang, L., Wu, W., Feng, D.: Related-Key Differential-Linear Attacks on Reduced AES-192. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 73–85. Springer, Heidelberg (2007)

A New 6-Round Impossible Differential of Rijndael-224 and 9-Round Attack with Lower Data Complexity

Assume we start with round 1 and there is only one nonzero byte of the input difference ΔX_1 whereas the other bytes are zero. One options is depicted in Figure 5 with nonzero byte at the first byte position. Then encrypt the input for 2.5 rounds from the `SB` operation in 1R to the `SR` operation in 3R to get the difference ΔX_3^{SR}. Given the output difference with three nonzero bytes in the first column, whereas the other bytes are zero. For Rijndael-224, the only option exists is given in Figure 5. After 3.5 rounds decryption (as depicted from the operation `ARK`* in round 6 to the operation `ARK` in round 3 in order to get the difference ΔX_3^{MC}). For the first column of the state X_3^{SR}, the number of nonzero bytes of ΔX_3^{SR} is one, while the maximum number of nonzero bytes of ΔX_3^{MC} is three. Since the branch number of the MDS matrix is five, there exists an contradiction. Therefore, we make up a 6-round impossible differential for Rijndael-224.

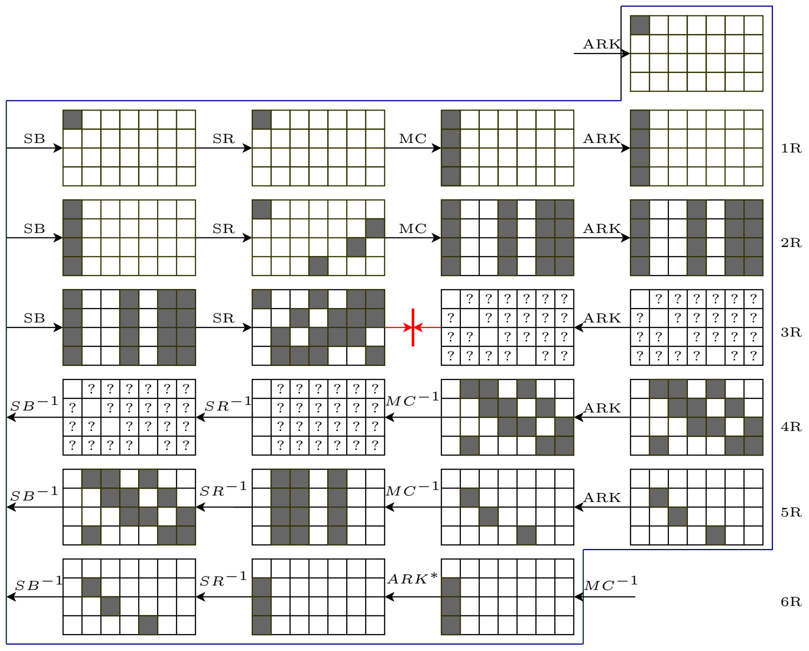


Fig. 5. The New 6-Round Impossible Differential of Rijndael-224

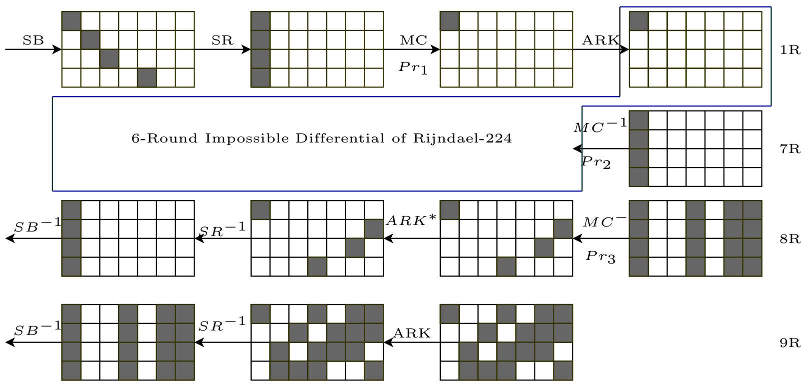


Fig. 6. The Improved 9-Round Attack on Rijndael-224 with lower Data Complexity

Cube Cryptanalysis of LBlock with Noisy Leakage[⋆]

Zhenqi Li[1], Bin Zhang[2], Yuan Yao[1], and Dongdai Lin[2]

[1] Institute of Software Chinese Academy of Sciences, Beijing
[2] SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, Beijing
{lizhenqi,zhangbin,yaoyuan}@is.iscas.ac.cn, ddlin@iie.ac.cn

Abstract. In this paper, we present some side channel cube attacks on LBlock, a lightweight block cipher proposed at ACNS 2011. It is shown that in the single bit leakage model, 14 bits of the secret key can be recovered with $2^{10.7}$ time and $2^{7.6}$ chosen plaintexts, captured the 44th state bit of the third round. In the Hamming weight leakage model, the full 80-bit key can be retrieved with only 2^{10} 32-round LBlock encryptions and $2^{11.1}$ chosen plaintexts, given the leakage of the second least significant bit (LSB) of the Hamming weight after the third round. We also provide a rigorous analysis on the error tolerance probabilities of our attacks and show that the full 80-bit key can be restored in 2^{30} 32-round LBlock encryptions with $2^{8.5}$ chosen plaintexts and at most 5.5% of the noisy leaked bits in the LSB of the Hamming weight after the second round. Many of the ideas in our attacks are applicable to other block ciphers as well.

Keywords: Cryptanalysis, Cube attack, Side channel attack, LBlock.

1 Introduction

RFID technology has been widely used in many real life applications nowadays, to name but a few, access control, parking management, identification, goods tracking and so on. To assure the security in such scenarios (weak computation ability, small storage space and strict power constraints), many lightweight block ciphers have been designed such as SEA [24], CGEN [21], HIGHT [9], DESL [13], PRESENT [2], KATAN/KTANTAN [4], MIBS [10], TWIS [18], LED [8], LBlock [28], Piccolo [23] and TWINE [25]. LBlock is proposed by Wu and Zhang at ACNS 2011. It is a 80-bit key Feistel-like block cipher with 64-bit block size

[⋆] This work was supported by the programs of the National Natural Science Foundation of China (Grant No. 60833008, 60603018, 61173134, 91118006, 61272476), the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDA06010701) and the National Grand Fundamental Research 973 Program of China(Grant No. 2013CB338002).

and 32 rounds. It is efficient not only in hardware but also in software. Current cryptanalysis [28,3,29,17,14] shows that it provides enough security margin against known cryptanalytic techniques.

Cube attack [5] was formally proposed by Dinur and Shamir at EUROCRYPT 2009. It is a generic key recovery attack, applicable to any cryptosystem in which at least a single bit can be represented by an unknown low degree multivariate polynomial in the secret and public variables. Side channel attacks typically exploit partial information leakage (time counting [19], physical probing, power consumption [11], electro-magnetic radiation [20] et al.) to recover the secret key. In this paper, we ignore these concrete issues and focus on the abstract leaked information hereafter. Side channel cube attack is a combination of cube and side channel attack. In such an attack, the adversary can obtain not only the plaintexts and the ciphertexts but also some restricted internal state information of the intermediate rounds. Dinur and Shamir applied side channel cube attack to AES and Serpent[6]. Since then, PRESENT is also found to be especially vulnerable to side channel cube attack [15,30,22].

In this paper, we present some side channel cube attacks on LBlock both in the single bit leakage model and Hamming weight leakage model. Combining the divide and conquer strategy with the cube-searching algorithm of large and complex algebraic systems, we can efficiently find good cubes based on the leaked information in the first few rounds. Based on the investigation of the mixing extent of the plaintext and key bits, we derive some important diffusion properties of the key bits, which can be utilized by a side channel cube attack. In real applications, the 0/1 value of the leaked information is likely to contain errors due to the noise and quantization problems. The original cube attack is extremely sensitive to errors. Therefore, we present a rigorous analysis of error tolerance of all our side channel cube attacks on LBlock. It is shown that the attacker can recover 50 key bits with 5.52% error tolerance and can obtain 56 key bits with 4.52% error tolerance in practical scenarios. Many of the ideas in our attacks are applicable to other block ciphers as well. Our attack results are summarized in Table 1.

Table 1. Our results on LBlock

Leakage model	Round	Leaked bit position	Data	Time	No. of key bits	Error tolerance
SB leakage	3	44th	$2^{7.6}$	$2^{10.7}$	14	2.42%
HW leakage	2	0th	$2^{8.5}$	$2^{15.9}$	50	5.52%
HW leakage	3	0th	$2^{10.0}$	$2^{17.0}$	67	2.40%
HW leakage	2	1st	$2^{8.9}$	$2^{16.4}$	56	4.52%
HW leakage	3	1st	$2^{11.1}$	$2^{17.2}$	70	0.62%

SB:Single bit. HW:Hamming weight.

The paper is organized as follows. The description of LBlock is provided in Section 2. We present a brief review of cube attacks in Section 3. In Section 4, the side channel cube attacks on LBlock based on the single bit leakage model and the Hamming weight leakage model are given respectively. The analysis of error tolerance on our side channel cube attacks is presented in Section 5. Finally, some conclusions are in Section 6.

2 Description of LBlock

2.1 Encryption Algorithm

The block length of LBlock is 64-bit, and the key length is 80-bit. It employs a variant Feistel structure and consists of 32 rounds. The encryption procedure is depicted in Fig. 1. Let $M = X_1||X_0$ denote the 64-bit plaintext, and then the data processing procedure can be expressed as follows.

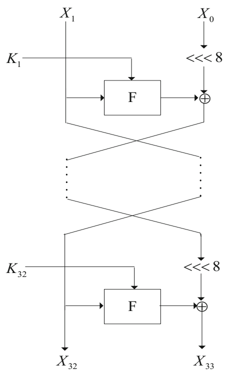


Fig. 1. Encryption procedure

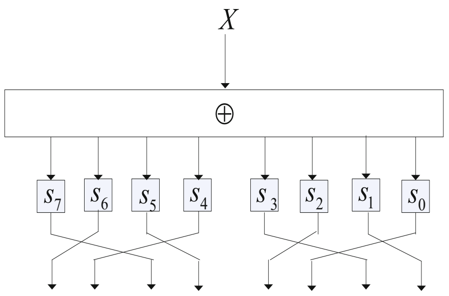


Fig. 2. Round Function F

1. For $i = 2, 3, ..., 33$, do

$$X_i = F(X_{i-1}, K_{i-1}) \oplus (X_{i-2} <<< 8)$$

2. Output $C = X_{32}||X_{33}$ as the 64-bit ciphertext
   (1) Round function F: The round function F is defined as follows, where S and P denote the confusion and diffusion functions which will be defined later.

$$F : \{0,1\}^{32} \times \{0,1\}^{32} \longrightarrow \{0,1\}^{32}$$
$$(X, K_i) \longrightarrow U = P(S(X \oplus K_i)$$

   Fig.2 shows the structure of round function F in detail.
   (2) Confusion function S: Confusion function S denotes the non-linear layer of round function F, and it consists of eight 4-bit S-boxes s_i in parallel.

$$S : \{0,1\}^{32} \longrightarrow \{0,1\}^{32}$$
$$Y = Y_7||Y_6||Y_5||Y_4||Y_3||Y_2||Y_1||Y_0 \longrightarrow Z =$$
$$Z_7||Z_6||Z_5||Z_4||Z_3||Z_2||Z_1||Z_0$$
$$Z_7 = s_7(Y_7), Z_6 = s_6(Y_6), Z_5 = s_5(Y_5), Z_4 = s_4(Y_4),$$
$$Z_3 = s_3(Y_3), Z_2 = s_2(Y_2), Z_1 = s_1(Y_1), Z_0 = s_0(Y_0).$$

(3) Diffusion function P: Diffusion function P is defined as a permutation of eight 4-bit words, and it can be expressed as the following equations.

$$P : \{0,1\}^{32} \longrightarrow \{0,1\}^{32}$$
$$Z = Z_7||Z_6||Z_5||Z_4||Z_3||Z_2||Z_1||Z_0 \longrightarrow U = U_7||U_6||U_5||U_4||U_3||U_2||U_1||U_0$$
$$U_7 = Z_6, U_6 = Z_4, U_5 = Z_7, U_4 = Z_5,$$
$$U_3 = Z_2, U_2 = Z_0, U_1 = Z_3, U_0 = Z_1.$$

2.2 Key Scheduling

The 80-bit master key K is stored in a key register and denoted as $K = k_{79}k_{78}k_{77}k_{76}......k_1k_0$. Output the leftmost 32 bits of current content of register K as round subkey K_1, and then operate as follows.

1. For $i = 1, 2, ..., 31$, update the key register K as follows:
   (a) $K <<< 29$
   (b) $[k_{79}k_{78}k_{77}k_{76}] = s_9[k_{79}k_{78}k_{77}k_{76}]$
   $[k_{75}k_{74}k_{73}k_{72}] = s_8[k_{75}k_{74}k_{73}k_{72}]$
   (c) $[k_{50}k_{49}k_{48}k_{47}k_{46}] \oplus [i]_2$, where $[i]_2$ is the binary form of i.
   (d) Output the leftmost 32 bits of current content of register K as round subkey K_{i+1}.

where s_8 and s_9 are two 4-bit S-boxes, and they are defined in Table 2.

3 Cube Attacks

Cube attack was formally introduced by Dinur and Shamir at Eurocrypt 2009 [5]. According to the comments and arguments of some researchers, cube attack has been studied under other names such as higher order differential attack [12] and algebraic IV differential attack [26][27] as well. Any output bit can be represented by a multivariate master polynomial $p(k_1, ..., k_n, v_1, ..., v_m)$ over $GF(2)$. The variables include secret variables k_i (key bits) and public variables v_i (plaintext bits in block ciphers and MACs, IV bits in stream ciphers).

In the off-line phase, the attacker chooses t_I randomly which can be indexed by the subset $I \subseteq \{1, ..., m\}$. The index of the subset I is defined as cube index. The polynomial can be represented as $p(k_1, ..., k_n, v_1, ..., v_m) = t_I \cdot p_{S(I)} + q(k_1, ..., k_n, v_1, ..., v_m)$ where $p_{S(I)}$ is called the superpoly of I in p. The polynomial is divided into two parts $p_{S(I)}$ and q by t_I. We assign all the public variables with all the possible combination of 0/1 values. Then the $p_{S(I)}$ becomes a polynomial including secret variables only. A maxterm of p is a term t_I such that $deg(p_{S(I)}) \equiv 1$, i.e. the superpoly of I in p is a linear polynomial which is not a constant. The $p_{S(I)}$ corresponding to the maxterm calls maxterm equation. Let the degree of the polynomial be d. According to Theorem. 1 of [5], Sum p that $t_I \in \{0,1\}^{d-1}$, then $\sum_{t_I \in \{0,1\}^{d-1}} p = \sum_{t_I \in \{0,1\}^{d-1}} (t_I p_{S(I)} + q) = p_{S(I)}$. We set other variables not involving in I to be constant (e.g. all 0s). Since the key can be chosen in this phase, it is easy to check whether a superpoly is linear by linear tests [1]. We choose secret variable vectors $x, y \in \{0,1\}^n$ randomly, and verify the equation $p_{S(I)}[0] + p_{S(I)}[x] + p_{S(I)}[y] = p_{S(I)}[x+y]$. The test always succeeds if $p_{S(I)}$ is linear. The attackers repeat the test N times, and a non-linear superpoly can be accepted with probability 2^{-N}. In this phase the attackers try to find as many maxterms and their equations as possible.

In the on-line phase, the secret key is fixed. The attackers choose plaintexts to get a system of linear equations and solve it to recover the key. The superpoly can be evaluated by summing over every possible assignment to its maxterm. If the degree of the maxterm is $d-1$, each sum requires 2^{d-1} evaluations of the derived polynomials.

4 Cube Attack on LBlock

In this section, we first analyze the diffusion properties of the key bits in the first few rounds of LBlock, then we give the side channel cube attack on LBlock based on the single bit leakage model and the Hamming weight leakage model respectively.

4.1 The Attack Round and Bit Position

In general, the choice of round r plays a vital role in the side channel cube attack on block ciphers. If r is small, e.g. $r = 1$ or $r = 2$, the complexity of chosen plaintext is minimized, but the number of key bits which can be recovered would be very few, the remaining key bits have to be exhaustively tested. If r gets bigger, the mixing of plaintexts and key bits will be much thorough, it is hard to find the maxterms with a low complexity since both the degree and the number of monomials will grow exponentially.

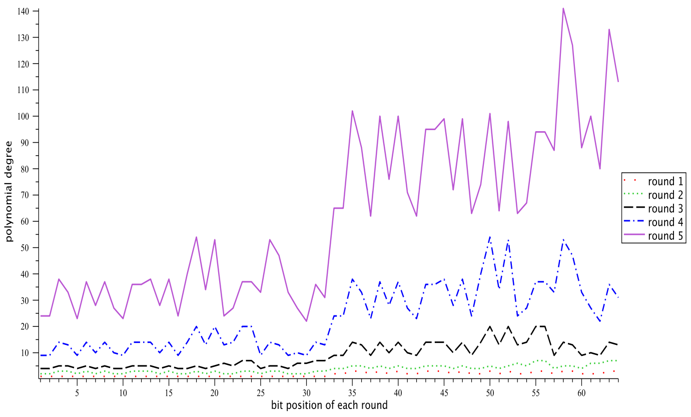


Fig. 3. Polynomial degree for the 64 bit positions of round 1,2,3,4 and 5

Fig.3 depicts the polynomial degree for the 64 bit positions[1] of round 1,2,3,4 and 5 of LBlock. It is easy to see that the degree of the polynomial grows exponentially with the increasing of LBlock round number. Besides, due to the intrinsic properties of Feistel network, only half of the state bits will be changed each round. Considering the efficiency of cube attack in which the polynomial degree of state bits should not be too large, we choose the right side state bits:$\{33, 34, ..., 64\}$ of the third round (or equivalently the left side state bits $\{1, 2, ..., 32\}$ of the fourth round). Experiments show that the polynomial degree of these state bits are around 15.

4.2 Diffusion of Key Bits

In order to recover more key bits in the single bit leakage model, it is helpful to observe the key bits diffusion of LBlock and select an appropriate leaked bit position. For each round, we keep two types of monomials, one involving a single key variable and the other only involving public variables. Then in the next round we compute the terms in the polynomial of the state bit which related to the selected terms only. And we discard other terms involving more than one key variables. In this way, we can control the number of the monomials in the first 3 rounds. We give an analysis of the initial key bits diffusion as follows.

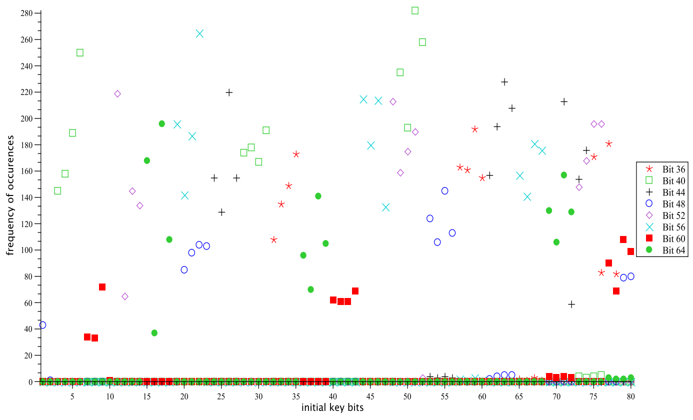


Fig. 4. Diffusion of key bits

Fig.4 exhibits the frequency of occurrences of the initial key bits in the multivariate polynomial of the state bits (36,36,40,44,48,52,56,60,64) in the third round (The distributions of initial key bits in other state bits are similar). It is easy to see that the distribution of the initial key bits is not uniform. For state bit 44 (symbol '+' indicates the frequency of occurrences), it only covers the following 16 key bits. The reason is that the speed of the key bits diffusion is not

[1] Bit positions from 1 to 64 corresponds to the state bits from right side to left side in each round of Fig.1.

fast. When the diffusion of key bits is complete in the latter round, the degree of polynomial will be much higher.

4.3 Attack in the Single Bit Leakage Model

In the single bit leakage model, we assume that only one internal state bit at certain round is available to the cryptanalyst. We employ the same strategy mentioned in section 4.2 to control the complexity of the multivariate polynomials. In this way, we can explicitly compute the multivariate polynomials for each state bit after the third round and treat the coefficient of the linear terms and constant terms as cubes. We apply the cube attack on each bit of the right side state bits:$\{33, 34, ..., 64\}$ of the third round. We found that at most 14 key bits can be recovered with $2^{7.6}$ chosen plaintexts[2] and $2^{10.7}$ time (This is the cost of Gaussian elimination [7] to solve the linear equation system). The corresponding leaked bit position is 44. The maxternms with linearly independent maxterm equations are listed in Table 3.(The attack results on other state bits are listed in Table 5 of appendix A).

Table 2. 14 maxterms and maxterm equations

Cube Indexes	Linear Equations	Cube Indexes	Linear Equations
$\{37, 39\}$	$1 + k_{56}$	$\{37, 40\}$	k_{55}
$\{39, 40\}$	$1 + k_{53}$	$\{29, 30, 31\}$	k_{71}
$\{29, 30, 45, 57\}$	k_{63}	$\{29, 30, 47, 57\}$	$1 + k_{61} + k_{62}$
$\{29, 31, 32, 57\}$	$k_{73} + k_{74}$	$\{29, 45, 46, 57\}$	$1 + k_{26} + k_{73}$
$\{30, 31, 32, 57\}$	$1 + k_{74}$	$\{30, 45, 47, 58\}$	k_{64}
$\{31, 46, 59, 60\}$	$k_{61} + k_{64}$	$\{32, 45, 47, 58\}$	k_{24}
$\{46, 47, 59, 60\}$	$k_{24} + k_{25}$	$\{31, 45, 46, 57, 60\}$	$1 + k_{27} + k_{63} + k_{64}$

Therefore, this attack can recover 14 key bits. Due to the incomplete diffusion of key bits in the third round of LBlock, the number of recovered key bits is very limited. In the following, we will show a better attack in the Hamming weight leakage model.

4.4 Attack in the Hamming Weight Leakage Model

In general, Hamming weight leakage model is a weaker leakage assumption, supported by many previously known practical results on side channel attacks. More precisely, let the internal state of the cipher $S = s_0...s_{L-1}$ be a binary string of length L. The Hamming weight of S is the number of bits with value 1 in the binary representation of S, which can be computed as $HW(S) = \sum_{i=0}^{L-1} s_i$ and has a value between 0 and L. Clearly, the Hamming weight can also be viewed as a boolean vector mapping $HW : \{0,1\}^L \rightarrow \{0,1\}^{|log_2 L+1|}$, where the LSB of

[2] Since the cube index size is different in Table 3, we choose the number of plaintexts as $N_{CP} = 3 \cdot 2^2 + 1 \cdot 2^3 + 9 \cdot 2^4 + 1 \cdot 2^5 \approx 2^{7.6}$.

$HW(S)$ is the exclusive OR of all bits from S, the most significant bit(MSB) of $HW(S)$ is the logic AND of all bits from S and each bit in between is a boolean function whose degree increases as the bit position gets closer to the MSB. We consider all bits starting from the LSB position towards the MSB position.

In our attacks, we assume that only one bit of the Hamming weight at certain round is available to the cryptanalyst. If the LSB of Hamming weight after the second round is leaked, 50 key bits can be retrieved with $2^{8.5}$ chosen plaintexts[3] and $2^{15.9}$ time. All these recovered key bits can be deduced from the 50 linearly independent maxterm equations listed in Table 3.

Table 3. 50 maxterms and maxterm equations

Cube Indexes	Maxterm Equations	Cube Indexes	Maxterm Equations
$\{1,33\}$	$k50+k51+k52$	$\{5,42\}$	$k57+k59+k60$
$\{9,53\}$	$1+k70+k71+k72$	$\{13,61\}$	$k78+k80$
$\{17,49\}$	$k66+k67+k68$	$\{21,57\}$	$1+k74+k76$
$\{25,37\}$	$1+k54+k56$	$\{29,45\}$	$k62+k63+k64$
$\{2,3,34\}$	$k49+k52$	$\{3,4,35\}$	$1+k49+k50$
$\{6,7,43\}$	$1+k57+k58$	$\{1,34,35\}$	$k52$
$\{2,33,34\}$	$1+k30+k51$	$\{3,33,34\}$	$1+k29$
$\{3,34,35\}$	$1+k31+k52$	$\{5,41,44\}$	$k59$
$\{5,43,44\}$	$k57$	$\{6,41,42\}$	$k35$
$\{6,42,43\}$	$k34$	$\{7,41,43\}$	$k33$
$\{9,54,55\}$	$k72$	$\{9,54,56\}$	$1+k71$
$\{10,11,54\}$	$1+k69+k72$	$\{10,53,54\}$	$1+k38$
$\{10,54,55\}$	$1+k39+k72$	$\{11,53,54\}$	$1+k37+k71$
$\{13,62,63\}$	$1+k80$	$\{13,62,64\}$	$1+k79$
$\{14,15,62\}$	$k77+k80$	$\{14,61,62\}$	$1+k42+k79$
$\{15,61,62\}$	$1+k41$	$\{17,50,51\}$	$k68$
$\{17,50,52\}$	$1+k67$	$\{18,19,51\}$	$1+k65+k66$
$\{21,58,59\}$	$1+k76$	$\{21,58,60\}$	$k75$
$\{22,23,58\}$	$1+k73+k76$	$\{25,38,39\}$	$1+k56$
$\{25,38,40\}$	$1+k55$	$\{26,27,38\}$	$1+k53+k56$
$\{26,37,38\}$	$k22+k55$	$\{27,37,38\}$	$k21$
$\{29,46,47\}$	$k64$	$\{29,46,48\}$	$k63$
$\{30,32,47\}$	$1+k61+k62$	$\{31,45,46\}$	$1+k27+k63$
$\{32,45,46\}$	$1+k26$	$\{32,46,47\}$	$k25+k64$
$\{37,38,40\}$	$1+k23$	$\{61,62,64\}$	$1+k43$

If the LSB of the Hamming weight after the third round is leaked, 67 key bits can be restored with $2^{10.0}$ chosen plaintexts and $2^{17.0}$ time, reducing the LBlock key searching space to 2^{13}, which is better than that of the second round. All these recovered key bits can be deduced from the 67 linearly independent maxterm equations listed in Table 6 of appendix A.

[3] Since the cube index size is different in Table 3, we choose the number of plaintexts as $N_{CP} = 8 \cdot 2^2 + 42 \cdot 2^3 \approx 2^{8.5}$.

We also extend the cube attack on the leakage of the second LSB of the Hamming weight after the second round. More precisely, the second LSB of the Hamming weight of S can be expressed as $[HW(S)]_2 = s_i \cdot s_j$ $(0 \leq i < j \leq L-1)$, where the degree of $[HW(S)]_2$ is 2 and suppose L to be an even number, which is often 8 due to the implementation of a block cipher on a 8-bit microcontroller. By utilizing this leaked bit, 56 key bits can be obtained with $2^{8.9}$ chosen plaintexts and $2^{16.4}$ time, leaving $k_1, k_2, ..., k_{19}$ and $k_{44}, k_{45}, ..., k_{48}$ to be recovered. All those recovered key bits can be deduced from the 56 linearly independent maxterm equations listed in Table 7 of appendix A.

Next, we further improve the above attack by using the divide-and-conquer strategy. If the second LSB of Hamming weight after the third round is leaked, we find that the multivariate polynomial of $[HW(S)]_2$ in the third round is hard to compute explicitly, since each s_i in the third round contains a lot of monomials with relatively high degree. We divide $[HW(S)]_2$ into L different groups and compute the explicit multivariate polynomial expression for each group. We use the same strategy mentioned in section 4.3 to search good cubes for each group, then apply the attack to LBlock based on all these cubes. Experimental results show that 70 key bits can be obtained with $2^{11.1}$ chosen plaintexts and $2^{17.2}$ time, please see Table 8 of appendix A for the 70 linearly independent equations.

These results show that the speed of the key bits diffusion of a block cipher has a great influence to the attack efficiency of the side channel cube attack. The attack under the single bit leakage model proves to be very effective to PRESENT [15,30] whose diffusion speed is very fast. However, it is not applicable for a block cipher with low diffusion speed, such as LBlock. It is inferior to the attack under the Hamming weight leakage model, which proves to be more efficient for both PRESENT [22] and LBlock.

5 Error Tolerance Side Channel Cube Attack

In real applications, the 0/1 value of the leaked information is likely to contain errors due to the noise and quantization problems. The original attack is extremely sensitive to errors, since it typically sums (modulo 2) lots of 0/1 values of a cube to get a single linear equation and repeats the summation over a number of different cubes to derive all the equations. In this section, we first give a brief introduction of Dinur-Shamir error correcting model [6] and point out its limitations when applying in the side channel cube attacks. Then we apply a modified version of Dinur-Shamir model to our attacks on LBlock

5.1 Dinur-Shamir Model

In the basic model, each leaked bit has three possible values: 0, 1 and $\perp$, where $\perp$ indicates a problematic measurement which cannot be relied upon. This model is closely related to erasure codes [16], in which the recipient of some communication knows which of the received bits are correct and which bits might have been flipped. Such flipped bits can be set as new variables in linear equations to overcome the uncertainty.

More precisely, let ϵ be the fraction of the $\perp$ values among all the measurements (leaked bits). Let n be the number of secret key variables. It is assumed that the errors are uniformly distributed and the leakage function is a d-random multivariate polynomial. The attacker chooses a big cube with $k \geq d + \log_d^n$ public variables. In the off-line phase, the attacker compute all the coefficients of all the $\binom{k}{d-1}$ linear equations which are determined by summing over all the possible subcubes of dimension $d-1$ in the big cube of dimension k. In the on-line phase, the attacker obtains 2^k leaked bits. Out of the 2^k values, $\epsilon \cdot 2^k$ values are $\perp$ due to the uncertainty (noise) in the measurement of the leakage function. The attacker assigns a new variable y_j to each one of these unknown values and sums both the known 0/1 values and the unknown y_j variables over each one of the $\binom{k}{d-1}$ overlapping subcubes of the big cube. The result of each summation is the sum of a subset of the y_j's, plus 0 or 1. The attacker treats all these unknown variables as the new key variables and obtains a system of $\binom{k}{d-1}$ linear equations in the $\epsilon \cdot 2^k + n$ variables y_j and k_i.

In order to solve the random looking linear system, the number of linear equations $\binom{k}{d-1}$ should be larger than $\epsilon \cdot 2^k + n$. That is $\binom{k}{d-1} \geq \epsilon \cdot 2^k + n$, then we can derive $\epsilon \leq \frac{\binom{k}{d-1} - n}{2^k} \approx \frac{1}{\sqrt{\Pi \cdot (d-1)}}$. In feasible attacks $k = 2(d-1) < 50$, and thus $\frac{1}{\sqrt{\Pi \cdot (d-1)}}$ is bigger than $\frac{1}{\sqrt{\Pi \cdot 25}} \approx 0.11$. Consequently, the attacker can find the complete key even when 11% of the leaked bits are too noisy to measure accurately.

However, considering the efficiency and feasibility, the side channel cube attacks are often applied to the early rounds of some cipher. The number of linear equations we can obtained is very limited. The assumption that the leakage function is a d-random multivariate polynomial, made in the model is thus not applicable in the real scenario. Therefore, we applied a modified version of Dinur-Shamir model to the attacks on LBlock in the following.

5.2 Side Channel Cube Attacks on LBlock with Noisy Leakage

In our side channel cube attack on LBlock, the number of measurements can be expressed as $N^*_{CP} = \sum_{i=1}^{m} n_i \cdot 2^{d_i}$, where n_i is the number of cubes with size d_i and m is the number of different sizes of cubes. We assume that the errors are uniformly distributed, then out of the N^*_{CP} values, $\epsilon \cdot N^*_{CP}$ values are $\perp$ due to the noise. We then obtained a system of L linear equations in the $\epsilon \cdot N^*_{CP} + n$ variables, where n is the number of the key variables and L is the number of linear equations we obtained in the off-line stage, satisfying $L = \sum_{i=1}^{m} n_i$.

In order to solve the random looking linear system, it is required that $L \geq \epsilon \cdot N^*_{CP} + n$. Considering the matrix of the linear system might not be nonsingular, we add a modifying factor θ representing the fraction of linear equations which is linearly dependent to other equations, thus modify the inequation to $L \cdot (1-\theta) \geq \epsilon \cdot N^*_{CP} + n$, therefore, we can derive

$$\epsilon \leq \frac{L \cdot (1-\theta) - n}{N^*_{CP}}.$$

We can thus recover the key when at most $(L \cdot (1-\theta) - n)/N^*_{CP}$ fraction of the leaked bits are $\perp$. In the single bit leakage attacks on the 44th of the third round in Section 4.3, we totally get $L = 172$ linear equations, which contain a lot of linearly dependent equations. After canceled all these linearly dependent equations, we obtained 14 maxterms and the corresponding linearly independently maxterm equations listed in Table 3. However, considering the random existence of new variable y_j when summing over the leakage bit for a cube, we can use those linearly dependent equations to solve the new linear system. The value of N^*_{CP} is different from that of N_{CP} in Section 4, since it count all those cancelled maxterms, thus $N^*_{CP} = 1 \cdot 2^1 + 3 \cdot 2^2 + 4 \cdot 2^3 + 50 \cdot 2^4 + 73 \cdot 2^5 + 41 \cdot 2^6 = 5806$. There exists $n = 14$ key variables in all the 172 linear equations. Given $\theta = 0.1$, we can get $\epsilon \leq 2.42\%$. Consequently, the attacker can retrieve 14 key bits even when 2.42% of the leaked bits are too noisy to measure accurately. We also apply the same model to other leakage attacks in section 4, the results are summarized in Table 4.

Table 4. Analysis of error tolerance of our attacks

Leakage model	Round	Bit position	L	n	N^*_{CP}	ϵ
SB leakage	3	44th	172	14	5806	$\leq 2.42\%$
HW leakage	2	0th	188	50	2160	$\leq 5.52\%$
HW leakage	3	0th	1199	67	42200	$\leq 2.40\%$
HW leakage	2	1st	868	56	16024	$\leq 4.52\%$
HW leakage	3	1st	13952	70	1990904	$\leq 0.62\%$

SB:Single bit. HW:Hamming weight.

From Table 4, it is easy to see that the error tolerance of attacks in the second round is higher than that of the third round, since the cube size in the second round is relatively smaller than that of the third round. This subtle difference will lead to a big gap between N^*_{CP} of the second round and the third round. The reason is that the high degree of the multivariate polynomial of the leakage bit will lead to the exponential increase in N^*_{CP}. Consequently, in our leakage model, it is strongly recommend that the attack should base on a polynomial whose degree is as low as possible in order to get higher error tolerance. Another way to increase the error tolerance is to obtain more linear equations in the off-line stage of cube attack.

6 Conclusion

In this paper, we have presented several side channel cube attacks on LBlock under the novel assumption of obtaining accurate leaked bits without any noise. Based on the Hamming weight leakage model, we can obtain all the 80-bit key with 2^{10} 32-round LBlock encryptions and $2^{11.1}$ chosen plaintexts with a special leaked bit of the second LSB of Hamming weight after the third round. We also present an rigorous analysis on the error tolerance of all our attacks on LBlock. Under our leakage model, we can recover all the 80-bit key with 2^{30} 32-round LBlock encryptions and $2^{8.5}$ chosen plaintexts when at most 5.52% of the leaked bits in the LSB of Hamming weight after the second round are too noisy to measure accurately in practical scenarios. To our knowledge, this is the most efficient cube cryptanalysis of LBlock. In order to get a higher error tolerance, we need to get more linear equations and the corresponding cubes' size should be as small as possible. Our further research will focus on how to enhance the error tolerance, how to identify the multivariate polynomial in deeper round of block cipher and identify cipher structures against side channel cube attacks.

References

1. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. Journal of Computer and System Sciences 47, 549–595 (1993)
2. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
3. Courtois, N.T., Sepehrdad, P., Sušil, P., Vaudenay, S.: ElimLin Algorithm Revisited. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 306–325. Springer, Heidelberg (2012)
4. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
5. Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009)
6. Dinur, I., Shamir, A.: Side Channel Cube Attacks on Block Ciphers. Cryptology ePrint Archive. Report 2009/127 (2009)
7. Farebrother, R.W.: Linear Least Squares Computations. STATISTICS: Textbooks and Monographs. Marcel Dekker (1988) ISBN 978-0-8247-7661-9
8. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED Block Cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)
9. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B.-S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: HIGHT: A New Block Cipher Suitable for Low-Resource Device. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)
10. Izadi, M., Sadeghiyan, B., Sadeghian, S.S., Khanooki, H.A.: MIBS: A New Lightweight Block Cipher. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 334–348. Springer, Heidelberg (2009)
11. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
12. Lai, X.: Higher Order Derivatives and Differential Cryptanalysis. In: Communications and Cryptography: Two Sides of One Tapestry, p. 227 (1994)

13. Leander, G., Paar, C., Poschmann, A., Schramm, K.: New Lightweight DES Variants. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 196–210. Springer, Heidelberg (2007)
14. Zhao, L., Nishide, T., Sakurai, K.: Differential Fault Analysis of Full LBlock. In: Schindler, W., Huss, S.A. (eds.) COSADE 2012. LNCS, vol. 7275, pp. 135–150. Springer, Heidelberg (2012)
15. Yang, L., Wang, M., Qiao, S.: Side Channel Cube Attack on PRESENT. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 379–391. Springer, Heidelberg (2009)
16. Luby, M.G., Mitzenmacher, M., Shokrollahi, M.A., Spielman, D.A.: Efficient erasure correcting codes. IEEE Transactions on Information Theory 47(2), 569–584 (2001)
17. Marine, M., Maria, N.P.: A related key impossible differential attack against 22 rounds of the lightweight block cipher LBlock. Information Processing Letters 112(16), 624–629 (2012)
18. Ojha, S.K., Kumar, N., Jain, K., Sangeeta: TWIS - A Lightweight Block Cipher. In: Prakash, A., Sen Gupta, I. (eds.) ICISS 2009. LNCS, vol. 5905, pp. 280–291. Springer, Heidelberg (2009)
19. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
20. Quisquater, J.J., Samyde, D.: A new tool for non-intrusive analysis of smart cards based on electro-magnetic emissions: the SEMA and DEMA methods(EB/OL). Eurocrypt rump session (2000)
21. Robshaw, M.J.B.: Searching for Compact Algorithms: CGEN. In: Nguyen, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 37–49. Springer, Heidelberg (2006)
22. Shekh Faisal, A.-L., Mohammad, R.R., Willy, S., Jennifer, S.: Extended Cubes: Enhancing the cube attack by Extracting Low-Degree Non-linear Equations. In: Cheung, B., Hui, L.C.K., Sandhu, R., Wong, D.S. (eds.) ASIACCS 2011, pp. 296–305 (2011)
23. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: *Piccolo*: An Ultra-Lightweight Blockcipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 342–357. Springer, Heidelberg (2011)
24. Standaert, F.-X., Piret, G., Gershenfeld, N., Quisquater, J.-J.: SEA: A Scalable Encryption Algorithm for Small Embedded Applications. In: Domingo-Ferrer, J., Posegga, J., Schreckling, D. (eds.) CARDIS 2006. LNCS, vol. 3928, pp. 222–236. Springer, Heidelberg (2006)
25. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: TWINE: A Lightweight, Versatile Block Cipher. In: ECRYPT Workshop on Lightweight Cryptography (November 2011)
26. Vielhaber, M.: Breaking ONE.TRIVIUM by AIDA and Algebraic IV Differential Attack. IACR Cryptology ePrint Archive, 413 (2007)
27. Vielhaber, M.: AIDA Breaks (BIVIUM A and B) in 1 Minute Dual Core CPU Time. IACR Cryptology ePrint Archive, 402 (2009)
28. Wu, W., Zhang, L.: LBlock: A Lightweight Block Cipher. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 327–344. Springer, Heidelberg (2011)
29. Liu, Y., Gu, D., Liu, Z., Li, W.: Impossible Differential Attacks on Reduced-Round LBlock. In: Ryan, M.D., Smyth, B., Wang, G. (eds.) ISPEC 2012. LNCS, vol. 7232, pp. 97–108. Springer, Heidelberg (2012)
30. Zhao, X.J., Wang, T., Guo, S.Z.: Improved Side Channel Cube Attacks on PRESENT. Cryptology ePrint Archive. Report 2011/165 (2011)

A Results of Side Channel Cube Attacks on LBlock

Table 5. Recovered key bits on other state bits

Bit Position	Recovered Key Bits
33	$k_{32}, k_{33}, k_{34}, k_{35}, k_{57}, k_{58}, k_{59}, k_{60}, k_{67}, k_{68}, k_{77}, k_{75} + k_{76}$
34	$k_{34}, k_{35}, k_{57}, k_{58}, k_{59}, k_{60}, k_{67}, k_{68}, k_{77}, k_{32} + k_{33}, k_{65} + k_{66}$
35	$k_{32}, k_{33}, k_{34}, k_{35}, k_{57}, k_{58}, k_{59}, k_{60}, k_{65}, k_{67}, k_{68}, k_{77}, k_{78}, k_{75} + k_{76}$
36	$k_{34}, k_{35}, k_{57}, k_{58}, k_{59}, k_{60}, k_{67}, k_{75}, k_{77}, k_{32} + k_{33}$
37	$k_{5}, k_{6}, k_{28}, k_{29}, k_{30}, k_{31}, k_{49}, k_{50}, k_{51}, k_{52}, k_{75}, k_{73} + k_{74}$
38	$k_{3}, k_{5}, k_{6}, k_{30}, k_{31}, k_{28}, k_{29}, k_{49}, k_{50}, k_{51}, k_{52}, k_{75}, k_{76}$
39	$k_{5}, k_{28}, k_{29}, k_{30}, k_{31}, k_{49}, k_{50}, k_{51}, k_{52}, k_{3} + k_{4}, k_{73} + k_{74}$
40	$k_{5}, k_{6}, k_{28}, k_{29}, k_{30}, k_{31}, k_{49}, k_{50}, k_{51}, k_{52}, k_{73}, k_{75}, k_{76}$
41	$k_{24}, k_{25}, k_{26}, k_{27}, k_{55}, k_{61}, k_{62}, k_{63}, k_{64}, k_{73}, k_{74}, k_{53} + k_{54}$
42	$k_{26}, k_{27}, k_{55}, k_{56}, k_{61}, k_{62}, k_{63}, k_{64}, k_{24} + k_{25}, k_{71} + k_{72}$
43	$k_{24}, k_{25}, k_{26}, k_{27}, k_{55}, k_{56}, k_{61}, k_{62}, k_{63}, k_{64}, k_{73}, k_{74}, k_{53} + k_{54}, k_{71} + k_{72}$
44	$k_{24}, k_{25}, k_{26}, k_{27}, k_{53}, k_{55}, k_{56}, k_{61}, k_{62}, k_{63}, k_{64}, k_{71}, k_{73}, k_{74}$
45	$k_{1}, k_{2}, k_{20}, k_{21}, k_{22}, k_{23}, k_{53}, k_{54}, k_{55}, k_{56}, k_{63}, k_{64}, k_{79} + k_{80}$
46	$k_{1}, k_{2}, k_{20}, k_{21}, k_{22}, k_{23}, k_{53}, k_{54}, k_{55}, k_{56}, k_{61}, k_{63}, k_{64}$
47	$k_{1}, k_{2}, k_{20}, k_{21}, k_{22}, k_{23}, k_{53}, k_{54}, k_{55}, k_{56}, k_{63}, k_{79}$
48	$k_{22}, k_{23}, k_{53}, k_{54}, k_{55}, k_{56}, k_{63}, k_{64}, k_{20} + k_{21}, k_{61} + k_{62}, k_{79} + k_{80}$
49	$k_{13}, k_{14}, k_{51}, k_{52}, k_{73}, k_{74}, k_{75}, k_{76}$
50	$k_{13}, k_{14}, k_{51}, k_{73}, k_{74}, k_{75}, k_{76}, k_{11} + k_{12}$
51	$k_{51}, k_{52}, k_{73}, k_{74}, k_{75}, k_{76}, k_{11} + k_{12}, k_{49} + k_{50}$
52	$k_{11}, k_{13}, k_{14}, k_{49}, k_{51}, k_{52}, k_{73}, k_{74}, k_{75}, k_{76}$
53	$k_{59}, k_{60}, k_{65}, k_{66}, k_{67}, k_{68}, k_{57} + k_{58}$
54	$k_{59}, k_{60}, k_{65}, k_{66}, k_{67}, k_{68}$
55	$k_{57}, k_{59}, k_{60}, k_{65}, k_{66}, k_{67}, k_{68}$
56	$k_{59}, k_{65}, k_{66}, k_{67}, k_{68}, k_{57} + k_{58}$
57	$k_{9}, k_{10}, k_{40}, k_{41}, k_{42}, k_{43}, k_{71}, k_{72}, k_{77}, k_{78}, k_{79}, k_{80}$
58	$k_{9}, k_{10}, k_{40}, k_{41}, k_{42}, k_{43}, k_{71}, k_{72}, k_{77}, k_{78}, k_{79}, k_{80}, k_{7} + k_{8}, k_{69} + k_{70}$
59	$k_{7}, k_{9}, k_{10}, k_{40}, k_{41}, k_{42}, k_{43}, k_{71}, k_{77}, k_{78}, k_{79}, k_{80}, k_{69} + k_{70}$
60	$k_{9}, k_{42}, k_{43}, k_{69}, k_{71}, k_{72}, k_{77}, k_{78}, k_{79}, k_{80}, k_{40} + k_{41}$
61	$k_{36}, k_{37}, k_{38}, k_{39}, k_{69}, k_{70}, k_{71}, k_{72}, k_{79}, k_{77} + k_{78}$
62	$k_{36}, k_{38}, k_{39}, k_{69}, k_{70}, k_{71}, k_{72}, k_{79}, k_{80}$
63	$k_{36}, k_{37}, k_{38}, k_{39}, k_{69}, k_{70}, k_{71}, k_{72}, k_{79}, k_{80}, k_{77} + k_{78}$
64	$k_{36}, k_{37}, k_{38}, k_{39}, k_{69}, k_{70}, k_{71}, k_{72}, k_{77}, k_{79}, k_{80}$

Table 6. 67 maxterms and maxterm equations

Cube Indexes	Maxterm Equations	Cube Indexes	Maxterm Equations
$\{2, 3, 4\}$	$k5 + k6$	$\{5, 6, 7\}$	$1 + k77$
$\{1, 2, 38\}$	$k6$	$\{1, 2, 40\}$	$1 + k4$
$\{2, 33, 37\}$	$k50 + k52$	$\{5, 42, 61\}$	$k57 + k59 + k60$
$\{9, 49, 53\}$	$1 + k70 + k71 + k72$	$\{13, 14, 15\}$	$1 + k9$
$\{13, 41, 61\}$	$k78 + k80$	$\{21, 22, 23\}$	$1 + k12$
$\{21, 22, 24\}$	$1 + k14$	$\{21, 45, 58\}$	$k73 + k76$
$\{25, 26, 27\}$	$1 + k80$	$\{25, 33, 37\}$	$1 + k54 + k56$
$\{29, 30, 31\}$	$k72$	$\{29, 30, 32\}$	$1 + k74$
$\{29, 45, 57\}$	$k62 + k64$	$\{5, 7, 8, 63\}$	$1 + k76 + k78$
$\{1, 2, 33, 35\}$	$1 + k52$	$\{1, 2, 33, 36\}$	$1 + k51$
$\{1, 2, 35, 36\}$	$1 + k49 + k50$	$\{5, 7, 44, 63\}$	$k59 + k76 + k78$
$\{6, 7, 43, 61\}$	$1 + k57 + k58$	$\{1, 33, 34, 37\}$	$k30 + k51$
$\{3, 33, 34, 37\}$	$k28$	$\{3, 33, 34, 39\}$	$1 + k6 + k29$
$\{3, 34, 35, 39\}$	$k4 + k31 + k52$	$\{5, 41, 43, 61\}$	$k60$
$\{6, 41, 42, 61\}$	$k35$	$\{6, 42, 43, 61\}$	$k34$
$\{7, 41, 43, 61\}$	$1 + k33$	$\{9, 10, 52, 53\}$	$1 + k71$
$\{9, 10, 52, 55\}$	$1 + k69 + k70$	$\{10, 49, 53, 54\}$	$1 + k38$
$\{10, 49, 54, 55\}$	$1 + k39 + k72$	$\{11, 49, 53, 54\}$	$k37 + k71$
$\{13, 15, 16, 43\}$	$1 + k8 + k10$	$\{13, 41, 62, 64\}$	$1 + k79$
$\{14, 15, 16, 43\}$	$k10$	$\{14, 41, 61, 62\}$	$1 + k42 + k79$
$\{15, 41, 61, 62\}$	$k41$	$\{17, 18, 49, 56\}$	$1 + k67$
$\{17, 18, 50, 54\}$	$k65 + k68$	$\{17, 18, 51, 56\}$	$1 + k65 + k66$
$\{17, 49, 51, 53\}$	$k68$	$\{21, 23, 24, 46\}$	$1 + k13 + k14$
$\{21, 45, 57, 60\}$	$1 + k75$	$\{25, 27, 28, 34\}$	$1 + k1 + k2$
$\{25, 33, 38, 39\}$	$1 + k56$	$\{25, 33, 38, 40\}$	$1 + k55$
$\{26, 27, 28, 34\}$	$k2$	$\{26, 33, 37, 38\}$	$k22 + k55$
$\{27, 33, 37, 38\}$	$1 + k21$	$\{29, 30, 45, 58\}$	$k63$
$\{29, 30, 47, 58\}$	$1 + k61 + k62$	$\{29, 45, 46, 58\}$	$k26 + k73$
$\{29, 45, 46, 60\}$	$k27 + k63$	$\{29, 46, 47, 57\}$	$1 + k64$
$\{31, 32, 46, 57\}$	$1 + k61 + k64$	$\{32, 46, 47, 57\}$	$k25 + k64$
$\{33, 37, 38, 40\}$	$1 + k23$	$\{41, 42, 62, 64\}$	$k32 + k33 + k59$
$\{41, 61, 62, 64\}$	$1 + k43$	$\{11, 50, 52, 53, 54\}$	$k36$
$\{15, 42, 43, 61, 62\}$	$1 + k40$	$\{27, 34, 35, 37, 38\}$	$k20$
$\{32, 45, 47, 58, 59\}$	$k24$		

Table 7. 56 maxterms and maxterm equations

Cube Indexes	Maxterm Equations	Cube Indexes	Maxterm Equations
$\{32, 47\}$	$1 + k64$	$\{32, 48\}$	$k63$
$\{1, 2, 34\}$	$k49 + k52$	$\{2, 4, 35\}$	$1 + k52$
$\{2, 4, 36\}$	$1 + k51$	$\{5, 7, 41\}$	$1 + k59$
$\{5, 7, 43\}$	$1 + k57 + k58$	$\{5, 8, 42\}$	$1 + k57 + k60$
$\{1, 33, 34\}$	$k29 + k30 + k51$	$\{2, 33, 34\}$	$k28$
$\{3, 35, 39\}$	$k49 + k50 + k52$	$\{5, 41, 42\}$	$1 + k35$
$\{5, 42, 43\}$	$k34 + k60$	$\{6, 42, 43\}$	$k34$
$\{8, 41, 42\}$	$1 + k32 + k33 + k59$	$\{9, 10, 54\}$	$1 + k69 + k72$
$\{9, 51, 53\}$	$1 + k71$	$\{9, 51, 55\}$	$1 + k69 + k70$
$\{9, 53, 54\}$	$k37 + k38 + k71$	$\{10, 12, 55\}$	$1 + k72$
$\{10, 53, 54\}$	$1 + k36$	$\{13, 14, 62\}$	$k77 + k80$
$\{13, 61, 62\}$	$k41 + k42 + k79$	$\{14, 16, 61\}$	$1 + k79$
$\{14, 16, 63\}$	$1 + k77 + k78$	$\{14, 61, 62\}$	$k40$
$\{14, 62, 63\}$	$k43$	$\{15, 62, 63\}$	$1 + k43 + k80$
$\{17, 18, 50\}$	$k65 + k68$	$\{18, 19, 51\}$	$1 + k68$
$\{18, 19, 52\}$	$1 + k67$	$\{18, 51, 55\}$	$k65 + k66 + k68$
$\{21, 23, 58\}$	$k73 + k76$	$\{22, 23, 57\}$	$k75$
$\{22, 23, 59\}$	$1 + k73 + k74$	$\{25, 27, 38\}$	$k53 + k56$
$\{25, 37, 38\}$	$1 + k22 + k55 + k56$	$\{26, 27, 37\}$	$1 + k55$
$\{26, 27, 39\}$	$1 + k53 + k54$	$\{27, 37, 38\}$	$1 + k20$
$\{27, 38, 39\}$	$k21 + k23 + k56$	$\{29, 30, 47\}$	$1 + k61 + k62$
$\{29, 31, 46\}$	$1 + k61 + k64$	$\{29, 45, 46\}$	$1 + k26$
$\{29, 46, 47\}$	$k25$	$\{30, 45, 47\}$	$1 + k24 + k64$
$\{33, 34, 36\}$	$k31$	$\{33, 34, 37\}$	$1 + k30 + k51$
$\{33, 37, 38\}$	$k56$	$\{42, 61, 62\}$	$1 + k42 + k79$
$\{46, 57, 58\}$	$k76$	$\{50, 53, 54\}$	$1 + k38$
$\{50, 54, 55\}$	$1 + k39$	$\{8, 41, 42, 61\}$	$k32 + k59$
$\{27, 35, 38, 39\}$	$k23 + k56$	$\{31, 45, 46, 60\}$	$1 + k27 + k63 + k64$

Table 8. 70 maxterms and maxterm equations

Cube Indexes	Maxterm Equations	Cube Indexes	Maxterm Equations
$\{10, 13, 14\}$	$1 + k9$	$\{10, 14, 15\}$	$1 + k10$
$\{1, 21, 22, 23\}$	$1 + k11 + k12$	$\{1, 21, 22, 24\}$	$k14$
$\{5, 17, 42, 64\}$	$1 + k57 + k60$	$\{6, 18, 41, 42\}$	$k75 + k76$
$\{10, 15, 44, 63\}$	$k77 + k78 + k80$	$\{11, 15, 50, 53\}$	$1 + k70 + k72$
$\{12, 13, 14, 43\}$	$k7$	$\{13, 14, 15, 54\}$	$1 + k9 + k69 + k72$
$\{17, 41, 42, 64\}$	$k32 + k33 + k59$	$\{21, 22, 24, 34\}$	$k14 + k49 + k52$
$\{25, 26, 27, 30\}$	$1 + k79 + k80$	$\{25, 26, 28, 30\}$	$k2$
$\{25, 27, 28, 29\}$	$k1 + k2$	$\{25, 29, 31, 32\}$	$1 + k71 + k72$
$\{25, 29, 45, 46\}$	$1 + k27 + k63$	$\{25, 30, 32, 48\}$	$1 + k63 + k71 + k72$
$\{26, 29, 35, 38\}$	$1 + k53 + k55 + k56$	$\{26, 29, 45, 46\}$	$k26 + k73$
$\{50, 52, 62, 63\}$	$1 + k67 + k80$	$\{3, 4, 21, 33, 37\}$	$1 + k6 + k51$
$\{1, 21, 23, 24, 45\}$	$k13 + k14$	$\{1, 22, 45, 47, 58\}$	$k73 + k75 + k76$
$\{2, 21, 33, 34, 37\}$	$1 + k6 + k30 + k51$	$\{2, 22, 33, 35, 36\}$	$1 + k5 + k6$
$\{3, 21, 34, 35, 37\}$	$1 + k5 + k31 + k52$	$\{3, 21, 34, 35, 38\}$	$1 + k31 + k52$
$\{3, 21, 34, 36, 38\}$	$k28 + k29 + k51$	$\{3, 21, 35, 37, 39\}$	$k49 + k50 + k52$
$\{5, 17, 42, 43, 63\}$	$1 + k34 + k60$	$\{5, 17, 42, 44, 63\}$	$1 + k35 + k59$
$\{8, 18, 41, 42, 63\}$	$1 + k32 + k59$	$\{9, 13, 15, 16, 43\}$	$1 + k7 + k8 + k10$
$\{9, 14, 41, 61, 62\}$	$1 + k9 + k42 + k79$	$\{9, 15, 16, 41, 64\}$	$1 + k9 + k79$
$\{10, 11, 12, 49, 62\}$	$k77 + k80$	$\{10, 14, 51, 53, 54\}$	$k38$
$\{10, 14, 51, 54, 55\}$	$k39 + k72$	$\{10, 14, 51, 54, 56\}$	$k38 + k71$
$\{10, 15, 43, 61, 62\}$	$1 + k40$	$\{10, 15, 43, 62, 63\}$	$k43 + k80$
$\{10, 44, 61, 63, 64\}$	$1 + k40 + k41$	$\{11, 13, 52, 53, 54\}$	$k36$
$\{11, 14, 51, 53, 54\}$	$1 + k36 + k37 + k71$	$\{17, 18, 19, 43, 54\}$	$1 + k57 + k58$
$\{17, 41, 42, 43, 63\}$	$1 + k35$	$\{17, 41, 43, 44, 64\}$	$k32 + k33 + k34$
$\{18, 41, 42, 50, 54\}$	$1 + k65 + k67 + k68$	$\{19, 41, 42, 44, 63\}$	$k35 + k75$
$\{21, 22, 24, 35, 36\}$	$k14 + k49$	$\{25, 26, 27, 33, 45\}$	$k62 + k64$
$\{25, 26, 27, 34, 46\}$	$1 + k61 + k64$	$\{25, 26, 29, 37, 39\}$	$1 + k56$
$\{25, 26, 29, 37, 40\}$	$1 + k55$	$\{25, 26, 29, 39, 40\}$	$1 + k53 + k54$
$\{25, 29, 32, 45, 47\}$	$k64 + k71 + k72$	$\{25, 29, 35, 37, 38\}$	$1 + k22 + k55$
$\{25, 32, 46, 47, 57\}$	$1 + k24 + k25 + k64$	$\{27, 29, 35, 37, 38\}$	$k20$
$\{27, 29, 35, 38, 39\}$	$k23 + k56$	$\{27, 29, 36, 37, 38\}$	$1 + k20 + k21$
$\{1, 2, 3, 40, 59, 60\}$	$1 + k73 + k74$	$\{5, 7, 44, 50, 51, 63\}$	$k59 + k68 + k75 + k76 + k78$
$\{1, 21, 33, 34, 35, 40\}$	$1 + k3 + k5$	$\{1, 23, 33, 34, 35, 40\}$	$k3 + k4 + k5$
$\{17, 18, 41, 43, 51, 56\}$	$1 + k65 + k66$	$\{21, 22, 23, 33, 35, 36\}$	$1 + k11$
$\{21, 33, 34, 35, 37, 40\}$	$1 + k28$	$\{26, 31, 45, 47, 48, 59\}$	$1 + k24$

Comprehensive Study of Integral Analysis on 22-Round LBlock

Yu Sasaki[1] and Lei Wang[2]

[1] NTT Secure Platform Laboratories, NTT Corporation
sasaki.yu@lab.ntt.co.jp
[2] Nanyang Technological University, Singapore
Wang.Lei@ntu.edu.sg

Abstract. The current paper presents an integral cryptanalysis in the single-key setting against light-weight block-cipher LBlock reduced to 22 rounds. Our attack uses the same 15-round integral distinguisher as the previous attacks, but many techniques are taken into consideration in order to achieve comprehensive understanding of the attack; choosing the best balanced-byte position, meet-in-the-middle technique to identify right key candidates, partial-sum technique, relations among subkeys, and combination of the exhaustive search with the integral analysis.

Keywords: LBlock, integral analysis, partial-sum, meet-in-the-middle.

1 Introduction

Block-ciphers are basic tools for secure communications which provide the confidentiality of the data. Recently, block-ciphers which can be implemented in resource constraint environment, e.g., RFID Tags for a sensor network, have received much attention. Such block-ciphers are called light-weight block-ciphers.

Many light-weight block-ciphers were designed so far. Some examples are HIGHT [1] proposed at CHES 2006 which were standardized by ISO as a 64-bit block-cipher [2], and PRESENT [3] proposed at CHES 2007 and CLEFIA [4] proposed at FSE 2007, which were standardized by ISO for the lightweight cryptography [5]. Many other designs were proposed independently of the ISO standards e.g., LBlock [6] proposed at ACNS 2011, Piccolo [7] proposed at CHES 2011, LED [8] proposed at CHES 2011, and TWINE [9] proposed at SAC 2012. Different designs provide different implementation characteristics, e.g., different tradeoff of area, throughput, and security, thus making a comparison and identifying good designs is very hard. Particularly security evaluation is hard because it takes long and usually requires evaluations by the third party.

Integral analysis is a cryptanalytic technique against symmetric-key primitives, which was firstly proposed by Daemen *et al.* to evaluate the security of SQUARE cipher [10], and was later unified as *integral analysis* by Knudsen and Wagner [11]. The crucial part is a construction of an *integral distinguisher*: an attacker prepares a set of plaintexts which contains all possible values for some bytes and has a constant value for the other bytes. All plaintexts in the set are passed to an encryption oracle. Then, the corresponding state after a few rounds has a certain property, *e.g.* the XOR of all texts in the set becomes 0 with probability 1. Throughout the paper, this property is called *balanced.* A key recovery attack can be constructed by using this property. The attacker appends a few rounds to the end of the distinguisher. After she obtains a set of the ciphertexts, she guesses a part of round keys and performs the partial decryption up to the balanced state. If the guess is correct, the XOR sum of the results always becomes 0. Hence, the key space can be reduced.

Several improved techniques are known for the integral analysis. Ferguson *et al.* proposed a technique called *partial-sum* [12]. It reduces the complexity of the partial decryption up to the balanced state by guessing each subkey byte one after another. Sasaki and Wang introduced *meet-in-the-middle techniques* for the key recovery phase of the integral analysis against block-ciphers with a Feistel network [13]. It separates the partial decryption into two independent parts, and thus the complexity can be reduced.

The integral analysis has already been applied to LBlock. Firstly, the designers proposed a 15-round integral distinguisher, and constructed an 18-round attack [6]. Then, Sasaki and Wang extended the attack up to 20 rounds [13]. Regarding other approaches, the designers proposed a 20-round impossible differential attack [6]. This was later extended up to 21 rounds by Liu *et al.* [14]. This is the current best attack on LBlock in the single-key setting. Regarding related-key attacks, Minier and Naya-Plasencia proposed a related-key impossible differential attack up to 23 rounds [15]. Liu *et al.* studied several related-key differential-based attacks at ICICS 2012 [16]. An optimization of the brute-force attack by a biclique technique was studied by Wang *et al.* at WISA 2012 [17]. In this paper, we do not discuss such an optimized brute force attack with a small advantage of a constant factor.

Very recently, we have realized that Wang *et al.* [17] cited an unpublished paper written in the Chinese language which claims a 22-round integral attack on LBlock [18]. All the information we have about [18] is obtained through [17], and is summarized as follows. 1) It is claimed that 22 rounds of LBlock are attacked with $2^{61.6}$ CPs and $2^{71.2}$ encryptions. 2) The integral distinguisher used in the attack is the same as the previous one in [6]. We emphasize that our work is independent of [18].

Our Contributions. In this paper, we present a comprehensive study of the integral analysis against LBlock. Our goal is extending the number of attacked rounds and optimize the complexity as much as possible by considering all previously known techniques. Specifically, we consider the following techniques;

– There are 4 possibilities of the balanced-byte position at the output of the integral distinguisher. We try all of them to identify the best choice.
– We optimize the complexity by using the meet-in-the-middle approach.
– We optimize the complexity by using the partial-sum technique.
– We analyze the key schedule function, and exploit subkey relations.
– We combine the exhaustive search with integral analysis. This can optimize the data complexity.

Table 1. Comparison of attack results

Model	Approach	#Rounds	Data	Time	Memory (bytes)	Reference
Single-key	Imp. Diff.	20	2^{63}	$2^{72.7}$	2^{68}	[6]
	Imp. Diff.	21	$2^{62.5}$	$2^{73.7}$	$2^{55.5}$	[14]
	Integral	18	2^{62}	2^{36}	2^{20}	[6]
	Integral	20	$2^{63.6}$	$2^{39.6}$	2^{35}	[13]
	Integral	22	$2^{61.6}$	$2^{71.2}$	not given	[18] †
	Integral	21	$2^{61.6}$	$2^{54.16}$ ‡	$2^{51.58}$	This paper
	Integral	22	2^{61}	$2^{70.00}$	2^{63}	This paper
Related-key	Imp. Diff.	23	2^{40}	2^{70}	not given	[15]

†: Unpublished independent work available only in Chinese.
‡: The attack requires $2^{61.58}$ memory access in order to process $2^{61.58}$ ciphertexts.

As a result, we construct a 21-round attack with (Data, Time, Memory) = $(2^{61.6}, 2^{54.16}, 2^{51.58})$, which is better than the previous 21-round impossible differential attack with (Data, Time, Memory) = $(2^{62.5}, 2^{73.7}, 2^{55.5})$. We then extend the attack by one more round, and obtain a 22-round attack with (Data, Time, Memory) = $(2^{61}, 2^{70.00}, 2^{63})$. The attack results are summarized in Table 1.

The 15-round integral distinguisher discovered by the designers [6] produces the balanced byte at 4 byte-positions, 0th, 2nd, 4th, and 6th bytes of the intermediate state after 15 rounds. The previous integral attacks [13,6] used the balanced byte at the 4th byte without any reasoning. Our analysis shows that the choice of the balanced-byte position is very sensitive when subkey relations are considered. Interestingly, as later explained in Table 2, using the balanced byte at the 6th byte for attacking 21 rounds and at the 2nd byte for attacking 22 rounds achieves significantly smaller complexity than the other 3 choices.

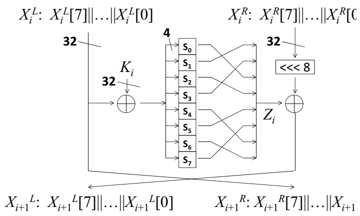
Our results indicate that the integral cryptanalysis is particularly useful for LBlock like structures. Indeed, LBlock is the almost only case that the integral cryptanalysis works more rounds than the impossible differential cryptanalysis.

2 Preliminaries

2.1 LBlock Specification

LBlock is a light-weight block-cipher proposed by Wu and Zhang [6]. The block size is 64 bits and the key size is 80 bits. It adopts a modified Feistel structure with 32 rounds, and its round function consists of the subkey addition, S-box transformations, and a permutation of the byte positions (1 byte is 4 bits).

Let $X_i^L \| X_i^R$, where $0 \leq i \leq 32$, be an internal state which is an input to the i-th round or an output from the $i-1$-th round. We further denote 8 bytes inside of X_i^L and X_i^R by $X_i^L = X_i^L[7] \| X_i^L[6] \| \cdots \| X_i^L[0]$ and $X_i^R = X_i^R[7] \| X_i^R[6] \| \cdots \| X_i^R[0]$, respectively. The plaintext is loaded into an internal state $X_0^L \| X_0^R$. The state $X_i^L \| X_i^R$ is updated with a 32-bit subkey $K_i =$



X_i^L: $X_i^L[7]\|\ldots\|X_i^L[0]$
X_i^R: $X_i^R[7]\|\ldots\|X_i^R[$
32
4
32
K_i
32
<<< 8
S_0
S_1
S_2
S_3
S_4
S_5
S_6
S_7
Z_i
X_{i+1}^L: $X_{i+1}^L[7]\|\ldots\|X_{i+1}^L[0]$
X_{i+1}^R: $X_{i+1}^R[7]\|\ldots\|X_{i+1}$

Fig. 1. LBlock round function

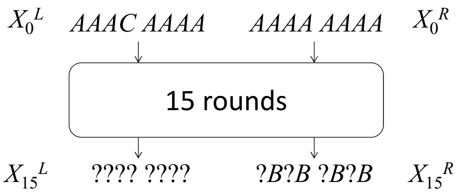


Fig. 2. 15-round integral distinguisher

$K_i[7]\|K_i[6]\|\cdots\|K_i[0]$ by $X^L_{i+1} = P\big(S(X^L_i \oplus K_i)\big) \oplus (X^R_i \lll 8)$, and $X^R_{i+1} = X^L_i$, where, $S(\cdot), P(\cdot)$, and $\lll 8$ represent an S-box layer, a permutation of the byte positions, and the left cyclic shift by 8 bits, respectively. In the S-box layer, each byte is updated according to the 4-bit to 4-bit S-boxes defined in the specification. Then, $P(x_7\|x_6\|x_5\|x_4\|x_3\|x_2\|x_1\|x_0)$ returns $(x_6\|x_4\|x_7\|x_5\|x_2\|x_0\|x_3\|x_1)$. These computations are described in Fig. 1. In this paper, we denote the state after the byte-position permutation in round i by Z_i. After 32 rounds, $X^L_{32}\|X^R_{32}$ are produced as the ciphertext.

Key Schedule Function. The key schedule function produces thirty-two 32-bit subkeys from an 80-bit secret key. Let κ_i, where $0 \le i \le 31$, be an 80-bit internal state for the key schedule function for round i. We denote each bit of κ_i by $\kappa_i[79], \kappa_i[78], \ldots, \kappa_i[0]$. We often denote several bits of κ_i by $\kappa_i[a, b, c, \cdots]$.

The leftmost 32 bits of κ_0, i.e., $\kappa_0[79, 78, \ldots, 48]$, are output as a 32-bit subkey for round 0, K_0. Then, the following is operated for $i = 1, 2, \ldots, 31$.

1. $\kappa_i \leftarrow \kappa_{i-1} \lll 29$.
2. Update $\kappa_i[79, 78, 77, 76]$ and $\kappa_i[75, 74, 73, 72]$ by $S_9(\kappa_i[79, 78, 77, 76])$ and $S_8(\kappa_i[75, 74, 73, 72])$ respectively, where S_8 and S_9 are 4-bit to 4-bit S-boxes.
3. Update $\kappa_i[50, 49, 48, 47, 46]$ by $\kappa_i[50, 49, 48, 47, 46] \oplus [i]_2$, where $[i]_2$ is a binary representation of the round index.
4. Output the leftmost 32 bits of κ_i as a 32-bit subkey K_i.

2.2 Notations for Integral Attack

To discuss integral distinguishers, the following notations are used in this paper.

"A **(Active)**" : all values appear exactly the same number in the set of texts.
"B **(Balanced)**" : the XOR of all texts in the set is 0.
"C **(Constant)**" : the value is fixed to a constant for all texts in the set.

We also use the following notations to describe the attack.

D: number of plaintexts to construct an integral distinguisher.
K_α: size of subkeys (in bits) recovered during the key recovery phase.

K_β: size of subkeys (in bits) exhaustively guessed after the key recovery phase.
$|B|$: size of the balanced state (in bits) to be checked in the key recovery phase. The key space is reduced by $|B|$ bits with analyzing a single set.

The previous integral attack, especially for LBlock, analyzed $K_\alpha/|B|$ plaintext sets to identify the right key of K_α. Then, K_β is recovered by the exhaustive search. The data complexity is $D \cdot (K_\alpha/|B|)$ and the time complexity is a sum of the one for the key recovery phase and 2^{K_β}. Several techniques can be applied to reduce the time complexity of the key recovery phase. Note that if D is much bigger than the time complexity for the key recovery phase and the exhaustive search, $D \cdot (K_\alpha/|B|)$ memory access for processing obtained ciphertexts is the bottle-neck of the time complexity.

2.3 Partial-Sum Technique

The partial-sum technique was introduced by Ferguson *et al.* [12]. The original attack target was AES. In the key recovery phase of the AES, the partial decryption involves 5 bytes of the key and 4 bytes of the ciphertext. Suppose that the number of data to be analyzed, n, is 2^{32} and the byte position b of each ciphertext is denoted by $c_{b,n}$. Then, the equation can be described as follows.

$$\bigoplus_{n=1}^{2^{32}} \Big[S_4\Big(S_0(c_{0,n} \oplus k_0) \oplus S_1(c_{1,n} \oplus k_1) \oplus S_2(c_{2,n} \oplus k_2) \oplus S_3(c_{3,n} \oplus k_3) \oplus k_4 \Big) \Big]. \quad (1)$$

With a straightforward method, the analysis takes $2^{32+40} = 2^{72}$ partial decryptions, while the partial-sum technique requires only 2^{48} partial decryptions. The idea is partially computing the sum by guessing each key byte one after another.

The analysis starts from 2^{32} texts $(c_{0,n}, c_{1,n}, c_{2,n}, c_{3,n})$. First, two key bytes k_0 and k_1 are guessed, and $S_0(c_{0,n} \oplus k_0) \oplus S_1(c_{1,n} \oplus k_1)$ is computed for each guess. Let $x_{i,n}$ be $\bigoplus_{p=0}^{i}(S_p(c_{p,n} \oplus k_p))$. Then, $S_0(c_{0,n} \oplus k_0) \oplus S_1(c_{1,n} \oplus k_1)$ can be represented by $x_{1,n}$, and Eq. (1) becomes $\bigoplus_{n=1}^{2^{32}} \Big[S_4\Big(x_{1,n} \oplus S_2(c_{2,n} \oplus k_2) \oplus S_3(c_{3,n} \oplus k_3) \oplus k_4 \Big) \Big]$. The original set includes 2^{32} texts, but now only 3-byte information (x_1, c_2, c_3) is needed. Hence, by only picking the values that appear odd times, the size of the data set is compressed into 3 bytes. For the second step, k_2 is guessed, and the size of the data set becomes 2 bytes (x_2, c_3). For the third step, k_3 is guessed, and the size of the data set becomes 1 byte (x_3). Finally, k_4 is guessed and Eq. (1) is computed. The complexity for the guess of k_0, k_1 is $2^{16} \times 2^{32} = 2^{48}$, for the guess of k_2 is $2^{16} \times 2^{8} \times 2^{24} = 2^{48}$. Similarly, the complexity is preserved to be 2^{48} until the end.

2.4 Previous Integral Analysis on LBlock

The designers showed a 15-round integral distinguisher [6], which is shown in Fig. 2. For a set of 2^{60} plaintexts with the form of $(AAAC\ AAAA\ AAAA\ AAAA)$,

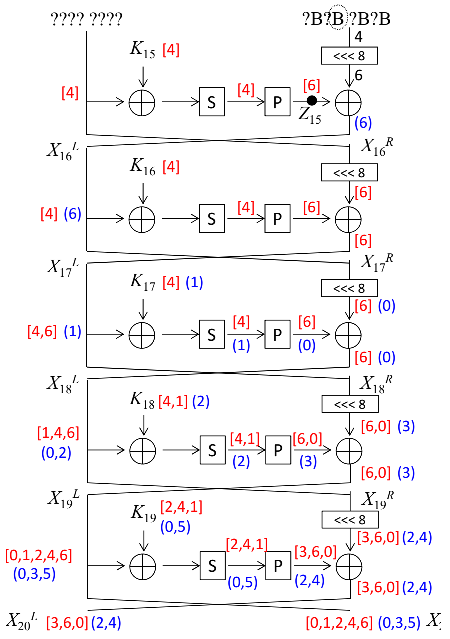


Fig. 3. 20-round attack on LBlock

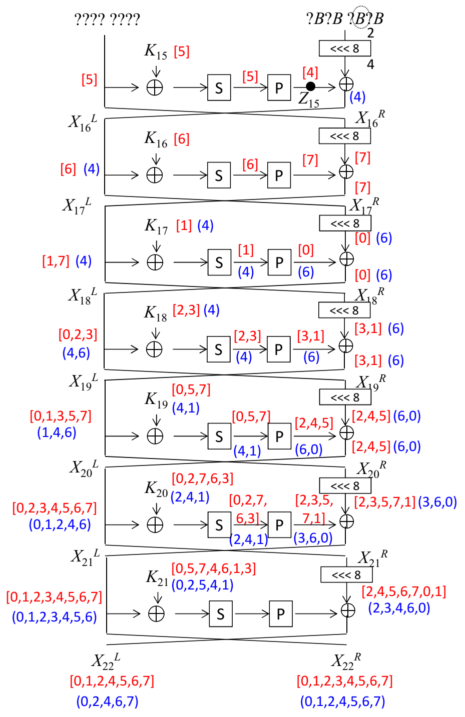


Fig. 4. 22-round attack on LBlock

the state after 15 rounds, $(X_{15}^L \| X_{15}^R)$, has the form of $(????\ ????\ ?B?B\ ?B?B)$. By using this property, the designers showed an 18-round key recovery attack. The attacker guesses a part of subkeys, and performs the partial decryption up to the fourth byte of X_{15}^R and checks if its sum is 0 or not. The partial decryption up to $X_{15}^R[4]$ involves 5 bytes of the ciphertext and 4 bytes of subkeys. The attacker first counts how many times each 5-byte value of the ciphertext appears and only picks the ones that appear odd times. Hence, at most $2^{4\times5} = 2^{20}$ values are stored in a memory. Then, for each guess of four key bytes, she computes the corresponding $X_{15}^R[4]$ and computes the sum. The attack complexity is $2^{20} \times 2^{16} = 2^{36}$ partial decryptions. With the analysis of a single 2^{60} plaintexts set, the key space is reduced by 1 byte. Therefore, to identify the right key, 4 sets of plaintexts need to be analyzed. Hence, the data complexity is $4 \times 2^{60} = 2^{62}$.

Sasaki and Wang introduced a meet-in-the-middle technique for the key recovery phase, and extended the number of attacked rounds up to 20 rounds [13]. The 5-round key-recovery phase is shown in Fig. 3. Because $\bigoplus X_{15}^R[4] = 0$ can be written as $\bigoplus Z_{15}[6] = \bigoplus X_{16}^L[6]$, the sum of $Z_{15}[6]$ and $X_{16}^L[6]$ can be computed independently, and right-key candidates are identified by checking their matches. The bottle-neck of the complexity, which is the partial decryption for $\bigoplus Z_{15}[6]$, involves 8 bytes of the ciphertext and 8 bytes of subkeys, and thus requires $2^{32}\times2^{32} = 2^{64}$ partial decryptions. Moreover, they applied the partial-sum technique, and the complexity for $\bigoplus Z_{15}[6]$ was reduced to 2^{36} partial decryptions. In this attack, 12 key bytes are guessed, and the key space is reduced by 1 byte with the analysis of a single 2^{60} plaintexts set. Therefore, 12 sets of plaintexts were analyzed, and the data complexity is $12 \times 2^{60} = 2^{63.6}$, which is very close to the full code book.

3 Combining Exhaustive Search for Data-Time Trade-Off

We explain a simple technique which gives the trade-off between the data complexity and time complexity. For example, we can convert the previous 20-round attack [13] with (Data, Time)=$(2^{63.7}, 2^{36})$ into the one with (Data, Time)=$(2^{62}, 2^{64})$, and thus can avoid the marginal improvement of the data complexity. Note that the complexity evaluation by Sasaki and Wang [13] is only for the key recovery phase. Actually, it requires $2^{63.7}$ memory access to process $2^{63.7}$ ciphertexts.

The approach is very simple. When we recover a part of subkeys, we stop identifying the unique right key for K_α, but reduce the key space into a sufficiently small size. Then, the reduced key space is exhaustively searched together with the remaining subkey bits K_β. Note that the exhaustive search only for K_α (independently of K_β) is impossible. K_α and K_β must be guessed together. With this method, the data complexity can be reduced with an extra cost for the exhaustive search. Let d be the number of sets to be analyzed. Then, the data complexity is $d \cdot D$. The key space for K_α is reduced into $K_\alpha - (d \cdot |B|)$, and the cost for the exhaustive search becomes $2^{K_\alpha - (d \cdot |B|) + K_\beta}$.

Let us apply this method to the previous 20-round attack [13] with (Data, Time)=$(2^{63.7}, 2^{36})$. More precisely, the parameters of this attack are $D = 2^{60}, K_\alpha = 48, K_\beta = 32, |B| = 4$. [13] chose $d = 12$, thus the data complexity is $12 \cdot 2^{60} = 2^{63.6}$ and the time complexity for the exhaustive search is $2^{0+32} = 2^{32}$. We now change the parameter d to $d = 4$. Then, the data complexity is $4 \cdot 2^{60} = 2^{62}$ and the time complexity for the exhaustive search is $2^{48-16+32} = 2^{64}$. Considering that the attack requires at least 2^{60} memory access to process the ciphertexts, the time complexity is now almost equally distributed, and we can avoid the marginal improvement of the data complexity.

Note that the data complexity of our attack is exactly the same as the expected values, while the previous attack in [6] uses double of the expected value to increase the success probability. Because our approach runs the exhaustive test, the right key can be identified with probability 1.

4 21-Round and 22-Round Attacks on LBlock

4.1 Overview without Considering the Key Schedule Function

As was done in [13], to detect the right key candidates, the attacker computes the sum of the target byte in Z_{15} and the sum of the target byte of X^L_{16} independently,

Table 2. Key space for 21- and 22-round attacks for all balanced-byte positions

Balanced-byte	21-round attack		22-round attack	
position	Key space for Z_{15}	K_α	Key space for Z_{15}	K_α
$X_{15}^R[0]$	50	63	62	75
$X_{15}^R[2]$	44	61	55	69
$X_{15}^R[4]$	47	63	63	75
$X_{15}^R[6]$	42	57	65	77

and find matches between two results. Due to the Feistel structure, the bottle-neck of the time complexity is the one for Z_{15}. Hence, how many subkey bytes relate to the computation for Z_{15} is important. The number of total subkey bytes, K_α, is also important to estimate the number of necessary text sets.

We firstly obtain such information for each of the balanced-byte position, $X_{15}^R[0], X_{15}^R[2], X_{15}^R[4]$, and $X_{15}^R[6]$. As a result, we found that such important factors are the same for all balanced-byte positions. In details, for 21-round attack, the computation for Z_{15} involves 13 subkey bytes and 12 ciphertext bytes, and K_α is 80 (20 bytes). For 22-round attack, the computation for Z_{15} involves 20 subkey bytes and 15 ciphertext bytes, and K_α is 128 (32 bytes).

21- and 22-round attacks are impossible only with the techniques in the previous attacks. We then analyze the key schedule function and exploit the relation between subkeys. If it is considered, the attack complexity is very different depending on the balanced-byte position.

4.2 Analysis of Key Schedule Function

What we do here is guessing several bits of the key state, and trace the guessed bit positions during several rounds. If the guess of several subkey bits reveals some information about other subkeys in different rounds, the number of guessed bits by the attacker can be reduced. We analyze the subkey relations for both of 21- and 22-rounds and for each balanced-byte position. As shown in Sect. 4.1, the number of guessed key bytes is too many, and thus we should choose the balanced-byte position with minimum key space. The results of the analysis is summarized in Table 2. The columns for "Key space for Z_{15}" show how many key bits must be guessed to compute the sum of the target byte in Z_{15} by considering the subkey relations, which is the bottle-neck of the time complexity. The columns for "K_α" show how many bits are guessed to analyze a single plaintext set, i.e., K_α is the number of elements of the union of the key space for Z_{15} and the key space for X_{16}^L. A smaller number indicates that overlaps of subkeys occur more frequently, and thus the number of guessed bits can be small. From Table 2, using $X_{15}^R[6]$ and $X_{15}^R[2]$ as the balanced-byte position would yield the best attack for 21 rounds and 22 rounds, respectively. It is particularly interesting that K_α for 22-round attack is significantly smaller for $X_{15}^R[2]$ than other balanced-byte positions.

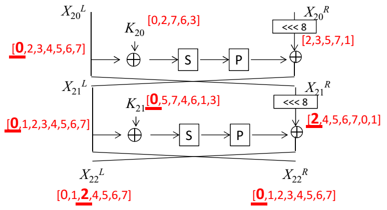


Fig. 5. Partial-sum with 1-byte guess

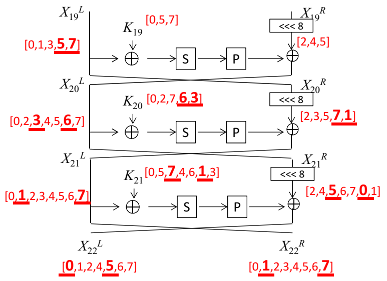


Fig. 6. Partial-sum with 3-byte guess

4.3 7-Round Key-Recovery Phase for 22-Round Attack

Details of the 7-round key-recovery phase are shown in Fig. 4. We follow the notations used in [13], where 20 key bytes and 15 ciphertext bytes related to the computation of $Z_{15}[4]$ are denoted by numbers in red square brackets. Similarly, 12 key bytes and 12 ciphertext bytes related to the computation of $X_{16}^L[4]$ are denoted by numbers in blue round brackets. The sum of $Z_{15}[4]$ and $X_{16}^L[4]$ are computed independently, and the bottle-neck is the computation for $Z_{15}[4]$. Hereafter, we mainly explain how to compute the sum of $Z_{15}[4]$.

High-Level Description. Because the procedure is very complicated, we first give the high-level description of the attack. The attack complexity depends on how we apply the partial-sum technique and how we utilize the subkey relations. In our attack, there are 2 patterns of the application of the partial-sum.

1-byte partial-sum: Suppose that we compute the sum for some byte of X_i^R. This always involves 1 byte of K_i and 2 bytes of (X_{i+1}^L, X_{i+1}^R). The 1-byte partial-sum is applied when (X_{i+1}^L, X_{i+1}^R) only relate to the computation of X_i^R. Namely, after we obtain the sum of X_i^R, we can discard the information on (X_{i+1}^L, X_{i+1}^R). An example is shown in Fig. 5, which we compute $X_{21}^R[0] = X_{20}^L[0] = (S_0(X_{22}^R[0] \oplus K_{21}[0]) \oplus X_{22}^L[2]) \ggg 8$, and $X_{22}^R[0]$ and $X_{22}^L[2]$ are only used to compute $X_{21}^R[0]$. Suppose, the number of texts to be analyzed is 2^{4N}, which consists of N-byte information ($X_{22}^R[0], X_{22}^L[2]$, and other $N-2$ bytes). For each 4-bit guess of $K_{21}[0]$, the attacker computes $X_{21}^R[0]$ for all 2^{4N} texts. Then she only picks $N-1$ byte tuple ($X_{21}^R[0]$ and other $N-2$ bytes) which appear odd times. The analysis requires 2^{4N+4} 1-round computations, and for each guess, the data is compressed into 2^{4N-4}. Therefore, the complexity of 2^{4N+4} is preserved until the end.

3-byte partial-sum: Suppose that we need to compute X_i^R by using 2-byte information of (X_{i+1}^L, X_{i+1}^R). The 3-byte partial-sum is applied when X_{i+1}^R is used not only for computing X_i^R but also for computing X_{i-1}^R. In this case, the data cannot be compressed after we guess 4 bits of K_i, and thus the attack complexity increases. An example is shown in Fig. 6, which $X_{22}^R[1,7]$

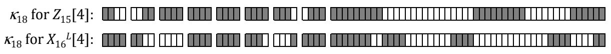

κ_{18} for $Z_{15}[4]$:
κ_{18} for $X_{16}^L[4]$:

Fig. 7. κ_{18} for computing $Z_{15}[4]$ and $X_{16}^L[4]$. Known bits are in grey.

are used not only for computing $X_{21}^R[6,3]$ but also for computing $X_{20}^R[7,5]$. We first guess two bytes of $K_{21}[1,7]$, and update the value of $X_{22}^L[0,5]$ to $X_{21}^R[6,3]$. This requires the complexity of 2^{4N+8}. Then, 1-byte partial-sum is applied for the computation of $X_{20}^R[5]$ (and then $X_{20}^R[7]$). In summary, the 3-byte partial-sum increases the complexity into $2^{(4N+8)+4}$.

The attack optimization also requires to consider the subkey relations, i.e., we need to arrange the computation order so that we can reduce the number of guessed bits by using the relations. This also makes the attack complicated. In our attack, these techniques are considered simultaneously by hand. In high-level, the complexity for computing the sum of $Z_{15}[4]$ can be explained as follows. The detailed procedure and and its evaluation are available in Appendix A.

- The analysis starts from 2^{60} texts, 15-byte information of the ciphertext.
- The 1-byte partial-sum is applied several times. At this stage, the complexity is preserved to be $2^{60+4} = 2^{64}$ 1-round computations.
- The 3-byte partial-sum is applied once, but thanks to the subkey relations, we can save the guess of 3 bits to update the value. At this stage, the complexity increases to $2^{(60+5)+4} = 2^{69}$ 1-round computations.
- Thanks to the subkey relations, the 1-byte partial-sum is applied by only guessing 2 bits. This occurs 2 times. At this stage, the complexity is $2^{(60+5-4)+4} = 2^{65}$ 1-round computations.
- The 3-byte partial-sum is applied once again. At this stage, the complexity increases to $2^{(60+5-4+8)+4} = 2^{73}$ 1-round computations.
- The remaining part can be computed with less complexity due to the subkey relations. Hence, intuitively, the bottle-neck of the complexity is 2^{73} 1-round computations.

Remaining Part of the Attack. The complexity for computing $\bigoplus X_{16}^L[4]$ must be estimated. We confirmed that $\bigoplus X_{16}^L[4]$ is computed with at most $12 \cdot 2^{48+8} < 2^{60}$ 1-round computations. Due to the limited space, we omit the details. We store the result of $\bigoplus X_{16}^L[4]$ together with the 48-bit guessed keys in a table $L_{X_{16}^L}$. After the analysis, we obtain a list $L_{X_{16}^L}$ with 2^{48} entries.

After we make two lists $L_{Z_{15}}$ and $L_{X_{16}^L}$, we identify the right key candidates by checking the match. By the condition $\bigoplus Z_{15}[4] = \bigoplus X_{16}^L[4]$, the key space can be reduced by 4 bits. Moreover, because the computations of $Z_{15}[4]$ and $X_{16}^L[4]$ share some key bits in common, we can reduce the key space more. The key state κ_{18} for computing $Z_{15}[4]$ and $X_{16}^L[4]$ are shown in Fig. 7. 25 key bits are overlapped between κ_{18} for computing $Z_{15}[4]$ and $X_{16}^L[4]$. Therefore, the key space can be reduced by $4 + 25 = 29$ bits in total. Note that 12 bits of κ_{18} are not included in both parts. Those 12 bits are exhaustively searched after the key space for other 68 bits of κ_{18} is sufficiently reduced.

Finally, we conclude the attack. We analyze 2 sets of 2^{60} texts. The key space for the guessed 68 bits of κ_{18} is reduced into $2^{56+48-(29*2)} = 2^{46}$. These 2^{46} and the other 12 bits are exhaustively searched with $2^{46+12} = 2^{58}$ LBlock computations. The data complexity is 2^{61} plaintexts. The time complexity is $2 \cdot (2^{73.46} + 2^{60})$ round functions and 2^{58} 22-round LBlock computations. The cost for computing one round function is regarded as 1/22 of 22-round LBlock computations. Hence, the total cost is $2 \cdot (2^{73.46} + 2^{60})/22 + 2^{58} \approx 2^{70.00}$ 22-round LBlock computations. The memory complexity is 2^{66} bits, which is 2^{63} bytes.

5 Concluding Remarks

In this paper, we presented a comprehensive study of the integral analysis against LBlock. We showed that the choice of the balanced-byte position is very sensitive when the subkey relations are considered. As a result, we achieved the 22-round attack with (Data, Time, Memory) = $(2^{61}, 2^{70.00}, 2^{63})$.

References

1. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B.-S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: HIGHT: A new block cipher suitable for low-resource device. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)
2. ISO/IEC 18033-3:2010: Information technology–Security techniques–Encryption Algorithms–Part 3: Block ciphers (2010)
3. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
4. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-bit blockcipher CLEFIA (extended abstract). In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 181–195. Springer, Heidelberg (2007)
5. ISO/IEC 29192-2:2011: Information technology–Security techniques–Lightweight cryptography–Part 2: Block ciphers (2011)
6. Wu, W., Zhang, L.: LBlock: A lightweight block cipher. In: Lopez, J., Tsudik, G. (eds.) ACNS 2011. LNCS, vol. 6715, pp. 327–344. Springer, Heidelberg (2011)
7. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: *Piccolo*: An ultra-lightweight blockcipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 342–357. Springer, Heidelberg (2011)
8. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED block cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)
9. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: TWINE: A lightweight block cipher for multiple platforms. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 339–354. Springer, Heidelberg (2013)

10. Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher SQUARE. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997)
11. Knudsen, L.R., Wagner, D.: Integral cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 112–127. Springer, Heidelberg (2002)
12. Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., Whiting, D.: Improved cryptanalysis of Rijndael. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 213–230. Springer, Heidelberg (2001)
13. Sasaki, Y., Wang, L.: Meet-in-the-middle technique for integral attacks against Feistel ciphers. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 234–251. Springer, Heidelberg (2013)
14. Liu, Y., Gu, D., Liu, Z., Li, W.: Impossible differential attacks on reduced-round LBlock. In: Ryan, M.D., Smyth, B., Wang, G. (eds.) ISPEC 2012. LNCS, vol. 7232, pp. 97–108. Springer, Heidelberg (2012)
15. Minier, M., Naya-Plasencia, M.: A related key impossible differential attack against 22 rounds of the lightweight block cipher LBlock. Inf. Process. Lett. 112(16), 624–629 (2012)
16. Liu, S., Gong, Z., Wang, L.: Improved related-key differential attacks on reduced-round LBlock. In: Chim, T.W., Yuen, T.H. (eds.) ICICS 2012. LNCS, vol. 7618, pp. 58–69. Springer, Heidelberg (2012)
17. Wang, Y., Wu, W., Yu, X., Zhang, L.: Security on LBlock against biclique cryptanalysis. In: Lee, D.H., Yung, M. (eds.) WISA 2012. LNCS, vol. 7690, pp. 1–14. Springer, Heidelberg (2012)
18. Li, Y.: Integral cryptanalysis on block ciphers. Institute of Software, Chinese Academy of Sciences, Beijing (2012) (in Chinese)

A Details of 22-Round Attack

The procedure to compute $\bigoplus Z_{15}[4]$ is as follows. Its summary is available in Table 3. The key state for the last seven rounds, $\kappa_{15}, \ldots, \kappa_{21}$ are shown in Fig. 8.

1. Query 2^{60} plaintexts which has the form of $(AAAC\ AAAA\ AAAA\ AAAA)$.
2. Count how many times each fifteen-byte value $X_{22}^L[0,1,2,4,5,6,7], X_{22}^R[0,1,2,3,4,5,6,7]$ appears, and pick the values which appear odd times.
3. Guess 4 bits of $K_{21}[0]$, and compute $X_{21}^R[0]$ with $X_{22}^L[2], X_{22}^R[0]$. Compress the data into 2^{56} texts of $(X_{22}^L[0,1,4,5,6,7], X_{22}^R[1,2,3,4,5,6,7], X_{21}^R[0])$.
4. Guess 4 bits of $K_{21}[4]$, and compute $X_{21}^R[4]$ with $X_{22}^L[6], X_{22}^R[4]$. Compress the data into 2^{52} texts of $(X_{22}^L[0,1,4,5,7], X_{22}^R[1,2,3,5,6,7], X_{21}^R[0,4])$.
5. Guess 4 bits of $K_{21}[6]$, and compute $X_{21}^R[5]$ with $X_{22}^L[7], X_{22}^R[6]$. Compress the data into 2^{48} texts of $(X_{22}^L[0,1,4,5], X_{22}^R[1,2,3,5,7], X_{21}^R[0,4,5])$.
6. Guess 4 bits of $K_{20}[0]$, and compute $X_{20}^R[0]$ with $X_{22}^R[2], X_{21}^R[0]$. Compress the data into 2^{44} texts of $(X_{22}^L[0,1,4,5], X_{22}^R[1,3,5,7], X_{21}^R[4,5], X_{20}^R[0])$. The known bits of the key state upto this step are shown in red in Fig. 8.
7. Guess $K_{21}[7]$. From Fig. 8, this can be done by 1-bit guess of $\kappa_{20}[47]$, which is colored in yellow. Compute $X_{21}^R[3]$ with $X_{22}^L[5], X_{22}^R[7]$. Update the data into 2^{44} texts of $(X_{22}^L[0,1,4], X_{22}^R[1,3,5,7], X_{21}^R[3,4,5], X_{20}^R[0])$.
8. Guess 4 bits of $K_{21}[1]$, and compute $X_{21}^R[6]$ with $X_{22}^L[0], X_{22}^R[1]$. Update the data into 2^{44} texts of $(X_{22}^L[1,4], X_{22}^R[1,3,5,7], X_{21}^R[3,4,5,6], X_{20}^R[0])$.

9. Guess 4 bits of $K_{20}[3]$, and compute $X^R_{20}[7]$ with $X^R_{22}[1], X^R_{21}[3]$. Compress the data into 2^{40} texts of $(X^L_{22}[1,4], X^R_{22}[3,5,7], X^R_{21}[4,5,6], X^R_{20}[0,7])$. The new known bits of the key state upto this step are shown in yellow in Fig. 8.
10. Guess $K_{19}[7]$. From Fig. 8, this can be done by 2-bit guess of $\kappa_{19}[78,79]$, which is colored in blue. Compute $X^R_{19}[3]$ with $X^R_{21}[5], X^R_{20}[7]$. Compress the data into 2^{36} texts of $(X^L_{22}[1,4], X^R_{22}[3,5,7], X^R_{21}[4,6], X^R_{20}[0], X^R_{19}[3])$.
11. Guess 4 bits of $K_{20}[6]$, and compute $X^R_{20}[5]$ with $X^R_{22}[7], X^R_{21}[6]$. Compress the data into 2^{32} texts of $(X^L_{22}[1,4], X^R_{22}[3,5], X^R_{21}[4], X^R_{20}[0,5], X^R_{19}[3])$.
12. Guess $K_{19}[5]$. From Fig. 8, this can be done by 2-bit guess of $\kappa_{19}[68,69]$, which is also colored in blue. Compute $X^R_{19}[2]$ with $X^R_{21}[4], X^R_{20}[5]$. Compress the data into 2^{28} texts of $(X^L_{22}[1,4], X^R_{22}[3,5], X^R_{20}[0], X^R_{19}[2,3])$.
13. Guess 4 bits of $K_{21}[3]$, and compute $X^R_{21}[7]$ with $X^L_{22}[1], X^R_{22}[3]$. Update the data into 2^{28} texts of $(X^L_{22}[4], X^R_{22}[3,5], X^R_{21}[7], X^R_{20}[0], X^R_{19}[2,3])$.
14. Guess 4 bits of $K_{21}[5]$, and compute $X^R_{21}[2]$ with $X^L_{22}[4], X^R_{22}[5]$. Update the data into 2^{28} texts of $(X^R_{22}[3,5], X^R_{21}[2,7], X^R_{20}[0], X^R_{19}[2,3])$.
15. Guess 4 bits of $K_{20}[7]$, and compute $X^R_{20}[3]$ with $X^R_{22}[5], X^R_{21}[7]$. Compress the data into 2^{24} texts of $(X^R_{22}[3], X^R_{21}[2], X^R_{20}[0,3], X^R_{19}[2,3])$. The new known bits of the key state upto this step are also shown in blue in Fig. 8.
16. From Fig. 8, $K_{18}[2]$ is already known. Compute $X^R_{18}[1]$ with $X^R_{20}[3], X^R_{19}[2]$. Compress the data into 2^{20} texts of $(X^R_{22}[3], X^R_{21}[2], X^R_{20}[0], X^R_{19}[3], X^R_{18}[1])$.
17. Guess 4 bits of $K_{20}[2]$, and compute $X^R_{20}[1]$ with $X^R_{22}[3], X^R_{21}[2]$. Update the data into 2^{20} texts of $(X^R_{21}[2], X^R_{20}[0,1], X^R_{19}[3], X^R_{18}[1])$.
18. From Fig. 8, $K_{18}[3]$ is already known. Compute $X^R_{18}[7]$ with $X^R_{20}[1], X^R_{19}[3]$. Compress the data into 2^{16} texts of $(X^R_{21}[2], X^R_{20}[0], X^R_{18}[1,7])$.
19. Guess $K_{19}[0]$. From Fig. 8, this can be done by 1-bit guess of $\kappa_{19}[51]$, which is colored in green. Compute $X^R_{19}[0]$ with $X^R_{21}[2], X^R_{20}[0]$. Compress the data into 2^{12} texts of $(X^R_{19}[0], X^R_{18}[1,7])$.
20. From Fig. 8, $K_{17}[1]$ is already known. Compute $X^R_{17}[6]$ with $X^R_{19}[0], X^R_{18}[1]$. Compress the data into 2^8 texts of $(X^R_{18}[7], X^R_{17}[6])$.
21. Guess $K_{16}[6]$. From Fig. 8, this can be done by 2-bit guess of $\kappa_{16}[73,74]$, which is colored in green. Compute $X^R_{16}[5]$ with $X^R_{18}[7], X^R_{17}[6]$. Compress the data into 2^4 texts of $X^R_{16}[5]$.
22. Guess 4 bits of $K_{15}[5]$, and compute the final sum for $Z_{15}[4]$. We store the result together with the guessed key value in a list $L_{Z_{15}}$ The new known bits of the key state upto this step are also shown in green in Fig. 8.

Complexity for $\bigoplus Z_{15}[4]$. The complexity for each step is estimated as a product of the previous data size and the total number of guessed bits, and is shown in the last column of Table 3. In total, it requires $2^{64}+2^{64}+2^{64}+2^{64}+2^{61}+2^{65}+2^{69}+2^{67}+2^{67}+2^{65}+2^{65}+2^{69}+2^{73}+2^{69}+2^{69}+2^{69}+2^{66}+2^{62}+2^{60}+2^{60} \approx 2^{73.46}$ 1-round computations. After Step 22, we obtain a list $L_{Z_{15}}$ with 2^{56} entries which contains 60-bit information; $\bigoplus Z_{15}[4]$ and 56-bit key values.

Table 3. Summary of the computation of $\bigoplus Z_{15}[4]$ in 22-round attack

Step	Guessed key	#guessed bits (total)	New value	Discarded values	#texts	Values in the set	Complexity
2		0			2^{60}	$X_{22}^{L}[0,1,2,4,5,6,7]$, $X_{22}^{R}[0,1,2,3,4,5,6,7]$	
3	$K_{21}[0]$	4	$X_{21}^{R}[0]$	$X_{22}^{L}[2]$, $X_{22}^{R}[0]$	2^{56}	$X_{22}^{L}[0,1,4,5,6,7]$, $X_{22}^{R}[1,2,3,4,5,6,7]$, $X_{21}^{R}[0]$	$2^{60+4}=2^{64}$
4	$K_{21}[4]$	8	$X_{21}^{R}[4]$	$X_{22}^{L}[6]$, $X_{22}^{R}[4]$	2^{52}	$X_{22}^{L}[0,1,4,5,7]$, $X_{22}^{R}[1,2,3,5,6,7]$, $X_{21}^{R}[0,4]$	$2^{56+8}=2^{64}$
5	$K_{21}[6]$	12	$X_{21}^{R}[5]$	$X_{22}^{L}[7]$, $X_{22}^{R}[6]$	2^{48}	$X_{22}^{L}[0,1,4,5]$, $X_{22}^{R}[1,2,3,5,7]$, $X_{21}^{R}[0,4,5]$	$2^{52+12}=2^{64}$
6	$K_{20}[0]$	16	$X_{20}^{R}[0]$	$X_{22}^{R}[2]$, $X_{21}^{R}[0]$	2^{44}	$X_{22}^{L}[0,1,4,5]$, $X_{22}^{R}[1,3,5,7]$, $X_{21}^{R}[4,5]$, $X_{20}^{R}[0]$	$2^{48+16}=2^{64}$
7	$K_{21}[7]$	17	$X_{21}^{R}[3]$	$X_{22}^{L}[5]$	2^{44}	$X_{22}^{L}[0,1,4]$, $X_{22}^{R}[1,3,5,7]$, $X_{21}^{R}[3,4,5]$, $X_{20}^{R}[0]$	$2^{44+17}=2^{61}$
8	$K_{21}[1]$	21	$X_{21}^{R}[6]$	$X_{22}^{L}[0]$	2^{44}	$X_{22}^{L}[1,4]$, $X_{22}^{R}[1,3,5,7]$, $X_{21}^{R}[3,4,5,6]$, $X_{20}^{R}[0]$	$2^{44+21}=2^{65}$
9	$K_{20}[3]$	25	$X_{20}^{R}[7]$	$X_{22}^{R}[1]$, $X_{21}^{R}[3]$	2^{40}	$X_{22}^{L}[1,4]$, $X_{22}^{R}[3,5,7]$, $X_{21}^{R}[4,5,6]$, $X_{20}^{R}[0,7]$	$2^{44+25}=2^{69}$
10	$K_{19}[7]$	27	$X_{19}^{R}[3]$	$X_{21}^{R}[5]$, $X_{20}^{R}[7]$	2^{36}	$X_{22}^{L}[1,4]$, $X_{22}^{R}[3,5,7]$, $X_{21}^{R}[4,6]$, $X_{20}^{R}[0]$, $X_{19}^{R}[3]$	$2^{40+27}=2^{67}$
11	$K_{20}[6]$	31	$X_{20}^{R}[5]$	$X_{22}^{R}[7]$, $X_{21}^{R}[6]$	2^{32}	$X_{22}^{L}[1,4]$, $X_{22}^{R}[3,5]$, $X_{21}^{R}[4]$, $X_{20}^{R}[0,5]$, $X_{19}^{R}[3]$	$2^{36+31}=2^{67}$
12	$K_{19}[5]$	33	$X_{19}^{R}[2]$	$X_{21}^{R}[4]$, $X_{20}^{R}[5]$	2^{28}	$X_{22}^{L}[1,4]$, $X_{22}^{R}[3,5]$, $X_{20}^{R}[0]$, $X_{19}^{R}[2,3]$	$2^{32+33}=2^{65}$
13	$K_{21}[3]$	37	$X_{21}^{R}[7]$	$X_{22}^{L}[1]$	2^{28}	$X_{22}^{L}[4]$, $X_{22}^{R}[3,5]$, $X_{21}^{R}[7]$, $X_{20}^{R}[0]$, $X_{19}^{R}[2,3]$	$2^{28+37}=2^{65}$
14	$K_{21}[5]$	41	$X_{21}^{R}[2]$	$X_{22}^{L}[4]$	2^{28}	$X_{22}^{R}[3,5]$, $X_{21}^{R}[2,7]$, $X_{20}^{R}[0]$, $X_{19}^{R}[2,3]$	$2^{28+41}=2^{69}$
15	$K_{20}[7]$	45	$X_{20}^{R}[3]$	$X_{22}^{R}[5]$, $X_{21}^{R}[7]$	2^{24}	$X_{22}^{R}[3]$, $X_{21}^{R}[2]$, $X_{20}^{R}[0,3]$, $X_{19}^{R}[2,3]$	$2^{28+45}=2^{73}$
16	$K_{18}[2]$	45	$X_{18}^{R}[1]$	$X_{20}^{R}[3]$, $X_{19}^{R}[2]$	2^{20}	$X_{22}^{R}[3]$, $X_{21}^{R}[2]$, $X_{20}^{R}[0]$, $X_{19}^{R}[3]$, $X_{18}^{R}[1]$	$2^{24+45}=2^{69}$
17	$K_{20}[2]$	49	$X_{20}^{R}[1]$	$X_{22}^{R}[3]$	2^{20}	$X_{21}^{R}[2]$, $X_{20}^{R}[0,1]$, $X_{19}^{R}[3]$, $X_{18}^{R}[1]$	$2^{20+49}=2^{69}$
18	$K_{18}[3]$	49	$X_{18}^{R}[7]$	$X_{20}^{R}[1]$, $X_{19}^{R}[3]$	2^{16}	$X_{21}^{R}[2]$, $X_{20}^{R}[0]$, $X_{18}^{R}[1,7]$	$2^{20+49}=2^{69}$
19	$K_{19}[0]$	50	$X_{19}^{R}[0]$	$X_{21}^{R}[2]$, $X_{20}^{R}[0]$	2^{12}	$X_{19}^{R}[0]$, $X_{18}^{R}[1,7]$	$2^{16+50}=2^{66}$
20	$K_{17}[1]$	50	$X_{17}^{R}[6]$	$X_{19}^{R}[0]$, $X_{18}^{R}[1]$	2^{8}	$X_{18}^{R}[7]$, $X_{17}^{R}[6]$	$2^{12+50}=2^{62}$
21	$K_{16}[6]$	52	$X_{16}^{R}[5]$	$X_{18}^{R}[7]$, $X_{17}^{R}[6]$	2^{4}	$X_{16}^{R}[5]$	$2^{8+52}=2^{60}$
22	$K_{15}[5]$	56	$Z_{15}[4]$	$X_{16}^{R}[5]$	2^{0}	$\bigoplus Z_{15}[4]$	$2^{4+56}=2^{60}$

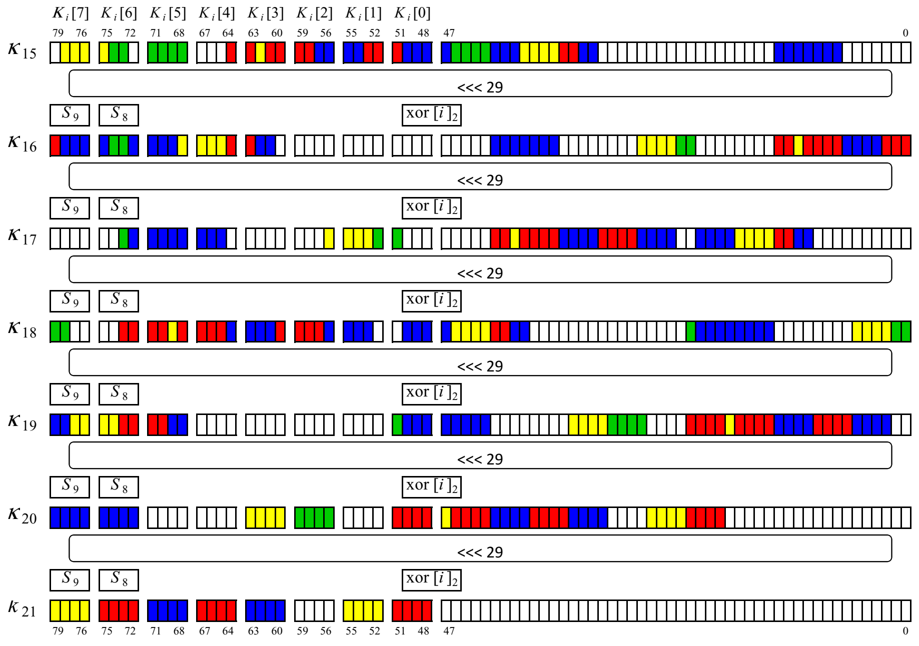


Fig. 8. Key state for the last 7 rounds

New Impossible Differential Attack on SAFER$_+$ and SAFER$_{++}$[⋆]

Jingyuan Zhao[1,2], Meiqin Wang[1,2,⋆⋆], Jiazhe Chen[1,2], and Yuliang Zheng[1,2,3]

[1] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan 250100, China
[2] School of Mathematics, Shandong University, Jinan 250100, China
[3] Department of Software and Information Systems, UNC Charlotte, 9201 University City Blvd, Charlotte, NC 28223, USA

Abstract. SAFER+ was a candidate block cipher for AES with 128-bit block size and a variable key sizes of 128, 192 or 256 bits. Bluetooth uses customized versions of SAFER+ for security. The numbers of rounds for SAFER+ with key sizes of 128, 192 and 256 are 8, 12 and 16, respectively. SAFER++, a variant of SAFER+, was among the cryptographic primitives selected for the second phase of the NESSIE project. The block size is 128 bits and the key size can take either 128 or 256 bits. The number of rounds for SAFER++ is 7 for keys of 128 bits, and 10 for keys of 256 bits. Both ciphers use PHT as their linear transformation. In this paper, we take advantage of properties of PHT and S-boxes to identify 3.75-round impossible differentials for SAFER++ and 2.75-round impossible differentials for SAFER+, which result in impossible differential attacks on 4-round SAFER+/128(256), 5-round SAFER++/128 and 5.5-round SAFER++/256. Our attacks significantly improve previously known impossible differential attacks on 3.75-round SAFER+/128(256) and SAFER++/128(256). Our attacks on SAFER+/128(256) and SAFER++/128(256) represent the best currently known attack in terms of the number of rounds.

Keywords: SAFER+, SAFER++, Impossible Differential, PHT, Bluetooth.

1 Introduction

SAFER+, designed by Massey, Khachatrian and Kuregian, was a candidate block cipher for AES with 128-bit block size and a variable key sizes of 128, 192 or

[⋆] This work was supported by NSFC Projects(No.61133013, No.61070244 and No.61103237), by 973 Project (No.2013CB834205) and No.2012GGX10135 as well as Research Foundation of Shandong University(No.2012JC018 and No.2010TS069).
[⋆⋆] Corresponding author.

256 bits, denoted by SAFER+/128, SAFER+/192 and SAFER+/256, respectively [8]. Since some weaknesses to the key schedules of SAFER+/192 and SAFER+/256 were discovered, Massey et al. changed the key schedule algorithms later. In this paper, we will use the remedied key schedule algorithms as in [12]. Bluetooth uses custom algorithms based on SAFER$_+$ for key derivation and authentication as MAC [4]. SAFER$_{++}$ was submitted to the NESSIE project [13] and was among the primitives selected for the second phase of this project [9]. The block size is 128-bit and the key size can be taken as 128-bit and 256-bit. The two ciphers have common S-boxes derived from exponentiation and discrete logarithm functions and share the Pseudo-Hadamard-like mixing transforms (PHT) but have different ways to use it. They also share the methods to perform key-mixing with two-commutative operations.

Several cryptanalytic results on SAFER$_+$ and SAFER$_{++}$ have been published. Nakahara et al. gave the non-homomorphic linear cryptanalysis for 3.25 rounds of SAFER+/128 and 3 rounds of SAFER++/128 and SAFER++/256 [10, 11]. Piret et al. gave the integral cryptanalysis for 4.25 rounds of SAFER++/128 and 4.75 rounds of SAFER++/256 [14]. Biryukov et al. gave the multiset attack on 4.5 rounds of SAFER++/128 and the boomerang attack on 5.5 rounds of SAFER++/128. For the impossible differential cryptanalysis, Nakahara et.al also gave the impossible differential cryptanalysis for 2.75 rounds of SAFER+/128 and SAFER++/128 [11, 12]. Then Behnam et al. claimed they could attack 4 rounds of SAFER++/128 with the impossible differential cryptanalysis [1], however, their attack only worked for 4-round SAFER++/128 without the final whitening-key layer, so their attack was a 3.75-round attack. Zheng et al. gave the impossible differential attacks on 3.75 rounds of SAFER+/128 (SAFER+/256) and 3.75 rounds of SAFER++/128 (SAFER++/256).

The impossible differential attack, which was independently proposed by Biham et al. [2] and Knudsen [5], is a popular cryptanalytic method. The attack starts with finding an input difference that can never result in an output difference, which will produce an impossible differential. By adding rounds before and/or after the impossible differential, one can collect pairs with certain plaintext and ciphertext differences. If there exists a pair that meets the input and output values of the impossible differential under some subkey bits, these bits must be wrong. In this way, we discard as many wrong keys as possible and exhaustively search the rest of the keys, this phase is called key recovery phase. The early abort technique is usually used during the key recovery phase, that is, one does not guess all the subkey bits at once, but guess some subkey bits instead to discard some pairs that do not satisfy certain conditions step by step. In this case, we can discard the unwished pairs as soon as possible to reduce the time complexity.

Our Contributions. By delicately choosing the positions and the number of the active S-boxes in the first round, we can identify 3.75 rounds impossible differentials for SAFER$_{++}$, which are significantly better than the previous 2.75-round impossible differentials [1, 16].

At the same time, we also identify 2.75-round impossible differentials for SAFER+. Although our impossible differentials work for the same number of rounds as those in [1, 16], they will result in less active S-boxes in the first round or the last round, and then the number of guessed subkey bytes will be reduced during the key recovery phase, so we can attack four full rounds with the final whitening key layer; the attack is better than the 3.75-round attack in [16] and the four-round attack without the final whitening key layer in [1]. Our attacks on bluetooth ciphers SAFER+/128 and SAFER+/256 are the best attacks according to the number of rounds. Specially, our attack on SAFER+/128 is the first attack on half of the full-round SAFER+.

Our attack on SAFER++/128 can work for 5 rounds with the final whitening key layer, which is much better than the previous impossible differential attack for 3.75 rounds in [1, 16]. However, the best attack on SAFER++/128 is the boomerang attack for 5.5 rounds [3]. Our attack on SAFER++/256 can work for 5.5 rounds. Although our attack on SAFER++/128 is not as good as those in [3], we greatly improve the impossible differential attacks in [1, 16] and our attacks are the best chosen plaintext attacks.

The only difference for the components of round functions for SAFER+ and SAFER++ is the linear transformation; the linear transformation of SAFER++ is more complicated than that of SAFER+, so the designers use less rounds for SAFER++ than SAFER+. It seems that the linear transformation for SAFER++ is much more secure than SAFER+, however, our attack shows that SAFER++ is less resistant to impossible differential attack than SAFER+, because the diffusion of the inverse linear layer for SAFER++ is much weaker.

We summarize our results of SAFER+ and SAFER++, as well as the major previous results in Table 1.

The rest of the paper is organized as follows. We give the brief descriptions of SAFER+ and SAFER++ in Sect. 2. Section 3 identifies the impossible differentials for SAFER+ and SAFER++. The impossible differential cryptanalysis of SAFER+/128 and SAFER+/256 is presented in Sect. 4. Section 5 gives the impossible differential cryptanalysis of SAFER++/128 and SAFER++/256. Finally, Sect. 6 concludes this paper.

2 Brief Descriptions of SAFER$_+$ and SAFER$_{++}$

This section contains short descriptions of SAFER+ and SAFER++. For more details, see [8,9]. SAFER+ (SAFER++) is a 128-bit SPN block ciphers with variable key sizes of 128, 192 or 256 bits, denoted by SAFER+/128, SAFER+/192 and SAFER+/256 (SAFER++/128 and SAFER++/256). The round function of SAFER+ (SAFER++) consists of an upper key layer, a nonlinear layer, a lower key layer and a linear transformation. After the final round, an additional key-addition whitening similar to the upper key layer is added. The numbers of rounds of SAFER+/128 and SAFER+/256 are 8 and 16, respectively. The numbers of rounds of SAFER++/128 and SAFER++/256 are 7 and 10, respectively. Among the components of the round functions of SAFER+

Table 1. Summary of attacks on SAFER+ and SAFER++

Cipher	Attack	#Rounds	Data	Time (Encryptions)	Memory (Bytes)	Source
+/128	ID	2.75	2^{64}CP	2^{58}	2^{104}	[12]
+/128	LNH	3.25	2^{101}KP	2^{141}	2^{108}	[10]
+/128	ID	3.75	2^{78}CP	2^{72}	2^{68}	[16]
+/128	**ID**	**4**	**$2^{122.4}$CP**	**2^{121}**	**$2^{87.4}$**	**Sect.4**
+/256	ID	3.75	2^{78}CP	2^{72}	2^{68}	[16]
+/256	**ID**	**4**	**$2^{124.4}$CP**	**2^{216}**	**$2^{89.4}$**	**Sect.4**
++/128	LNH	3	2^{81}KP	2^{105}	2^{88}	[10]
++/128	ID	2.75	2^{64}CP	2^{58}	2^{104}	[12]
++/128	Integral	4	2^{64}CP	2^{117}	2^{71}	[14]
++/128	Integral	4.25	–	–	–	[3]
++/128	Multiset	4.5	2^{48}CP	2^{100}	2^{55}	[3]
++/128	Boomerang	5.5	2^{108}CP/ACC	2^{116}	2^{55}	[3]
++/128	ID	3.75	2^{23}CP	2^{84}	2^{75}	[1]
++/128	ID	3.75	2^{78}CP	2^{63}	2^{62}	[16]
++/128	**ID**	**5**	**2^{124}CP**	**2^{121}**	**2^{97}**	**Sect.5**
++/256	LNH	3	2^{81}KP	2^{105}	2^{88}	[10]
++/256	Integral	4	2^{64}CP	2^{149}	2^{71}	[14]
++/256	Integral	4.75	–	–	–	[14]
++/256	ID	3.75	2^{78} CP	2^{71}	2^{70}	[16]
++/256	**ID**	**5.5**	**2^{124}CP**	**2^{246}**	**2^{97}**	**Sect.5**

CP: Chosen Plaintext; KP: Known Plaintext; ACC: Adaptive Chosen Ciphertext
ID: Impossible Differential; LNH: Linear(Non-Homomorphic).

and SAFER++, only the linear transformation is different. SAFER+ uses a 2-point pseudo Hadamard transformation(2-PHT) while SAFER++ uses a 4-point pseudo Hadamard transformation(4-PHT).

2.1 The Keyed Non-linear Layer

Since SAFER+ and SAFER++ are byte-oriented ciphers, the input plaintext block is initially splited into 16 bytes to combine with the 16 bytes subkey. Bytes 0, 3, 4, 7, 8, 11, 12, and 15 of the subkey are XORed to the corresponding bytes of the block, while bytes 1, 2, 5, 6, 9, 10, 13, and 14 of the subkey are combined with the corresponding bytes using addition modulo 256. The nonlinear layer is based on two different 8-to-8 bit functions, X and L,

$$\mathrm{X}(a) = (45^a \bmod 257) \bmod 256,$$
$$\mathrm{L}(a) = \log_{45}{}^a \bmod 257,$$

with the special case that L(0) = 128, making X and L mutually inverse. We call the layer including X and L as S-box layer. In this layer, bytes 0, 3, 4, 7, 8, 11, 12, and 15 are sent through the function X, and L is applied to bytes 1, 2, 5, 6, 9, 10, 13, and 14. The lower key layer mixes a 16-byte subkey to the output blocks from the X and L functions. Bytes 2, 3, 6, 7, 10, 11, 14 and 15 of the subkey are XORed to the corresponding bytes of the block and bytes 1, 4, 5, 8, 9, 12, 13 and 16 of the subkey and blocks are combined using addition modulo 256.

2.2 The Linear Layer

The linear transformation of SAFER+ (SAFER++) is constructed by two parts: the first is a permutation and the second is a 2-PHT(4-PHT) to two group of 2-branch(four group of 4-branch). The 2-PHT(4-PHT) can be implemented with two(six) modular additions. The linear layers can be expressed by matrices.

2.3 The Key Schedule

The key schedule of SAFER++ is same as that of SAFER+ for the same key size and the key schedules of 128-bit and 256-bit master keys are different. Firstly we introduce the 128-bit key schedule: $K=(k^1, k^2,\cdots, k^{16})$ is the 128-bit master key. From the 16 bytes of master key we get the 17-th byte:

$$k^{sp1} = \bigoplus_{i=1}^{16} k^i.$$

The 256-bit mater key is $K=(k^1,\cdots, k^{16}, k^{17},\cdots, k^{32})$. Different from 128-bit key schedule, the 256-bit master key is splitted into two 128-bit blocks. The first one is used to produce the upper key layer of each round and the final key addition, and the second one is used to produce the lower key layer of each round. k^{sp1} is computed as in SAFER+/128 and SAFER++/128. In addition, another subkey byte k^{sp2} can be computed with

$$k^{sp2} = \bigoplus_{i=17}^{32} k^i.$$

The recovered master key was marked in the Fig.2 and Fig.4. We do not depict them here in detail.

3 Impossible Differentials of SAFER$_+$ and SAFER$_{++}$

In this section, we will show how to identify 2.75 rounds impossible differentials for SAFER+ and 3.75 rounds impossible differentials for SAFER++ .

3.1 Notations

In this paper we use the following notations: T_r^I denotes the input of the r-th round, T_r^U, T_r^S, T_r^L and T_r^A denote the output values of the upper key layer, the S-boxes, the lower key layer and the linear layer in round r, respectively. So $T_r^I = T_{r-1}^A$ for $r \geq 2$. Δ represents the modular subtraction difference in $\mathbb{F}_{2^8}$. $*$ means the undetermined value. $(\Delta T_r^i)_j$ stands for the j-th byte of ΔT_r^i, $0 \leq j \leq 15$. C_j means the j-th byte of the ciphertext, $0 \leq j \leq 15$.

3.2 Impossible Differentials of SAFER$_+$ and SAFER$_{++}$

Firstly, we will introduce three propositions related with S-boxes, XOR and the modular addition.

Proposition 1 (see [7]). *For any byte pair (p, p'), if $(p-p') \equiv 0x80 \ (mod\, 256)$, then the output difference $X(p) \boxminus X(p')$ is always odd.*

Proposition 2 (see [6]). *For any byte pair (p, p'), $p \oplus p' = 0x80$ always means $(p - p') \equiv 0x80 \ (mod\, 256)$, and vice versa.*

Proposition 3 (see [16]). *For any given byte pair (p, p'), if $p \oplus p'$ is odd, then $(p \boxplus k) \oplus (p' \boxplus k)$ is odd. Also, if $p \boxplus p'$ is odd, $(p \oplus k) \boxminus (p' \oplus k)$ is odd. Here, k can take any value in $\mathbb{Z}_{256}$.*

Based on the propositions, we can get 2.75-round impossible differentials for SAFER+ and 3.75-round impossible differentials for SAFER++.

Theorem 1. *For SAFER+, if the output difference of the S-boxes in the first round ΔT_1^S is $(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 80_x, 0)$ and the output difference of the upper key layer in the fourth round ΔT_4^U is (0, a, 0, 0, 0, b, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), where a and b are any non-zero values. Such 2.75-round differential in Fig. 1 is impossible if a and b satisfy one of the three following conditions: $a + b = 0$, $8a + b = 0$, $a + 8b = 0$.*

Theorem 2. *For SAFER++, if the output difference of the S-boxes in the first round ΔT_1^S is $(0, 80_x, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 80_x, 0)$, and the output difference of the upper key layer in the fifth round is*

$$\Delta T_5^U = (0, a, -a, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),$$

where a is any non-zero value, such 3.75-round differential is impossible in Fig. 3.

The proof of the above theorems is available in the full version of this paper [17].

4 Impossible Differential Attacks on SAFER$_+$

In this section, we will use our 2.75-round impossible differential to recover the keys for four rounds of SAFER+/128 in Fig.2 and four rounds of SAFER+/256. First of all, in order to filter out the pairs as soon as possible, we derive the relation between the ciphertext bytes difference in Proposition 4.

Proposition 4. *For four full-round of SAFER+/128 or SAFER+/256, if the pairs have the difference ΔT_5^U=(0, a, −a, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), the differences for their corresponding ciphertext pairs have the following relations,*

$$\Delta C_1 - \Delta C_2 = 0, 2\Delta C_1 - \Delta C_6 = 0, \Delta C_5 - \Delta C_{10} = 0, \tag{1}$$

$$\Delta C_5 + \Delta C_9 - 5\Delta C_{13} = 0, \Delta C_6 + 2\Delta C_{14} - 6\Delta C_{13} = 0, \Delta C_1 + \Delta C_5 + \Delta C_6 - 7\Delta C_{13} = 0. \tag{2}$$

4.1 Impossible Differential Attack on SAFER$_+$/128

By placing the 2.75-round impossible differential on round 0.5-3.25, we can attack from round 1 to round 4. This is described in Fig.2. In order to show the effect of the key schedule, we denote our guessed subkey bits with their related master key bytes instead of themselves in Fig.2.

Data Collection. We first construct $2^{114.4}$ structures of plaintexts, where in each structure the plaintext byte P_{14} takes all values, whereas the other bytes are fixed. For each structure, ask for the encryption of the plaintexts to get the corresponding ciphertexts. In order to filter out the wrong pairs with Equation (1) in Proposition 4, we construct a hash table indexed by $(C_1 - C_2|2C_1 - C_6|C_5 - C_{10})$ and put 2^8 corresponding ciphertexts into the hash table. Then we combine the ciphertext pairs in the same entity in the hash table which can satisfy Equation (1) in Proposition 4. On average there are about $2^{15}/2^{24} = 2^{-9}$ remaining pairs for each structure. Then we will further filter out the wrong pairs with Equation (2) in Proposition 4, so we construct another hash table indexed by $(C_5 + C_9 - 5C_{13}|C_6 + 2C_{14} - 6C_{13}|C_1 + C_5 + C_6 - 7C_{13})$ and put the 2^{-9} pairs into the hash table. There will remain pairs in the same entity in the hash table which can satisfy Equation (2) in Proposition 4. Now we can easily get the value of A and B in Fig.2 from the ciphertext difference for each remaining ciphertext pair. On average, there are $2^{-9}/2^{24} = 2^{-33}$ remaining pairs for each structure.

Key Recovery. In order to find if there are pairs obtained from the data collection phase that may follow the differential in Fig.2, we need to guess the key bits and sieve the pairs in round 1 and round 4. From Fig.2, in round 1, we need to guess the 15-th subkey byte in the upper key layer which is related to the master key byte k^{15}. In round 4, 16 subkey bytes of the lower key layer which are related to the master key bytes (k^9, k^{10}, k^{11}, …, k^{16}, k^{sp1}, k^1, k^2,…, k^7) and we will guess partial bits for these 16 key bytes. We also need to guess the second and the sixth subkey bytes of the upper key layer which are related to the master key bytes (k^9, k^{13}). We proceed the key recovery phase for the remaining pairs as follows:

- Step 1. For all 2^8 possible values for the 15-th subkey byte of the upper key layer of the first round which depends on k^{15}, encrypt each plaintext of 2^{-33} remaining pairs for $\frac{1}{2}$ round to get the output differences of the S-boxes in the first round, which should satisfy $(\Delta T_1^S)_{14} = 80_x$. Then the number of remaining pairs is about 2^{-41}. The total number of guessed subkey bits in this step is 8.
- Step 2.
  - Step 2.1 In the final whitening key layer, there are eight XOR operations. As we get the ciphertext differences for the eight bytes, we can directly get the value for the least significant bit of $(\Delta T_4^A)_j, j \in \{0, 3, 4, 7, 8, 11, 12, 15\}$ without guessing the corresponding subkey value. Because we have known the value for A and B in the data collection

phase, we can derive the difference values for the eight least significant bits from A and B. Then we can sieve the pairs with the eight conditions, as a result, $2^{-41}/2^8 = 2^{-49}$ pairs remain for each structure.

- Step 2.2 For all 2^8 values of the least significant bits of the eight subkey bytes which depend on k^9, k^{12}, k^{13}, k^{16}, k^{sp1}, k^3, k^4, and k^7, respectively, compute the second least significant bits for $(\Delta T_4^A)_j, j \in \{0,3,4,7,8,11,12,15\}$ for all remaining pairs and verify if they equal to the corresponding values obtained from A and B. If not, we discard the pair. In a similar way, we guess the eight subkey bytes from the second least significant bit to the seventh least significant bit one by one and sieve the pairs according the conditions derived from A and B. As a result, about $2^{-49}/2^{8*7} = 2^{-105}$ pairs are obtained. The total number of new guessed subkey bits in this step is 56.

– Step 3. In this step, we will compute the value for a and b corresponding to $(\Delta T_4^U)_1$ and $(\Delta T_4^U)_5$.
so in order to calculate the values of them the bits depending on the following keys should be guessed:
$(T_4^U)_1$: k^{13}, k^{14}, k^{sp1}, k^1, k^6, k^9; the seven least significant bits of k^{11}, k^2, k^3, k^7; the six least significant bits of k^{10}, k^{12}, k^{15}, k^{16}; the five least significant bits of k^4; the four least significant bits of k^5.
$(T_4^U)_5$: k^9, k^{13}, k^{14}, k^{sp1}, k^1; the seven least significant bits of k^{10}, k^{15}, k^{16}, k^4, k^6; the six least significant bits of k^{11}, k^{12}, k^5, k^7; the five least significant bits of k^2; the four least significant bits of k^3.
Here some subkey bits have been guessed in the previous steps, so the total number of the new involved subkey bits in this step is 54.
For each pair obtained from Step 2.2, compute a and b to verify if they satisfy any one of the three relations for the three impossible differentials. If so, the 54-bit subkey should be discarded. After processing all the pairs, if any values for the 54-bit subkey remain, we output them with the guessed 64-bit subkey, and exhaustively search them with the remaining 10 bits subkey by trial encryption. Otherwise, we try another guess for 64-bit subkey from Step 1 and Step 2.

The data complexity of the attack is $2^{122.4}$ chosen plaintexts. In the data collection phase, the time complexity is about $2^{122.4} \times 3 = 2^{124}$ modular subtraction operations which is equivalent to 2^{119} encryptions and the memory complexity is about $2^{81.4} \times 2 \times 32 = 2^{87.4}$ bytes for the remaining pairs. In Step 1, the time complexity is about $2 \times 2^8 \times 2^{114.4-33} \times \frac{1}{2} \times \frac{1}{16} \times \frac{1}{4} = 2^{83.4}$ encryptions and the memory complexity for remaining pairs is less than that in the data collection phase. In Step 2, the time complexity is about $2 \times 2^8 \times 2^{114.4-41} \times 8 \times 8 \approx 2^{88.4}$ XOR operations and $2^{87.4}$ modular subtraction operations. The memory complexity for remaining pairs is less than that in Step 1. In Step 3, The expected number of remaining 118-bit subkey guesses is about $2^{118} \times (1-\frac{3}{2^8})^{2^{114.4-105}} \approx 2^{108}$. Since each of the remaining key guesses has to be exhaustively searched with the other 2^{10} key values, so the time complexity of this step is about $2 \times 2^{118} \times [1 + (1-\frac{3}{2^8}) + (1-\frac{3}{2^8})^2 + \ldots + (1-\frac{3}{2^8})^{2^{9.4}}] \times \frac{2}{16} \times \frac{1}{4} + 2^{108+10} \approx 2^{120.7}$

encryptions. Thus the total time complexity is about 2^{121} encryptions and the memory complexity is about $2^{87.4}$ bytes.

4.2 Impossible Differential Attack on SAFER$_+$/256

The only difference between the attacks on SAFER+/256 and SAFER+/128 is the difference in the key schedule. The detailed attack procedure is available in the full version of this paper [17].

The data complexity of the attack is $2^{124.4}$ chosen plaintexts. The total time complexity is 2^{216} encryptions and the memory complexity is about $2^{89.4}$ bytes.

5 Impossible Differential Attacks on SAFER$_{++}$

In this section, we will use the 3.75-round impossible differentials for SAFER++ in Section 3 to recover the keys for SAFER++/128 and SAFER++/256. First of all, in order to filter out the pairs as soon as possible, we derive the relations between the ciphertext bytes difference in Proposition 5.

Proposition 5. *For five full rounds of SAFER++/128 or SAFER++/256, if the pairs have the difference ΔT_5^U=(0, a, b, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), their corresponding output difference has the following relations,*

$$(\Delta T_6^U)_5 - (\Delta T_6^U)_{10} = 0, (\Delta T_6^U)_1 - (\Delta T_6^U)_9 = 0, (\Delta T_6^U)_6 - (\Delta T_6^U)_{14} = 0, \quad (3)$$

$$\begin{array}{r} (\Delta T_6^U)_1 + (\Delta T_6^U)_6 - 3(\Delta T_6^U)_5 = 0, \\ (\Delta T_6^U)_2 + (\Delta T_6^U)_{13} - 5(\Delta T_6^U)_5 = 0, \\ 3(\Delta T_6^U)_1 + (\Delta T_6^U)_{13} - 7(\Delta T_6^U)_5 = 0. \end{array} \quad (4)$$

5.1 Impossible Differential Attack on SAFER$_{++}$/128

By placing the 3.75-round impossible differential on round 0.5-4.25, we can attack SAFER++/128 from round 1 to round 5. This is described in Fig.4.

Data Collection. We first construct 2^{108} structures of plaintexts, where in each structure the plaintext bytes P_1 and P_{14} take all values, whereas the other bytes are fixed. For each structure, ask for the encryption of the plaintexts to get the corresponding ciphertexts. In order to filter out the wrong pairs with Equation (3) and Equation (4) in Proposition 5, we construct two hash tables indexed by $(C_5 - C_{10}|C_1 - C_9|C_6 - C_{14})$ and $(C_1 + C_6 - 3C_5|C_2 + C_{13} - 5C_5|3C_1 + C_{13} - 7C_5)$ and put the pairs into the hash tables. Now we can easily get the value of A and B in Fig.4 from the ciphertext difference for any remaining ciphertext pair. On average, there are 2^{-17} remaining pairs for each structure.

Key Recovery. In order to find if there are pairs obtained from the data collection phase that may follow the differential in Fig.4, we need to guess the key bits and sieve the pairs in round 1 and round 5. From Fig.4, in round 1, we need to guess the second and 15-th subkey bytes in the upper key layer which are related to the master key bytes k^2 and k^{15}, respectively. In round 5, 16 final whitening subkey bytes are related to the master key bytes $(k^{11}, k^{12}, k^{13}, \ldots, k^{16}, k^{sp1}, k^1, k^2,$
$\ldots, k^9)$ and we will guess partial bits for the 16 subkey bytes. We also need to guess the second and the third subkey bytes of the lower key layer in round 5 which are related to the master key bytes (k^{11}, k^{12}), respectively. We proceed the key recovery phase for the remaining pairs as follows:

- Step 1. For all 2^{16} possible values for the second and the 15-th bytes of upper key layer of the first round which depend on k^2 and k^{15}, for each structure encrypt each plaintext pair of the 2^{-17} remaining pairs for $\frac{1}{2}$ round to get the output differences of the S-boxes in the first round, which should satisfy $(\Delta T_1^S)_1 = (\Delta T_1^S)_{14} = 80_x$. Then the number of remaining pairs is about 2^{-33}. The total number of guessed subkey bits in this step is 16.
- Step 2.
  - Step 2.1 In the final whitening key layer, there are eight XOR operations. As we get the ciphertext differences for the eight bytes, we can directly get the value for the least significant bit of $(\Delta T_5^A)_j, j \in \{0, 3, 4, 7, 8, 11, 12, 15\}$ without guessing the corresponding subkey value. Because we have known the value for A and B in the data collection phase, we can derive the 8 bits difference for the least significant bits from A and B. Then we can sieve the pairs with the eight conditions, as a result, $2^{-33}/2^8 = 2^{-41}$ pairs remain for each structure.
  - Step 2.2 For all 2^8 values for the least significant key bit of eight subkey bytes which depend on k^{11}, k^{14}, k^{15}, k^1, k^2, k^5, k^6, k^9, respectively, compute the second least significant bits for $(\Delta T_5^A)_j, j \in \{0, 3, 4, 7, 8, 11, 12, 15\}$ for all remaining pairs and verify if they equal to the corresponding value derived from A and B. If not, we discard the pair. In a similar way, we guess eight subkey bytes from the second least significant bits to the seventh least significant bits one by one which depend on k^{11}, k^{14}, k^{15}, k^1, k^2, k^5, k^6, k^9, respectively, then we sieve the pairs according to the conditions derived from A and B. As a result, about $2^{-41}/2^{8*7} = 2^{-97}$ pairs are obtained. The total number of new guessed subkey bits in this step is 42.
- Step 3. In this step, we will compute the value for a and $-a$ corresponding to $(\Delta T_5^U)_1$ and $(\Delta T_5^U)_2$. Similar to the attack on SAFER+, we only guess the subkey bits that are necessary, the total number of the new involved subkey bits in this step is 52.

  For each pair obtained from Step 2.2, compute the value for $(\Delta T_5^U)_1$ and $(\Delta T_5^U)_2$ to verify if $(\Delta T_5^U)_1 = -(\Delta T_5^U)_2$. If so, the 52-bit subkey should be discarded. After processing all the pairs, if any values for the 52-bit subkey remain, we output them with the guessed 58-bit subkey, and exhaustively search them with the remaining 18 bits key by trial encryption. Otherwise, we try another guess for 58-bit subkey from Step 1 and Step 2.

The data complexity of the attack is 2^{124} chosen plaintexts. In the data collection phase, the time complexity is about $2^{124} \times 3 = 2^{125.6}$ modular subtraction operations, which is equivalent to $2^{120.6}$ times of encryptions and the memory complexity is about $2^{91} \times 2 \times 32 = 2^{97}$ bytes for the remaining pairs. In Step 1, the time complexity is about $2 \times 2^{16} \times 2^{91} \times \frac{1}{2} \times \frac{1}{16} \times 14 = 2^{101}$ encryptions. In Step 2, the time complexity is about $2 \times 2^{16} \times 2^{75} \times 8 \times 8 \approx 2^{98}$ XOR operations and 2^{97} modular subtraction operations. In Step 3, the expected number of remaining 110-bit key guesses is about $2^{110} \times (1 - \frac{1}{2^8})^{2^{108-97}} \approx 2^{100}$. Sine each of the remaining key guesses has to be exhaustively searched with the other 2^{18} key values, so the time complexity of this step is about $2 \times 2^{110} \times [1 + (1 - \frac{1}{2^8}) + (1 - \frac{1}{2^8})^2 + \ldots + (1 - \frac{1}{2^8})^{2^{11}}] \times \frac{2}{16} \times \frac{1}{4} + 2^{118} \approx 2^{118}$ encryptions. Thus the total time complexity is about 2^{118} encryptions and the memory complexity is about 2^{97} bytes.

5.2 Impossible Differential Attack on SAFER++/256

Put the 3.75-round impossible differential from round 0.5 to round 4.25 and we will recover the key for 5.5-round SAFER++/256. The detailed attack procedure is available in the full version of this paper [17]. The data complexity of the attack is 2^{124} chosen plaintexts. The memory complexity is about 2^{97} bytes. The time complexity is 2^{246} encryptions.

6 Conclusion

This paper introduces impossible differential attacks on SAFER+ and SAFER++ block ciphers. We first derive 2.75-round and 3.75-round impossible differentials for SAFER+ and SAFER++, which improves the previous 2.75-round impossible differentials for SAFER++. With the impossible differentials, attacks on 4-round SAFER+/128(256), 5-round SAFER++/128 and 5.5-round SAFER++/256 can be achieved. Our method can also be applied to other ciphers that have similar structures to SAFER+.

References

1. Behnam, B., Taraneh, E., Mohammad, R.A.: Impossible Differential Cryptanalysis of SAFER++. In: Proceedings of the 2008 International Conference on Security Management, SAM 2008, pp. 10–14. CSREA Press (2008)
2. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 12–23. Springer, Heidelberg (1999)
3. Biryukov, A., De Cannière, C., Dellkrantz, G.: Cryptanalysis of SAFER++. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 195–211. Springer, Heidelberg (2003)

4. BLUETOOTH SPECIFICATION Version 1.0B (November 29, 1999), `http://www.bluetooth.com/link/spec/bluetooth_b.pdf`
5. Knudsen, L.: DEAL-A 128-bit Block Cipher. NIST AES proposal. Technial report 151 (February 21, 1998) (retrieved February 27, 2007)
6. Knudsen, L.: A Detailed Analysis of SAFER K. Journal of Cryptplogy 13(4), 417–436 (2000)
7. Massey, J.L.: SAFER K-64: One Year Later. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 212–241. Springer, Heidelberg (1995)
8. Massey, J.L., Khachatrian, G.H., Kuregian, M.K.: 1st AES Conference on Nomination of SAFER+ as Candidate Algorithm for The Advanced Encryption Standard, California, USA (June 1998), `http://csrc.nist.gov/encryption/aes/`
9. Massey, J.L., Khachatrian, G.H., Kuregian, M.K.: 1st NESSIE Workshop on The SAFER++ Block Encryption Algorithm, Heverlee, Belgium (November 2000), `http://cryptonessie.org`
10. Nakahara Jr., J., Preneel, B., Vandewalle, J.: Linear Cryptanalysis of Reduced-Round Versions of the SAFER Block Cipher Family. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 244–261. Springer, Heidelberg (2001)
11. Nakahara, J.: Cryptanalysis and Design of Block Ciphers. PhD thesis. Katholidke University, Leuven (2003)
12. Nakahara, J., Preneel, B.: Impossible Differential Attacks on Reduced-Round SAFER Ciphers. NESSIE Public Report, NES/DOC/KUL/WP5/30/1 (2003)
13. NESSIE Project–New European Schemes for Signatures, Integrity and Encryption, `http://cryptonessie.org`
14. Piret, G., Quisauater, J.: Integral Cryptanalysis on Reduced-Round SAFER++–A Way to Extend The Attack? (2003), `http://eprint.iacr.org/2003/033.pdf`
15. Yemo, Y., Park, I.: Optimization of Integral Cryptanalysis on Reduced-Round SAFER++. Joho Shori Gakkai Shinpojiumu Ronbunshu 2003(15) (2003) (published in Japan)
16. Zheng, S., Wang, C.L., Yang, Y.X.: A New Impossible Differential Attack on SAFER Ciphers. Computers and Electrical Engineering 36, 180–189 (2010)
17. Zhao, J., Wang, M., Chen, J., Zheng, Y.: New Impossible Differential Attack on SAFER$_+$ and SAFER$_{++}$. IACR ePrint Archive report (2012)

A Appendix

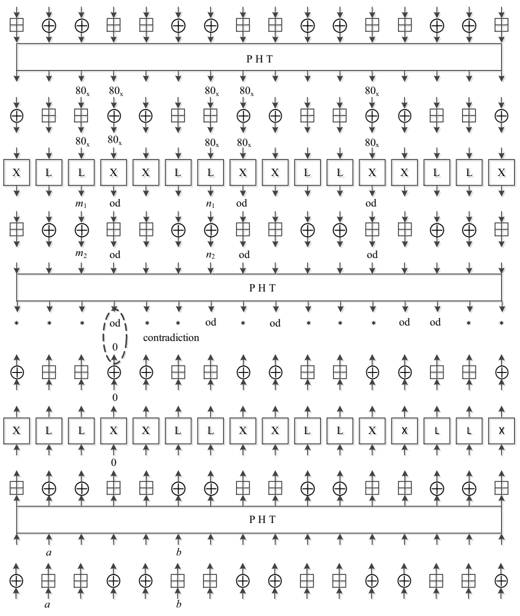


Fig. 1. 2.75-Round Impossible Differential of SAFER+

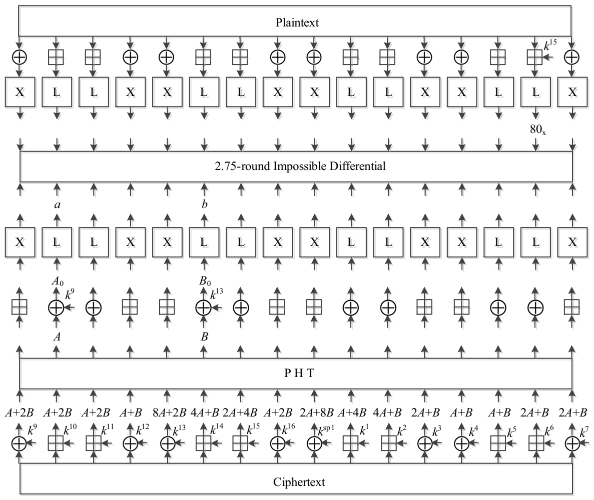


Fig. 2. Impossible Differential Attack on SAFER+128

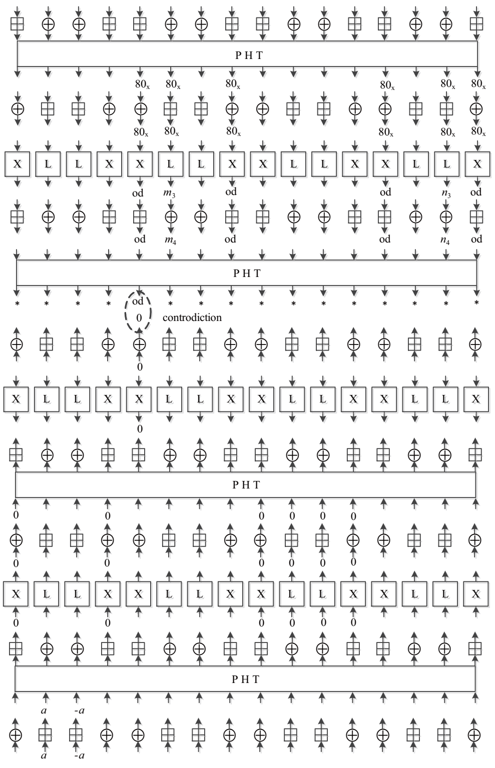


Fig. 3. 3.75-Round Impossible Differential of SAFER++

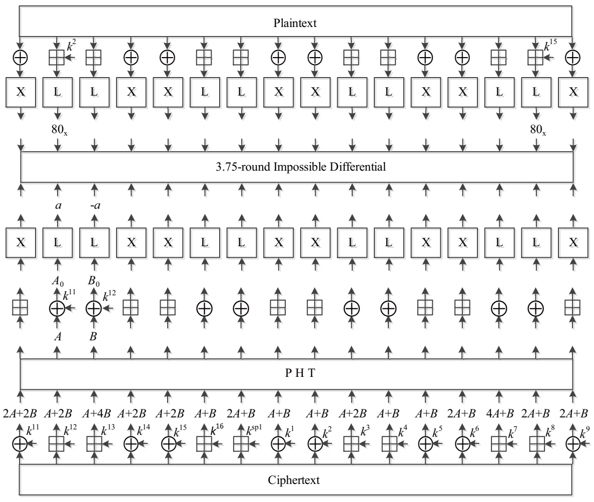


Fig. 4. Impossible Differential Attack on SAFER++128

An Information-Theoretically Secure Threshold Distributed Oblivious Transfer Protocol

Christian L.F. Corniaux and Hossein Ghodosi

James Cook University, Townsville QLD 4811, Australia
chris.corniaux@my.jcu.edu.au, hossein.ghodosi@jcu.edu.au

Abstract. The unconditionally secure Distributed Oblivious Transfer (DOT) protocol presented by Blundo, D'Arco, De Santis, and Stinson at SAC 2002 allows a receiver to contact k servers and obtain one out of n secrets held by a sender.

Once the protocol has been executed, the sender does not know which secret was selected by the receiver and the receiver knows nothing of the secrets she did not choose. In addition, the receiver's privacy is guaranteed against a coalition of $k - 1$ servers and similarly, the sender's security is guaranteed against a coalition of $k - 1$ servers. However, after the receiver has obtained a secret, she is able to learn all secrets by corrupting one server only. In addition, an external mechanism is required to prevent the receiver from contacting more than k servers.

The one-round DOT protocol we propose is information-theoretically secure, allows the receiver to contact k servers or more, and guarantees the sender's security, even if the receiver corrupts $k - 1$ servers after having obtained a secret.

Keywords: Cryptographic Protocol, Distributed Oblivious Transfer, Commodity Based Model, Information-Theoretic Security.

1 Introduction

Oblivious Transfer (OT) protocols allow two parties to exchange, in total privacy, one or more secret messages. The first OT protocol, introduced by Rabin [13], enables a sender to transmit a message to a receiver in such a way that the receiver gets the message with probability $\frac{1}{2}$ while the sender does not know whether the message was received. Even, Goldreich and Lempel [8] introduced a variant of the original OT for a contract signature application. This OT, identified as OT-$\binom{2}{1}$, is an exchange protocol between a receiver and a sender who has two secret messages; the receiver chooses one of the two messages and the sender transmits the chosen message to the receiver. At the end of the protocol, the sender does not know which message was selected and the receiver knows nothing of the other message.

A major drawback with OT-$\binom{2}{1}$ and with the more general OT-$\binom{n}{1}$ proposed by Brassard, Crépeau and Roberts [6] is the restriction in the availability of the secret messages, because if the unique sender is unavailable, the receiver cannot execute the protocol. To increase the availability of messages, the sender may distribute them to m servers, like in the first unconditionally secure Distributed Oblivious Transfer (DOT) protocol introduced by Gertner and Malkin [10] in 1997. However, Gertner and Malkin's protocol does not guarantee the messages' confidentiality against curious or corrupted servers.

In 2000, Naor and Pinkas [11] proposed an unconditionally secure DOT protocol which takes non-fully trusted servers into account: servers are only provided with parts – called *shares* – of the original messages. This DOT protocol was generalized to n secrets by Blundo, D'Arco, De Santis and Stinson [4,5]. Both protocols are composed of two phases: (i) the *set-up phase* and (ii) the *transfer phase.* During the set-up phase, the sender generates and sends shares of his secrets to all the servers. In the transfer phase, the receiver chooses the index of a secret, selects k servers $(1 < k \leq m)$ and sends them requests. From the k responses the receiver is able to determine the chosen secret.

Blundo et al. also defined a security model composed of four fundamental conditions that every DOT protocol should satisfy:

C_1. Correctness – The receiver is able to determine the chosen secret once she has received information from the k contacted servers.
C_2. Receiver's privacy – A coalition of up to $k-1$ servers cannot obtain any information on the choice of the receiver.
C_3. Sender's privacy with respect to $k-1$ servers and the receiver – A coalition of up to $k-1$ servers with the receiver does not obtain any information about the secrets.
C_4. Sender's privacy with respect to a "greedy" receiver – Given the transcript of the interaction with k servers, a coalition of up to $k-1$ dishonest servers and the receiver does not obtain any information about secrets which were not chosen by the receiver.

As it has been pointed out by Blundo et al. in [4,5], the protocol introduced by Naor and Pinkas only satisfies conditions C_1 and C_2. Their own protocol satisfies conditions C_1, C_2 and C_3 only. Actually, they have proven that condition C_4 cannot be guaranteed with a one-round DOT protocol – a round being defined as a set of consistent requests/responses exchanged between the receiver and k servers.

Besides, Nikov, Nikova, Preneel and Vanderwalle have demonstrated [12] that more generally, if the receiver's privacy is guaranteed against a coalition of $k_{\mathcal{R}}$ servers and the sender's security against a coalition of $k_{\mathcal{S}}$ servers, including when a secret had already been obtained, then the parameters $k_{\mathcal{S}}$ and $k_{\mathcal{R}}$ must satisfy the inequality $(k_{\mathcal{S}}+1)+(k_{\mathcal{R}}+1) < k$.

Recently, Beimel, Chee, Wang and Zhang [2] introduced communication-efficient DOT protocols. These protocols, based on information-theoretic private information retrieval (PIR) protocols, require that the number of servers contacted by the receiver is pre-determined.

In this paper, we introduce an information-theoretically secure threshold DOT protocol. That is, the number of servers the receiver needs to contact to obtain a secret is not limited to k. Moreover, unlike other unconditionally secure DOT protocols, our protocol satisfies security conditions C_1, C_2 for a coalition of any size, C_3 and C_4. Actually, to circumvent the impossibility result established by Blundo et al., we use the commodity-based model introduced by Beaver [1]. More precisely, our protocol is based on Rivest's trusted initializer OT protocol [15]. In this protocol, an additional party – the trusted initializer – is involved in the set-up phase; he generates and distributes random values, but receives nothing from other parties (in particular, he obtains neither the sender's secrets, nor the receiver's choice). In addition, our protocol has an efficiency similar to the efficiency of the full protocol presented by Blundo et al. [4,5].

This paper is organized as follows: in Sect. 2, we give an overview of the OT protocol proposed by Rivest [15]. In Sect. 3, we introduce some definitions and notations, as well as our security model. The protocol is described in Sect. 4 and the security is analysed in Sect. 5. The last section is devoted to the performance of the protocol.

2 Background

The OT-$\binom{2}{1}$ protocol presented by Rivest [15] is based on the protocol introduced by Bennett, Brassard, Crépeau and Skubiszewska [3], adapted to the trusted initializer model.

We assume that a sender $\mathcal{S}$ holds two secrets $w_0, w_1 \in \{0,1\}^\ell$ ($\ell \in \mathbb{N}^* = \{1,2,\dots\}$) and that a receiver $\mathcal{R}$ wishes to learn the secret w_e ($e = 0$ or $e = 1$).

In the set-up phase, the trusted initializer $\mathcal{T}$ gives to $\mathcal{S}$ two random ℓ-bit strings r_0 and r_1. Then, $\mathcal{T}$ selects a random bit d and sends the pair (d, r_d) to $\mathcal{R}$.

In the transfer phase, $\mathcal{R}$ selects the index e of one secret and transmits $c = e \oplus d$ to $\mathcal{S}$. $\mathcal{S}$ replies with two values $f_0 = w_0 \oplus r_c$ and $f_1 = w_1 \oplus r_{1-c}$. To obtain w_e, $\mathcal{R}$ calculates $f_e \oplus r_d$.

Clearly, the receiver obtains one secret only and the sender cannot determine which secret was chosen by the receiver.

3 Preliminaries

3.1 Notations and Definitions

The setting of the DOT protocol described in this paper encompasses a sender $\mathcal{S}$ who owns n secrets $w_1, \dots, w_n$ ($n > 1$) in a finite field $\mathbb{K} = \mathbb{F}_p$ (p prime), a receiver $\mathcal{R}$ who wishes to learn a secret w_e ($1 \le e \le n$), a trusted initializer $\mathcal{T}$ who generates random elements of $\mathbb{K}$ and m servers $S_1, \dots, S_m$. We assume that $p > \max(n, w_1, \dots, w_n, m)$ and that all operations are executed in $\mathbb{K}$.

Our protocol is composed of three phases: a set-up phase, a commodity acquisition phase and a transfer phase. In the set-up phase, for each secret the sender generates shares thanks to Shamir's (k, m)-threshold secret sharing schemes [16] ($1 < k \le m$). Then, the sender distributes the shares to the m servers and does not intervene in the rest of the protocol. In the commodity acquisition phase, the receiver contacts the trusted initializer who generates and distributes consistent masks to the m servers and to the receiver. The trusted initializer's presence is only required in this phase. In the transfer phase, the receiver has to contact t servers ($k \leq t \leq m$) to collect enough shares to construct w_e.

The protocol requires the availability of private communication channels between the trusted initializer and the servers, between the trusted initializer and the receiver and between the sender and the servers. The receiver sends requests to the servers thanks to a private broadcast channel and collects responses thanks to private channels between servers and herself. We assume that private channels are secure, i.e., any party is unable to eavesdrop on them and that all channels guarantee that communications cannot be tampered with.

The set $\{1, \ldots, n\}$ of natural numbers is denoted $[n]$. The additive group of univariate polynomials of degree at most k with coefficients in $\mathbb{K}$ is denoted $\mathbb{K}_k[X]$. In addition, by an abuse of language, a polynomial and its corresponding polynomial function will not be differentiated.

Since security conditions are linked to the quantity of information received by parties, it seems appropriate to use Shannon's entropy function [17], and more generally information theory, to demonstrate the security of our protocol. The following definitions and properties will be used in the paper (for more details on information theory, see for example [7]).

An element v of a finite field V is described by a discrete random variable $\boldsymbol{V}$ over a finite set $\mathcal{V}$. The probability distribution $\Pr(\boldsymbol{V})$ is associated with $\boldsymbol{V}$.

Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be two random variables.

- The *entropy* of $\boldsymbol{X}$ is $H(\boldsymbol{X}) = -\sum_{x\in\mathcal{X}} \Pr(\boldsymbol{X}=x)\log_2 \Pr(\boldsymbol{X}=x)$.
- The *joint entropy* $H(\boldsymbol{X},\boldsymbol{Y})$ of $\boldsymbol{X}$ and $\boldsymbol{Y}$ (joint distribution $\Pr(\boldsymbol{X},\boldsymbol{Y})$) is

$$H(\boldsymbol{X},\boldsymbol{Y}) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} \Pr(\boldsymbol{X}=x,\boldsymbol{Y}=y)\log_2 \Pr(\boldsymbol{X}=x,\boldsymbol{Y}=y).$$

- The *conditional entropy* $H(\boldsymbol{X} \mid \boldsymbol{Y})$ of $\boldsymbol{X}$ given $\boldsymbol{Y}$ is defined as

$$H(\boldsymbol{X} \mid \boldsymbol{Y}) = \sum_{y\in\mathcal{Y}} \Pr(\boldsymbol{Y}=y) H(\boldsymbol{X} \mid \boldsymbol{Y}=y),$$

  where the entropy $H(\boldsymbol{X} \mid \boldsymbol{Y}=y)$ is

$$H(\boldsymbol{X} \mid \boldsymbol{Y}=y) = -\sum_{x\in\mathcal{X}} \Pr(\boldsymbol{X}=x \mid \boldsymbol{Y}=y)\log_2 \Pr(\boldsymbol{X}=x \mid \boldsymbol{Y}=y).$$

Note that if $\Pr(\boldsymbol{X} = x) = 0$, then we adopt the convention that $\Pr(\boldsymbol{X}=x)\log_2 \Pr(\boldsymbol{X}=x) = 0$.

Let $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{Z}$ and $\boldsymbol{X}_i$ ($i \in [n]$) be random variables. We use the following properties in the security demonstrations:

$$H(\boldsymbol{X}) \geq H(\boldsymbol{X} \mid \boldsymbol{Y}) \tag{1}$$
$$H(\boldsymbol{X}, \boldsymbol{Y}) = H(\boldsymbol{X}) + H(\boldsymbol{Y} \mid \boldsymbol{X}) = H(\boldsymbol{Y}) + H(\boldsymbol{X} \mid \boldsymbol{Y}) \tag{2}$$
$$0 \leq H(\boldsymbol{X}) \leq \log_2 |\mathcal{X}| \tag{3}$$
$$\text{If } H(\boldsymbol{Y} \mid \boldsymbol{Z}) = 0 \text{ then } H(\boldsymbol{X} \mid \boldsymbol{Y}) \geq H(\boldsymbol{X} \mid \boldsymbol{Z}) \tag{4}$$
$$\text{If } H(\boldsymbol{Y} \mid \boldsymbol{Z}) = 0 \text{ then } H(\boldsymbol{X} \mid \boldsymbol{Y}, \boldsymbol{Z}) = H(\boldsymbol{X} \mid \boldsymbol{Z}) \tag{5}$$
$$H(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{Y}) = H(\boldsymbol{Z} \mid \boldsymbol{X}) \text{ iff } H(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{Z}) = H(\boldsymbol{Y} \mid \boldsymbol{X}) \tag{6}$$
$$\text{If } H(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{Y}) = H(\boldsymbol{Z}) \text{ then } H(\boldsymbol{X} \mid \boldsymbol{Y}, \boldsymbol{Z}) = H(\boldsymbol{X} \mid \boldsymbol{Y}) \tag{7}$$

and

$$\text{If } H(\boldsymbol{X} \mid \boldsymbol{Y}, \boldsymbol{Z}) = H(\boldsymbol{X}) \text{ then } H(\boldsymbol{X} \mid \boldsymbol{Y}) = H(\boldsymbol{X} \mid \boldsymbol{Z}) = H(\boldsymbol{X}) \tag{8}$$

3.2 Security Model

The point of the paper is not to propose a verifiable DOT protocol. This is why we assume that all parties wish to complete the protocol to allow the receiver to obtain the chosen secret. In particular, the trusted initializer and the sender are honest. However, even if they are not malicious, servers may actively collaborate to determine the receiver's choice (C_2) or the sender's secrets (C_3). The receiver may also actively cheat, either while cooperating with a coalition of active cheating servers (C_3), or by corrupting servers after having obtained a secret (C_4). In this latter case, the receiver has access to all data held by the corrupted servers.

4 Protocol Description

The key idea underlying our t-out-of-n DOT protocol is to extend Rivest's OT protocol in two directions:

1. Generalization to n secrets
2. Introduction of a distributed model with m servers

Furthermore, to prevent the servers from learning the sender's secrets, they receive shares of the secrets held by the sender. These shares are generated thanks to Shamir's secret sharing schemes [16].

In addition, to guarantee that the contacted servers do not receive requests related to different secrets, they all receive the same request and this request is broadcast.

The full protocol is described in Fig. 1.

5 Security of the Protocol

5.1 Formal Model

To prove the security of our protocol we use a formal model similar to the model introduced by Blundo et al. [4,5]. In this model, we assume that the

Input	The sender $\mathcal{S}$ contributes with n secrets $w_1, \dots, w_n \in \mathbb{K}$ The trusted initializer $\mathcal{T}$ generates m sets of n random masks and randomly chooses one of the n sets The receiver $\mathcal{R}$ chooses an index $e \in [n]$, and contributes with a cyclic permutation $\pi \in \mathfrak{S}_n$
Output	$\mathcal{R}$ receives w_e, while $\mathcal{S}$ and $\mathcal{T}$ receive nothing.

Set-up Phase

1 - *Preparation of shares.* For each secret w_i $(i \in [n])$, the sender $\mathcal{S}$ generates, thanks to Shamir's $(k,\ m)$-threshold secret sharing scheme, a sharing polynomial F_i of degree at most $k-1$, such that $F_i(0) = w_i$.

2 - *Distribution of shares.* To each server S_j $(j \in [m])$, $\mathcal{S}$ transmits the n shares $F_1(j), \dots, F_n(j)$.

Commodity Acquisition Phase

1 - *Preparation of masks.* The trusted initializer $\mathcal{T}$ generates mn random masks $r_{j,i} \in \mathbb{K}$ $(j \in [m],\ i \in [n])$ and one random index $s \in [n]$.

2 - *Distribution of masks.* $\mathcal{T}$ distributes the n masks $r_{j,1}, \dots, r_{j,n}$ to the server S_j $(j \in [m])$ and the index s as well as the m masks $r_{1,s}, \dots, r_{m,s}$ to the receiver $\mathcal{R}$.

Transfer Phase

1 - *Selection of the secret index and generation of the corresponding request.* The receiver $\mathcal{R}$ chooses a secret index e and generates the cyclic permutation $\pi \in \mathfrak{S}_n$ which satisfies $\pi(e) = s$.

2 - *Selection of servers and broadcast of a query.* $\mathcal{R}$ selects a subset $\mathcal{I} \subset [m]$ of $t \geq k$ indices and broadcasts a query containing the first cyclic permutation item, $\pi(1)$, as well as the list $\mathcal{I}$.

3 - *Responses of the servers.* Each server S_ℓ such that $\ell \in \mathcal{I}$ returns $\mu_{\ell,i} = F_i(\ell) + r_{\ell,\pi(i)}$ $(i \in [n])$ to $\mathcal{R}$.

4 - *Construction of the requested secret.* For each of the t responses $\mu_{\ell,e}$, $\mathcal{R}$ calculates the share $\mu_{\ell,e} - r_{\ell,s} = F_e(\ell)$, interpolates F_e and obtains $w_e = F_e(0)$.

Fig. 1. Protocol Overview

parties execute publicly known programs whose data are private. These data are described by the following discrete random variables shown on Fig. 2.

By extension, if $\boldsymbol{X}_j$ is a random variable which describes a datum x_j held by a server S_j $(j \in [m])$ and $G = \{\, j_1, \dots, j_t \,\}$ $(t \in [m])$, we denote $\boldsymbol{X}_G = (\, \boldsymbol{X}_{j_1}, \dots, \boldsymbol{X}_{j_t} \,)$ the random variable describing the sequence $(\, x_{j_1}, \dots, x_{j_t} \,)$. By simplification, $\boldsymbol{X}_{[m]}$ is denoted $\boldsymbol{X}$.

- Each secret $w_i \in \mathbb{K}$ $(i \in [n])$ is described by a variable $\boldsymbol{W}^i$ and the sequence of secrets $w_1, \dots, w_n$ by the variable $\boldsymbol{W} = (\, \boldsymbol{W}^1, \dots, \boldsymbol{W}^n \,)$. Moreover, if $e \in [n]$, we denote $\boldsymbol{W}^{\bar{e}}$ the sequence $(\, \boldsymbol{W}^1, \dots, \boldsymbol{W}^{e-1}, \boldsymbol{W}^{e+1}, \dots, \boldsymbol{W}^n \,)$.
- The secret index $e \in [n]$ chosen by $\mathcal{R}$ is described by the random variable $\boldsymbol{E}$.

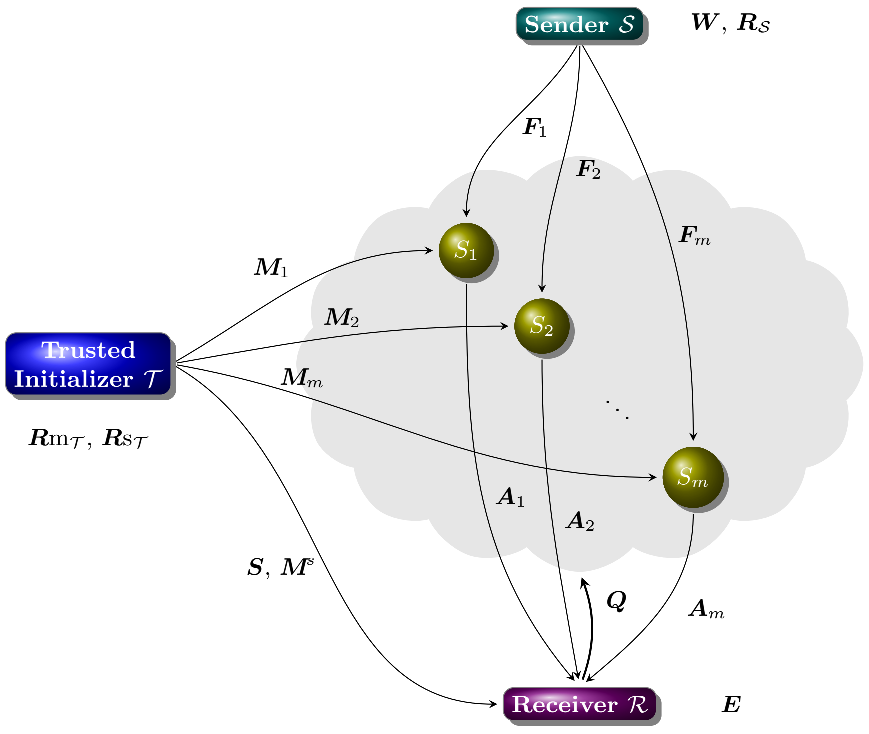


Fig. 2. Random Variables

- The random variable $\boldsymbol{M}_j^i$ ($j \in [m]$, $i \in [n]$) corresponds to the mask $r_{j,i}$ and the random variable $\boldsymbol{M}_j$ ($j \in [m]$) to the n ordered masks $(\, r_{j,1}, \ldots, r_{j,n} \,)$ distributed by $\mathcal{T}$ to the server S_j. Similarly, the random variable $\boldsymbol{F}_j^i$ ($i \in [n]$, $j \in [m]$) corresponds to the share $F_i(j)$ and the random variable $\boldsymbol{F}_j$ ($j \in [m]$) to the n shares $(\, F_1(j), \ldots, F_n(j) \,)$ distributed by $\mathcal{S}$ to the server S_j. By simplification $\boldsymbol{M}_j^{[n]} = (\, \boldsymbol{M}_j^1, \ldots, \boldsymbol{M}_j^n \,)$ is denoted $\boldsymbol{M}_j$ and $\boldsymbol{F}_j^{[n]} = (\, \boldsymbol{F}_j^1, \ldots, \boldsymbol{F}_j^n \,)$ is denoted $\boldsymbol{F}_j$.
- In addition, the random index $s \in [n]$ chosen by $\mathcal{T}$ is described by the random variable $\boldsymbol{S}$. The notation $\boldsymbol{M}_j^s$ corresponds to the random variable describing $r_{j,s}$ and $\boldsymbol{M}^s$ is a shorthand for $(\, \boldsymbol{M}_1^s, \ldots, \boldsymbol{M}_m^s \,)$.
- The cyclic permutation $\pi \in \mathfrak{S}_n$ is described by the random variable $\boldsymbol{Q}$:

$$H(\boldsymbol{Q} \mid \boldsymbol{E}, \boldsymbol{S}) = 0. \tag{9}$$

- The transcript $T_j = (Q, A_j)$ is composed of a query $Q = \pi$ described by the random variable $\boldsymbol{Q}$ and of an answer $A_j = (\, F_j(1) + r_{j,\pi(1)}, \ldots, F_j(n) + r_{j,\pi(n)} \,)$ described by the random variable $\boldsymbol{A}_j$. The random variable describing the answer $F_j(i) + r_{j,\pi(i)}$ ($j \in [m]$, $i \in [n]$) is denoted $\boldsymbol{A}_j^i$.
- A few uniform random variables are held by the parties involved in the protocol to allow them to produce private data:

- The trusted initialiser $\mathcal{T}$ holds two uniform random inputs, $\boldsymbol{R}\mathrm{m}_{\mathcal{T}}$, to generate the random masks $r_{j,i}$ $(i \in [n], j \in [m])$,

$$H(\boldsymbol{M}_j^i \mid \boldsymbol{R}\mathrm{m}_{\mathcal{T}}) = 0, \tag{10}$$

  and $\boldsymbol{R}\mathrm{s}_{\mathcal{T}}$, to determine the secret index s,

$$H(\boldsymbol{S} \mid \boldsymbol{R}\mathrm{s}_{\mathcal{T}}) = 0. \tag{11}$$

  Note that since $H(\boldsymbol{M}^s \mid \boldsymbol{M}, \boldsymbol{S}) = 0$ then

$$H(\boldsymbol{M}^s \mid \boldsymbol{R}\mathrm{m}_{\mathcal{T}}, \boldsymbol{R}\mathrm{s}_{\mathcal{T}}) = 0. \tag{12}$$

- The sender $\mathcal{S}$ holds a uniform random input $\boldsymbol{R}_{\mathcal{S}}$ to generate the shares $F_j(i)$ $(i \in [n], j \in [m])$:

$$H(\boldsymbol{F}_j^i \mid \boldsymbol{W}^i, \boldsymbol{R}_{\mathcal{S}}) = 0. \tag{13}$$

To show properties C_1, C_2, C_3 and C_4 is equivalent to show properties listed in Table 1.

Table 1. Security Conditions from an Information Theory Viewpoint

Security Condition	Number of Servers	Property
C_1	$k \leq \lvert G \rvert \leq m$	$H(\boldsymbol{W}^e \mid \boldsymbol{E} = e, \boldsymbol{S}, \boldsymbol{M}^s, \boldsymbol{Q}, \boldsymbol{A}_G) = 0$
C_2	$\lvert G \rvert \leq m$	$H(\boldsymbol{E} \mid \boldsymbol{F}_G, \boldsymbol{M}_G, \boldsymbol{Q}) = H(\boldsymbol{E})$
C_3	$\lvert G \rvert \leq k-1$	$H(\boldsymbol{W} \mid \boldsymbol{F}_G, \boldsymbol{M}_G, \boldsymbol{E}, \boldsymbol{S}, \boldsymbol{M}^s) = H(\boldsymbol{W})$
C_4	$k \leq \lvert G \rvert \leq m$ $\lvert G' \rvert \leq k-1$	$H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}_{G'}, \boldsymbol{M}_{G'}, \boldsymbol{E} = e, \boldsymbol{S} = s, \boldsymbol{Q} = \pi, \boldsymbol{A}_G, \boldsymbol{M}^s =$ $(r_{1,s}, \ldots, r_{m,s})) = H(\boldsymbol{W}^{\bar{e}})$

5.2 Correctness

Theorem 1. *The protocol is correct (condition C_1 is satisfied), i.e. if all parties follow the protocol, the receiver obtains the chosen secret w_e by contacting t servers S_j where $j \in G = \mathcal{I} = \{j_1, \ldots, j_t\}$ $(k \leq t \leq m)$.*

Proof.

To demonstrate that $H(\boldsymbol{W}^e \mid \boldsymbol{E} = e, \boldsymbol{S}, \boldsymbol{M}^s, \boldsymbol{Q}, \boldsymbol{A}_G) = 0$ is equivalent to demonstrate that once the protocol has been executed, $\Pr(\boldsymbol{W}^e = w_e \mid \boldsymbol{E} = e, \boldsymbol{S}, \boldsymbol{M}^s, \boldsymbol{Q}, \boldsymbol{A}_G) = 1$.

Once $\mathcal{R}$ has chosen e, the cyclic permutation $\pi \in \mathfrak{S}_n$ such that $\pi(e) = s$ is determined. The response sent by the server S_ℓ $(\ell \in \mathcal{I})$ then contains the value $\mu_{\ell,e} = F_e(\ell) + r_{\ell,\pi(e)} = F_e(\ell) + r_{\ell,s}$. Since $\mathcal{R}$ knows $r_{\ell,s}$, she is able to calculate the t shares $F_e(\ell)$, to interpolate F_e (degree at most $k-1 < t$) and to determine $w_e = F_e(0)$. It follows that $\Pr(\boldsymbol{W}^e = w_e \mid \boldsymbol{E} = e, \boldsymbol{S}, \boldsymbol{M}^s, \boldsymbol{Q}, \boldsymbol{A}_G) = 1$. □

5.3 Receiver's Privacy against a Coalition of Servers

Theorem 2. *The protocol guarantees the receiver's privacy against a coalition of h servers S_j where $j \in G = \{ j_1, \ldots, j_h \}$ $(0 \leq h \leq m)$, i.e. condition C_2 is satisfied.*

Proof.
To show that $H(\boldsymbol{E} \mid \boldsymbol{F}_G, \boldsymbol{M}_G, \boldsymbol{Q}) = H(\boldsymbol{E})$, first we demonstrate that

$$H(\boldsymbol{E} \mid \boldsymbol{F}_G, \boldsymbol{M}_G, \boldsymbol{Q}) = H(\boldsymbol{E} \mid \boldsymbol{Q})$$

and second that

$$H(\boldsymbol{E} \mid \boldsymbol{Q}) = H(\boldsymbol{E}).$$

For the first part of the demonstration, we adapt a technique applied by Beimel, Chee, Wang and Zhang [2] in a similar context.

1. First, we show that the conditional entropy of $\boldsymbol{E}$ given $\boldsymbol{F}_G$, $\boldsymbol{M}_G$ and $\boldsymbol{Q}$ satisfies

$$H(\boldsymbol{E} \mid \boldsymbol{F}_G, \boldsymbol{M}_G, \boldsymbol{Q}) = H(\boldsymbol{E} \mid \boldsymbol{Q}).$$

For this purpose, thanks to property (6), we show that

$$H(\boldsymbol{F}_G, \boldsymbol{M}_G \mid \boldsymbol{E}, \boldsymbol{Q}) = H(\boldsymbol{F}_G, \boldsymbol{M}_G \mid \boldsymbol{Q}).$$

The choice of the receiver is independent from the data held by the trusted initializer, by the sender, by the servers and by herself at the end of the commodity acquisition phase, so

$$H(\boldsymbol{E} \mid \boldsymbol{R}\mathrm{m}_{\mathcal{T}}, \boldsymbol{R}\mathrm{s}_{\mathcal{T}}, \boldsymbol{W}, \boldsymbol{R}_{\mathcal{S}}, \boldsymbol{F}, \boldsymbol{M}, \boldsymbol{S}, \boldsymbol{M}^{s}) = H(\boldsymbol{E}). \tag{14}$$

If we apply property (8), we obtain the particular case

$$H(\boldsymbol{E} \mid \boldsymbol{F}_G, \boldsymbol{M}_G, \boldsymbol{R}\mathrm{s}_{\mathcal{T}}) = H(\boldsymbol{E}). \tag{15}$$

Similarly, the uniform random variable $\boldsymbol{R}\mathrm{s}_{\mathcal{T}}$ held by the trusted initializer is independent from the uniform random variable $\boldsymbol{R}\mathrm{m}_{\mathcal{T}}$, from the sender's data and from the data held by the servers at the end of the commodity acquisition phase. It follows

$$H(\boldsymbol{R}\mathrm{s}_{\mathcal{T}} \mid \boldsymbol{R}\mathrm{m}_{\mathcal{T}}, \boldsymbol{W}, \boldsymbol{R}_{\mathcal{S}}, \boldsymbol{F}, \boldsymbol{M}) = H(\boldsymbol{R}\mathrm{s}_{\mathcal{T}}). \tag{16}$$

Once more, if we apply property (8), we obtain the particular case

$$H(\boldsymbol{R}\mathrm{s}_{\mathcal{T}} \mid \boldsymbol{F}_G, \boldsymbol{M}_G) = H(\boldsymbol{R}\mathrm{s}_{\mathcal{T}}). \tag{17}$$

The joint entropy between $\boldsymbol{F}_G$ and $\boldsymbol{M}_G$ is

$$\begin{aligned}
H(\boldsymbol{F}_G, \boldsymbol{M}_G) &\geq H(\boldsymbol{F}_G, \boldsymbol{M}_G \mid \boldsymbol{Q}) && \text{(from (1))}\\
&\geq H(\boldsymbol{F}_G, \boldsymbol{M}_G \mid \boldsymbol{Q}, \boldsymbol{E}) && \text{(from (1))}\\
&\geq H(\boldsymbol{F}_G, \boldsymbol{M}_G \mid \boldsymbol{R}\mathrm{s}_{\mathcal{T}}, \boldsymbol{E}) && \text{(from (9), (11) and (4))}\\
&= H(\boldsymbol{F}_G, \boldsymbol{M}_G \mid \boldsymbol{R}\mathrm{s}_{\mathcal{T}}) && \text{(from (15) and (7))}\\
&= H(\boldsymbol{F}_G, \boldsymbol{M}_G). && \text{(from (17) and (7))}
\end{aligned}$$

Therefore, $H(\boldsymbol{F}_G, \boldsymbol{M}_G \mid \boldsymbol{Q}) = H(\boldsymbol{F}_G, \boldsymbol{M}_G \mid \boldsymbol{Q}, \boldsymbol{E})$ and from property (6), $H(\boldsymbol{E} \mid \boldsymbol{Q}, \boldsymbol{F}_G, \boldsymbol{M}_G) = H(\boldsymbol{E} \mid \boldsymbol{Q})$.

2. To prove that $H(\boldsymbol{E} \mid \boldsymbol{Q}) = H(\boldsymbol{E})$, thanks to property (6), it is sufficient to show that $H(\boldsymbol{Q} \mid \boldsymbol{E}) = H(\boldsymbol{Q})$.
First, we observe that given a secret index e and a cyclic permutation π, the random index s is uniquely determined: $s = \pi(e)$. Therefore, in terms of entropy, it follows that

$$H(\boldsymbol{S} \mid \boldsymbol{Q}, \boldsymbol{E}) = 0. \tag{18}$$

Second, the conditional joint entropy between $\boldsymbol{Q}$ and $\boldsymbol{S}$ given $\boldsymbol{E}$ is

$$\begin{aligned} H(\boldsymbol{Q}, \boldsymbol{S} \mid \boldsymbol{E}) &= H(\boldsymbol{Q} \mid \boldsymbol{E}) + H(\boldsymbol{S} \mid \boldsymbol{Q}, \boldsymbol{E}) && \text{(from (2))} \\ &= H(\boldsymbol{Q} \mid \boldsymbol{E}) && \text{(from (18))} \end{aligned}$$

and also

$$\begin{aligned} H(\boldsymbol{Q}, \boldsymbol{S} \mid \boldsymbol{E}) &= H(\boldsymbol{S} \mid \boldsymbol{E}) + H(\boldsymbol{Q} \mid \boldsymbol{E}, \boldsymbol{S}) && \text{(from (2))} \\ &= H(\boldsymbol{S} \mid \boldsymbol{E}) && \text{(from (9))} \end{aligned}$$

It follows that $H(\boldsymbol{Q} \mid \boldsymbol{E}) = H(\boldsymbol{S} \mid \boldsymbol{E})$.
If we apply property (8) to equality (14), we obtain the particular case $H(\boldsymbol{E} \mid \boldsymbol{S}) = H(\boldsymbol{E})$ which, combined with property (6) gives $H(\boldsymbol{S} \mid \boldsymbol{E}) = H(\boldsymbol{S})$. Therefore, $H(\boldsymbol{Q} \mid \boldsymbol{E}) = H(\boldsymbol{S})$. Moreover, because the random variable $\boldsymbol{S}$ is uniform, it holds $H(\boldsymbol{S}) = \log_2 n$ and because the number of cyclic permutations of $\mathfrak{S}_n$ is n, we can write:

$$\log_2 n \geq H(\boldsymbol{Q}) \geq H(\boldsymbol{Q} \mid \boldsymbol{E}) = H(\boldsymbol{S}) = \log_2 n.$$

It follows that $H(\boldsymbol{Q} \mid \boldsymbol{E}) = H(\boldsymbol{Q})$ and from (6) that $H(\boldsymbol{E} \mid \boldsymbol{Q}) = H(\boldsymbol{E})$.

We have shown that $H(\boldsymbol{E} \mid \boldsymbol{Q}, \boldsymbol{F}_G, \boldsymbol{M}_G) = H(\boldsymbol{E} \mid \boldsymbol{Q})$ and $H(\boldsymbol{E} \mid \boldsymbol{Q}) = H(\boldsymbol{E})$. We conclude $H(\boldsymbol{E} \mid \boldsymbol{F}_G, \boldsymbol{M}_G, \boldsymbol{Q}) = H(\boldsymbol{E})$. □

5.4 Sender's Security against a Coalition of the Receiver and Servers

Theorem 3. *The protocol guarantees the sender's security against a coalition of the receiver and h servers S_j where $j \in G = \{j_1, \ldots, j_h\}$ $(0 \leq h \leq k-1)$, before the protocol is executed (condition C_3 is satisfied).*

Proof.
The demonstration is symmetrical to the previous demonstration. First, we demonstrate that $H(\boldsymbol{W} \mid \boldsymbol{F}_G, \boldsymbol{M}_G, \boldsymbol{E}, \boldsymbol{S}, \boldsymbol{M}^s) = H(\boldsymbol{W} \mid \boldsymbol{F}_G)$ and second that the secrets are independent from the shares received by any set of h servers ($h \leq k-1$) in the set-up phase, i.e., $H(\boldsymbol{W} \mid \boldsymbol{F}_G) = H(\boldsymbol{W})$. These two demonstrations will allow us to show that the secrets are independent from the data held by a coalition between the receiver and h servers.

1. The uniform random variable $\boldsymbol{R}\mathrm{m}_{\mathcal{T}}$ held by the trusted initializer is independent from the data held by the sender, by the servers and by herself at the end of the commodity acquisition phase, except masks. It follows

$$H(\boldsymbol{R}\mathrm{m}_{\mathcal{T}} \mid \boldsymbol{R}\mathrm{s}_{\mathcal{T}}, \boldsymbol{W}, \boldsymbol{R}_{\mathcal{S}}, \boldsymbol{F}) = H(\boldsymbol{R}\mathrm{m}_{\mathcal{T}}). \tag{19}$$

If we apply property (8), we obtain the particular case

$$H(\boldsymbol{R}\mathrm{m}_{\mathcal{T}} \mid \boldsymbol{R}\mathrm{s}_{\mathcal{T}}, \boldsymbol{W}, \boldsymbol{F}_G) = H(\boldsymbol{R}\mathrm{m}_{\mathcal{T}}). \tag{20}$$

Likewise, from (14) and (8) we can write

$$H(\boldsymbol{E} \mid \boldsymbol{W}, \boldsymbol{F}_G, \boldsymbol{R}\mathrm{s}_{\mathcal{T}}, \boldsymbol{R}\mathrm{m}_{\mathcal{T}}) = H(\boldsymbol{E}), \tag{21}$$

and from (16) and (8) we can write

$$H(\boldsymbol{R}\mathrm{s}_{\mathcal{T}} \mid \boldsymbol{W}, \boldsymbol{F}_G) = H(\boldsymbol{R}\mathrm{s}_{\mathcal{T}}). \tag{22}$$

The conditional entropy of $\boldsymbol{W}$ given $\boldsymbol{F}_G$ is

$$\begin{aligned} H(\boldsymbol{W} \mid \boldsymbol{F}_G) &\geq H(\boldsymbol{W} \mid \boldsymbol{F}_G, \boldsymbol{M}_G, \boldsymbol{E}, \boldsymbol{S}, \boldsymbol{M}^s) && \text{(from (1))} \\ &\geq H(\boldsymbol{W} \mid \boldsymbol{F}_G, \boldsymbol{E}, \boldsymbol{R}\mathrm{m}_{\mathcal{T}}, \boldsymbol{R}\mathrm{s}_{\mathcal{T}}) && \text{(from (10), (11), (12) and (4))} \\ &= H(\boldsymbol{W} \mid \boldsymbol{F}_G). && \text{(from (21), (22), (20) and (7))} \end{aligned}$$

We conclude that $H(\boldsymbol{W} \mid \boldsymbol{F}_G, \boldsymbol{M}_G, \boldsymbol{E}, \boldsymbol{S}, \boldsymbol{M}^s) = H(\boldsymbol{W} \mid \boldsymbol{F}_G)$.

2. It is well-known that Shamir's secret sharing scheme [16] is perfect, i.e, for $i \in [n]$, we have $H(\boldsymbol{W}^i \mid \boldsymbol{F}^i_G) = H(\boldsymbol{W}^i)$. The n secrets $w_1, \ldots, w_n$ are shared thanks to independent schemes; Therefore, the previous equality may easily be generalized to a vector of secrets $(\boldsymbol{W}^1, \ldots, \boldsymbol{W}^n)$. It follows that $H(\boldsymbol{W} \mid \boldsymbol{F}^1_G, \ldots, \boldsymbol{F}^n_G) = H(\boldsymbol{W})$.
Since $\boldsymbol{F}^1_G, \ldots, \boldsymbol{F}^n_G = \boldsymbol{F}^{[n]}_G = \boldsymbol{F}_G$, we obtain $H(\boldsymbol{W} \mid \boldsymbol{F}_G) = H(\boldsymbol{W})$.

We have demonstrated that $H(\boldsymbol{W} \mid \boldsymbol{F}_G, \boldsymbol{M}_G, \boldsymbol{E}, \boldsymbol{S}, \boldsymbol{M}^s) = H(\boldsymbol{W} \mid \boldsymbol{F}_G)$ and that $H(\boldsymbol{W} \mid \boldsymbol{F}_G) = H(\boldsymbol{W})$. We conclude

$$H(\boldsymbol{W} \mid \boldsymbol{F}_G, \boldsymbol{M}_G, \boldsymbol{E}, \boldsymbol{S}, \boldsymbol{M}^s) = H(\boldsymbol{W}).$$

□

5.5 Sender's Security against a "Greedy" Receiver

Theorem 4. *The protocol guarantees the sender's security against a coalition of the receiver and h servers $S_{j'}$ where $j' \in G' = \{ j'_1, \ldots, j'_h \}$ $(0 \leq h \leq k-1)$, after the protocol has been executed (condition C_4 is satisfied).*

Proof.
We assume that in the transfer phase of the protocol, t servers S_j are contacted by the receiver, where $j \in G = \{ j_1, \ldots, j_t \}$ $(1 < t \leq m)$. We introduce a random variable $\boldsymbol{K} = (\boldsymbol{E}, \boldsymbol{S}, \boldsymbol{M}^s)$ describing the data $K = (e, s, (r_{1,s}, \ldots, r_{m,s}))$. The theorem is demonstrated in four steps:

- First, we demonstrate that,

$$H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}_{G'}, \boldsymbol{M}_{G'}, \boldsymbol{A}_G, \boldsymbol{K} = K, \boldsymbol{Q} = \pi)$$
$$= H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{A}^{\bar{e}}_{G\setminus G'}, \boldsymbol{K} = K).$$

- Second, we show that

$$H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{A}^{\bar{e}}_{G\setminus G'}, \boldsymbol{K} = K)$$
$$= H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K} = K).$$

- Third, we show that

$$H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K} = K) = H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{K} = K).$$

- Lastly, we show that $H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{K} = K) = H(\boldsymbol{W}^{\bar{e}})$.

1. The random variable $\boldsymbol{A}_G$ may be decomposed under the form $\boldsymbol{A}_G = (\boldsymbol{A}_{G\setminus G'}, \boldsymbol{A}_{G'})$. Since $H(\boldsymbol{A}_{G'} \mid \boldsymbol{F}_{G'}, \boldsymbol{M}_{G'}, \boldsymbol{Q} = \pi) = 0$, we apply property (5) which yields

$$H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}_{G'}, \boldsymbol{M}_{G'}, \boldsymbol{A}_G, \boldsymbol{K} = K, \boldsymbol{Q} = \pi)$$
$$= H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}_{G'}, \boldsymbol{M}_{G'}, \boldsymbol{A}_{G\setminus G'}, \boldsymbol{K} = K, \boldsymbol{Q} = \pi).$$

The random variable $\boldsymbol{A}_{G\setminus G'}$ may be decomposed under the form $\boldsymbol{A}_{G\setminus G'} = (\boldsymbol{A}^{\bar{e}}_{G\setminus G'}, \boldsymbol{A}^{e}_{G\setminus G'})$. Since for $j \in [m]$, we have $F_e(j) = (F_e(j) + r_{j,\pi(e)}) + r_{j,\pi(e)} = (F_e(j) + r_{j,\pi(e)}) + r_{j,s}$, it holds that

$$H(\boldsymbol{F}^{e}_{G\setminus G'} \mid \boldsymbol{A}^{e}_{G\setminus G'}, \boldsymbol{M}^{s}_{G\setminus G'}) = 0 \text{ and } H(\boldsymbol{A}^{e}_{G\setminus G'} \mid \boldsymbol{F}^{e}_{G\setminus G'}, \boldsymbol{M}^{s}_{G\setminus G'}) = 0.$$

Applying (5) we obtain

$$H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}_{G'}, \boldsymbol{M}_{G'}, \boldsymbol{A}_{G\setminus G'}, \boldsymbol{K} = K, \boldsymbol{Q} = \pi)$$
$$= H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}_{G'}, \boldsymbol{F}^{e}_{G\setminus G'}, \boldsymbol{M}_{G'}, \boldsymbol{A}^{\bar{e}}_{G\setminus G'}, \boldsymbol{K} = K, \boldsymbol{Q} = \pi).$$

From properties (9) and (5), we obtain

$$H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}_{G'}, \boldsymbol{F}^{e}_{G\setminus G'}, \boldsymbol{M}_{G'}, \boldsymbol{A}^{\bar{e}}_{G\setminus G'}, \boldsymbol{K} = K, \boldsymbol{Q} = \pi)$$
$$= H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}_{G'}, \boldsymbol{F}^{e}_{G\setminus G'}, \boldsymbol{M}_{G'}, \boldsymbol{A}^{\bar{e}}_{G\setminus G'}, \boldsymbol{K} = K).$$

Because $\boldsymbol{M}_{G'} = (\boldsymbol{M}^{s}_{G'}, \boldsymbol{M}^{\bar{s}}_{G'})$ and $\boldsymbol{F}_{G'} = (\boldsymbol{F}^{e}_{G'}, \boldsymbol{F}^{\bar{e}}_{G'})$, we can apply (5). It follows

$$H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}_{G'}, \boldsymbol{F}^{e}_{G\setminus G'}, \boldsymbol{M}_{G'}, \boldsymbol{A}^{\bar{e}}_{G\setminus G'}, \boldsymbol{K} = K)$$
$$= H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{A}^{\bar{e}}_{G\setminus G'}, \boldsymbol{K} = K).$$

2. To prove that
$$H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{A}^{\bar{e}}_{G\setminus G'}, \boldsymbol{K}=K) = H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K}=K),$$
thanks to property (7) and Lemma 1, it is enough to show that
$$H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K}, \boldsymbol{W}^{\bar{e}}) = H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'}).$$
We have:
$$\begin{aligned}
&H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'}, \boldsymbol{M}^{\bar{s}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K}=K, \boldsymbol{W}^{\bar{e}})\\
&= H(\boldsymbol{M}^{\bar{s}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K}=K, \boldsymbol{W}^{\bar{e}})\\
&+ H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K}=K, \boldsymbol{W}^{\bar{e}}, \boldsymbol{M}^{\bar{s}}_{G\setminus G'})\\
&= H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K}=K, \boldsymbol{W}^{\bar{e}})\\
&+ H(\boldsymbol{M}^{\bar{s}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K}=K, \boldsymbol{W}^{\bar{e}}, \boldsymbol{A}^{\bar{e}}_{G\setminus G'}).
\end{aligned}$$
For $i \in [n], i \neq e$ and $j \in [m]$, we have $F_i(j) = (F_i(j) + r_{j,\pi(i)}) + r_{j,\pi(i)}$. Using property (9), it holds that
$$H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K}=K, \boldsymbol{W}^{\bar{e}}, \boldsymbol{M}^{\bar{s}}_{G\setminus G'}) = 0$$
and symetrically
$$H(\boldsymbol{M}^{\bar{s}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K}=K, \boldsymbol{W}^{\bar{e}}, \boldsymbol{A}^{\bar{e}}_{G\setminus G'}) = 0.$$
It follows that
$$\begin{aligned}
&H(\boldsymbol{M}^{\bar{s}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K}=K, \boldsymbol{W}^{\bar{e}})\\
&= H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K}=K, \boldsymbol{W}^{\bar{e}}).
\end{aligned}$$
Each mask $r_{j,i}$ ($i \in [n]$, $j \in [m]$) is randomly generated by the trusted initializer and is independent from the other variables held by the different parties at the beginning of the transfer phase. More precisely, if $G_1 \subset [m]$, $G_2 \subset [m]$, $H_1 \subset [n]$ and $H_2 \subset [n]$ are four subsets such that $G_1 \cap G_2 = \emptyset$ or $H_1 \cap H_2 = \emptyset$, we have
$$H(\boldsymbol{M}^{H_1}_{G_1} \mid \boldsymbol{E}, \boldsymbol{S}, \boldsymbol{W}, \boldsymbol{F}, \boldsymbol{M}^{H_2}_{G_2}) = H(\boldsymbol{M}^{H_1}_{G_1}). \tag{23}$$
If we apply property (8) and Lemma 1 (See Appendix A), we obtain the particular case
$$H(\boldsymbol{M}^{\bar{s}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K}=K, \boldsymbol{W}^{\bar{e}}) = H(\boldsymbol{M}^{\bar{s}}_{G\setminus G'}).$$
Therefore, $H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K}=K, \boldsymbol{W}^{\bar{e}}) = H(\boldsymbol{M}^{\bar{s}}_{G\setminus G'})$. Furthermore, the random variable $\boldsymbol{M}^{\bar{s}}_{G\setminus G'}$ is uniform, so
$$H(\boldsymbol{M}^{\bar{s}}_{G\setminus G'}) = \log_2 p^{(n-1)\times|G\setminus G'|}.$$

Thus,

$$\begin{aligned} H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'}) &\geq H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K} = K, \boldsymbol{W}^{\bar{e}}) \qquad \text{(from (1))} \\ &= \log_2 p^{(n-1)\times|G\setminus G'|}. \end{aligned}$$

By property (3), $H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'}) \leq \log_2 p^{(n-1)\times|G\setminus G'|}$.
It follows that $H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'}) = \log_2 p^{(n-1)\times|G\setminus G'|} = H(\boldsymbol{M}^{\bar{s}}_{G\setminus G'})$. We conclude

$$H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K} = K, \boldsymbol{W}^{\bar{e}}) = H(\boldsymbol{M}^{\bar{s}}_{G\setminus G'}) = H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'}).$$

Applying property (8), we obtain $H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K} = K, \boldsymbol{W}^{\bar{e}}) = H(\boldsymbol{A}^{\bar{e}}_{G\setminus G'})$ and consequently

$$H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{A}^{\bar{e}}_{G\setminus G'}, \boldsymbol{K} = K) = H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K} = K).$$

3. Once again, we apply property (8) and Lemma 1 to (23) and obtain the particular case

$$H(\boldsymbol{M}^{\bar{s}}_{G'} \mid \boldsymbol{W}^{\bar{e}}, \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{K} = K) = H(\boldsymbol{M}^{\bar{s}}_{G'}).$$

It follows, from property (7), that

$$H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{M}^{\bar{s}}_{G'}, \boldsymbol{K} = K) = H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{K} = K).$$

4. Thanks to Lagrange's interpolation theorem, we can write $H(\boldsymbol{F}^{e}_{G''}, \boldsymbol{W}^{e} \mid \boldsymbol{F}^{e}_{G}) = 0$ and $H(\boldsymbol{F}^{e}_{G} \mid \boldsymbol{F}^{e}_{G''}, \boldsymbol{W}^{e}) = 0$ where G'' is a set of $k-1$ distinct non-null indices. In particular, if $G'' = G'$ ($|G'| = h < k$), we obtain $H(\boldsymbol{F}^{e}_{G'}, \boldsymbol{W}^{e} \mid \boldsymbol{F}^{e}_{G}) = 0$ and $H(\boldsymbol{F}^{e}_{G} \mid \boldsymbol{F}^{e}_{G'}, \boldsymbol{W}^{e}) = 0$. Using property (5), it follows that

$$H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{K}) = H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}_{G'}, \boldsymbol{W}^{e}, \boldsymbol{K}).$$

In Sect. 5.4, we have demonstrated that if $|G| < k$ then

$$H(\boldsymbol{W} \mid \boldsymbol{F}_G, \boldsymbol{M}_G, \boldsymbol{E}, \boldsymbol{S}, \boldsymbol{M}^{s}) = H(\boldsymbol{W}).$$

Applying this property to G' and combining it with property (8) gives

$$H(\boldsymbol{W} \mid \boldsymbol{F}_{G'}, \boldsymbol{K}) = H(\boldsymbol{W}).$$

From property (6), $H(\boldsymbol{W} \mid \boldsymbol{F}_{G'}, \boldsymbol{K}) = H(\boldsymbol{W})$ involves $H(\boldsymbol{F}_{G'}, \boldsymbol{K} \mid \boldsymbol{W}) = H(\boldsymbol{F}_{G'}, \boldsymbol{K})$ and from property (7), $H(\boldsymbol{F}_{G'}, \boldsymbol{K} \mid \boldsymbol{W}) = H(\boldsymbol{F}_{G'}, \boldsymbol{K} \mid \boldsymbol{W}^{e}, \boldsymbol{W}^{\bar{e}}) = H(\boldsymbol{F}_{G'}, \boldsymbol{K})$ involves $H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}_{G'}, \boldsymbol{K}, \boldsymbol{W}^{e}) = H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{W}^{e})$. We assume that the secrets are independent; consequently, $H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{W}^{e}) = H(\boldsymbol{W}^{\bar{e}})$, which allows us to conclude $H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}_{G'}, \boldsymbol{W}^{e}, \boldsymbol{K}) = H(\boldsymbol{W}^{\bar{e}})$, i.e., $H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{K}) = H(\boldsymbol{W}^{\bar{e}})$. Using Lemma 1, it follows that $H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}^{\bar{e}}_{G'}, \boldsymbol{F}^{e}_{G}, \boldsymbol{K} = K) = H(\boldsymbol{W}^{\bar{e}})$.

The demonstrations of the four steps above yield that $H(\boldsymbol{W}^{\bar{e}} \mid \boldsymbol{F}_{G'}, \boldsymbol{M}_{G'}, \boldsymbol{E} = e, \boldsymbol{S} = s, \boldsymbol{M}^{s} = (\,r_{1,s}, \ldots, r_{m,s}\,), \boldsymbol{Q} = \pi, \boldsymbol{A}_G) = H(\boldsymbol{W}^{\bar{e}})$. □

6 Efficiency Consideration

Clearly, the number of shares returned by the servers to the receiver is higher with the proposed protocol (linear communication complexity in n) than with Beimel, Chee, Wang and Zhang's DOT protocols [2] (sublinear communication complexity in n for some PIR protocols). However, in this section, we show that the performance of Blundo et al.'s DOT protocol [4,5] and of our protocol are similar.

In Table 2, we list the main computations performed by each party, for Blundo et al.'s DOT protocol and for our DOT protocol.

Table 2. Computation Efficiency of DOT protocols

	Blundo et al.'s DOT Protocol	Our DOT Protocol
Set-up Phase		
$\mathcal{S}$	$2(n-1)$ random masks in $\mathbb{K}^*$, $2n$ sharing polynomials and $2mn$ shares	n sharing polynomials and mn shares
Commodity Acquisition Phase		
$\mathcal{T}$		mn random masks in $\mathbb{K}$ 1 random number in $[n]$
Transfer Phase		
$\mathcal{R}$	$(n-1)$ sharing polynomials and $k(n-1)$ shares, 4 polynomial interpolations	1 cyclic permutation of $\mathfrak{S}_n$, 1 polynomial interpolation
S_j $(j \in \mathcal{I})$	2 $(n-1)$-tuple scalar products and 2 additions	n additions

Similarly, in Table 3, we list for each protocol the number of shares exchanged between the sender and the servers, the receiver and the servers, and between the trusted initializer and (1) the sender and (2) the receiver in the case of our protocol. We assume that in both protocols, k servers are contacted by the receiver, i.e., $t = k$ in our protocol.

The operations performed off-line (set-up and commodity acquisition phases) for both protocols are close, but in our protocol these operations are distributed between the sender and the trusted initializer. As for the on-line operations, our protocol is more efficient than Blundo et al.'s one: on the receiver's side, only one cyclic permutation and one interpolation are required (vs. the generation of $k(n-1)$ shares from $(n-1)$ sharing polynomials and four interpolations in the case of Blundo et al.'s protocol), whereas on the servers' side, only n additions are required (vs. $2(n-1)$-tuple scalar products and two additions in the case of Blundo et al.'s protocol).

The number of shares distributed by the sender in the set-up phase is around $3n$ in Blundo et al.'s protocol and $2n$ in our protocol. However, our protocol requires an additional distribution of $m(n+1)$ shares by the trusted initializer in the commodity acquisition phase. In the transfer phase, the request sent to

Table 3. Communication Efficiency of DOT protocols (shares)

	Blundo et al.'s DOT Protocol	Our DOT Protocol
Set-up Phase		
$\mathcal{S} \to S_j$ $(j \in [m])$	$2n$ shares, $n-1$ elements of $\mathbb{K}$	n shares
Commodity Acquisition Phase		
$\mathcal{T} \to S_j$ $(j \in \mathcal{I})$		n masks
$\mathcal{T} \to \mathcal{R}$		1 index, m masks
Transfer Phase		
$\mathcal{R} \to S_j$ $(j \in \mathcal{I})$	$n-1$ shares	$t = k$ server indices, 1 number in $[n]$ (nota: broadcast data)
$S_j \to \mathcal{R}$ $(j \in \mathcal{I})$	$2n$ shares	n shares

a server contains $n-1$ shares (Blundo et al.'s protocol) whereas the broadcast request contains $k+1$ integers (our protocol). The receiver collects two times more shares in Blundo et al.'s protocol than in our protocol.

We also note that our DOT protocol can easily be extended to a DOT-$\binom{n}{\ell}$; instead of choosing one set of random masks, the trusted initializer randomly selects ℓ sets of random masks and distributes them to the receiver in the commodity acquisition phase, with the corresponding indices $s_1, \ldots, s_\ell$. In this scenario, the receiver selects ℓ indices $e_1, \ldots, e_\ell$ and generates a random permutation π, instead of a cyclic permutation, such that $\pi(e_1) = s_1, \ldots, \pi(e_\ell) = s_\ell$. The operations executed by the servers are the same as in the case where the receiver wishes to obtain one secret only. On reception of the responses, the receiver has to interpolate ℓ polynomials to determine the ℓ chosen secrets. Therefore, in our protocol, due to the constant number of operations performed by the servers and to the constant number of data exchanged between the servers and the receiver, the communication and computation performance, relative to ℓ, improves when ℓ increases. Blundo et al.'s DOT protocol would need to be executed ℓ times for ℓ secrets, which would be less efficient than our protocol.

In a similar vein, the protocol may easily be extended to a verifiable DOT, with the simple requirement that enough shares are collected by the receiver to identify – and discard – incorrect shares returned by malicious servers. Thus, a Reed-Solomon codes [14] decoding algorithm like the algorithm introduced by Gao [9] would allow the receiver to determine the chosen secret in spite of $u \leq \frac{t-k}{2}$ malicious servers.

Acknowledgements. We would like to thank the anonymous reviewers of ICISC 2012 for their helpful comments.

References

1. Beaver, D.: Commodity-based cryptography. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, pp. 446–455. ACM (1997)
2. Beimel, A., Chee, Y.M., Wang, H., Zhang, L.F.: Communication-efficient distributed oblivious transfer. Journal of Computer and System Sciences 78(4), 1142–1157 (2012)
3. Bennett, C.H., Brassard, G., Crépeau, C., Skubiszewska, M.-H.: Practical Quantum Oblivious Transfer. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 351–366. Springer, Heidelberg (1992)
4. Blundo, C., D'Arco, P., De Santis, A., Stinson, D.R.: New Results on Unconditionally Secure Distributed Oblivious Transfer. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 291–309. Springer, Heidelberg (2003)
5. Blundo, C., D'Arco, P., De Santis, A., Stinson, D.R.: On Unconditionally Secure Distributed Oblivious Transfer. Journal of Cryptology 20(3), 323–373 (2007)
6. Brassard, G., Crépeau, C., Robert, J.M.: All-or-Nothing Disclosure of Secrets. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 234–238. Springer, Heidelberg (1987)
7. Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2nd edn. John Wiley & Sons, Inc., Hoboken (2006)
8. Even, S., Goldreich, O., Lempel, A.: A Randomized Protocol for Signing Contracts. Communications of the ACM 28, 637–647 (1985)
9. Gao, S.: A new algorithm for decoding Reed-Solomon codes. In: Bhargava, V.K., Poor, H.V., Tarokh, V., Yoon, S. (eds.) Communications, Information and Network Security, pp. 55–68. Kluwer Academic Publishers (2003)
10. Gertner, Y., Malkin, T.: Efficient Distributed (n choose 1) Oblivious Transfer. Tech. rep., MIT Lab of Computer Science (1997)
11. Naor, M., Pinkas, B.: Distributed Oblivious Transfer. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 205–219. Springer, Heidelberg (2000)
12. Nikov, V., Nikova, S., Preneel, B., Vandewalle, J.: On Unconditionally Secure Distributed Oblivious Transfer. In: Menezes, A., Sarkar, P. (eds.) INDOCRYPT 2002. LNCS, vol. 2551, pp. 395–408. Springer, Heidelberg (2002)
13. Rabin, M.O.: How to Exchange Secrets with Oblivious Transfer. Tech. rep., Aiken Computation Lab, Harvard University (1981)
14. Reed, I., Solomon, G.: Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics 8(2), 300–304 (1960)
15. Rivest, R.L.: Unconditionally Secure Commitment and Oblivious Transfer Schemes Using Private Channels and a Trusted Initializer (1999) (unpublished manuscript)
16. Shamir, A.: How to Share a Secret. Communications of the ACM 22(11), 612–613 (1979)
17. Shannon, C.E.: A Mathematical Theory of Communication. Bell System Technology Journal 27, 379–423, 623–656 (1948)

A Conditional Entropy with Fixed Condition

Let $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$ be three random variables.

Lemma 1. *If $H(\boldsymbol{X} \mid \boldsymbol{Y}, \boldsymbol{Z}) = H(\boldsymbol{X})$ then for $z_i \in \mathcal{Z}$ we have $H(\boldsymbol{X} \mid \boldsymbol{Y}, \boldsymbol{Z} = z_i) = H(\boldsymbol{X})$.*

Proof.
Because $H(\boldsymbol{X} \mid \boldsymbol{Y}, \boldsymbol{Z}) = H(\boldsymbol{X})$, the variables $\boldsymbol{X}$ and $(\,\boldsymbol{Y}, \boldsymbol{Z}\,)$ are independent. Their corresponding probabilities satisfy the relation $\Pr(\boldsymbol{X} = x, \boldsymbol{Y} = y, \boldsymbol{Z} = z) = \Pr(\boldsymbol{X} = x)\Pr(\boldsymbol{Y} = y, \boldsymbol{Z} = z)$ for $(\,x, y, z\,) \in \mathcal{X}\times\mathcal{Y}\times\mathcal{Z}$. That is, $\Pr(\boldsymbol{X} = x \mid \boldsymbol{Y} = y, \boldsymbol{Z} = z) = \Pr(\boldsymbol{X} = x, \boldsymbol{Y} = y, \boldsymbol{Z} = z)/\Pr(\boldsymbol{Y} = y, \boldsymbol{Z} = z) = \Pr(\boldsymbol{X} = x)$. Hence

$$\begin{aligned}
&H(\boldsymbol{X} \mid \boldsymbol{Y}, \boldsymbol{Z} = z_i) \\
&\qquad = \sum_{y\in\mathcal{Y}} \Pr(\boldsymbol{Y} = y) \times H(\boldsymbol{X} \mid \boldsymbol{Y} = y, \boldsymbol{Z} = z_i) \\
&\qquad = \sum_{y\in\mathcal{Y}} \Big(\Pr(\boldsymbol{Y} = y) \\
&\qquad\qquad \times - \sum_{x\in\mathcal{X}} \Pr(\boldsymbol{X} = x \mid \boldsymbol{Y} = y, \boldsymbol{Z} = z_i) \log_2 \Pr(\boldsymbol{X} = x \mid \boldsymbol{Y} = y, \boldsymbol{Z} = z_i)\Big) \\
&\qquad = \sum_{y\in\mathcal{Y}} \Big(\Pr(\boldsymbol{Y} = y) \times - \sum_{x\in\mathcal{X}} \Pr(\boldsymbol{X} = x) \log_2 \Pr(\boldsymbol{X} = x)\Big) \\
&\qquad = \sum_{y\in\mathcal{Y}} \Pr(\boldsymbol{Y} = y) \times H(\boldsymbol{X}) \\
&\qquad = H(\boldsymbol{X})
\end{aligned}$$

□

Practically Efficient Multi-party Sorting Protocols from Comparison Sort Algorithms

Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi

NTT Secure Platform Laboratories, NTT Corporation
3-9-11, Midori-cho, Musashino-shi, Tokyo 180-8585 Japan
{hamada.koki,kikuchi.ryo,ikarashi.dai,chida.koji,
takahashi.katsumi}@lab.ntt.co.jp

Abstract. Sorting is one of the most important primitives in various systems, for example, database systems, since it is often the dominant operation in the running time of an entire system. Therefore, there is a long list of work on improving its efficiency. It is also true in the context of secure multi-party computation (MPC), and several MPC sorting protocols have been proposed. However, all existing MPC sorting protocols are based on less efficient sorting algorithms, and the resultant protocols are also inefficient. This is because only a method for converting data-oblivious algorithms to corresponding MPC protocols is known, despite the fact that most efficient sorting algorithms such as quicksort and merge sort are not data-oblivious. We propose a simple and general approach of converting non-data-oblivious comparison sort algorithms, which include the above algorithms, into corresponding MPC protocols. We then construct an MPC sorting protocol from the well known efficient sorting algorithm, quicksort, with our approach. The resultant protocol is *practically* efficient since it significantly improved the running time compared to existing protocols in experiments.

Keywords: Multi-party protocol, sorting, comparison sort, secret sharing, unconditional security.

1 Introduction

With the growth in information technology, the use of personal data is also increasing. Therefore, awareness concerning privacy issues has been growing, and systems that use sensitive data without breaching privacy are needed. Secure multi-party computation (MPC) is a technique that enables the creation of such secure systems, and frameworks, such as FairplayMP [3], Sharemind [6], SEPIA [7], TASTY [17], and VIFF [13], have been implemented. MPC protocols allow a set of participants (*parties*) to compute a function privately. That is, when a function is represented as $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, each party with its private input x_i obtains only the output y_i and nothing else. In a typical MPC framework, input and output values are in secret-shared form. Namely, x_i and y_i are the shares of input and output values, respectively. Although any function can be computed securely by using a circuit representation of the function [4,15], it is not easy to design practically efficient MPC protocols for complex algorithms, such as database operations. Therefore, proposals have been made to construct specific and efficient MPC protocols as building blocks, e.g., computing bit-decomposition and comparison [10,25], and modulo reduction [24].

Sorting is one of the most important primitives in various systems, for example, database systems, since it is frequently conducted and comparatively time-consuming. The importance of a sorting algorithm is known, and there is a long list of work on improving its efficiency. To obtain a practically efficient sorting algorithm, researchers not only investigated computational complexity but also experimental performance. Although computational complexity is a good asymptotic metric of efficiency, sometimes an inferior (in the sense of computational complexity) sorting algorithm exceeds the experimental performance of superior ones. For example, quicksort is more popular than merge sort since quicksort often performs better even though its computational complexity is worse than that of the merge sort algorithm. One of the most famous classes of sorting is *comparison sorts*. A comparison sort determines the sorted order based only on comparisons between the input elements. Comparison sorts include a number of well-known and efficient sorting algorithms, such as quicksort, shell sort, heapsort, and merge sort.

In the context of MPC protocols, sorting is also a very important primitive. MPC sorting protocols are often required in various database operations and have many applications such as cooperative IDS [20], oblivious RAM [12] and private set intersection [19]. Therefore, a number of MPC sorting protocols has been proposed [16,3,20,32]. However, they are based on less efficient sorting algorithms, and the resultant protocols are also inefficient. One of the main causes is the obstacle in constructing MPC protocols.

1.1 Obstacle for Using Well-known Algorithms

We say that an algorithm is *data-dependent* if the control flow of the algorithm depends on data values, and an algorithm that is not data-dependent is said to be *data-oblivious*. Generally speaking, there is a large obstacle when one constructs a practically efficient MPC protocol from a well-known algorithm. That is, MPC protocols should be data-oblivious while most efficient algorithms are not. Furthermore, how to convert data-dependent algorithms to data-oblivious algorithms is not known.

To illustrate this obstacle during the conversion from data-dependent algorithms to MPC protocols, let us consider the following two algorithms. Both algorithms receive a sequence of values $a_1, \ldots, a_m \in \mathbb{Z}_p = \{0, 1, \ldots, p-1\}$, where p is a prime, as input, and the output is the number of non-zero values in $a_1, \ldots, a_m$. [1]

```
CountNonZero1($a_1, \ldots, a_m$):
1: $c = 0$.
2: for $i = 1$ to $m$ do
3:     $c = c + ((a_i)^{p-1} \mod p)$.
4: return $c$.
```

```
CountNonZero2($a_1, \ldots, a_m$):
1: $c = 0$.
2: for $i = 1$ to $m$ do
3:     if $a_i \neq 0$ then
4:         $c = c + 1$.
5: return $c$.
```

[1] $(a_i)^{p-1} \mod p = \begin{cases} 0 \text{ if } a_i = 0 \\ 1 \text{ otherwise} \end{cases}$ holds by Fermat's little theorem.

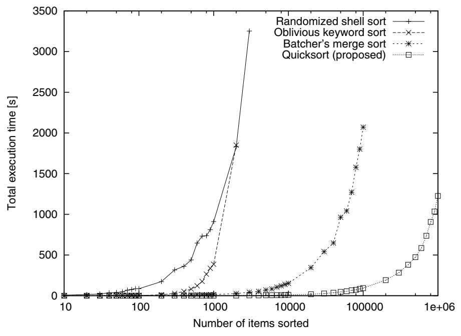


Fig. 1. Running time of four compared sorting implementations. Number of elements on x-axis is on log-scale.

The running time of the first algorithm is $O(m \log p)$ since $(a_i)^{p-1} \mod p$ is computed with $O(\log p)$ multiplications over $\mathbb{Z}_p$ by using the exponentiation by squaring technique, and that of the second algorithm is $O(m)$. Therefore, the second algorithm seems more efficient than the first one.

Next, let us consider the case when we convert these algorithms to MPC protocols. For the first algorithm, we need only minor modifications: We replace the values $a_1, \ldots, a_m$ and c with secret-shared values (or values in other forms depending on the MPC environment), and replace operations applied to these values, such as additions and multiplications, with corresponding MPC subprotocols. [2]

The resulting protocol requires only $O(m \log p)$ invocations of subprotocols. For the second algorithm, it is not enough to apply the same modifications as the first one since the second algorithm has an **if** condition, and the result of the **if** condition discloses the information that $a_i = 0$ or not. Even if the result is hidden, the branch of subsequent processes discloses the information. To avoid these disclosures naively, we have to execute both cases of the **if** condition. Therefore, the resulting protocol requires $\Omega(2^m)$ invocations of subprotocols.

This significant difference between the complexities of the converted protocols is due to the fact that the first algorithm is data-oblivious while the second algorithm is data-dependent. Thus, the naive method used to convert data-oblivious algorithms to MPC protocols does not work when the algorithm is data-dependent.

Above obstacle also occurs in the area of sorting. Therefore, all existing MPC sorting protocols are based on specific sorting algorithms, which are data-oblivious but less efficient. This is one of the main causes of the large gap on efficiency between MPC sorting protocols and well known sorting algorithms.

1.2 Contributions

In this paper we show that in the areas of comparison sort one can efficiently convert data-dependent algorithms to MPC protocols with a simple approach. Furthermore, we

[2] We have no need for applying expensive exponentiation protocols since p is a public constant value.

propose a practically efficient MPC sorting protocol from the well known sorting algorithm quicksort. Note that we discriminate *protocol* and *algorithm* such that the former is used in a multi-party sense and the other is in an ordinal one, and say an algorithm or protocol is *practically efficient* if it not only has less computational complexity but also delivers good experimental results.

When trying to convert comparison sort algorithms to MPC protocols, an obstacle to conversion for data-dependent algorithms occurs: The next pair of elements to be compared depends on the outcome of previous comparisons in sorting algorithms. Therefore, well known and practically efficient comparison sort algorithms, such as quicksort, have not been applied to MPC protocols.

To overcome the above obstacle, we use a simple approach of *shuffling before sorting*. That is, the parties first shuffle the input (in an MPC sense) and then use a comparison sort algorithm, e.g., quicksort or merge sort, with minor modifications on the shuffled secret-shared values. Roughly speaking, although the data-dependent comparison leaks the order of compared elements, the order is randomized by the shuffling and has no relation to the inputs of the protocol. Therefore, we can straightforwardly construct MPC sorting protocols from comparison sort algorithms after shuffling.

We next show that our approach can construct a practically efficient MPC sorting protocol. We concretely construct an MPC sorting protocol from the quicksort algorithm with our approach. Our protocol uses $O(m \log m)$ comparisons in $O(\log m)$ rounds on average, which are comparable to other existing protocols. We describe a precise complexity comparison in Sect. 4. Furthermore, we implement the proposed quicksort protocol and other existing sorting protocols [2,31,32] on $(2, 3)$-Shamir's secret-sharing scheme with corruption tolerance $t = 1$. This setting is reasonable since our aim is to produce a practically efficient sorting protocol and the performance of MPC protocols does not scale well based on the number of parties. As a result, our proposed quicksort protocol sorts 32-bit words and $1,000,000$ secret-shared values in $1,227$ seconds, while existing sorting protocols cannot sort within $3,600$ seconds. We describe an intuitive graph in Fig. 1 and precise experimental results in Sect. 4.

1.3 Related Work

Some circuit-based sorting algorithms are known as sorting networks. Since sorting networks are constructed in a circuit style and circuit-based algorithms are data-oblivious, they can be efficiently applied to MPC protocols. Ajtai et al. proposed an asymptotically optimal sorting network known as the AKS sorting network, which exhibits a complexity of $O(m \log m)$ comparisons, where m is the number of input shares [1]. However, this algorithm is not practical since its constant factor is very high. On the other hand, Batcher's merge sort [2] is more efficient unless m is quite large [21]. This algorithm exhibits a complexity of $O(m \log^2 m)$ comparisons with a lower constant factor.

Goodrich proposed a data-oblivious sort called randomized shell sort [16]. Similar to sorting networks, data-oblivious sorts are also efficiently applied to MPC protocols. Although randomized shell sort returns a wrong output with low probability, it exhibits a complexity of $O(m)$ rounds and $O(m \log m)$ comparisons.

Wang et al. reported experimental results of some sorting algorithms [31]. Their implementation is based on the MPC system Fairplay [23]. The running times of Batcher's merge sort [2] and randomized shell sort [16] for 256 input values are approximately $3,000$ and $6,200$ seconds, respectively.

Jónsson et al. studied a general technique to hide the number of input values for sorting protocols [20]. They also implemented Batcher's merge sort [2] and other sorting protocols on the MPC system Sharemind [6]. Their implementation is optimized using a technique called vectorization, and the vectorized Batcher's merge sort sorts $16,384$ secret shared values in 210 seconds.

Zhang proposed data-oblivious sorting algorithms [32] based on bead sort. All of Zhang's algorithms exhibit complexities of constant rounds and $O(Rm)$ or $O(m^2)$ comparisons depending on the algorithm, where R represents the range of input values. Since these algorithms are data-oblivious, we can convert them to multi-party sorting protocols by using a circuit-based technique while keeping their complexities.

2 Preliminaries

2.1 Assumptions and Notations

We focus on secret-sharing-based MPC. For simplicity, n parties $P_1, \ldots, P_n$ are connected by secure channels. All values used in secret-sharing schemes belong to a field K. We use $[\![s]\!]_{P_i}$ to denote a *share* for P_i where a *secret value* is $s \in K$. Let $\mathbb{Q}$ be a coalition of parties and $[\![s]\!]_{\mathbb{Q}}$ denote a set of shares $\{[\![s]\!]_{P_i} \mid P_i \in \mathbb{Q}\}$. When $\mathbb{U}$ represents all parties, we simply denote $[\![s]\!]_{\mathbb{U}}$ as $[\![s]\!]$ and call it *shared values*. We call some elements related to secret-sharing scheme as follows;

– s: secret value,
– $[\![s]\!]_{P_i}$: share (for a party P_i),
– $[\![s]\!] = [\![s]\!]_{\mathbb{U}} = \{[\![s]\!]_{P_1}, \ldots, [\![s]\!]_{P_n}\}$: shared value.

We use $[i]$ to denote a set $\{1, 2, \ldots, i\}$.

2.2 Security Model

We consider unconditional, perfect security against a semi-honest adversary with static corruption of at most t. This means that the adversary can execute unbounded computation, must follow a protocol, and can corrupt at most t parties only before the protocol is conducted. More technically, we say that a protocol is secure if there is a simulator that simulates the view of corrupted parties from the inputs and outputs of the protocol. We use $\mathbb{I} = \{P_{i_1}, P_{i_2}, \ldots, P_{i_t}\} \subset \mathbb{U}$ to denote the parties that are corrupted. Due to space limitation, the formal definition of the security against a semi-honest adversary with static corruption appears in Appendix A.

2.3 Complexity Metrics in MPC

We use two metrics, *round complexity* and *the number invocations of the comparison protocol*, to evaluate the overall running time of protocols. The round complexity of a protocol is the number of rounds of parallel invocations of the communication. Because the comparison protocol is a dominant factor of the complexity of communications, we measure the amount of data transmitted by the parties with the number of invocations of the comparison protocol.

2.4 Secret-Sharing Scheme

We focus on a class of secret-sharing schemes called (k, n)-threshold. This means that the shares are shared by the n parties in such a way that any coalition of k or more parties can together reconstruct the secret, but no coalition fewer than k parties can. Shamir's secret-sharing scheme [28] belongs to this class. We assume that the corruption tolerance t satisfies $t < \min(k, n - k)$. We say $[\![s]\!]$ is *uniformly random* if it is uniformly randomly chosen from the set of possible shared values whose secret value is s.

A secret-sharing scheme Π_{SS} is a pair of algorithms, *dealing* and *revealing*. The dealing algorithm takes a secret value s as input and outputs a uniformly random shared value. The revealing algorithm takes at least k shares and outputs the secret value s.

2.5 Shuffling, Comparison, and Reveal Protocols

We introduce some existing MPC protocols used as building blocks of our protocol.

Our protocols are designed to be used as building blocks in the paradigm of computing on shared values, which is one of the most common paradigms for MPC protocols [8]. In this paradigm, secret values are preliminary shared with a secret-sharing scheme to all parties that participate in MPC protocols. Then MPC protocols take secret-shared values as inputs from each party and output the result in secret-shared form. The result is finally recovered by the revealing algorithm of the secret-sharing scheme.

Comparison protocol. The comparison protocol [10,25] receives two shared values and outputs a shared value of the comparison result of the inputs. More precisely, the comparison protocol accepts $[\![a]\!]_{P_i}, [\![b]\!]_{P_i}$ from each $P_i \in \mathbb{U}$ as input and outputs $[\![c]\!]_{P_i}$ to each $P_i \in \mathbb{U}$ such that $c = 1$ if $a \leq b$ and $c = 0$ otherwise. We assume that K is totally ordered and denote this protocol as "$[\![c]\!] \leftarrow [\![a \leq b]\!]$". We formally define the comparison protocol with the following function $f^{\mathsf{CMP}}_{\Pi_{\mathsf{SS}}}$.

$f^{\mathsf{CMP}}_{\Pi_{\mathsf{SS}}}$: *On inputting* $[\![x]\!]_{P_i}$ *and* $[\![y]\!]_{P_i}$ *from each* $P_i \in \mathbb{U}$*, it reveals* x *and* y *with the revealing algorithm of* Π_{SS}*, sets* $z = 1$ *if* $x \leq y$ *and* $z = 0$ *otherwise, and generates* $[\![z]\!]$ *with the dealing algorithm of* Π_{SS}*. Finally, it outputs* $[\![z]\!]_{P_i}$ *to each* $P_i \in \mathbb{U}$*.*

The comparison protocol proposed by Nishide and Ohta [25] exhibits the complexity of $O(1)$ rounds and $O(\ell)$ invocations of multiplication protocols where ℓ is the bit-length of K.

Shuffling protocol. The shuffle protocol receives some shared values and outputs renewed shared values where their secret values are uniformly randomly permuted. More precisely, the shuffle protocol accepts $[\![a_1]\!]_{P_i}, \dots, [\![a_m]\!]_{P_i}$ from each $P_i \in \mathbb{U}$ and outputs $[\![b_1]\!]_{P_i}, \dots, [\![b_m]\!]_{P_i}$ to each $P_i \in \mathbb{U}$ such that $b_j = a_{\pi(j)}$ for a uniformly random permutation $\pi : [m] \to [m]$ and every $j \in [m]$. A run of this protocol is denoted as

$$[\![b_1]\!], \dots, [\![b_m]\!] \leftarrow \mathsf{Shuffle}([\![a_1]\!], \dots, [\![a_m]\!]).$$

We formally define the shuffling protocol with the following function $f^{\mathsf{Shuffle}}_{\Pi_{\mathsf{SS}}}$.

$f^{\mathsf{Shuffle}}_{\Pi_{\mathsf{SS}}}$: *On inputting* $([\![a_1]\!]_{P_i}, \dots, [\![a_m]\!]_{P_i})$ *from each* $P_i \in \mathbb{U}$*, it reveals* $a_1, \dots, a_m$ *with the revealing algorithm of* Π_{SS}*, selects a permutation* $\pi : [m] \to [m]$ *uniformly* *at random, sets $b_i = a_{\pi(i)}$ for $i \in [m]$, and generates $[\![b_1]\!], \ldots, [\![b_m]\!]$ with the dealing algorithm of* Π_{SS}. *Finally, it outputs* $([\![b_1]\!]_{P_i}, \ldots, [\![b_m]\!]_{P_i})$ *to each* $P_i \in \mathbb{U}$.

Laura et al. proposed efficient shuffling protocols [22]. One of their protocols exhibits the complexity of $O(2^n/\sqrt{n})$ rounds and $O(2^n n^{3/2} m \log m)$ communications. When the number of parties is constant, it exhibits $O(1)$ rounds and $O(m \log m)$ communications. We use this protocol as the shuffling protocol.

Reveal protocol. The reveal protocol accepts $[\![x]\!]_{P_i}$ from each $P_i \in \mathbb{U}$ and outputs x to each $P_i \in \mathbb{U}$. This protocol just has a role of the reveal algorithm in a multi-party setting. A run of this protocol is denoted as

$$x \leftarrow \mathsf{Reveal}([\![x]\!]).$$

We formally define the reveal protocol with the following function $f^{\mathsf{Reveal}}_{\Pi_{\mathsf{SS}}}$.

$f^{\mathsf{Reveal}}_{\Pi_{\mathsf{SS}}}$: *On inputting $[\![x]\!]_{P_i}$ from each $P_i \in \mathbb{U}$, it reveals x with the revealing algorithm of* Π_{SS} *and outputs x to each $P_i \in \mathbb{U}$.*

The reveal protocol can be easily constructed in a semi-honest model by distributing all shares among all parties. Even in the malicious model it can be constructed by using secret-sharing schemes *secure against cheating* [27,26].

3 MPC Sorting Protocols

In this section, we propose an approach of constructing efficient sorting protocols, and then we apply our approach to the quicksort algorithm. For simplicity, we split the construction of our quicksort protocol with two steps: we begin by describing the construction with restricted inputs and later show how to remove this restriction. We also discuss further extensions of our approach.

We assume that the following protocols can be executed on Π_{SS}; shuffling, comparison, and reveal. For example, Shamir's secret-sharing scheme satisfies this condition.

3.1 Our Approach of Constructing Efficient Sorting Protocols

To construct an efficient sorting protocol, it is natural to try to construct an MPC sorting protocol that emulates practically efficient sorting algorithms. However, this approach has to solve a certain problem; When trying to convert well-known sorting algorithms to MPC protocols, the problem with most practically efficient sorting algorithms is that they are data-dependent. On the other hand, if an MPC protocol changes its behavior according to the input, it might violate privacy. Therefore, all existing sorting protocols use less efficient data-oblivious sorting algorithms. Consequently, we have to fill the gap between data-dependency and data-obliviousness to construct MPC sorting protocols from well-known sorting algorithms.

Sorting algorithms which determine the sorted order based only on comparisons between the input elements are called comparison sorts. Comparison sorts include a number of practically efficient sorting algorithms, such as quicksort, shell sort, heapsort, insertion sort, and merge sort. However, comparison sorts are essentially data-dependent since the next pair of elements to be compared depends on the outcome of previous comparisons. Therefore no comparison sort, including well known quicksort algorithm, has been applied to MPC protocols.

To solve the above problem, we use a simple approach of *shuffling before sorting*. Our approach consists of the following modifications to the original comparison sort algorithm.

1. We apply the shuffling protocol to the inputs at the first step.
2. We execute as the same to the original (data-dependent) comparison sort algorithm, except to replace the comparison operation with a continuous execution of comparison and reveal protocols.

In the execution of the protocol, the revealed result of comparison seems to leak ordinal information. However, the ordinal information is randomized by the shuffling at the first step, so it leaks no information about true inputs. This approach is quite simple and effective for constructing practically efficient sorting protocols. Our approach is also quite general since, to our knowledge, all of practically efficient comparison sort algorithms can be converted to MPC protocols with our approach.

3.2 Quicksort Protocol

Now, we concretely construct an MPC sorting protocol, which we call quicksort protocol, from the quicksort algorithm with our approach. Note that we assume that the secret values of inputs are distinct here and discuss the unrestricted input case in the following subsection.

The sorting function is defined as follows.

$f^{\mathsf{Sorting}}_{\Pi_{\mathsf{SS}}}$: *On inputting* $(\llbracket a_1 \rrbracket_{P_i}, \ldots, \llbracket a_m \rrbracket_{P_i})$ *from each* $P_i \in \mathbb{U}$, *it reveals* $a_1, \ldots, a_m$ *with the reveal algorithm of* Π_{SS}, *sorts* $(a_1, \ldots, a_m)$ *to* $(b_1, \ldots, b_m)$ *such that* $b_i \leq b_{i+1}$ *for* $i \in [m-1]$, *and generates* $\llbracket b_1 \rrbracket, \ldots, \llbracket b_m \rrbracket$ *with the dealing algorithm of* Π_{SS}. *Finally, it outputs* $(\llbracket b_1 \rrbracket_{P_i}, \ldots, \llbracket b_m \rrbracket_{P_i})$ *to each* $P_i \in \mathbb{U}$.

We describe our quicksort protocol constructed by applying our approach in Protocol 1. Next we discuss the property of our quicksort protocol.

Correctness. Our quicksort protocol has two differences compared to the original quicksort algorithm. The first difference is comparison; however, this has no effect on execution since the replicated protocols simply emulate the original. The second difference is an additional shuffling step inserted at the beginning of our quicksort protocol. Since the secret values of the input shared values are distinct, the order of the secret values of the output is unique. Therefore, the first shuffling step does not affect the results.

Security. Roughly speaking, the shuffling and comparison protocols are secure, and the swapping operation is just a local computation. Therefore, the only possible information leakage is the revealed results from the comparisons. However, the results of each comparison have no relation to the input. This is because the input shared values are shuffled in the first step by the shuffling protocol. We formally claim the following theorem.

Theorem 1. *Protocol 1 t-privately reduces* $f^{\mathsf{Sorting}}_{\Pi_{\mathsf{SS}}}$ *to* $f^{\mathsf{Shuffle}}_{\Pi_{\mathsf{SS}}}$, $f^{\mathsf{CMP}}_{\Pi_{\mathsf{SS}}}$, *and* $f^{\mathsf{Reveal}}_{\Pi_{\mathsf{SS}}}$.

The proof of the theorem appears in Appendix B.

Protocol 1. Quicksort protocol

Notation: $[\![b_1]\!], \ldots, [\![b_m]\!] \leftarrow \mathsf{Quicksort}([\![a_1]\!], \ldots, [\![a_m]\!])$
Input: Shared values $[\![a_1]\!], \ldots, [\![a_m]\!]$.
Output: Shared values $[\![b_1]\!], \ldots, [\![b_m]\!]$ where $b_1 \leq \cdots \leq b_m$.

```
1: Unless this is a recursively called execution, apply the shuffling protocol to $[\![a_1]\!], \ldots, [\![a_m]\!]$.
2: if $1 < m$ then
3:     $p, [\![e_1]\!], \ldots, [\![e_m]\!] \leftarrow \mathsf{Partition}([\![a_1]\!], \ldots, [\![a_m]\!])$.
4:     $[\![b_1]\!], \ldots, [\![b_{p-1}]\!] \leftarrow \mathsf{Quicksort}([\![e_1]\!], \ldots, [\![e_{p-1}]\!])$.
5:     Let $[\![b_p]\!] = [\![e_p]\!]$.
6:     $[\![b_{p+1}]\!], \ldots, [\![b_m]\!] \leftarrow \mathsf{Quicksort}([\![e_{p+1}]\!], \ldots, [\![e_m]\!])$.
7: else
8:     Let $([\![b_1]\!], \ldots, [\![b_m]\!]) = ([\![a_1]\!], \ldots, [\![a_m]\!])$.
9: return $[\![b_1]\!], \ldots, [\![b_m]\!]$.
```

Notation: $p, [\![e_1]\!], \ldots, [\![e_m]\!] \leftarrow \mathsf{Partition}([\![a_1]\!], \ldots, [\![a_m]\!])$
Input: Shared values $[\![a_1]\!], \ldots, [\![a_m]\!]$.
Output: Position p and shared values $[\![e_1]\!], \ldots, [\![e_m]\!]$.

```
1: Let $i = 0$.
2: for $j = 1$ to $m - 1$ do
3:     $[\![c]\!] \leftarrow [\![a_j \leq a_m]\!]$.
4:     $c \leftarrow \mathsf{Reveal}([\![c]\!])$.
5:     if $c = 1$ then
6:         Let $i = i + 1$.
7:         Swap $[\![a_i]\!]$ and $[\![a_j]\!]$.
8: Let $p = i + 1$.
9: Swap $[\![a_p]\!]$ and $[\![a_m]\!]$.
10: Let $([\![e_1]\!], \ldots, [\![e_m]\!]) = ([\![a_1]\!], \ldots, [\![a_m]\!])$.
11: return $p, [\![e_1]\!], \ldots, [\![e_m]\!]$.
```

Complexity. There are only two subprotocols that matter in terms of complexity. One is the shuffling protocol and the other is the comparison protocol. As described previously, we use the shuffling protocol proposed by Laura et al. [22], which exhibits a complexity of $O(1)$ rounds and $O(m \log m)$ communications when the number of parties n is constant. Since the quicksort algorithm requires $\Omega(m \log m)$ invocations of comparison, we have no need to take into account the complexity of the shuffling protocol.

The number of invocations of the comparison protocol is exactly the same as that of comparisons in the original quicksort algorithm. With a naive implementation, therefore, our quicksort protocol exhibits a complexity of $O(m \log m)$ rounds and $O(m \log m)$ comparisons.

We can improve the round complexity of the main part of the proposed quicksort protocol to $O(\log m)$ by setting the invocations of the comparison protocols to be parallel. First, we claim that the depth of the recursive calls is $\Theta(\log m)$ on average. Since our quicksort protocol shuffles the input in the first step, the input to the main part of the quicksort protocol is uniformly randomized. When the input is assumed to be uniformly randomized, the depth of the recursive calls for the quicksort algorithm is known to be $\Theta(\log m)$ on average [9]. Additionally, we can easily confirm that we can make the invocations of the subprotocol Partition parallel at each depth. Thus, through parallel implementation, our quicksort protocol exhibits a complexity of $O(\log m)$ rounds and $O(m \log m)$ comparisons on average.

3.3 Sorting Duplicated Values

When there are duplicate inputs in its secret values, our quicksort protocol may leak information regarding the input. For example, if the protocol invokes two comparison protocols $[\![a \leq b]\!]$ and $[\![b \leq a]\!]$ s.t. $a = b$, the results of the comparisons reveal the existence of a pair of shared values with identical secret values. Another example is the case when all the values are same. In this case, the results of comparisons are all true, and this implies many values are same with high probability.

We can easily address this problem, for example, by the following steps. Let m be the number of input shares and add $\lceil \log_2 m \rceil$ bits, which we call a tie breaker, to every input share in the least significant positions. Then, we execute the protocol treating the modified input as the input. The above modification gives the identical shared values strict order; therefore, solving the problem. Furthermore, depending on how we make the tie breaker, we can give the proposed quicksort protocol certain features. If we shuffle the tie breaker, the duplicated values are uniformly and randomly ordered. To generate a sorting protocol while retaining the original order of the duplicated items (such a sorting operation is called *stable*), we arrange the tie breakers in ascending order.

3.4 Further Extensions

Beyond sorting, our approach must be applied to many other data-dependent algorithms. We illustrate a *selection algorithm* which is for finding the k-th smallest number in a list. This includes finding the minimum, maximum, and median elements often executed in the database operation. For example, we can obtain the median MPC protocol that exhibits $O(\log m)$ rounds and $O(m)$ comparisons in the average case from Hoare's algorithm [18] and also obtain the protocol that exhibits the same rounds and comparisons even in the worst case from Blum's algorithm [5].

Our approach seems to be secure even in the malicious model if the shuffling, reveal, and comparison protocols are also secure in the malicious model. However, we are interested in constructing a practically efficient MPC protocol, and to our knowledge, there is no secret-sharing scheme providing practically efficient shuffling, reveal, and comparison protocols simultaneously. Therefore, we only give the proof in the semi-honest model in this paper.

4 Evaluation

In this section, we evaluate our quicksort protocol. We compare this protocol with other existing sorting protocols both asymptotically and experimentally. As a result, we show that our quicksort protocol exhibits a comparable computational complexity and significantly improved the running time in an experiment.

Table 1. Complexities of sorting protocols. m and R represent the number of the input values and the range of input values, respectively.

Sorting protocol	Rounds		Invocations of comparison	
	Average	Worst	Average	Worst
AKS sorting network [1]	$O(\log m)$	$O(\log m)$	$O(m \log m)$	$O(m \log m)$
Randomized shell sort [16]	$O(m)$	$O(m)$	$O(m \log m)$	$O(m \log m)$
Batcher's merge sort [2]	$O(\log^2 m)$	$O(\log^2 m)$	$O(m \log^2 m)$	$O(m \log^2 m)$
Oblivious arrayless bead sort [32]	$O(1)$	$O(1)$	$O(Rm)$	$O(Rm)$
Oblivious keyword sort [32]	$O(1)$	$O(1)$	$O(m^2)$	$O(m^2)$
Quicksort (proposed)	$O(\log m)$	$O(m)$	$O(m \log m)$	$O(m^2)$

Table 2. Performance of sorting protocols. m represents the number of the input values. The "N/A" means that the execution did not finish in 3,600 seconds.

Sorting protocol	$m = 10$	$m = 10^2$	$m = 10^3$	$m = 10^4$	$m = 10^5$	$m = 10^6$
Randomized shell sort [16]	6.356[s]	86.355[s]	911.376[s]	N/A	N/A	N/A
Oblivious keyword sort [32]	0.335[s]	3.392[s]	387.128[s]	N/A	N/A	N/A
Batcher's merge sort [2]	1.331[s]	4.139[s]	14.285[s]	152.168[s]	2070.890[s]	N/A
Quicksort (proposed)	0.247[s]	0.488[s]	1.410[s]	9.859[s]	93.674[s]	1226.267[s]

4.1 Complexity Analysis

We first evaluated our quicksort protocol from an asymptotic perspective. As described in Sect. 3, this protocol exhibits a complexity of $O(\log m)$ rounds and $O(m \log m)$ comparisons on average, where m is the number of the input values. We summarize the complexities of ours and existing sorting protocols in Table 1 by taking into account parallelism.

As mentioned repeatedly, we are interested in practically efficient protocols; therefore, we stress the average case rather than the worst case. Our quicksort protocol requires $O(m \log m)$ comparisons on average, which is asymptotically optimal for comparison sorts. Our quicksort protocol is superior to randomized shell sort [16] and Batcher's merge sort [2] in either rounds or comparisons. The AKS sorting network [1] has the same complexity on average. The oblivious arrayless bead sort [32] exhibits $O(1)$ rounds and $O(Rm)$ comparisons where R is the range of secret values. This algorithm is quite efficient when R is small, e.g., the secret value belongs to $\{0, 1\}$. However, when R is large, e.g., $R = 2^{32}$, it becomes quite inefficient. The oblivious keyword sort [32] has a comparable complexity to ours. This exhibits a complexity of constant rounds that is superior to ours but $O(m^2)$ comparisons that is inferior on average.

4.2 Experimental Results

As the quicksort often outperforms other sorting algorithms with $O(m \log m)$ comparisons in practice [30], the experiment results are very important for practical use. We implemented our quicksort protocol and existing sorting protocols, such as the randomized shell sort [16], the oblivious keyword sort [32] and Batcher's merge sort [2], for comparison. The AKS sorting network [1] was not implemented since this algorithm is not of practical interest. We also did not implement the oblivious arrayless bead sort [32]. This is because in many applications such as Oblivious RAM the range of numbers, R, is large; therefore, this sort protocol becomes quite inefficient.

We implemented sorting protocols on (2, 3)-Shamir's secret-sharing scheme with corruption tolerance $t = 1$. This is because MPC protocols generally do not scale well as the number of participants increases, and such MPC protocols executed by a few participants can be building blocks of ones executed by many participants [11]. For better performance, we implemented component protocols secure against a semi-honest adversary. This implies all the implemented sorting protocols are also secure against such an adversary. We implemented the comparison protocol proposed by Damgård et al. [10] as a building block of all sorting protocols. The quicksort protocol additionally uses the shuffling protocol proposed by Laura et al. [22]. Our implementation of the randomized shell sort and Batcher's merge sort protocols are based on circuit representations. That is, we replaced the comparators in the original algorithms to comparator protocols constructed by comparisons, multiplications, and additions. We implemented all of them on C++ and compiled by g++ 4.6.1. All values are in $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$, where p is a prime number 4294967291 and satisfies $2^{31} < p < 2^{32}$, that is, 32-bit words.

We then timed how long the running time of these protocols is. All the experiments were conducted on three laptop machines with an Intel Core i5 2540M 2.6-GHz CPU and 8 GB of physical memory. These three machines were connected to a 1-Gbps LAN. The running times of the sorting protocols are shown in Fig. 1, and detailed times in some cases are summarized in Table 2 where m is the number of input shared values.

As expected, our quicksort protocol allowed us to consider large inputs size. The results show that our quicksort protocol is much faster than randomized shell sort and oblivious keyword sort, and about ten to twenty times faster than Batcher's merge sort. Consequently, the proposed quicksort protocol significantly improved the running time of the existing sorting protocols. In other words, our quicksort protocol is practically efficient.

5 Conclusion

We proposed a simple and general approach, shuffling before sorting, for converting data-dependent but efficient comparison sort algorithms to MPC sorting protocols. We then constructed a quicksort protocol from the quicksort algorithm with our approach. The resultant protocol is practically efficient since it has comparable computational complexity and significantly improved the running time compared to existing protocols in experiments.

References

1. Ajtai, M., Komlós, J., Szemerédi, E.: An O(n log n) sorting network. In: STOC, pp. 1–9. ACM (1983)
2. Batcher, K.E.: Sorting networks and their applications. In: AFIPS Spring Joint Computing Conference, pp. 307–314 (1968)

3. Ben-David, A., Nisan, N., Pinkas, B.: Fairplaymp: a system for secure multi-party computation. In: Ning, P., Syverson, P.F., Jha, S. (eds.) ACM Conference on Computer and Communications Security, pp. 257–266. ACM (2008)
4. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: [29], pp. 1–10
5. Blum, M., Floyd, R.W., Pratt, V.R., Rivest, R.L., Tarjan, R.E.: Time bounds for selection. J. Comput. Syst. Sci. 7(4), 448–461 (1973)
6. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A framework for fast privacy-preserving computations. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 192–206. Springer, Heidelberg (2008)
7. Burkhart, M., Strasser, M., Many, D., Dimitropoulos, X.A.: Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. In: USENIX Security Symposium, pp. 223–240. USENIX Association (2010)
8. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: [29], pp. 11–19
9. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
10. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 285–304. Springer, Heidelberg (2006)
11. Damgård, I., Ishai, Y.: Constant-round multiparty computation using a black-box pseudorandom generator. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 378–394. Springer, Heidelberg (2005)
12. Damgård, I., Meldgaard, S., Nielsen, J.B.: Perfectly secure oblivious RAM without random oracles. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 144–163. Springer, Heidelberg (2011)
13. Geisler, M.: Cryptographic Protocols: Theory and Implementation. PhD thesis, University of Aarhus (2010)
14. Goldreich, O.: The Foundations of Cryptography. Basic Applications, vol. 2. Cambridge University Press (2004)
15. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC, pp. 218–229. ACM (1987)
16. Goodrich, M.T.: Randomized shellsort: A simple oblivious sorting algorithm. In: SODA, pp. 1262–1277 (2010)
17. Henecka, W., Kögl, S., Sadeghi, A.R., Schneider, T., Wehrenberg, I.: Tasty: tool for automating secure two-party computations. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM Conference on Computer and Communications Security, pp. 451–462. ACM (2010)
18. Hoare, C.A.R.: Algorithm 65: find. Commun. ACM 4(7), 321–322 (1961)
19. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: NDSS (2012)
20. Jónsson, K.V., Kreitz, G., Uddin, M.: Secure multi-party sorting and applications. IACR Cryptology ePrint Archive 2011, 122 (2011)
21. Knuth, D.E.: Art of Computer Programming, 2nd edn. Sorting and Searching, vol. 3, ch. 5. Addison-Wesley Professional (1998)
22. Laur, S., Willemson, J., Zhang, B.: Round-efficient oblivious database manipulation. In: Lai, X., Zhou, J., Li, H. (eds.) ISC 2011. LNCS, vol. 7001, pp. 262–277. Springer, Heidelberg (2011)
23. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system. In: USENIX Security Symposium, pp. 287–302 (2004)

24. Ning, C., Xu, Q.: Multiparty computation for modulo reduction without bit-decomposition and a generalization to bit-decomposition. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 483–500. Springer, Heidelberg (2010)
25. Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 343–360. Springer, Heidelberg (2007)
26. Obana, S., Araki, T.: Almost optimum secret sharing schemes secure against cheating for arbitrary secret distribution. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 364–379. Springer, Heidelberg (2006)
27. Ogata, W., Kurosawa, K.: Optimum secret sharing scheme secure against cheating. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 200–211. Springer, Heidelberg (1996)
28. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)
29. Simon, J. (ed.): Proceedings of the 20th Annual ACM Symposium on Theory of Computing, STOC, Chicago, Illinois, USA, May 2-4. ACM (1988)
30. Skiena, S.S.: The Algorithm Design Manual, 2nd edn. Springer Publishing Company, Incorporated (2008)
31. Wang, G., Luo, T., Goodrich, M.T., Du, W., Zhu, Z.: Bureaucratic protocols for secure two-party sorting, selection, and permuting. In: ASIACCS, pp. 226–237 (2010)
32. Zhang, B.: Generic constant-round oblivious sorting algorithm for MPC. In: Boyen, X., Chen, X. (eds.) ProvSec 2011. LNCS, vol. 6980, pp. 240–256. Springer, Heidelberg (2011)

A Formal Definition of the Security

We give the formal definition of the security against a semi-honest adversary with static corruption. Let $\mathbf{x} = (x_1, \ldots, x_n)$, $\mathbf{x}_{\mathbb{I}} = (x_{i_1}, \ldots, x_{i_t})$, $f_i(\mathbf{x})$ be the i-th output of $f(\mathbf{x})$, and $f_{\mathbb{I}}(\mathbf{x}) = (f_{i_1}(\mathbf{x}), \ldots, f_{i_t}(\mathbf{x}))$. We denote the view of P_i during the protocol execution of ρ on inputs $\mathbf{x}$ as $\textsc{view}^{\rho}_{P_i}(\mathbf{x}) = (x_i, r_i; \mu_1, \ldots, \mu_\ell)$ where r_i is P_i's random tape, and μ_j is the j-th message that P_i received in the protocol execution. We also denote the output of P_i as $\textsc{output}^{\rho}_{P_i}(\mathbf{x})$.

We are now ready to define the security notion in the presence of semi-honest adversaries.

Definition 1 ([14]). *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a probabilistic n-ary functionality, ρ be a protocol,* $\textsc{view}^{\rho}_{\mathbb{I}}(\mathbf{x}) = (\textsc{view}^{\rho}_{P_{i_1}}(\mathbf{x}), \ldots, \textsc{view}^{\rho}_{P_{i_t}}(\mathbf{x}))$, *and*

$$\textsc{output}^{\rho}(\mathbf{x}) = (\textsc{output}^{\rho}_{P_1}(\mathbf{x}), \ldots, \textsc{output}^{\rho}_{P_n}(\mathbf{x})).$$

We say that ρ t-privately computes f if there exists $\mathcal{S}$ such that for all $\mathbb{I} \subset \mathbb{U}$ of cardinality of at most t and all $\mathbf{x}$ *, it holds that*

$$\{(\mathcal{S}(\mathbb{I}, \mathbf{x}_{\mathbb{I}}, f_{\mathbb{I}}(\mathbf{x})), f(\mathbf{x}))\} \equiv \left\{(\textsc{view}^{\rho}_{\mathbb{I}}(\mathbf{x}), \textsc{output}^{\rho}(\mathbf{x}))\right\}.$$

It is well known that a protocol satisfying the above security notions can be securely composed with other protocols in a semi-honest setting. To explain this composition property, we introduce the security notion for a protocol that computes a function with the help of an oracle.

Definition 2 ([14]). *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a probabilistic n-ary functionality, $g : (\{0,1\}^*)^m \to (\{0,1\}^*)^m$ be a probabilistic m-ary functionality, and ρ be a protocol. We say that ρ t-privately reduces g to f if ρ privately computes g with an oracle access of the functionality of f.*

We introduce an informal description of the composition theorem. Suppose that a protocol Π^g privately reduces g to f and a protocol Π^f privately computes f. Then the protocol $\Pi^{g|f}$, which is the same as Π^g except that all oracle calls are substituted by the executions of Π^f, privately computes g. This implies that we can treat a constitutive protocol as a black box to prove the security of a high-level protocol.

B Proof of Theorem 1

Let $[\![b'_1]\!], \dots, [\![b'_m]\!]$ be the shuffled (and renewed) shared values in the Step 1 of Quicksort(). The view of adversaries consists of their inputs $[\![a_1]\!]_{\mathbb{I}}, \dots, [\![a_m]\!]_{\mathbb{I}}$, random tapes, $[\![b'_1]\!]_{\mathbb{I}}, \dots, [\![b'_m]\!]_{\mathbb{I}}$, $[\![c]\!]_{\mathbb{I}}$, and c. The output consists of $[\![b_1]\!]_{\mathbb{I}}, \dots, [\![b_m]\!]_{\mathbb{I}}$. Note that the adversaries have no view of the subprotocols Shuffle$(\cdot)$, $[\![\cdot \le \cdot]\!]$, and Reveal$(\cdot)$ since the execution of these protocols are substituted with the oracle invocation of functionalities $f^{\mathsf{Shuffle}}_{\Pi_{\mathsf{SS}}}$, $f^{\mathsf{CMP}}_{\Pi_{\mathsf{SS}}}$, and $f^{\mathsf{Reveal}}_{\Pi_{\mathsf{SS}}}$, respectively.

We construct the simulator $\mathcal{S}$ as follows. Inputs and outputs are the same as those of adversaries, and $\mathcal{S}$ selects random tapes uniformly at random.

As for $[\![b'_1]\!]_{\mathbb{I}}, \dots, [\![b'_m]\!]_{\mathbb{I}}$ and c, let $\pi' : [m] \to [m]$ be a permutation that satisfies $[\![b_i]\!] = [\![b'_{\pi'(i)}]\!]$ $(i \in [m])$. There exists exactly one such permutation since $\{b_1, \dots, b_m\}$ is distinct and $([\![b_1]\!], \dots, [\![b_m]\!])$ is a permutated sequence from $([\![b'_1]\!], \dots, [\![b'_m]\!])$ by the swap operations executed in Step 7 or Step 9 of Partition$(\cdot)$. Once π' is perfectly simulated, $[\![b'_i]\!]_{\mathbb{I}}$ is also perfectly simulated by setting $[\![b_{\pi'^{-1}(i)}]\!]_{\mathbb{I}}$ as the simulated shares and c is also perfectly simulated by setting the value

$$c' = \begin{cases} 1 \text{ if } \pi'^{-1}(i) \le \pi'^{-1}(j) \\ 0 \text{ otherwise} \end{cases}$$

when $[\![b'_i \le b'_j]\!]$ is executed. Now we claim that $\mathcal{S}$ perfectly simulates π' by selecting just a uniformly random permutation. By the correctness of the shuffling and quicksort protocols, $b'_i = a_{\pi_r(i)}$ and $b_i = a_{\pi_s(i)}$ $(i \in [m])$ hold for a fixed (according to $a_1, \dots, a_m$) permutation $\pi_s : [m] \to [m]$ and a uniformly random permutation $\pi_r : [m] \to [m]$. $\pi_s = \pi_r \circ \pi'$ holds and this implies $\pi' = \pi_r{}^{-1} \circ \pi_s$. Therefore, π' is uniformly random.

As for $[\![c]\!]_{\mathbb{I}}$, $\mathcal{S}$ picks $|\mathbb{I}|$ uniformly random numbers and sets them as the simulated values for $[\![c]\!]_{\mathbb{I}}$. Since $[\![c]\!]_{\mathbb{I}}$ is the output shares of Reveal$(\cdot)$, the above simulation is perfect.

Thus, $\mathcal{S}$ perfectly simulates the view of adversaries. □

Provably Secure Certificateless One-Way and Two-Party Authenticated Key Agreement Protocol

Lei Zhang

Shanghai Key Laboratory of Trustworthy Computing
Software Engineering Institute
East China Normal University, Shanghai, China
leizhang@sei.ecnu.edu.cn

Abstract. Key agreement protocols are one of the fundamental primitives in cryptography. In this paper, we formalize the security model for certificateless one-way and two-party authenticated key agreement protocols and propose a concrete certificateless one-way and two-party authenticated key agreement protocol. The security of our protocol is proven under the computational Diffie-Hellman, square computational Diffie-Hellman and gap bilinear Diffie-Hellman assumptions. As for efficiency, the protocol requires only one pass and has low communication overhead.

Keywords: key agreement, authentication, certificateless cryptography, one-way.

1 Introduction

Key agreement (KA) is one of the fundamental cryptographic primitives in cryptography. It allows two or more parties agree on a session key in such a way that both influence the outcome. One of the most famous protocol for KA was proposed by Diffie and Hellman [7]. However, the basic Diffie-Hellman protocol does not authenticate the two communication entities. Therefore, an active adversary who has control over the communication channel can mount a man-in-the-middle attack [11]. Authenticated KA [19,20] enables two or more parties to establish a shared session key over an insecure channel.

Two party KA protocols can be classified into three types [15], i.e., *non-interactive*, *one-way* and *one-round*. In a non-interactive KA protocol, no information needs to be transmitted between two entities. However, the session key generated in a non-interactive KA is derived only from long-term private keys. Hence, they cannot offer any form of forward secrecy. In a one-round KA protocol, both entities require to transmit information to each other during the protocol. It usually offers better security properties than other two types of key agreement protocols. In a one-way KA protocol, only one entity is required to transmit information to the other during the protocol. One-way KA protocols are very useful in the condition when the trade-off between security and efficiency is considered. When a message needs to be encrypted with a shared session key, they require two less message flows than the one-round protocols and at the same time provide better security properties than the non-interactive ones.

Authenticated KA protocols can be designed in different public key cryptosystems. In traditional public key cryptosystem, the management of certificates is usually a big problem. To eliminate such cost, Shamir [17] introduced identity-based public key cryptosystem (ID-PKC). In ID-PKC, the public key of an entity is just its identity (such as its IP address). However, key escrow problem comes with ID-PKC. That is the PKG (who helps an entity to generate its private key) knows the private keys of all the entities in the system. Therefore, in an identity-based one-way and two-party authenticated KA protocol [8,15], the PKG can always compute the session key. Certificateless public key cryptography (CL-PKC) may successfully solve this drawback. In CL-PKC, the KGC (who is used to help an entity to generate its private key) only has access to the partial private key of an entity. An entity's full private key is composed of the partial private key comes from the KGC and a secret information chosen by itself. Since the KGC does not hold the full private key of the entity in the system, it cannot represent any entity to do cryptographical operations without being detected.

The first certificateless authenticated KA protocol was proposed by Al-Riyami and Paterson [1]. Later, several certificateless two-party authenticated KA protocols [9,10,12,13,16,20] have been presented. Among them, the authors in [10,16,20] defined the security models for certificateless authenticated KA protocols respectively. These protocols require both entities to transmit information to the other. Therefore, they are one-round KA protocols. The first certificateless one-way and two-party authenticated KA protocol is presented in [5]. However, no formal security analysis is provided for the protocol in [5].

Our contribution: In this paper, we propose a formal security model for certificateless one-way and two-party authenticated KA protocols and propose a concrete certificateless one-way and two-party authenticated KA protocol based on bilinear maps. Our protocol captures the common security requirements of one-way and two-party authenticated AK protocols [15], i.e., *known-key security*, *unknown key-share*, *random number compromise security*, *sender's key-compromise impersonation*, *sender's forward security* and *no key control* (See Section3.1). Our protocol is efficient and has low communication cost. The security of our protocol is proven under the assumptions that the computational Diffie-Hellman, square computational Diffie-Hellman and gap bilinear Diffie-Hellman problems are hard.

Paper organization: The rest of the paper is organized as follows: Section 2 gives some preliminaries. In Section 3, we introduce the security model for certificateless one-way and two-party authenticated KA protocols. Our efficient certificateless one-way and two-party authenticated KA protocol is proposed in Section 4. In Section 5, we prove the security of our protocol. Section 6 concludes our paper.

2 Preliminaries

2.1 Bilinear Maps

Let G_1 be an additive group of prime order q and G_2 be a multiplicative group of the same order. Let P denote a generator of G_1. A map $\hat{e} : G_1 \times G_1 \longrightarrow G_2$ is called a bilinear map if it satisfies the following properties: 1) Bilinearity: $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$ for all $P, Q \in G_1, a, b \in Z_q^*$; 2) Non-degeneracy: There exists $P, Q \in G_1$ such that $\hat{e}(P, Q) \neq 1$ 3)Computability: There exists an efficient algorithm to compute $\hat{e}(P, Q)$ for any $P, Q \in G_1$.

2.2 Mathematical Problems

Here we present some mathematical problems, which are related to the security of our key agreement protocol.

Let G_1, G_2, and $\hat{e} : G_1 \times G_1 \longrightarrow G_2$ be groups and bilinear map as specified in Section 2.1.

Computational Diffie-Hellman (CDH) Problem: Given a generator P of $\mathbb{G}_1$ and (aP, bP) for unknown $a, b \in \mathbb{Z}_q^*$, compute abP.

When $a = b$, the above CDH problem degenerate to the following square computational Diffie-Hellman problem.

Square Computational Diffie-Hellman (SCDH) Problem: Given a generator P of $\mathbb{G}_1$ and aP for unknown $a \in \mathbb{Z}_q^*$, compute a^2P.

It was shown that the CDH problem and the SCDH problem are polynomial time equivalent [18].

Bilinear Diffie-Hellman (BDH) Problem: Given a randomly chosen $P \in G_1$, as well as aP, bP, cP (for random unknown $a, b, c \in Z_q^*$), compute $\hat{e}(P, P)^{abc}$.

When $a = c$, the above BDH problem degenerate to the following bilinear square Diffie-Hellman problem.

Bilinear Square Diffie-Hellman (BSDH) Problem: Given a randomly chosen $P \in G_1$, as well as aP, bP (for random unknown $a, b \in Z_q^*$), compute $\hat{e}(P, P)^{a^2b}$.

The BDH problem and the BSDH problem are proved to be polynomial time equivalent [18].

Decisional Bilinear Diffie-Hellman (DBDH) Problem: Given a randomly chosen $P \in G_1$, as well as aP, bP, cP (for random unknown $a, b, c \in Z_q^*$) and $v \in G_2$, decide whether $v = \hat{e}(P, P)^{abc}$.

When $a = c$, the above DBDH problem degenerate to the following decision bilinear square Diffie-Hellman problem.

Decisional Bilinear Square Diffie-Hellman (DBSDH) Problem: Given a randomly chosen $P \in G_1$, as well as aP, bP (for random unknown $a, b \in Z_q^*$) and $v \in G_2$, decide whether $v = \hat{e}(P, P)^{a^2b}$.

Gap Bilinear Diffie-Hellman (GBDH) Problem [2,14]: Given a randomly chosen $P \in G_1$, as well as aP, bP and cP (for random unknown $a, b, c \in Z_q^*$), compute $\hat{e}(P,P)^{abc}$ with the help of the DBDH oracle $\mathcal{D}(aP,bP,cP|v)$. If $\hat{e}(P,P)^{abc}=v$, $\mathcal{D}(aP,bP,cP|v)$ output 1; otherwise, it outputs 0, where $v \in G_2$.

The intractability of the GBDH problem means that it is hard to solve the BDH problem although one has access to a DBDH oracle. When $a=c$, the above GBDH problem degenerate to the following gap bilinear square Diffie-Hellman problem.

Gap Bilinear Square Diffie-Hellman (GBSDH) Problem: Given a randomly chosen $P \in G_1$, as well as aP, bP (for random unknown $a, b \in Z_q^*$), compute $\hat{e}(P,P)^{a^2b}$ with the help of the DBSDH oracle $\mathcal{DS}(aP,bP|v)$. If $\hat{e}(P,P)^{a^2b} = v$, $\mathcal{DS}(aP,bP|v)$ output 1; otherwise, it outputs 0, where $v \in G_2$.

The GBSDH problem is a special case of the GBDH problem. For simplicity, in this paper, the GBSDH problem is included in the GBDH problem.

2.3 Certificateless One-Way and Two-Party Authenticated Key Agreement Protocol

A certificateless one-way and two-party authenticated KA protocol is defined by following six algorithms:

- Setup: This algorithm is run by the KGC. It takes as input a security parameter k and returns a master-key and a list of system parameters params.
- Partial-Private-Key-Extract: This algorithm is run by the KGC. It takes as input an entity's identity ID_i, a parameter list params and a master-key to produce the entity's partial private key D_i.
- Set-Secret-Value: This algorithm is run by an entity. On input a parameter list params, an entity's identity ID_i, this algorithm produces the entity's secret value x_i.
- Set-Private-Key: It is run by an entity that accepts a parameter list params, the entity's identity ID_i, partial private key D_i and secret value x_i to produce a private key S_i for that entity.
- Set-Public-Key: It is run by an entity that takes as input a parameter list params, an entity's identity ID_i and secret value x_i to produce a public key P_i for the entity.
- Key-Agreement: This algorithm accepts a parameter list params, (S_A, ID_A, P_A) for sender A, (S_B, ID_B, P_B) for receiver B to produce a session key K.

3 Security Model

In this section, we define the security model for certificateless one-way and two-party authenticated KA protocols. Our model is based on the security models in [10,16,20] derived from [3,4,6].

3.1 Desirable Attributes

Our security model intends to capture the following security attributes [15]:

1. *Known-key security*: Each run of the KA protocol have to result in a unique session key. The compromise of one session key should not compromise other session keys.
2. *Unknown key-share*: An entity A must not be coerced into sharing a key with any entity C if A thinks that it is sharing the key with another entity B.
3. *Random number compromise security*: The compromise of a random input of sender A should not compromise A's private key or the established session keys.
4. *Sender's key-compromise impersonation*: If an adversary compromises the sender A's private key. He can impersonate A, but he cannot impersonate other entities in the presence of A.
5. *Sender's forward security*: If private keys of senders are compromised, the secrecy of previously established session keys should not be affected.
6. *No key control*: Neither entity should be able to force the session key to be a preselected value. In other words, for a preselected session key, none of the protocol participants can find the corresponding random input. We note that the definition of no key control in this paper is the same as that in [15].

3.2 The Model

Two types of adversaries with different capabilities are generally considered in certificateless key agreement protocols [1]. They are known as type I adversary and type II adversary:

– **Type I adversary.** This type of adversary does not have access to the master-key, but has the ability to replace the public key of any entity with a value of his choice.
– **Type II adversary.** This type of adversary has access to the master-key but cannot perform public key replacement.

In [20], the ability of type II adversary is strengthened. A type II adversary is also allowed to replace the public key of any entity except the target one. Our model is designed to capture the properties described in Section 3.1 as well as the ability of Type I and strengthened type II adversaries.

Our model is specified via the following two games between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. Both games include a set of protocol participants, each participant has a public/private key pair. These participants are modeled by oracles. We use the notation $\prod_{i,j}^{n}$, meaning a participant i believing that it is communicating with another participant j for the n-th time. $\mathcal{A}$ is either a type I or type II adversary. $\mathcal{A}$ is modeled by a probabilistic polynomial time Turing Machine and has access to all the oracles in the game. $\mathcal{A}$ can relay, modify, delay, interleave and delete messages. All communications go through $\mathcal{A}$. Participant oracles only respond to queries by $\mathcal{A}$ and do not communicate directly amongst themselves. We call $\mathcal{A}$ is *benign*, if he is deterministic and restricts its action to choosing a pair of oracles $\prod_{i,j}^{n}$ and $\prod_{j,i}^{t}$ and then faithfully conveying each flow from one oracle to the other.

Before defining the games, we first define the following oracles:

– Create: The input of this oracle is an identity ID_i of a participant i. On receiving such a oracle query, $\mathcal{C}$ generates the public and private keys for this participant.
– Public-Key: The input of this oracle is an identity ID_i of a participant i. The output of this oracle is the public key P_i of i.
– Partial-Private-Key: The input of this oracle is an identity ID_i of a participant i. The output of this oracle is the partial private key D_i of i.
– Secret-Value: The input of this oracle is an identity ID_i of a participant i. The output of this oracle is the secret value x_i of i.
– Corrupt: The input of this oracle is an identity ID_i of a participant i. The output of this oracle is the private key S_i of i.
– Public-Key-Replacement: The input of this oracle is (ID_i, P_i'). On receiving such a oracle query, $\mathcal{C}$ sets P_i' as the new public key of the participant i. $\mathcal{C}$ will record this replacement which will be used later.
– Send: The input of this oracle $(\prod_{i,j}^{n}, M)$. In this case, participant i assumes the message M has been sent by participant j. $\mathcal{A}$ may also make a special Send query λ to an oracle $\prod_{i,j}^{n}$, which instructs i to initiate a protocol run with j. An oracle is an *initiator* oracle if the first message it has received is λ. If an oracle does not receive a message λ as its first message, then it is a *responder* oracle.
– Session-Key-Reveal: On receiving the Session-Key-Reveal query on $\prod_{i,j}^{n}$, this oracle outputs the session key held by $\prod_{i,j}^{n}$.
– Random-Number-Reveal: On receiving the Random-Number-Reveal query on $\prod_{i,j}^{n}$, this oracle outputs the random number held by $\prod_{i,j}^{n}$. Since, only the sender will choose a random number, we require that $\prod_{i,j}^{n}$ is an *initiator* oracle.

An oracle $\prod_{i,j}^{n}$ exists in one of the following several possible states:

– *Accepted*: An oracle is in an accepted state, if it decides to accept, holding a session key, after receipt of properly formulated messages.
– *Rejected*: An oracle is in a rejected state, if it decides not to establish a session key and to abort the protocol.
– *State **: An oracle is in state *, if it has not made any decision to accept or reject.
– *Opened*: An oracle is in an opened state, if it has answered a session key reveal query.
– *Corrupted*: An oracle is in a corrupted state, if it has answered a corrupt query.

Definition 1 (Matching conversation). *Let the session ID be the concatenation of the identities and public keys of the protocol participants, and, the other messages transmitted in a session. Two oracles $\prod_{i,j}^{n}$ and $\prod_{j,i}^{t}$ are said to have a matching conversation with each other if they have the same session ID.*

Game 1

At the beginning of this game, $\mathcal{C}$ runs the Setup algorithm to obtain the master-key and the system parameter list params. If $\mathcal{A}$ is a Type I adversary, $\mathcal{C}$ sends params to $\mathcal{A}$ and keeps the master-key secret; otherwise, $\mathcal{A}$ is a Type II adversary, $\mathcal{C}$ sends params with master-key to $\mathcal{A}$.

$\mathcal{A}$ is allowed to access Create, Public-Key, Partial-Private-Key, Secret-Value, Corrupt, Public-Key-Replacement, Send, Session-Key-Reveal and Random-Number-Reveal oracles.

Test: At some point in $\mathcal{A}$'s attack, he may choose one of the oracles, say $\prod_{i,j}^{n}$, to ask a single Test query. This oracle must be fresh (See Definition 2). To answer the query, the oracle flips a fair coin $\theta \in \{0, 1\}$, and returns the session key held by $\prod_{i,j}^{n}$ if $\theta = 0$, or else a random session key if $\theta = 1$.

Definition 2 (Freshness). *An oracle $\prod_{i,j}^{n}$ is fresh if (1) $\prod_{i,j}^{n}$ is in the state Accepted; (2) $\prod_{i,j}^{n}$ is not in the state Opened; (3) party i and j are not corrupted; (4) there is no oracle $\prod_{j,i}^{t}$ in the state Opened having a matching conversation with $\prod_{i,j}^{n}$; (5) if $\mathcal{A}$ is a Type I adversary, $\mathcal{A}$ has never requested the partial private key of participants i and j; if $\mathcal{A}$ is a Type II adversary, $\mathcal{A}$ has never replaced the public key of participants i and j, and has never requested the secret value of participants i and j.*

After the Test query, $\mathcal{A}$ can continue querying the oracles except that he cannot reveal the test oracle $\prod_{i,j}^{n}$ or $\prod_{j,i}^{t}$ which is matching conversation with $\prod_{i,j}^{n}$ (if it exists); and he cannot corrupt party i and j; and if $\mathcal{A}$ is a Type I adversary, he cannot request the partial private key of participants i and j; and if $\mathcal{A}$ is a Type II adversary, he cannot replace the public key of participants i and j, and cannot request the secret value of participants i and j. At the end of $\mathcal{A}$'s attack, he must output a bit θ' as his guess for θ. $\mathcal{A}$'s advantage, denoted ϵ, is defined as:

$$\epsilon = |\Pr[\theta' = \theta] - 1/2|$$

is the probability that $\mathcal{A}$ can distinguish the session key held by the tested oracle from a random string.

Definition 3. *A protocol is a secure certificateless one-way and two-party authenticated AK protocol if:*

1. *In the presence of the benign adversary on $\prod_{i,j}^{n}$ and $\prod_{j,i}^{t}$, both oracles always accept holding the same session key, and this key is distributed uniformly at random;*

2. *For any adversary $\mathcal{A}$ whether of type I or type II, ϵ is negligible in the above game.*

The above model captures the known-key security, unknown key-share and random number compromise security properties of certificateless one-way and two-party authenticated KA protocols. However, it does not allow $\mathcal{A}$ to get the private key of the sender. Therefore, it does not capture the sender's key-compromise impersonation and sender's forward security properties.

Game 2

To capture the sender's key-compromise impersonation and sender's forward security properties, we define the second game. In this game, $\mathcal{A}$ is allowed to access Create, Public-Key, Partial-Private-Key, Secret-Value, Corrupt, Public-Key-Replacement, Send, Session-Key-Reveal and Random-Number-Reveal oracles.

Test: At some point in his attack, $\mathcal{A}$ may choose one of the oracles, say $\prod_{i,j}^{n}$, to ask a single Test query. It requires that $\prod_{i,j}^{n}$ is an *initiator*. Further, this oracle must be fresh (See Definition 4). To answer the query, the oracle flips a fair coin $\theta \in \{0, 1\}$, and returns the session key held by $\prod_{i,j}^{n}$ if $\theta = 0$, or else a random key sampled from $\prod_{i,j}^{n}$ if $\theta = 1$.

Definition 4 (Freshness). *An oracle $\prod_{i,j}^{n}$ is fresh if (1) $\prod_{i,j}^{n}$ is in the state Accepted; (2) $\prod_{i,j}^{n}$ is not in the state Opened; (3) party j is not corrupted; (4) there is no oracle $\prod_{j,i}^{t}$ in the state Opened having a matching conversation with $\prod_{i,j}^{n}$; (5) if $\mathcal{A}$ is a Type I adversary, $\mathcal{A}$ has never requested the partial private key of participant j; if $\mathcal{A}$ is a Type II adversary, $\mathcal{A}$ has never replaced the public key of participant j, and has never requested the secret value of participant j; (6) $\prod_{i,j}^{n}$ has not answered a random number reveal query.*

After the Test query the adversary can continue querying the oracles except that he cannot reveal the test oracle $\prod_{i,j}^{n}$ or its partner $\prod_{j,i}^{t}$ (if it exists); and he cannot corrupt party j; and if $\mathcal{A}$ is a Type I adversary, he cannot request the partial private key of participant j; if $\mathcal{A}$ is a Type II adversary, he cannot replace the public key of participant j, and cannot request the secret value of participant j; and he cannot request the Random-Number-Reveal query on $\prod_{i,j}^{n}$. At the end of $\mathcal{A}$'s attack, he must output a bit θ' as his guess for θ. $\mathcal{A}$'s advantage, denoted ϵ, is defined as:

$$\epsilon = |\Pr[\theta' = \theta] - 1/2|$$

is the probability that $\mathcal{A}$ can distinguish the session key held by the tested oracle from a random string.

Definition 5. *A certificateless one-way and two-party authenticated KA protocol holds sender's key-compromise impersonation and sender's forward security if:*

1. *In the presence of the benign adversary on $\prod_{i,j}^{n}$ and $\prod_{j,i}^{t}$, both oracles always accept holding the same session key, and this key is distributed uniformly at random;*
2. *ϵ is negligible for any adversary $\mathcal{A}$ whether of type I or type II in Game 2.*

4 Our Protocol

In this section, we propose our concrete protocol. It comes as follows:

- Setup: On input a security parameter k, the KGC selects a cyclic additive group G_1 with prime order q, a cyclic multiplicative group G_2 of the same order, a generator $P \in G_1$ and a bilinear map $\hat{e} : G_1 \times G_1 \longrightarrow G_2$; chooses a random master-key $s \in Z_q^*$ and set $P_0 = sP$; chooses cryptographic hash functions $H_1 : \{0,1\}^* \longrightarrow G_1$, $H_2 : \{0,1\}^* \longrightarrow \{0,1\}^l$. The system parameters params=$(q, G_1, G_2, \hat{e}, P, P_0, H_1, H_2)$.
- Partial-Private-Key-Extract: This algorithm accepts params, an entity's identity ID_i and generates the partial private key for the entity as follows:
  1. Compute $Q_i = H(ID_i)$.
  2. Output the partial private key $D_i = sQ_i$.
- Set-Secret-Value: This algorithm accepts params and an entity's identity ID_i, and selects a random $x_i \in Z_q^*$. It outputs x_i as the entity's secret value.
- Set-Private-Key: This algorithm takes as input params, an entity's partial private key D_i and secret value x_i. The output of the algorithm is the private key $S_i = (x_i, D_i)$.
- Set-Public-Key: This algorithm accepts params and an entity's secret value $x_i \in Z_q^*$ to produce the entity's public key $P_i = x_iP$.
- Key-Agreement: Assume a sender A with identity ID_A, private key $S_A = (x_A, D_A)$ and public key $P_A = x_AP$, and, a receiver B with identity ID_B, private key $S_B = (x_B, D_B)$ and public key $P_B = x_BP$ want to establish a session key. A picks a random $r \in Z_q^*$, computes $U = rP$, and sends (ID_A, P_A, U) to B. A and B can establish their session key as follows:
  A computes:
  $$K_{AB} = H_2(ID_A, ID_B, P_A, P_B, U, rP_B, x_AP_B, \hat{e}(D_A, Q_B), \hat{e}(rP_0, Q_B)).$$
  B computes:
  $$K_{BA} = H_2(ID_A, ID_B, P_A, P_B, U, x_BU, x_BP_A, \hat{e}(Q_A, D_B), \hat{e}(U, D_B)).$$
  The session key is $K = K_{AB} = K_{BA}$.

5 Security Analysis

In this section, we prove that our protocol captures the security attributes in Section 3.1.

Lemma 1. *In the presence of the benign adversary on $\prod_{i,j}^n$ and $\prod_{j,i}^t$, both oracles always agree on the same session key, and this key is distributed uniformly at random.*

Proof. Suppose that the two oracles follow the protocol and the adversary is benign. In this case, since $K_{AB} = K_{BA}$ by the bilinearity of the bilinear map, both oracles agree on the same session key. Since r is random, based on the properties of cryptographic hash functions, the session key is uniformly distributed over $\{0,1\}^l$.

Lemma 2. *Our protocol is a secure certificateless ony-way and two-party authenticated KA protocol against type I adversary in the random oracle model assuming the GBDH problem is intractable. Specifically, suppose in the attack, a type I adversary $\mathcal{A}$ who makes at most q_{H_1} times H_1 queries, q_{H_2} times H_2 queries, q_c times* Corrupt *queries, q_{sr} times* Session-Key-Reveal *queries, wins the game with advantage ϵ. Then there exists an algorithm $\mathcal{C}$ to solve the GBDH problem with advantage $\epsilon' \geqslant \frac{1}{q_{H_2}e^3}(\frac{3}{q_c+q_{sr}+3})^3\epsilon$.*

Proof. See Appendix A.

Lemma 3. *Our protocol is a secure certificateless ony-way and two-party authenticated KA protocol against type II adversary in the random oracle model assuming the SCDH problem is intractable. Specifically, suppose in the attack, a type II adversary $\mathcal{A}$ who makes at most q_{H_2} times H_2 queries, q_c times* Corrupt *queries, q_{sr} times* Session-Key-Reveal *queries, wins the game with advantage ϵ. Then there exists an algorithm $\mathcal{C}$ to solve the SCDH problem with advantage $\epsilon' \geqslant \frac{1}{q_{H_2}e^3}(\frac{3}{q_c+q_{sr}+3})^3\epsilon$.*

Proof. Due to the page limitation, it will be presented in this full version of this paper.

Theorem 1. *Our protocol is a secure certificateless ony-way and two-party authenticated KA protocol.*

Proof. The theorem follows directly from Lemma 1, 2 and 3.

Lemma 4. *Our protocol has sender's key-compromise impersonation and sender's forward security against type I adversary in the random oracle model provided the GBDH problem is intractable. Specifically, suppose in the attack, an adversary $\mathcal{A}$ who makes at most q_{H_1} times H_1 queries, q_{H_2} times H_2 queries, q_c times* Corrupt *queries, q_{sr} times* Session-Key-Reveal *queries, q_{rr} times* Random-Number-Reveal *queries, wins the game with advantage ϵ. Then there exists an algorithm $\mathcal{C}$ to solve the GBDH problem with advantage $\epsilon' \geqslant \frac{1}{2q_{H_2}e^2}(\frac{2}{q_c+q_{sr}+q_{rr}+3})^2\epsilon$.*

Proof. It will be presented in this full version of this paper.

Lemma 5. *Our protocol has sender's key-compromise impersonation and sender's forward security against type II adversary in the random oracle model provided the CDH problem is intractable. Specifically, suppose in the attack, an adversary $\mathcal{A}$ who makes at most q_{H_2} times H_2 queries, q_c times* Corrupt *queries, q_{sr} times* Session-Key-Reveal *queries, q_{rr} times* Random-Number-Reveal *queries, wins the game with advantage ϵ. Then there exists an algorithm $\mathcal{C}$ to solve the CDH problem with advantage $\epsilon' \geqslant \frac{1}{2q_{H_2}e^3}(\frac{2}{q_c+q_{sr}+q_{rr}+2})^2\epsilon$.*

Proof. It will be presented in this full version of this paper.

Theorem 2. *Our protocol has sender's key-compromise impersonation and sender's forward security.*

Proof. The theorem follows directly from Lemma 1, 4 and 5.

Theorem 3. *Our protocol captures no key control.*

Proof. Since the value r is selected by A, it is easy to see that B cannot control the session key. A cannot do this either comes from the fact that for a predetermined session key K to find r such that $H_2(ID_A, ID_B, P_A, P_B, U, rP_B, x_AP_B, \hat{e}(D_A, Q_B), \hat{e}(rP_0, Q_B)) = K$ is computationally impossible.

6 Conclusion

One-way and two-party authenticated KA protocols are important tools in certificateless cryptography. In this paper, we have presented an efficient certificateless one-way and two-party authenticated KA protocol. To generate a session key, our protocol only requires one pass and has low communication overhead. The security of our protocol is based on the hardness of the CDH, SCDH and GBDH problems.

Acknowledgments and Disclaimer. This work was supported in part by the the NSF of China under Grants 61202465, 61021004, 11061130539, 61103222; the Shanghai NSF under Grant No. 12ZR1443500, 11ZR1411200; the Shanghai Chen Guang Program (12CG24); the Open Project of Shanghai Key Laboratory of Trustworthy Computing (No. 07dz22304201101); the Fundamental Research Funds for the Central Universities; EU FP7 under Projects "DwB" and "Inter-Trust"; the Spanish Government under Projects CTV-09-634, PTA2009-2738-E, TSI-020302-2010-153, TIN2009-11689, TIN2011-27076-C03-01,CONSOLIDER INGENIO 2010 "ARES" CSD2007-0004, and TSI2007-65406-C03-01.

References

1. Al-Riyami, S.S., Paterson, K.G.: Certificateless public key cryptography. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 452–473. Springer, Heidelberg (2003)
2. Baek, J., Safavi-Naini, R., Susilo, W.: Efficient multi-receiver identity-based encryption and its application to broadcast encryption. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 380–397. Springer, Heidelberg (2005)
3. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994)
4. Blake-Wilson, S., Johason, D., Menezes, A.: Key agreement protocols and their security analysis. In: Darnell, M.J. (ed.) Cryptography and Coding 1997. LNCS, vol. 1355, pp. 30–45. Springer, Heidelberg (1997)

5. Chen, W., Zhang, L., Qin, B., Wu, Q., Zhang, H.: Certificateless one-way authenticated two-party key agreement protocol. In: IEEE IAS 2009, pp. 483–486 (2009)
6. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001)
7. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory 22(6), 644–654 (1976)
8. Gorantla, M., Boyd, C., Nieto, J.: ID-based one-pass authenticated key establishment. In: Sixth Australasian Conference on Information Security, vol. 81, pp. 39–46 (2008)
9. Li, X., Zhang, Y., Zhang, G.: A new certificateless authenticated key agreement protocol for SIP with different KGCs, Security and Communication Networks, doi:10.1002/sec.595
10. Lippold, G., Boyd, C., Nieto, J.G.: Strongly secure certificateless key agreement. In: Shacham, H., Waters, B. (eds.) Pairing 2009. LNCS, vol. 5671, pp. 206–230. Springer, Heidelberg (2009)
11. Lowe, G.: An attack on the Needham-Schroeder public-key authentication protocol. Information Processing Letters 56(3), 131–133 (1995)
12. Luo, M., Wen, Y., Zhao, H.: An enhanced authentication and key agreement mechanism for SIP using certificateless public-key cryptography. In: IEEE ICYCS 2008, pp. 1577–1582 (2008)
13. Mandt, T.K., Tan, C.H.: Certificateless authenticated two-party key agreement protocols. In: Okada, M., Satoh, I. (eds.) ASIAN 2006. LNCS, vol. 4435, pp. 37–44. Springer, Heidelberg (2008)
14. Okamoto, T., Pointcheval, D.: The gap-problems: a new class of problems for the security of cryptographic schemes. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 104–118. Springer, Heidelberg (2001)
15. Okamoto, T., Tso, R., Okamoto, E.: One-way and two-party authenticated ID-based key agreement protocols using pairing. In: Torra, V., Narukawa, Y., Miyamoto, S. (eds.) MDAI 2005. LNCS (LNAI), vol. 3558, pp. 122–133. Springer, Heidelberg (2005)
16. Swanson, C., Jao, D.: A Study of two-party certificateless authenticated key-agreement protocols. In: Roy, B., Sendrier, N. (eds.) INDOCRYPT 2009. LNCS, vol. 5922, pp. 57–71. Springer, Heidelberg (2009)
17. Shamir, A.: Identity based cryptosystems and signature schemes. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985)
18. Zhang, F., Safavi-Naini, R., Susilo, W.: An efficient signature scheme from bilinear pairings and its applications. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 277–290. Springer, Heidelberg (2004), Full version: `http://www.uow.edu.au/~wsusilo/PKC04.pdf`
19. Zhang, L., Wu, Q., Qin, B., Domingo-Ferrer, J.: Provably secure one-round identity-based authenticated asymmetric group key agreement protocol. Information Sciences 181(19), 4318–4329 (2011)
20. Zhang, L., Zhang, F., Wu, Q., Domingo-Ferrer, J.: Simulatable certificateless two-party authenticated key agreement protocol. Information Sciences 180(6), 1020–1030 (2010)

A Proof of Lemma 2

Proof. Suppose $\mathcal{C}$ is given an arbitrary input (P, aP, bP, cP), where $a = c$. We show how can $\mathcal{C}$ use $\mathcal{A}$ to solve GBDH problem, i.e. to compute $\hat{e}(P,P)^{abc}$ with the help of the DBDH oracle. All queries by the adversary $\mathcal{A}$ now pass through $\mathcal{C}$. Firstly, $\mathcal{C}$ sets $P_0 = bP$, then selects the system parameter $\mathsf{params}{=}(q, G_1, G_2, \hat{e}, P, P_0, H_1, H_2)$, and gives params to $\mathcal{A}$.

H_1 queries: $\mathcal{C}$ maintains a list $\mathsf{H_1list}$ which is initially empty. On input ID_i, $\mathcal{C}$ first checks whether this query has been queried. If so, $\mathcal{C}$ finds the tuple $(coin_i, ID_i, \pi_i, Q_i)$ on $\mathsf{H_1list}$ and returns Q_i as the answer; otherwise, he flips a coin $coin_i \in \{0,1\}$ that yields 1 with probability δ and 0 with probability $1-\delta$. If $coin_i = 1$, $\mathcal{C}$ selects $\pi_i \in Z_q^*$ at random, sets $Q_i = \pi_i aP$, adds $(coin_i, ID_i, \pi_i, Q_i)$ to $\mathsf{H_1list}$ and returns Q_i as the answer; else, selects $\pi_i \in Z_q^*$ at random, sets $Q_i = \pi_i P$, adds $(coin_i, ID_i, \pi_i, Q_i)$ to $\mathsf{H_1list}$ and returns Q_i as the answer.

H_2 queries: $\mathcal{C}$ maintains a list $\mathsf{H_2list}$ which is initially empty. On input $(ID_A^i, ID_B^i, P_A^i, P_B^i, U_i, X_i, Y_i, u_i, v_i)$, $\mathcal{C}$ checks whether this query has been queried. If so, $\mathcal{C}$ finds the tuple $(ID_A^i, ID_B^i, P_A^i, P_B^i, U_i, X_i, Y_i, u_i, v_i, h_i)$ on $\mathsf{H_2list}$ and returns h_i as the answer; otherwise, $\mathcal{C}$ does the following:

- If there's a tuple $(coin_i', n, ID_I, ID_J, P_I, P_J, U_{i,j}, r_{i,j}, K_{i,j})$ on SList (See Send queries) such that $ID_I = ID_A^i, ID_J = ID_B^i, P_I = P_A^i, P_J = P_B^i, U_i = U_{i,j}, \hat{e}(X_i, P) = \hat{e}(U_{i,j}, P_J), \hat{e}(Y_i, P) = \hat{e}(P_I, P_J)$, $\mathcal{D}(P_0, H_1(ID_I), H_1(ID_J)|u_i) = 1, \mathcal{D}(U_{i,j}, P_0, H_1(ID_J)|v_i) = 1, K_{i,j} \neq \perp$, set $h_i = K_{i,j}$, add $(ID_A^i, ID_B^i, P_A^i, P_B^i, U_i, X_i, Y_i, u_i, v_i, h_i)$ to $\mathsf{H_2list}$ and return h_i as the answer.
- Else, set h_i to be a random value in $\{0,1\}^l$, add $(ID_A^i, ID_B^i, P_A^i, P_B^i, U_i, X_i, Y_i, u_i, v_i, h_i)$ to $\mathsf{H_2list}$ and return h_i as the answer.

Create queries: $\mathcal{C}$ maintains an initially empty list CList. On input an identity ID_i, if ID_i has been submitted previously, $\mathcal{C}$ does nothing; otherwise, $\mathcal{C}$ chooses a random $x_i \in Z_q^*$, computes the public key $P_i = x_i P$, submits ID_i to H_1 and recovers the tuple $(coin_i, ID_i, \pi_i, Q_i)$ from $\mathsf{H_1list}$. If $coin_i = 0$, $\mathcal{C}$ computes $D_i = \pi_i P_0$, adds (ID_i, x_i, D_i, P_i) to CList; else, sets $D_i = \perp$, adds (ID_i, x_i, D_i, P_i) to CList.

$\mathsf{Public\text{-}Key}$ queries: On input an identity ID_i, $\mathcal{C}$ first submits ID_i to the Create oracle, then recovers the tuple (ID_i, x_i, D_i, P_i) from CList and then returns P_i as the answer.

$\mathsf{Secret\text{-}Value}$ queries: On input an identity ID_i, $\mathcal{C}$ first submits ID_i to the Create oracle and then recovers the tuple (ID_i, x_i, D_i, P_i) from CList. If $P_i \neq x_i P$, $\mathcal{C}$ returns $\perp$; else he returns x_i as the answer. Note that if $\mathcal{A}$ has made a $\mathsf{Public\text{-}Key\text{-}Replacement}$ query on ID_i, then $P_i \neq x_i P$.

$\mathsf{Corrupt}$ queries: On input ID_i, $\mathcal{C}$ submits ID_i to the Create oracle and recovers (ID_i, x_i, D_i, P_i) from CList. If $D_i = \perp$, $\mathcal{C}$ aborts (Event 1); else if $P_i \neq x_i P$, returns $(\perp, D_i)$ as the answer; else returns (x_i, D_i) as the answer.

$\mathsf{Partial\text{-}Private\text{-}Key}$ queries: On input an identity ID_i, $\mathcal{C}$ submits ID_i to the $\mathsf{Corrupt}$ oracle. If $\mathcal{C}$ does not abort, he returns D_i as the answer.

Public-Key-Replacement queries: On input (ID_i, P'_i), $\mathcal{C}$ submits ID_i to the Create oracle, then recovers (ID_i, x_i, D_i, P_i) from CList, and then sets $P_i = P'_i$.

Send queries: $\mathcal{C}$ keeps a list SList which is initially empty. One receiving a Send query $\prod_{i,j}^n(M)$, $\mathcal{C}$ submits ID_i and ID_j to the Create oracle, recovers the tuples (ID_i, x_i, D_i, P_i) and (ID_j, x_j, D_j, P_j) from CList, flips a coin $coin'_i \in \{0,1\}$ that yields 1 with probability δ and 0 with probability $1-\delta$. If $M = \lambda$, $\mathcal{C}$ randomly chooses $r_{i,j} \in Z_q^*$, computes $U_{i,j} = r_{i,j}P$, sets $K_{i,j} = \perp, ID_I = ID_i, ID_J = ID_j, P_I = P_i, P_J = P_j$, adds $(coin'_i, n, ID_I, ID_J, P_I, P_J, U_{i,j}, r_{i,j}, K_{i,j})$ to SList and returns $U_{i,j}$ as the answer. Else, $\mathcal{C}$ sets $r_{i,j} = \perp, U_{i,j} = M, K_{i,j} = \perp, ID_I = ID_j, ID_J = ID_i, P_I = P_j, P_J = P_i$, adds $(coin'_i, n, ID_I, ID_J, P_I, P_J, U_{i,j}, r_{i,j}, K_{i,j})$ to SList and returns $U_{i,j}$ as the answer.

Session-Key-Reveal queries: On receiving a Session-Key-Reveal query on $\prod_{i,j}^n$, $\mathcal{C}$ finds the tuple $(coin'_i, n, ID_I, ID_J, P_I, P_J, U_{i,j}, r_{i,j}, K_{i,j})$ on SList, then does the following:

– If $coin'_i = 1$, abort (Event 2).
– Else if $K_{i,j} \neq \perp$, return $K_{i,j}$ as the answer.
– Else if there's a tuple $(ID_A^i, ID_B^i, P_A^i, P_B^i, U_i, X_i, Y_i, u_i, v_i, h_i)$ on H$_2$list such that $ID_I = ID_A^i, ID_J = ID_B^i, P_I = P_A^i, P_J = P_B^i, U_i = U_{i,j}, \hat{e}(X_i, P) = \hat{e}(U_{i,j}, P_J), \hat{e}(Y_i, P) = \hat{e}(P_I, P_J)$, $\mathcal{D}(P_0, H_1(ID_I), H_1(ID_J)|u_i) = 1$, $\mathcal{D}(U_{i,j}, P_0, H_1(ID_J)|v_i) = 1$, set $K_{i,j} = h_i$ and return $K_{i,j}$ as the answer.
– Else, set $K_{i,j}$ to be a random value in $\{0,1\}^l$ and return $K_{i,j}$ as the answer.

Random-Number-Reveal queries: On receiving a Random-Number-Reveal query on $\prod_{i,j}^n$, $\mathcal{C}$ finds the tuple $(coin'_i, n, ID_I, ID_J, P_I, P_J, U_{i,j}, r_{i,j}, K_{i,j})$ on SList and returns $r_{i,j}$ as the answer. As denoted in Section 3.2, it requires that $\prod_{i,j}^n$ is an *initiator* oracle.

Test query: At some point in the simulation, $\mathcal{A}$ asks a Test query on $\prod_{i,j}^n$. $\mathcal{C}$ recovers the tuple $(coin'_i, n, ID_I, ID_J, P_I, P_J, U_{i,j}, r_{i,j}, K_{i,j})$ from SList, submits ID_i and ID_j to H_1, finds the tuples $(coin_i, ID_i, \pi_i, Q_i)$ and $(coin_j, ID_j, \pi_j, Q_j)$ on H$_1$list. If $coin_i \neq 1$ or $coin_j \neq 1$ or $coin' \neq 1$, $\mathcal{C}$ aborts (Event 3). Otherwise, $\mathcal{C}$ simply outputs a random value $h \in \{0,1\}^l$.

Once $\mathcal{A}$ finishes his queries and returns his guess bit, $\mathcal{C}$ randomly chooses u_ρ from H$_2$list and returns $(u_\rho)^{(\pi_i\pi_j)^{-1}}$ as the response to the GBDH challenge.

In the above simulation, all the responses of the oracles are uniformly distributed in the message space. Hence, if $\mathcal{C}$ does not abort, $\mathcal{A}$ cannot find any inconsistency between the simulation and the real world. Therefore, $\mathcal{A}$ can win the game with probability $\Pr[\theta = \theta'] = \epsilon$. It remains to determine the probability that $\mathcal{C}$ outputs the required u_ρ.

In our simulation, $\mathcal{C}$ will abort if Event 1 or Event 2 or Event 3 happens. We must calculate $\Pr[\neg\texttt{Event } 1 \wedge \neg\texttt{Event } 2 \wedge \neg\texttt{Event } 3]$. By our setting, it is easy to get

$$\Pr[\neg\texttt{Event } 1 \wedge \neg\texttt{Event } 2 \wedge \neg\texttt{Event } 3] \geqslant \frac{1}{e^3}\left(\frac{3}{q_c + q_{sr} + 3}\right)^3$$

It is now easy to see that $\mathcal{C}$ solves the GBDH problem with probability $\epsilon' \geqslant \frac{1}{q_{H_2}e^3}\left(\frac{3}{q_c+q_{sr}+3}\right)^3\epsilon$. This concludes the proof.

A CCA-Secure Identity-Based Conditional Proxy Re-Encryption without Random Oracles

Kaitai Liang[1], Zhen Liu[1,2], Xiao Tan[1],
Duncan S. Wong[1], and Chunming Tang[3]

[1] Department of Computer Science, City University of Hong Kong, China
{kliang4,zhenliu7,xiaotan4}@student.cityu.edu.hk, duncan@cityu.edu.hk
[2] Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
liuzhen@sjtu.edu.cn
[3] School of Mathematics and Information Science, Guangzhou University, China
ctang@gzhu.edu.cn

Abstract. Although a few unidirectional single-hop Identity-Based Proxy Re-Encryption (IBPRE) systems are available in the literature, none of them is CCA secure in the standard model. Besides, they can not support *conditional re-encryption property*, which allows a delegator to specify a condition for ciphertexts so that the proxy can re-encrypt ciphertexts only if the re-encryption key corresponding to the same condition is given. This paper, for the first time, proposes a new unidirectional single-hop Identity-Based Conditional Proxy Re-Encryption (IBCPRE) scheme that not only captures the property of IBPRE (i.e. identity-based re-encryption), but also supports conditional re-encryption. Moreover, the scheme can be proved secure against adaptive condition and adaptive identity chosen-ciphertext attacks in the standard model.

Keywords: Unidirectional Conditional Proxy Re-encryption, Identity-Based Encryption, Single Hop, Standard Model.

1 Introduction

First defined by Blaze, Bleumer and Strauss [3], Proxy Re-Encryption (PRE) extends the traditional Public Key Encryption (PKE) to support the decryption rights delegation, in which a semi-trusted *proxy* is allowed to transform a ciphertext under Alice's public key into a ciphertext under Bob's public key using a re-encryption key given by Alice. The proxy, however, learns nothing about the plaintext. If ciphertexts can be transformed from Alice to Bob and to Carol, and so on, then the scheme is *multi-hop.* If ciphertexts can be transformed to Bob only, then the scheme is a *single-hop* PRE. PRE can be further categorized into *bidirectional* PRE and *unidirectional* PRE. In bidirectional PRE, a re-encryption key allows ciphertexts to be transformed from Alice to Bob and vise versa. In the unidirectional setting, a re-encryption key only allows ciphertexts to be transformed from Alice to Bob or from Bob to Alice.

PRE can offer practical solutions to many network applications where the delegation of decryption rights is required, such as secure e-mail forwarding, secure files systems [1] and cloud storage systems. In many cloud storage systems, to protect the confidentiality of the data, a system user (say Alice) often encrypts her data with a content key before uploading to the cloud server. When sharing the data with multiple system users, Alice may directly deliver the content key to the users or alternatively, she may first encrypt the content key under each user's public key and then upload the ciphertexts to the cloud so that the users can recover the content key using their respective secret key. The inconvenience with the two strategies above is that Alice has to be responsible for either the delivery of the content key or the generation of the ciphertexts of the content key (which are intended for multiple users), and meanwhile, Alice has to be on-line.

Using traditional PRE, Alice is able to utilize the proxy's (i.e. the cloud's) abundant computational power to re-encrypt the ciphertext of the content key so that her encryption workload is lessened. Meanwhile, PRE allows Alice to share the data when she is off-line. Specifically, Alice encrypts the content key using her public key and uploads the ciphertext to the cloud. And before being off-line, Alice (i.e. the delegator) first specifies the delegatees, next generates and sends the re-encryption keys to the cloud server (i.e. the proxy). The cloud server then uses the re-encryption keys to re-encrypt the ciphertext of Alice's content key and forwards the resulting ciphertexts to the delegatees, so that the delegatees can access the data using the content key. Indeed, the server is kept from knowing either the content key or the content (of the data).

To employ traditional PRE in the identity-based cryptographic setting, Green and Ateniese [13] proposed the first Identity-Based Proxy Re-Encryption (IBPRE), which allows the proxy to transform an encryption under Alice's identity (e.g., email address) to a new ciphertext computed under Bob's identity. This is similar to identity-based encryption (IBE) but a major difference is that IBPRE in addition supports the delegation of decryption rights, i.e. allowing *identity-based re-encryption*. This paper deals with the case of unidirectional single-hop IBPRE. Despite there are some unidirectional single-hop IBPRE schemes in the literature, how to construct one that is secure against *chosen-ciphertext attacks* (CCA) in the standard model still remains open. In this paper, we focus on such an open problem.

A problem incurred by employing either traditional PRE or IBPRE in cloud storage systems is that the re-encryption power of the cloud server cannot be controlled. More specifically, the server can re-encrypt *all* ciphertexts of Alice's content keys to Bob as long as the corresponding re-encryption key is given. It might be a potential risk for access control as Alice might want to share the data labeled "Monday" but not the one tagged with "Thursday" with Bob.

To solve the problem above in the PRE setting, Conditional Proxy Re-Encryption (CPRE) (e.g., [20,25,26]) is proposed. A CPRE is a type of PRE providing *conditional re-encryption* to capture a fine-grained control over the delegation. That is, Alice is allowed to specify a condition for a ciphertext so that the cloud server can re-encrypt the ciphertext to Bob only if the re-encryption key corresponding to the same condition is given. However, as far as we know, there is no solution for IBPRE to capture conditional re-encryption property in the standard model. This paper also focuses on filling such a gap.

1.1 Our Contributions

In this paper we first formalize the definition and security models for IBCPRE. Specifically, in our definition a condition is required as an auxiliary input to the re-encryption key, encryption and decryption algorithms. Regarding the security models, we first define the *adaptive condition and adaptive identity chosen-ciphertext security* (IND-aCon-aID-CCA) at original ciphertext. We next define the IND-aCon-aID-CCA security at re-encrypted ciphertext, which can be regarded as a weaker notion when compared with the one defined in [15].

There are two open problems in the literature of IBPRE: one is how to construct a CCA-secure unidirectional single-hop IBPRE without random oracles, and the other is how to extend IBPRE to support conditional re-encryption. This paper, for the first time, answers the problems affirmatively by proposing a new unidirectional single-hop IBPRE scheme with conditional re-encryption (i.e. IBCPRE). Moreover, the new scheme can be proved IND-aCon-aID-CCA secure in the standard model. Besides, our scheme also captures collusion resistance (that is, the proxy cannot compromise the entire secret key of the delegator even if the proxy colludes with the corresponding delegatee).

Here we further describe the difficulty of constructing a CCA-secure unidirectional single-hop IBPRE scheme in the standard model. The construction is not trivial even if an IBE scheme (e.g., [5]) is given as a building block. In Green and Ateniese's IBPRE [13], an eligible decryptor who has either the secret key sk_{id} of the delegator or the secret key $sk_{id'}$ of the delegatee can recover σ and the plaintext m so that he/she can verify the validity of the decryption of the original ciphertext (resp. re-encrypted ciphertext) by checking $A \stackrel{?}{=} g^{H_4(\sigma,m)}$ and $D \stackrel{?}{=} H_3(id||\langle A,B,C\rangle)^{H_4(\sigma,m)}$ (resp. $A \stackrel{?}{=} g^{H_4(\sigma,m)}$) (for more details, the reader is referred to [13]). Using such a verifying technique in the random oracle model, the scheme can capture CCA security.

The technique above, however, is not suitable for constructing IBPRE in the standard model. Intuitively, the CHK transformation [7] might be a possible approach to make a unidirectional single-hop IBPRE scheme secure against CCA in the standard model. Nevertheless, we show that the manner cannot be trivially employed in IBPRE. Suppose an IBPRE scheme without random oracles is secure against *chosen-plaintext attacks* (CPA) based on an IBE scheme (e.g., [24]), and its original ciphertext is (A, B, C). In re-encryption, suppose the proxy can generate (at least) a new component A' and output (A', B, C) as the re-encrypted ciphertext so that the delegatee can recover the plaintext from A', B, C using his secret key. Can we simply apply the CHK transformation to achieve CCA security? Unfortunately, the above manner seems unwieldy. Note that for simplicity we omit the transformation details and only discuss the signature part. If we sign (A, B, C) by the CHK transformation, then the proxy cannot output the re-encrypted ciphertext (A', B, C) without invalidating the signature. To keep the validity of the signature, the proxy might choose to output (A, B, C), the corresponding signature (for A, B, C) and A' as re-encrypted ciphertext. Despite the validity of (A, B, C) can be verified by the signature, the validity of A' cannot be guaranteed. But if we only sign (B, C), then A can be arbitrarily mutated which leads to the invalidity of the decryption value. In Section 3, we propose a solution to overcome the above difficulty.

1.2 Related Work

Following the concept of decryption rights delegation introduced by Mambo and Okamoto [17], Blaze, Bleumer and Strauss [3] formalized proxy re-encryption and proposed the first CPA-secure PRE scheme. Later on, many classical PRE schemes, such as [1,8,14,15], have been proposed.

Employing traditional PRE in the identity-based cryptographic setting, Green and Ateniese [13] defined the notion of IBPRE and proposed two unidirectional IBPRE schemes in the random oracle model: one is CPA-secure multi-hop IBPRE and the other is CCA-secure single-hop IBPRE. The schemes, however, are not collusion resistant. Note that Ivan and Dodis [14] also proposed an IBPRE in which a trusted private key generator (PKG) delegates decryption rights for all system users. Their construction differs from that of Green and Ateniese. In this paper we mainly focus on the previous works of unidirectional single-hop IBPRE.

In 2007, two CPA-secure IBPRE schemes without random oracles were proposed by Matsuo [18]. Later on, Wang et al. [22,23] proposed two IBPRE schemes in the random oracle model: one is CPA secure and supports the revocability of proxy's re-encryption rights, and the other is CCA secure and allows the proxy to be malicious (rather than being semi-trusted). In [22], Wang et al. claimed that their scheme could achieve CCA security by combining the 2-level HIBE of Waters with the CHK transformation. However, the manner cannot be trivially used to convert a CPA-secure IBPRE to a CCA-secure one in the standard model. Please refer to the discussion in Section 1.1. In 2011, Minzuno and Doi [19] proposed an IBPRE scheme in the standard model with CPA security. Previous IBPRE schemes require PKG to participate into the generation of the re-encryption key (i.e. the re-encryption key generation is interactive).

Following the first IBPRE scheme [13] without any interaction (i.e. non-interactive) in the re-encryption key generation, Tang et al. [21] proposed a CPA-secure IBPRE scheme with random oracles, in which the delegator and the delegatee can come from different domains. Recently, two CPA-secure IBPRE schemes without random oracles were proposed by Luo et al. [16]: one is single-hop and the other is multi-hop.

Both traditional PRE and IBPRE have a potential risk for access control in the sense that they allow the proxy to re-encrypt all ciphertexts of the delegator without any discrimination. To solve the problem, Type-Based PRE [20] (in 2008) and Conditional PRE (CPRE) [25,26] (in 2009) were proposed to guarantee that the proxy can re-encrypt a ciphertext tagged with a specific condition only if a re-encryption key corresponding to the same condition is generated by the delegator.

The aforementioned unidirectional single-hop IBPRE schemes neither achieve CCA security in the standard model nor support conditional re-encryption. We choose some previous unidirectional single-hop IBPRE schemes which are related to our work, and summarize the comparison of properties in Table 1. To the best of our knowledge, our scheme is the first CCA-secure unidirectional single-hop IBPRE scheme without random oracles supporting conditional re-encryption.

Table 1. Property Comparison

Schemes	Security	Without ROM	Collusion Resistance	Conditional Re-Encryption	Non-Interactivity
IBPRE [19]	CPA	✓	✓	✗	✗
IBPRE [16]	CPA	✓	✓	✗	✓
IBPRE [13]	CCA	✗	✗	✗	✓
IBPRE [23]	CCA	✗	✓	✗	✗
IBPRE [11]	CCA	✗	✗	✗	✓
Our IBCPRE	CCA	✓	✓	✓	✓

2 Definition and Security Models

As of [13], we refer to the original ciphertext and the re-encrypted ciphertext as the second-level ciphertext and the first-level ciphertext, respectively. Unless stated otherwise, by an IBCPRE we mean a unidirectional single-hop IBCPRE.

2.1 Definition of Identity-Based Conditional Proxy Re-Encryption

Definition 1. ***(IBCPRE)*** *An Identity-Based Conditional Proxy Re-Encryption (IBCPRE) scheme consists of the following algorithms:*

1. $(mpk, msk) \leftarrow Setup(1^\lambda)$*: on input a security parameter* $\lambda \in \mathbb{N}$*, output a master public key* mpk *and a master secret key* msk*.*
2. $sk_{ID} \leftarrow KeyGen(mpk, msk, ID)$*: on input* mpk*,* msk*, and an identity* $ID \in \{0,1\}^*$*, output a secret key* sk_{ID}*.*
3. $rk_{w|ID_i \rightarrow ID_j} \leftarrow ReKeyGen(mpk, sk_{ID_i}, ID_j, w)$*: on input* mpk*, the secret key* sk_{ID_i} *of an identity* ID_i*, an identity* ID_j*, and a condition* $w \in \{0,1\}^*$*, output a re-encryption key* $rk_{w|ID_i \rightarrow ID_j}$ *from* ID_i *to* ID_j *under* w*.*
4. $C^{(2)}_{(ID_i,w)} \leftarrow Enc(mpk, ID_i, w, m)$*: on input* mpk*, an identity* ID_i*, a condition* w *and a plaintext* $m \in \{0,1\}^\lambda$*, output a second-level ciphertext* $C^{(2)}_{(ID_i,w)}$*.*
5. $C^{(1)}_{(ID_j,w)} \leftarrow ReEnc(mpk, rk_{w|ID_i \rightarrow ID_j}, ID_i, w, C^{(2)}_{(ID_i,w)})$*: on input* mpk*, a re-encryption key* $rk_{w|ID_i \rightarrow ID_j}$*, an identity* ID_i*, a condition* w *and a second-level ciphertext* $C^{(2)}_{(ID_i,w)}$*, output a first-level ciphertext* $C^{(1)}_{(ID_j,w)}$*.*

6. *$m \leftarrow Dec_2(mpk, ID_i, sk_{ID_i}, w, C^{(2)}_{(ID_i,w)})$: on input mpk, an identity ID_i and the corresponding secret key sk_{ID_i}, a condition w and a second-level ciphertext $C^{(2)}_{(ID_i,w)}$, output a plaintext m or $\perp$ for failure.*
7. *$m \leftarrow Dec_1(mpk, ID_i, ID_j, sk_{ID_j}, w, C^{(1)}_{(ID_j,w)})$: on input mpk, an identity ID_i, an identity ID_j and the corresponding secret key sk_{ID_j}, a condition w and a first-level ciphertext $C^{(1)}_{(ID_j,w)}$, output a plaintext m or $\perp$ for failure.*

For simplicity, we omit *mpk* in the expression of the algorithms in the rest of the paper.

Correctness: For any $\lambda \in \mathbb{N}$, any identities $ID_i, ID_j \in \{0,1\}^*$, where $i \neq j$, $i, j \in \{1, ..., poly(\lambda)\}$, any condition $w \in \{0,1\}^*$ and any message $m \in \{0,1\}^\lambda$, if $(mpk, msk) \leftarrow Setup(1^\lambda)$, $sk_{ID} \leftarrow KeyGen(msk, ID)$, for all ID used in the system, $rk_{w|ID_i \to ID_j} \leftarrow ReKeyGen(sk_{ID_i}, ID_j, w)$, $C^{(2)}_{(ID_i,w)} \leftarrow Enc(ID_i, w, m)$, and $C^{(1)}_{(ID_j,w)} \leftarrow ReEnc(rk_{w|ID_i \to ID_j}, ID_i, w, C^{(2)}_{(ID_i,w)})$, we have $Dec_2(ID_i, sk_{ID_i}, w, C^{(2)}_{(ID_i,w)}) = m$; $Dec_1(ID_i, ID_j, sk_{ID_j}, w, C^{(1)}_{(ID_j,w)}) = m$.

2.2 Security Models

We start with the formalization of IND-aCon-aID-CCA security at second-level ciphertext as follows.

Definition 2. *An IBCPRE scheme is IND-aCon-aID-CCA-secure at second-level ciphertext if no probabilistic polynomial time (PPT) adversary $\mathcal{A}$ can win the game below with non-negligible advantage. In the game, $\mathcal{B}$ is the game challenger and λ is the security parameter.*

1. ***Setup.*** *$\mathcal{B}$ runs $Setup(1^\lambda)$ and sends mpk to $\mathcal{A}$.*
2. ***Query Phase I.*** *$\mathcal{A}$ is given access to the following oracles.*
   (a) *$Extract(ID)$: given an identity ID, return $sk_{ID} \leftarrow KeyGen(msk, ID)$, and ID is considered as corrupted.*
   (b) *$ReKeyExtract(ID_i, ID_j, w)$: given two distinct identities ID_i and ID_j, and a condition w, return $rk_{w|ID_i \to ID_j} \leftarrow ReKeyGen(sk_{ID_i}, ID_j, w)$, where $sk_{ID_i} \leftarrow KeyGen(msk, ID_i)$.*
   (c) *$ReEnc(ID_i, ID_j, w, C^{(2)}_{(ID_i,w)})$: given two distinct identities ID_i and ID_j, a condition w and a second-level ciphertext $C^{(2)}_{(ID_i,w)}$, return a first-level ciphertext $C^{(1)}_{(ID_j,w)} \leftarrow ReEnc(rk_{w|ID_i \to ID_j}, ID_i, w, C^{(2)}_{(ID_i,w)})$, where $rk_{w|ID_i \to ID_j} \leftarrow ReKeyGen(sk_{ID_i}, ID_j, w)$, $sk_{ID_i} \leftarrow KeyGen(msk, ID_i)$.*
   (d) *$Dec_2(ID_i, w, C^{(2)}_{(ID_i,w)})$: given an identity ID_i, a condition w and a second-level ciphertext $C^{(2)}_{(ID_i,w)}$, return $m \leftarrow Dec_2(ID_i, sk_{ID_i}, w, C^{(2)}_{(ID_i,w)})$, where $sk_{ID_i} \leftarrow KeyGen(msk, ID_i)$.*

   (e) *$Dec_1(ID_i, ID_j, w, C^{(1)}_{(ID_j,w)})$: given two distinct identities ID_i and ID_j, a condition w, and a first-level ciphertext $C^{(1)}_{(ID_j,w)}$, return $m \leftarrow Dec_1(ID_i,\ ID_j,\ sk_{ID_j},\ w,\ C^{(1)}_{(ID_j,w)})$, where $sk_{ID_j} \leftarrow KeyGen(msk,\ ID_j)$.*

3. ***Challenge.*** *$\mathcal{A}$ outputs two equal-length plaintexts m_0, m_1, a target identity ID^* and a target condition w^* to $\mathcal{B}$. If the following queries*
   - *$Extract(ID^*)$, and*
   - *$ReKeyExtract(ID^*, ID_j, w^*)$ and $Extract(ID_j)$ for any identity ID_j*

   are never made, $\mathcal{B}$ outputs $C^{(2)*}_{(ID^*,w^*)} = Enc(ID^*, w^*, m_b)$, where $b \in_R \{0,1\}$.
4. ***Query Phase II.*** *$\mathcal{A}$ makes further queries as in Query Phase I except the following:*
   (a) *$Extract(ID)$ if $ID = ID^*$;*
   (b) *$ReKeyExtract(ID^*, ID_j, w^*)$ and $Extract(ID_j)$ for any identity ID_j;*
   (c) *$ReEnc(ID^*, ID_j, w^*, C^{(2)*}_{(ID^*,w^*)})$ and $Extract(ID_j)$ for any identity ID_j;*
   (d) *$Dec_2(ID^*, w^*, C^{(2)*}_{(ID^*,w^*)})$; and*
   (e) *$Dec_1(ID^*, ID_j, w^*, C^{(1)}_{(ID_j,w^*)})$ for any ID_j and $C^{(1)}_{(ID_j,w^*)}$, if $(ID_j,\ w^*,\ C^{(1)}_{(ID_j,w^*)})$ is a derivative of $(ID^*,\ w^*,\ C^{(2)*}_{(ID^*,w^*)})$. As of [8], the derivative of $(ID^*,\ w^*,\ C^{(2)*}_{(ID^*,w^*)})$ is defined as follows.*
      i. *If $\mathcal{A}$ has issued a re-encryption key query on $(ID^*,\ ID_j,\ w^*)$ to obtain the re-encryption key $rk_{w^*|ID^* \to ID_j}$, and computed $C^{(1)}_{(ID_j,w^*)} \leftarrow ReEnc(rk_{w^*|ID^* \to ID_j}, ID^*, w^*, C^{(2)*}_{(ID^*,w^*)})$, then $(ID_j, w^*, C^{(1)}_{(ID_j,w^*)})$ is a derivative of $(ID^*, w^*, C^{(2)*}_{(ID^*,w^*)})$.*
      ii. *If $\mathcal{A}$ has issued a re-encryption query on $(ID^*, ID_j, w^*, C^{(2)*}_{(ID^*,w^*)})$ and obtained $C^{(1)}_{(ID_j,w^*)}$, then $(ID_j, w^*, C^{(1)}_{(ID_j,w^*)})$ is a derivative of $(ID^*, w^*, C^{(2)*}_{(ID^*,w^*)})$.*
5. ***Guess.*** *$\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$. If $b' = b$, $\mathcal{A}$ wins.*

The advantage of $\mathcal{A}$ is defined as $\epsilon = Adv^{IBCPRE-2nd}_{\mathcal{A}}(1^\lambda) = |Pr[b' = b] - \frac{1}{2}|$.

The definition of IND-aCon-aID-CCA security at first-level ciphertext for IBCPRE can be defined in an identical method. Due to limited space, the definition is provided in the full paper.

3 Constructions

Our IBCPRE scheme is constructed based on Waters IBE [24], a strongly existential unforgeable one-time signature scheme [2], a pseudorandom function family [12] and a Target Collision Resistant (TCR) hash function .

3.1 Preliminaries

Bilinear Pairings. Let *BSetup* be an algorithm that on input the security parameter λ, outputs the parameters of a bilinear map as $(q, g, \mathbb{G}_1, \mathbb{G}_2, e)$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are multiplicative cyclic groups of prime order q, $|q| = \lambda$, and g is a random generator of $\mathbb{G}_1$. The mapping $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ has three properties: (1) *Bilinearity*: for all $a, b \in_R \mathbb{Z}_q^*$, $e(g^a, g^b) = e(g, g)^{ab}$; (2) *Non-degeneracy*: $e(g, g) \neq 1_{\mathbb{G}_2}$, where $1_{\mathbb{G}_2}$ is the unit of $\mathbb{G}_2$; (3) *Computability*: e can be efficiently computed.

The Decisional Bilinear Diffie-Hellman Assumption. We review the Decisional Bilinear Diffie-Hellman (DBDH) problem (symmetric case) [4] as follows. Let g be a random generator of group $\mathbb{G}_1$.

Definition 3. *Given the tuple $(g, g^a, g^b, g^c, T) \in \mathbb{G}_1^4 \times \mathbb{G}_2$, the DBDH problem is to decide whether $T = e(g, g)^{abc}$, where $a, b, c \in_R \mathbb{Z}_q^*$. Define $Adv_{\mathcal{A}}^{DBDH} = |Pr[\mathcal{A}(g, g^a, g^b, g^c, e(g, g)^{abc}) = 1] - Pr[\mathcal{A}(g, g^a, g^b, g^c, T) = 1]|$ as the advantage of $\mathcal{A}$ in winning the DBDH problem. We say that the DBDH assumption holds in $\mathbb{G}_1$ if no PPT algorithm has non-negligible advantage.*

Strongly Existential Unforgeable One-Time Signatures. A strongly existential unforgeable one-time signature (OTS) [2] consists of the following algorithms:

1. $(K_{\mathrm{S}}, K_{\mathrm{V}}) \leftarrow Sign.KeyGen(1^\lambda)$: on input a security parameter $\lambda \in \mathbb{N}$, the algorithm outputs a signing/verification key pair $(K_{\mathrm{S}}, K_{\mathrm{V}})$.
2. $\sigma \leftarrow Sign(K_{\mathrm{S}}, M)$: on input the signing key K_{S} and a message $M \in \Gamma_{Sig}$, the algorithm outputs a signature σ, where Γ_{Sig} is the message space of a signature scheme.
3. $1/0 \leftarrow Verify(K_{\mathrm{V}}, \sigma, M)$: on input the verification key K_{V}, a signature σ and a message M, the algorithm outputs 1 when σ is a valid signature of M, and output 0 otherwise.

Remark. In this paper we assume the verification key K_{V} is n-bit length.

Definition 4. *A signature scheme is one-time strongly unforgeable chosen message attack secure if the advantage $Adv_{\mathcal{A}}^{OTS}(1^\lambda)$ is negligible for any PPT adversary $\mathcal{A}$ in the following experiment.*

$$\begin{aligned} &Adv_{\mathcal{A}}^{OTS}(1^\lambda) = Pr[Verify(K_V, \sigma^*, M^*) = 1 : (K_S, K_V) \leftarrow Sign.KeyGen(1^\lambda);\\ &(M, State) \leftarrow \mathcal{A}(K_V); \sigma \leftarrow Sign(K_S, M); (M^*, \sigma^*) \leftarrow \mathcal{A}(K_V, \sigma, State);\\ &(M^*, \sigma^*) \neq (M, \sigma)], \end{aligned}$$

where λ is the security parameter, State is the state information, $M, M^* \in \Gamma_{Sig}$.

3.2 Identity-Based Encryption Scheme

We first review the construction of Waters IBE scheme [24]. The details of definition and security model of IBE can be found in [24].

1. $\mathit{Setup}(1^\lambda)$: run $(q, g, \mathbb{G}_1, \mathbb{G}_2, e) \leftarrow \mathit{BSetup}(1^\lambda)$, choose $\alpha \in_R \mathbb{Z}_q^*$, $g_2, u', u_1, ..., u_n \in_R \mathbb{G}_1$, set $g_1 = g^\alpha$ and $U = \{u_i | 1 \leq i \leq n\}$. The master public key is $\mathit{mpk} = (\lambda, g, g_1, g_2, u', U)$ and the master secret key is $\mathit{msk} = g_2^\alpha$.
2. $\mathit{KeyGen}(\mathit{mpk}, \mathit{msk}, \mathit{ID})$: output a secret key $\mathit{sk}_{\mathit{ID}}$=$(\mathit{sk}_{\mathit{ID}_1}, \mathit{sk}_{\mathit{ID}_2})$=$(g_2^\alpha \cdot (u' \prod_{i \in \mathcal{V}} u_i)^r, g^r)$, where $r \in_R \mathbb{Z}_q^*$, ID is an n-bit string and $\mathcal{V}$ is the set of all i for which the i-th bit of ID is set to 1.
3. $\mathit{Enc}(\mathit{mpk}, \mathit{ID}, m)$: compute the ciphertext $C = (C_1, C_2, C_3) = (m \cdot e(g_1, g_2)^t, g^t, (u' \prod_{i \in \mathcal{V}} u_i)^t)$, where $t \in_R \mathbb{Z}_q^*$, $m \in \mathbb{G}_2$.
4. $\mathit{Dec}(\mathit{mpk}, \mathit{sk}_{\mathit{ID}}, C)$: recover a message $m = \frac{C_1 \cdot e(\mathit{sk}_{\mathit{ID}_2}, C_3)}{e(\mathit{sk}_{\mathit{ID}_1}, C_2)}$.

By Theorem 1 stated in [24], we have the following theorem.

Theorem 1. *Waters IBE scheme is CPA secure assuming the DBDH assumption holds.*

We next extend Waters IBE scheme to support hybrid encryption without losing CPA security. Specifically, we modify the system to admit m to be encoded as a λ-bit string, and to employ a pseudorandom function, which takes a function key (in $\mathbb{G}_2$) and a ciphertext component (in $\mathbb{G}_1$) as input and outputs a λ-bit pseudorandom string, to hide m symmetrically. Below is the extension.

1. $\mathit{Setup}(1^\lambda)$: choose a pseudorandom function $\mathit{PRF} : \mathbb{G}_2 \times \mathbb{G}_1 \rightarrow \{0,1\}^\lambda$ and add PRF to mpk.
2. $\mathit{KeyGen}(\mathit{mpk}, \mathit{msk}, \mathit{ID})$: same as that of Waters IBE scheme.
3. $\mathit{Enc}(\mathit{mpk}, \mathit{ID}, m)$: compute $C = (C_0, C_1, C_2, C_3) = ([\mathit{PRF}(\sigma, C_2)] \oplus m, \sigma \cdot e(g_1, g_2)^t, g^t, (u' \prod_{i \in \mathcal{V}} u_i)^t)$, where $t \in_R \mathbb{Z}_q^*$, $\sigma \in_R \mathbb{G}_2$, $m \in \{0,1\}^\lambda$.
4. $\mathit{Dec}(\mathit{mpk}, \mathit{sk}_{\mathit{ID}}, C)$: recover $\sigma = \frac{C_1 \cdot e(\mathit{sk}_{\mathit{ID}_2}, C_3)}{e(\mathit{sk}_{\mathit{ID}_1}, C_2)}$ and $m = C_0 \oplus [\mathit{PRF}(\sigma, C_2)]$.

We refer to Waters IBE scheme with the extension above as Type-I modified Waters IBE (Type-I mWIBE). The CPA security of the above scheme depends on the DBDH assumption and the pseudorandomness of PRF.

Theorem 2. *The Type-I mWIBE scheme is CPA secure assuming the DBDH assumption holds and PRF is a pseudorandom function family.*

Due to limited space, the proof of Theorem 2 is provided in the full paper.

Employing the technique introduced in [9], we can transform the Type-I mWIBE scheme to capture CCA security. Below is the transformation.

1. $\mathit{Setup}(1^\lambda)$: run $(q, g, \mathbb{G}_1, \mathbb{G}_2, e) \leftarrow \mathit{BSetup}(1^\lambda)$, choose $\alpha \in_R \mathbb{Z}_q^*$, $g_2, u'_1, u'_2, u_{1,1}, ..., u_{1,n}, u_{2,0}, u_{2,1}, ..., u_{2,n} \in_R \mathbb{G}_1$, a pseudorandom function $\mathit{PRF} : \mathbb{G}_2 \times \mathbb{G}_1 \rightarrow \{0,1\}^\lambda$, set $g_1 = g^\alpha$, $U_1 = \{u_{1,i} | 1 \leq i \leq n\}$ and $U_2 = \{u_{2,i} | 1 \leq i \leq n\}$. The master public key is $\mathit{mpk} = (\lambda, g, g_1, g_2, u'_1, u'_2, u_{2,0}, U_1, U_2, \mathit{PRF}, (\mathit{Sign.KeyGen}, \mathit{Sign}, \mathit{Verify}))$, and the master secret key is $\mathit{msk} = g_2^\alpha$.
2. $\mathit{KeyGen}(\mathit{mpk}, \mathit{msk}, \mathit{ID})$: output a secret key $\mathit{sk}_{\mathit{ID}}$=$(\mathit{sk}_{\mathit{ID}_1}, \mathit{sk}_{\mathit{ID}_2})$=$(g_2^\alpha \cdot (u' \prod_{i \in \mathcal{V}} u_i)^r, g^r)$, where $r \in_R \mathbb{Z}_q^*$.
3. $\mathit{Enc}(\mathit{mpk}, \mathit{ID}, m)$: run $(K_{\mathrm{S}}, K_{\mathrm{V}}) \leftarrow \mathit{Sign.KeyGen}(1^\lambda)$, generate the ciphertext $C = (K_{\mathrm{V}}, C_0, C_1, C_2, C_3, C_4, C_5) = (K_{\mathrm{V}}, [\mathit{PRF}(\sigma, C_2)] \oplus m, \sigma \cdot e(g_1, g_2)^t,$

g^t, $(u'_1 \prod_{i \in \mathcal{V}} u_{1,i})^t$, $(u'_2 u_{2,0} \prod_{i \in \mathcal{K}} u_{2,i})^t$, $Sign(K_S, (C_0, C_1, C_2, C_3, C_4)))$, where $t \in_R \mathbb{Z}_q^*$, $\sigma \in_R \mathbb{G}_2$, $m \in \{0,1\}^\lambda$, and $\mathcal{K}$ is the set of all i for which the i-th bit of K_V is set to 1.

4. $Dec(mpk, sk_{ID}, C)$: check $e(g, C_4) \overset{?}{=} e(C_2, u'_2 u_{2,0} \prod_{i \in \mathcal{K}} u_{2,i})$, $Verify(K_V, C_5, (C_0, C_1, C_2, C_3, C_4)) \overset{?}{=} 1$. If the equations do not hold, output $\perp$. Otherwise, compute $\sigma = \frac{C_1 \cdot e(sk_{ID_2}, C_3)}{e(sk_{ID_1}, C_2)}$ and $m = C_0 \oplus [PRF(\sigma, C_2)]$.

As stated in [9], Waters IBE scheme can be transformed to capture CCA security in the same manner above. We refer to the transformed Waters IBE scheme as Type-II mWIBE. By the security argument in [6], Theorem 1 and 2, we have the following theorem.

Theorem 3. *Suppose the Type-I mWIBE scheme (resp. Waters IBE scheme) is secure against CPA and the underlying one-time signature scheme ($Sign.KeyGen$, $Sign$, $Verify$) is strongly existential unforgeable, the transformed Type-I mWIBE scheme (resp. the Type-II mWIBE scheme) is CCA secure.*

We further modify the transformed Type-I mWIBE scheme as follows. In algorithm *Enc*, we notice that $(C_0, C_1, C_2, C_3, C_4)$ have to be signed so that they can be fixed by C_5 and K_V. However, it is unnecessary to sign all of them to capture CCA security. We utilize PRF to verify the validity of C_1 so that the signature C_5 can be only made for (C_0, C_2, C_3, C_4). Specifically, we first modify C_0 as $C_0 = [PRF(\sigma, C_2)]^{\lambda_1 - \lambda} || [PRF(\sigma, C_2)]_\lambda \oplus m$, where λ_1 is a security parameter and PRF is now required to output a λ_1-bit pseudorandom string. Then, we only sign (C_0, C_2, C_3, C_4) with K_S. In algorithm *Dec*, a decryptor can check the integrity of (C_0, C_2, C_3, C_4) by verifying C_5 with K_V. Meanwhile, the integrity of C_1 can be verified by $[PRF(\sigma, C_2)]^{\lambda_1 - \lambda} \overset{?}{=} [C_0]^{\lambda_1 - \lambda}$. Hence, the scheme still captures CCA security even if C_0 and C_5 are modified as above.

Besides, the transformed Type-I mWIBE scheme (which is a 2-level HIBE) can be naturally extended to a 3-level HIBE without losing CCA security as follows. Note that the additional level is for an n-bit condition w. We first define new parameters $u'_3 \in_R \mathbb{G}_1$ and $U_3 = (u_{3,i}) \in_R \mathbb{G}_1^n$ and add them to mpk, and next add a new component $C_6 = (u'_3 \prod_{i \in \xi_w} u_{3,i})^t$ to ciphertext C, where ξ_w is the set of all i for which the i-bit of w is set to 1. Finally, we sign C_6 as well as C_0, C_2, C_3 and C_4, i.e. $C_5 = Sign(K_S, (C_0, C_2, C_3, C_4, C_6))$.

It is not difficult to see that the two modifications above do not affect the CCA security of the transformed Type-I mWIBE scheme. We refer to the transformed Type-I mWIBE scheme with the two modifications as Type-III mWIBE. By the security argument in [6] and Theorem 3, we have the following theorem.

Theorem 4. *Suppose the transformed Type-I mWIBE scheme is secure against CCA, the Type-III mWIBE scheme is CCA secure.*

3.3 A New Unidirectional Single-Hop IBCPRE Scheme

We now start describing our IBCPRE scheme. Note that we allow identities and conditions to be arbitrary length bit-string, but they have to be hashed by a TCR hash function $H_0 : \{0,1\}^* \rightarrow \{0,1\}^n$ beforehand.

1. $Setup(1^\lambda)$. Run $(q, g, \mathbb{G}_1, \mathbb{G}_2, e) \leftarrow BSetup(1^\lambda)$. Let $w \in \{0,1\}^n$ be an n-bit condition string. Choose $\alpha \in_R \mathbb{Z}_q^*$, $g_2, u_1', u_2', u_3', u_{3,0} \in_R \mathbb{G}_1$, three random n-length sets $U_1 = \{u_{1,i} | 1 \leq i \leq n\}$, $U_2 = \{u_{2,i} | 1 \leq i \leq n\}$, $U_3 = \{u_{3,i} | 1 \leq i \leq n\}$, $u_{1,i}$, $u_{2,i}$, $u_{3,i} \in_R \mathbb{G}_1$, a pseudorandom function $PRF : \mathbb{G}_2 \times \mathbb{G}_1 \rightarrow \{0,1\}^{\lambda_1}$, and a TCR hash function $H_1 : \mathbb{G}_2 \rightarrow \mathbb{G}_1$, where λ_1 is a security parameter. The master secret key is $msk = g_2^\alpha$, the master public key is $mpk = (\lambda, \lambda_1, g, g_1, g_2, u_1', u_2', u_3', u_{3,0}, U_1, U_2, U_3, PRF, H_1, (Sign.KeyGen, Sign, Verify))$, where $g_1 = g^\alpha$.
2. $KeyGen(msk, ID)$. Output $sk_{ID} = (sk_{ID_1}, sk_{ID_2}) = (g_2^\alpha \cdot (u_1' \prod_{i \in \mathcal{V}_{ID}} u_{1,i})^r, g^r)$, where $r \in_R \mathbb{Z}_q^*$, $ID \in \{0,1\}^n$, and let $\mathcal{V}_{ID}$ be the set of all i for which the i-bit of ID is set to 1.
3. $Enc(ID_i, w, m)$. Run $(K_S, K_V) \leftarrow Sign.KeyGen(1^\lambda)$, choose $t \in_R \mathbb{Z}_q^*, \sigma \in_R \mathbb{G}_2$, generate the ciphertext: $C_0 = [PRF(\sigma, C_2)]^{\lambda_1 - \lambda} || [PRF(\sigma, C_2)]_\lambda \oplus m$, $C_1 = e(g_1, g_2)^t \cdot \sigma$, $C_2 = g^t$, $C_3 = (u_1' \prod_{i \in \mathcal{V}_{ID_i}} u_{1,i})^t$, $C_4 = (u_2' \prod_{i \in \xi_w} u_{2,i})^t$, $C_5 = (u_3' u_{3,0} \prod_{i \in \mathcal{X}_{K_V}} u_{3,i})^t$, $C_6 = Sign(K_S, (C_0, C_2, C_3, C_4, C_5))$, and output $C^{(2)}_{(ID_i, w)} = (K_V, C_0, C_1, C_2, C_3, C_4, C_5, C_6)$, where $ID_i \in \{0,1\}^n$, $m \in \{0,1\}^\lambda$, let ξ_w, $\mathcal{X}_{K_V}$, $\mathcal{V}_{ID_i}$ be the sets of all i for which the i-bit of w, K_V, ID_i is set to 1, respectively.
4. $ReKeyGen(sk_{ID_i}, ID_j, w)$. Choose $\rho, t' \in_R \mathbb{Z}_q^*$, $\theta \in_R \mathbb{G}_2$, compute $rk_0 = sk_{ID_{i_1}} \cdot (u_2' \prod_{i \in \xi_w} u_{2,i})^\rho$, $rk_1 = g^\rho$, $rk_2 = sk_{ID_{i_2}} \cdot H_1(\theta)$, $rk_3 = e(g_1, g_2)^{t'} \cdot \theta$, $rk_4 = g^{t'}$, $rk_5 = (u_1' \prod_{i \in \mathcal{V}_{ID_j}} u_{1,i})^{t'}$, $rk_6 = (u_3' u_{3,0} \prod_{i \in \mathcal{X}_{K_V'}} u_{3,i})^{t'}$, $rk_7 = Sign(K_S', (rk_3, rk_4, rk_5, rk_6))$, and output $rk_{w|ID_i \rightarrow ID_j} = (K_V', rk_0, rk_1, rk_2, rk_3, rk_4, rk_5, rk_6, rk_7)$, where $ID_j \in \{0,1\}^n$, $(K_S', K_V') \leftarrow Sign.KeyGen(1^\lambda)$.
5. $ReEnc(rk_{w|ID_i \rightarrow ID_j}, ID_i, w, C^{(2)}_{(ID_i, w)})$.
   (a) Verify

$$\begin{aligned} & e(g, C_3) \stackrel{?}{=} e(C_2, u_1' \prod_{i \in \mathcal{V}_{ID_i}} u_{1,i}), e(g, C_4) \stackrel{?}{=} e(C_2, u_2' \prod_{i \in \xi_w} u_{2,i}), \\ & e(g, C_5) \stackrel{?}{=} e(C_2, u_3' u_{3,0} \prod_{i \in \mathcal{X}_{K_V}} u_{3,i}), \\ & Verify(K_V, C_6, (C_0, C_2, C_3, C_4, C_5)) \stackrel{?}{=} 1. \end{aligned} \quad (1)$$

   If Eq. (1) does not hold, output $\perp$. Otherwise, proceed.

   (b) Compute $C_1' = \frac{C_1 \cdot e(rk_2, C_3)}{e(rk_0, C_2)/e(rk_1, C_4)}$, and output the first-level ciphertext $C^{(1)}_{(ID_j, w)} = (K_V, C_0, C_1', C_2, C_3, C_4, C_5, C_6, K_v', rk_3, rk_4, rk_5, rk_6, rk_7)$.
6. $Dec_2(ID_i, sk_{ID_i}, w, C^{(2)}_{(ID_i, w)})$.

(a) Verify Eq. (1). If Eq. (1) does not hold, output $\perp$. Otherwise, proceed.

(b) Compute $\sigma = C_1 \cdot \frac{e(sk_{ID_{i_2}}, C_3)}{e(sk_{ID_{i_1}}, C_2)}$, and output $m = [C_0]_\lambda \oplus [PRF(\sigma, C_2)]_\lambda$ if $[PRF(\sigma, C_2)]^{\lambda_1 - \lambda} = [C_0]^{\lambda_1 - \lambda}$ holds. Otherwise, output $\perp$.

7. $Dec_1(ID_i, ID_j, sk_{ID_j}, w, C^{(1)}_{(ID_j, w)})$.

(a) Verify

$$e(g, rk_5) \stackrel{?}{=} e(rk_4, u'_1 \prod_{i \in \mathcal{V}_{ID_j}} u_{1,i}), e(g, rk_6) \stackrel{?}{=} e(rk_4, u'_3 u_{3,0} \prod_{i \in \mathcal{X}_{K'_V}} u_{3,i}),$$
$$Verify(K'_V, rk_7, (rk_3, rk_4, rk_5, rk_6)) \stackrel{?}{=} 1. \quad (2)$$

If Eq. (2) does not hold, output $\perp$. Otherwise, proceed.

(b) Compute $\theta = rk_3 \cdot \frac{e(sk_{ID_{j_2}}, rk_5)}{e(sk_{ID_{j_1}}, rk_4)}$.

(c) Verify Eq. (1). If Eq. (1) does not hold, output $\perp$. Otherwise, proceed.

(d) Compute $\sigma = C'_1 / e(H_1(\theta), C_3)$, and output $m = [C_0]_\lambda \oplus [PRF(\sigma, C_2)]_\lambda$ if $[PRF(\sigma, C_2)]^{\lambda_1 - \lambda} = [C_0]^{\lambda_1 - \lambda}$ holds. Otherwise, output $\perp$.

Correctness: It is easy to verify that the plaintexts of the first-level and second-level ciphertexts can be recovered correctly if the ciphertexts are computed via the description above. We hence skip the details.

Theorem 5. *Suppose the DBDH assumption holds, PRF is a pseudorandom function family, $(Sign.KeyGen, Sign, Verify)$ is a strongly existential unforgeable one-time signature scheme and H_1 is a TCR hash function, our IBCPRE scheme is IND-aCon-aID-CCA secure at second-level ciphertext.*

The proof of Theorem 5 is provided in Appendix A.

Remark. Despite a malicious proxy can collude with the delegatee to recover the second component of the delegator's secret key and obtain the decryption rights of the ciphertexts which are only encrypted under the delegator's identity and condition w, the proxy cannot compromise the entire secret key of the delegator.

4 Concluding Remarks

In this paper, we tackled the open problems of the existing IBPRE schemes by proposing a new unidirectional single-hop IBCPRE scheme which achieves identity-based re-encryption and conditional re-encryption. We also showed that our scheme can be proved IND-aCon-aID-CCA secure in the standard model. To the best of our knowledge, our scheme is the first CCA-secure unidirectional single-hop IBCPRE without random oracles.

This paper also motivates some open problems, for example, how to construct CCA-secure IBCPRE in the standard model supporting OR gates on conditions.

References

1. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Trans. Inf. Syst. Secur. 9(1), 1–30 (2006)
2. Bellare, M., Shoup, S.: Two-tier signatures, strongly unforgeable signatures, and fiat-shamir without random oracles. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 201–216. Springer, Heidelberg (2007)
3. Blaze, M., Bleumer, G., Strauss, M.J.: Divertible protocols and atomic proxy cryptography. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 127–144. Springer, Heidelberg (1998)
4. Boneh, D., Boyen, X.: Efficient selective-ID secure identity-based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004)
5. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
6. Boneh, D., Katz, J.: Improved efficiency for CCA-secure cryptosystems built using identity-based encryption. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 87–103. Springer, Heidelberg (2005)
7. Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 207–222. Springer, Heidelberg (2004)
8. Canetti, R., Hohenberger, S.: Chosen-ciphertext secure proxy re-encryption. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) ACM Conference on Computer and Communications Security, pp. 185–194. ACM (2007)
9. Chu, C.-K., Tzeng, W.-G.: Identity-based proxy re-encryption without random oracles. In: Garay, J.A., Lenstra, A.K., Mambo, M., Peralta, R. (eds.) ISC 2007. LNCS, vol. 4779, pp. 189–202. Springer, Heidelberg (2007)
10. Coron, J.S.: On the exact security of full domain hash. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 229–235. Springer, Heidelberg (2000)
11. Emura, K., Miyaji, A., Omote, K.: An identity-based proxy re-encryption scheme with source hiding property, and its application to a mailing-list system. In: Camenisch, J., Lambrinoudakis, C. (eds.) EuroPKI 2010. LNCS, vol. 6711, pp. 77–92. Springer, Heidelberg (2011)
12. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM 33(4), 792–807 (1986)
13. Green, M., Ateniese, G.: Identity-based proxy re-encryption. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 288–306. Springer, Heidelberg (2007)
14. Ivan, A.A., Dodis, Y.: Proxy cryptography revisited. In: NDSS. The Internet Society (2003)
15. Libert, B., Vergnaud, D.: Unidirectional chosen-ciphertext secure proxy re-encryption. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 360–379. Springer, Heidelberg (2008)
16. Luo, S., Shen, Q., Chen, Z.: Fully secure unidirectional identity-based proxy re-encryption. In: Kim, H. (ed.) ICISC 2011. LNCS, vol. 7259, pp. 109–126. Springer, Heidelberg (2012)
17. Mambo, M., Okamoto, E.: Proxy cryptosystems: Delegation of the power to decrypt ciphertexts. IEICE Transactions E80-A(1), 54–63 (1997)
18. Matsuo, T.: Proxy re-encryption systems for identity-based encryption. In: Takagi, T., Okamoto, T., Okamoto, E., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 247–267. Springer, Heidelberg (2007)

19. Mizuno, T., Doi, H.: Secure and efficient ibe-pke proxy re-encryption. IEICE Transactions 94-A(1), 36–44 (2011)
20. Tang, Q.: Type-based proxy re-encryption and its construction. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 130–144. Springer, Heidelberg (2008)
21. Tang, Q., Hartel, P., Jonker, W.: Inter-domain identity-based proxy re-encryption. In: Yung, M., Liu, P., Lin, D. (eds.) Inscrypt 2008. LNCS, vol. 5487, pp. 332–347. Springer, Heidelberg (2009)
22. Wang, L., Wang, L., Mambo, M., Okamoto, E.: Identity-based proxy cryptosystems with revocability and hierarchical confidentialities. In: Soriano, M., Qing, S., López, J. (eds.) ICICS 2010. LNCS, vol. 6476, pp. 383–400. Springer, Heidelberg (2010)
23. Wang, L., Wang, L., Mambo, M., Okamoto, E.: New identity-based proxy re-encryption schemes to prevent collusion attacks. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 327–346. Springer, Heidelberg (2010)
24. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)
25. Weng, J., Deng, R.H., Ding, X., Chu, C.K., Lai, J.: Conditional proxy re-encryption secure against chosen-ciphertext attack. In: Li, W., Susilo, W., Tupakula, U.K., Safavi-Naini, R., Varadharajan, V. (eds.) ASIACCS, pp. 322–332. ACM (2009)
26. Weng, J., Yang, Y., Tang, Q., Deng, R.H., Bao, F.: Efficient conditional proxy re-encryption with chosen-ciphertext security. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) ISC 2009. LNCS, vol. 5735, pp. 151–166. Springer, Heidelberg (2009)

A Proof of Theorem 5

Proof. Suppose there is an adversary $\mathcal{A}$ who can break the IND-aCon-aID-CCA security at second-level ciphertext of our IBCPRE scheme with non-negligible probability ϵ. We then construct a reduction algorithm $\mathcal{B}$ to break the CCA-secure Type-III mWIBE scheme using $\mathcal{A}$. Let $\mathcal{B}_1$ be the challenger of the Type-III mWIBE in the CCA experiment. Note that $\mathcal{B}$ maintains the following tables which are initially empty.

1. SKT: records the tuples $(coin_i, ID_i, sk_{ID_i})$, which are the information of the secret keys.
2. RKT: records the tuples $(coin_z,\ ID_i,\ ID_j,\ w,\ rk_{w|ID_i \to ID_j},\ \theta,\ tag_1,\ tag_2)$, which are the results of the queries to $ReKeyExtract(ID_i, ID_j, w)$, where tag_1, tag_2 denote that the re-encryption key is a valid/random key.
3. RET: records the tuples $(ID_i,\ ID_j,\ w,\ C^{(1)}_{(ID_j,w)},\ tag_1,\ tag_2)$, which are the results of the queries to $ReEnc(ID_i,\ ID_j,\ w,\ C^{(2)}_{(ID_i,w)})$, where tag_1, tag_2 denote that the first-level ciphertext is generated under either a valid re-encryption key or a random one.

1. **Setup.** $\mathcal{B}_1$ sends $mpk = (\lambda,\ \lambda_1,\ g,\ g_1,\ g_2,\ u'_1,\ u'_2,\ u'_3,\ u_{3,0},\ U_1,\ U_2,\ U_3,\ PRF,$ $(Sign.KeyGen, Sign, Verify))$ to $\mathcal{B}$. Then $\mathcal{B}$ chooses a TCR hash function $H_1 : \mathbb{G}_2 \to \mathbb{G}_1$, adds H_1 to mpk and forwards mpk to $\mathcal{A}$.

2. **Query Phase I.** $\mathcal{A}$ issues a series of queries to which $\mathcal{B}$ responds as follows.
   (a) $Extract(ID)$: $\mathcal{B}$ first uses the Coron's technique [10] to flip a biased $coin_i \in \{0,1\}$ such that $Pr[coin_i = 1] = \vartheta$ and $Pr[coin_i = 0] = 1 - \vartheta$, where ϑ will be determined later.
      – If $coin_i = 0$, $\mathcal{B}$ outputs a random bit in $\{0,1\}$ and **aborts**.
      – Otherwise, $\mathcal{B}$ forwards the query to the secret key extraction oracle of Type-III mWIBE, obtains the secret key sk_{ID}, returns sk_{ID} to $\mathcal{A}$ and adds $(1, ID, sk_{ID})$ to SKT.
   (b) $ReKeyExtract(ID_i,\ ID_j,\ w)$: $\mathcal{B}$ first checks whether there is a tuple $(*, ID_i, ID_j, w, rk_{w|ID_i \to ID_j}, \theta, *, *)$ in RKT. If yes, $\mathcal{B}$ directly responds $rk_{w|ID_i \to ID_j}$ to $\mathcal{A}$, where $*$ is the wildcard. Otherwise, $\mathcal{B}$ first flips a biased $coin_z \in \{0,1\}$ for w and next generates the re-encryption key for $\mathcal{A}$ as follows.
      – If there is no tuple $(1, ID_i, sk_{ID_i})$ in SKT, $\mathcal{B}$ flips a biased $coin_i$ for ID_i. If $coin_i = 1$, $\mathcal{B}$ queries the secret key extraction oracle of Type-III mWIBE to obtain sk_{ID_i}, generates the re-encryption key $rk_{w|ID_i \to ID_j}$ via algorithm $ReKeyGen$ as in the real scheme, returns $rk_{w|ID_i \to ID_j}$ to $\mathcal{A}$ and adds $(1, ID_i, sk_{ID_i})$ and $(*, ID_i, ID_j, w, rk_{w|ID_i \to ID_j}, \theta, 1, 0)$ to SKT and RKT, respectively, where $\theta \in_R \mathbb{G}_2$. If $coin_i = 0$, $\mathcal{B}$ first sets $rk_0 = \delta_1, rk_1 = \delta_2, rk_2 = \delta_3$ and constructs rk_3, rk_4, rk_5, rk_6, rk_7 to encrypt a $\theta \in_R \mathbb{G}_2$ as in the real scheme, where $\delta_1, \delta_2, \delta_3 \in_R \mathbb{G}_1$. $\mathcal{B}$ next forwards the re-encryption key to $\mathcal{A}$ and adds $(*, ID_i, ID_j, w, rk_{w|ID_i \to ID_j}, \theta, 0, 1)$ to RKT.
      – Otherwise, $\mathcal{B}$ uses sk_{ID_i} to generate the re-encryption key via algorithm $ReKeyGen$ as in the real scheme, returns the re-encryption key to $\mathcal{A}$ and adds $(*, ID_i, ID_j, w, rk_{w|ID_i \to ID_j}, \theta, 1, 0)$ to RKT, where $\theta \in_R \mathbb{G}_2$.
   (c) $ReEnc(ID_i, ID_j, w, C^{(2)}_{(ID_i,w)})$: $\mathcal{B}$ first checks whether there is a tuple $(*, ID_i, ID_j, w, rk_{w|ID_i \to ID_j}, \theta, *, *)$ in RKT. If yes, $\mathcal{B}$ generates $C^{(1)}_{(ID_j,w)}$ using $rk_{w|ID_i \to ID_j}$ as in the real scheme, returns $C^{(1)}_{(ID_j,w)}$ to $\mathcal{A}$ and adds $(ID_i,\ ID_j,\ w,\ C^{(1)}_{(ID_j,w)},\ *, *)$ to RET. Otherwise, $\mathcal{B}$ first issues $ReKeyExtract(ID_i, ID_j, w)$ to obtain the re-encryption key, next generates $C^{(1)}_{(ID_j,w)}$ and adds the corresponding tuple to RET as above.
   (d) $Dec_2(ID_i, w, C^{(2)}_{(ID_i,w)})$: $\mathcal{B}$ first verifies Eq. (1). If the equation does not hold, $\mathcal{B}$ output $\perp$. Otherwise, $\mathcal{B}$ proceeds. If $(1, ID_i, sk_{ID_i}) \in SKT$, $\mathcal{B}$ recovers m using sk_{ID_i}. Otherwise, $\mathcal{B}$ forwards the query to the decryption oracle of Type-III mWIBE to obtain m.
   (e) $Dec_1(ID_i, ID_j, w, C^{(1)}_{(ID_j,w)})$: $\mathcal{B}$ first verifies Eq. (1) and Eq. (2). If the equations do not hold, $\mathcal{B}$ output $\perp$. Otherwise, $\mathcal{B}$ proceeds.
      i. If $(ID_i, ID_j, w, C^{(1)}_{(ID_j,w)}, 0, 1) \in RET \vee (*, ID_i, ID_j, w, rk_{w|ID_i \to ID_j}, \theta, 0, 1) \in RKT$ holds, $\mathcal{B}$ recovers $rk_{w|ID_i \to ID_j}$ from RKT and computes $C_1 = C_1' \cdot \frac{e(\delta_1, C_2)/e(\delta_2, C_4)}{e(\delta_3, C_3)}$. Then $\mathcal{B}$ issues $(ID_i, w, C^{(2)}_{(ID_i,w)})$ to the decryption oracle of Type-III mWIBE to obtain m.

   ii. Otherwise, $\mathcal{B}$ checks whether there is a tuple $(1, ID_j, sk_{ID_j})$ in SKT. If yes, $\mathcal{B}$ recovers m using sk_{ID_j}. Otherwise, $\mathcal{B}$ issues (ID_j, C') to the decryption oracle of Type-III mWIBE[1], obtains θ and then recovers m as in the real scheme, where $C' = (K'_{\mathrm{V}}, rk_3, rk_4, rk_5, rk_6, rk_7)$.

3. **Challenge.** When $\mathcal{A}$ decides that Query Phase I is over, it outputs m_0, m_1, ID^* and w^* to $\mathcal{B}$. If either the case that there is a tuple $(1, ID^*, sk_{ID^*})$ in SKT or the case that the $coin_z$ for w^* is equivalent to 1 holds, $\mathcal{B}$ outputs a random bit in $\{0,1\}$ and **aborts**. Otherwise, $\mathcal{B}$ sends the challenge message (m_0, m_1, ID^*, w^*) to $\mathcal{B}_1$, obtains the challenge ciphertext $C^{(2)*}_{(ID^*,w^*)}$ from $\mathcal{B}_1$ and returns $C^{(2)*}_{(ID^*,w^*)}$ to $\mathcal{A}$.
4. **Query Phase II.** $\mathcal{A}$ makes further queries as in Query Phase I with the constraints defined in Definition 2, and $\mathcal{B}$ responds as in Query Phase I except that in $ReKeyExtract$, if $ID_i \neq ID^*$, $\mathcal{B}$ responds as in Query Phase I, otherwise, $\mathcal{B}$ generates a random re-encryption key as in Query Phase I.
5. **Guess.** When $\mathcal{A}$ outputs a guess bit $b' \in \{0,1\}$, $\mathcal{B}$ outputs b'.

If $\mathcal{B}$ does not abort, $\mathcal{A}$'s view is identical to the real scheme except for the case that $\mathcal{B}$ outputs the random re-encryption keys in step (b) of the simulation. Let us consider the indistinguishability of the random re-encryption key and the valid one. It is not difficult to see that δ_1, δ_2 and δ_3 (which are randomly chosen by $\mathcal{B}$) must be able to take the form of rk_0, rk_1 and rk_2 (which are the components of the valid re-encryption key), respectively. Hence, the above indistinguishability relies on the indistinguishability between the encryption of $\theta \in_R \mathbb{G}_2$ (which is chosen by $\mathcal{B}$) and the encryption of the θ (which is used to hide $sk_{ID_{i_2}}$ in the real scheme). If there exists an adversary $\mathcal{A}_1$ who can distinguish the two encryptions, then we can construct a reduction algorithm $\mathcal{B}_2$ to break the CCA security of the Type-II mWIBE scheme via using $\mathcal{A}_1$.

Now we analyze the advantage of $\mathcal{B}$ using the same manner introduced in [11]. We have that $\mathcal{B}$ does not abort in the simulation with the probability $\vartheta^{q_{sk}} \cdot (1-\vartheta)^2$, which is maximized at $\vartheta_{opt} = \frac{q_{sk}}{2(1+q_{sk})}$, where q_{sk} is the total number of secret key extraction queries. Using ϑ_{opt}, $\mathcal{B}$ dose not abort with probability at least $\frac{1}{2\dot{e}(1+q_{sk})}$, where $\dot{e}$ is the base of the natural logarithm. Therefore, the advantage of $\mathcal{B}$ is at least $\frac{\epsilon}{2\dot{e}(1+q_{sk})}$.

This completes the proof of Theorem 5. $\square$

[1] We cannot issue (ID_j, C') to the decryption oracle of Type-III mWIBE directly. But we can modify the CCA-secure Type-III mWIBE scheme (by removing the PRF and the condition parts) to achieve the Type-II mWIBE scheme which is used to encrypt θ as in algorithm $ReKeyGen$. From Theorem 1 and 3, we can see that the Type-II mWIBE scheme is CCA secure assuming the DBDH assumption holds and $(Sign.KeyGen, Sign, Verify)$ is a strongly existential unforgeable one-time signature scheme.

Ciphertext Policy Multi-dimensional Range Encryption

Kohei Kasamatsu[1,2], Takahiro Matsuda[2,⋆], Goichiro Hanaoka[2], and Hideki Imai[1]

[1] Chuo University, Japan
{kasamatsu-kohei,h-imai}@imailab.sakura.ne.jp
[2] National Institute of Advanced Industrial Science and Technology, Japan
{t-matsuda,hanaoka-goichiro}@aist.go.jp

Abstract. There are many applications in which services are provided only if some values associated with some confidential (encrypted) data are within a specific range. In this paper, we propose the notion of (ciphertext-policy) range encryption (RE) that can be used in many of such applications. RE is a type of public key encryption with additional functionality where an encryptor can freely specify a range to a ciphertext so that it can be decrypted only if the values associated with the key belong to the range. We propose a concrete RE scheme based on the time-specific encryption scheme by Kasamatsu et al. (SCN2012). Our RE scheme is selectively secure under the weak bilinear Diffie-Hellman inversion assumption.

1 Introduction

There are many applications in which services are provided only if some values associated with a user's encrypted data are within a certain range, while preserving the data confidential. For example, an Internet advertising company may want to distribute information only for those who are adult and whose income is greater than some threshold. In such a case, it will be useful if we have an encryption scheme with the following property: a decryption key is associated with values, and an encryptor can freely specify a range to a ciphertext so that it can be decrypted only if the values associated with the key belong to the range.

In this paper, we propose the notion of (ciphertext-policy) *multi-dimensional range encryption* (RE) that realizes such a requirement. More specifically, in RE, a user's information (attribute) is encoded as a point $(p_1, p_2, \ldots)$ in a multi-dimensional space, and an authority distributes to each user a decryption key that corresponds to the point. An encryptor specifies a (multi-dimensional) range $([x_1, y_1], [x_2, y_2], \ldots)$, and sends a ciphertext that corresponds the range. A receiver of the ciphertext can decrypt it only if the point associated with his/her decryption key belongs to the range associated with the ciphertext, i.e. if it holds that $p_i \in [x_i, y_i]$ for all i. In fact, we actually consider a more flexible "threshold" version of such an encryption scheme. That is, an encryption scheme has

⋆ The second author is supported by a JSPS Fellowship for Young Scientists.

a threshold K such that the ciphertext can be decrypted only if the number of indices i satisfying $p_i \in [x_i, y_i]$ is greater than or equal to the threshold K. We refer to an encryption scheme in which the total number of dimensions treated in the scheme is L and the threshold is K as *K-out-of-L dimensional RE.* In this paper, we propose a concrete RE scheme by extending the time-specific encryption (TSE) scheme by Kasamatsu et al. [14].

1.1 Background

Time-Specific Encryption and the Construction in [14]. Paterson and Quaglia [17] proposed the notion of time-specific encryption (TSE), and several instantiations of it. In TSE, there is a trusted agent that periodically issues the information sk_t that is associated with each time slot t, and a ciphertext c is generated in such a way that it is associated with a time interval $[t_L, t_R]$ (called decryption time interval). The ciphertext c can be decrypted by using sk_t only if $t \in [t_L, t_R]$.[1] By regarding time as a point in one dimensional space, TSE can be viewed as a 1-out-of-1 dimensional RE scheme.

As mentioned earlier, our construction of a RE scheme is based on the TSE scheme by Kasamatsu et al. [14]. [14] constructed a TSE scheme based on a forward-secure encryption (FSE) scheme [1,8] which is obtained from a hierarchical identity-based encryption (HIBE) scheme by Boneh, Boyen, and Goh [5] (we call it BBG-HIBE). The Kasamatsu et al.'s TSE construction is based on the similarity of the functionalities of FSE and TSE. In fact, FSE can be viewed as a special case of TSE in which the "starting point" of an interval is fixed, by the following simple observation: In FSE, a ciphertext and a decryption key are associated with a time, and a decryption key can be updated periodically (without updating a corresponding public key). If a ciphertext is associated with time t, the ciphertext can be decrypted using any decryption key $sk_{t'}$ corresponding to time t' satisfying $t' \leq t$ by updating the decryption key $sk_{t'}$ into sk_t corresponding to time t and using it for decryption. Therefore a ciphertext of FSE for time t can be considered as a ciphertext of TSE for the decryption time interval of $[1, t]$. By reversing the role of time in a trivial manner, from a FSE scheme one can realize a TSE scheme in which the "end point" of a decryption time interval is fixed. The basic idea behind the TSE scheme in [14] is to combine these "restrictive" TSE schemes to obtain a TSE scheme in which we can specify an arbitrary decryption time interval of a ciphertext. However, a naive combination by multiple encryption [21,9] of these two "restrictive" TSE schemes does not work, because a decryption key of such a naive combination consists of two independently generated decryption keys of the underlying two restrictive TSE schemes, but an adversary who obtains several different decryption keys can decompose them to re-construct a new decryption key that allows the adversary

[1] We note that in [17], Paterson and Quaglia proposed three different settings of TSE: the plain setting, the public-key setting, and the identity-based setting. In the latter two settings, a ciphertext is additionally associated with a public-key or an identity of a receiver, and to decrypt the ciphertext the receiver's secret key is also required. The explanation of TSE here is about the plain setting.

to decrypt a challenge ciphertext. In [14], Kasamatsu et al. solved the above technical problem by using a specific instantiation of FSE obtained from the BBG-HIBE scheme [4]. More specifically, they combined two keys from the restrictive TSE schemes (obtained from the FSE scheme based on the BBG-HIBE scheme) in an inseparable manner, by relying on an algebraic structure of the BBG-HIBE scheme.

1.2 Our Contribution

In this paper, we propose the notion of RE and its efficient construction. Our construction of a concrete RE scheme is based on the time-specific encryption scheme by Kasamatsu et al. [14], and we show its selective security under the weak bilinear Diffie-Hellman inversion assumption [5, Sect. 2.3].

Here, we explain a basic idea for our proposed RE scheme. As mentioned in the previous subsection, TSE can be viewed as 1-out-of-1 dimensional RE. One might intuitively think that a K-out-of-L dimensional RE scheme can be obtained by combining this 1-out-of-1 dimensional RE with a K-out-of-L secret sharing scheme by a naive multiple encryption methodology. That is, a plaintext is split into shares using a K-out-of-L secret sharing scheme, each share is encrypted by an independent 1-out-of-1 dimensional RE scheme, and a ciphertext of a K-out-of-L dimensional RE scheme consists of ciphertexts of the underlying 1-out-of-1 dimensional RE schemes.

Unfortunately, this naive approach does not work, because it is vulnerable for collusion attacks, with a similar reason to the case of constructing TSE from FSE as we explained in the previous section. More specifically, a decryption key of such a RE scheme must consist of independently generated decryption keys of the underlying 1-out-of-1 dimensional RE scheme, but an adversary who obtains several different decryption keys can decompose them and re-construct a decryption key that allows the adversary to decrypt a challenge ciphertext. We solve this technical problem by using a splitting of not a plaintext, but a master secret key. More specifically, we split the master secret key of the TSE scheme by Kasamatsu et al. [14] using a K-out-of-L secret sharing so that each share of the master secret key can be used as the master secret key of the TSE scheme in [14]. Such an approach will not work in general, but we show that it works for the particular TSE scheme (i.e. 1-out-of-1 dimensional RE) of [14].

1.3 Related Work

The functionality of RE (i.e. the conditions for decryption via range membership) can be generally realized by functional encryption [16,3] which includes ciphertext-policy attribute-based encryption (CP-ABE) [2] and inner-product encryption [10] as special cases. However, the data size of RE schemes obtained from this general methodology are at least linear in $N \cdot L$ where the RE schemes support the maximal size of the range N, and thus will not be practical.

As mentioned in Sect. 1.1, TSE treats an interval as a condition for decryption, and can be viewed as 1-out-of-1 dimensional RE. TSE is introduced by

Paterson and Quaglia [17], and they constructed generic constructions of TSE schemes from identity-based encryption [18,6] and broadcast encryption [12,7]. Kasamatsu et al. [14] showed a concrete TSE scheme based on the BBG-HIBE scheme [5] and a generic construction of a TSE scheme from any FSE scheme.

Shi, Bethencourt, Chan, Song, and Perrig [11] proposed multi-dimensional range query over encrypted data (MRQED). In MRQED, as opposed to RE, a ciphertext is associated with a point in a multi-dimensional space, and a decryption key is associated with a (multi-dimensional) range. The ciphertext can be decrypted only if each coordinate of the point is within the range for all dimensions. (Therefore, by our terminology, MRQED could be called key-policy L-out-of-L RE.) By using MRQED, we can realize applications in which an administrator allows controlled access of data to a user of the applications, such as audit log systems for network and finance. For example, in the case of a network audit log system, an auditor can request, to a system administrator, for a decryption key so that it allows him/her to decrypt an encrypted log for port numbers, IP addresses, and time stamps that belong to a certain range.

2 Preliminaries

In this section, we formally introduce the definition of FSE and bilinear groups, and describe the decisional ℓ-weak Bilinear Diffie-Hellman Inversion (ℓ-wBDHI) assumption.

Notation. For integers x, y with $0 \leq x \leq y$, we denote by $[x, y]$ the interval containing all time periods from x to y inclusive for representing discrete L-dimensional space. Furthermore, for a positive integer $n \in \mathbb{N}$, we define $[n] = \{1, \ldots, n\}$. $x \overset{\$}{\leftarrow} y$ denotes that x is chosen uniformly at random from y. $x \leftarrow y$ denotes x is output from y if y is an algorithm, or y is assigned to x otherwise. "PPT" denotes *probabilistic polynomial time*. We say that a function $f(k)$ is negligible (in k) if $f(k) < 1/p(k)$ for any positive polynomial $p(k)$ and all sufficiently large k

2.1 Forward-Secure Encryption

FSE [1,8] has the property that the threat of key exposure is confined to some span by updating the secret key at each time unit. This scheme realizes the property by using the functionality that a receiver can update the previous secret key d_{t-1} to the next secret key d_t without interacting with any outside entity and without updating the corresponding public key. Here, we provide a formal definition of FSE by following [8] but slightly customized for our purpose.

An FSE scheme is defined by the four algorithms ($\mathtt{Gen}, \mathtt{Upd}, \mathtt{Enc}, \mathtt{Dec}$), which has the associated message space MSP. The key generation algorithm $\mathtt{Gen}(1^\lambda, N)$ takes a security parameter 1^λ and the total number of time periods N as input, and outputs a public key pk and an initial secret key d_0. The key update algorithm $\mathtt{Upd}(pk, i, j, d_i)$ takes pk, an index $i < N$ of a previous time period, an

```
$\mathtt{Gen}_{\mathtt{BBG}}(1^k, N)$ :
  $\alpha \xleftarrow{\$} \mathbb{Z}_q$; $g \xleftarrow{\$} \mathbb{G}$; $u \xleftarrow{\$} \mathbb{G}$
  $R \leftarrow e(g^\alpha, g)$; $d_0 \leftarrow g^\alpha$
  $h_i \xleftarrow{\$} \mathbb{G}$ for all $i \in [0, N]$
  Let $\boldsymbol{h} := (h_0, \ldots, h_N)$
  $PK \leftarrow (g, u, \boldsymbol{h}, R)$
  Return $(PK, d_0)$.

$\mathtt{Enc}_{\mathtt{BBG}}(PK, i, \sigma, M; s \in \mathbb{Z}_q)$ :
  Parse $PK$ as $(g, u, \boldsymbol{h}, R)$.
  $C_1 \leftarrow R^s \cdot M$
  $C_2 \leftarrow g^s$
  $C_3 \leftarrow H_\sigma(i, \boldsymbol{h}, u)^s$
  Return $C \leftarrow (C_1, C_2, C_3)$.
```

```
$\mathtt{Upd}_{\mathtt{BBG}}(PK, d_i, j, \sigma)$: (where $j > i$)
  Parse $PK$ as $(g, u, \boldsymbol{h}, R)$.;  $r \xleftarrow{\$} \mathbb{Z}_q$
  If $i = 0$ then
    $d_j \leftarrow (d_0 \cdot H_\sigma(j, \boldsymbol{h}, u)^r, g^r, \{h_\ell^r\}_{\ell \in [j+1, N]})$
  Else (i.e. $i \neq 0$)
    Parse $d_i$ as $(a_0, a_1, b_{i+1}, \ldots, b_N)$.
    $D_1 \leftarrow a_0 \cdot (\prod_{k=i+1}^{j} b_k^\sigma) \cdot H_\sigma(j, \boldsymbol{h}, u)^r$
    $d_j \leftarrow (D_1, a_1 \cdot g^r, \{b_v \cdot h_v^r\}_{v \in [j+1, N]})$
  End If
  Return $d_j$.

$\mathtt{Dec}_{\mathtt{BBG}}(d_i, C)$ :
  Parse $C$ as $(C_1, C_2, C_3)$.;  Parse $d_i$ as $(D_1, D_2, \ldots)$.
  Return $M \leftarrow \frac{C_1 \cdot e(C_3, D_2)}{e(C_2, D_1)}$.
```

Fig. 1. Basic BBG-FSE: The FSE scheme obtained from the BBG HIBE scheme, where $H_\sigma(i, \boldsymbol{h} = (h_0, \ldots, h_N), u) = h_0 \cdot (\prod_{k=1}^{i} h_k^\sigma) \cdot u$

index $j > i$ for the current time period, and a secret key d_i (corresponding to the period i) as input, and outputs a secret key d_j for the time period j. The encryption algorithm $\mathtt{Enc}(pk, i, M)$ takes pk, $i < N$, and a message $M \in MSP$ as input, and outputs a ciphertext c. The decryption algorithm $\mathtt{Dec}(pk, d_{i'}, c)$ takes pk, $d_{i'}$, and c as input, and outputs either M or a failure symbol $\perp$. We require, for all $\lambda \in \mathbb{N}$, all $N \in \mathbb{N}$, all $(pk, d_0) \leftarrow \mathtt{Gen}(1^\lambda, N)$, all indices $i \in [0, N-1]$ (for specifying time periods), and all messages $M \in MSP$, that $\mathtt{Dec}(pk, \mathtt{Upd}(pk, 0, i, d_0), \mathtt{Enc}(pk, i, M)) = M$. The security of FSE guarantees that even if an adversary learns SK_i for some i, messages encrypted during all time periods prior to i remain secret. We omit the explanation of security of FSE. For details, we would like to refer the reader to [8].

We note that Canetti et al. [8] defined only the "sequential update" algorithm for FSE. That is, in their syntax, the key update algorithm only allows an update from a secret key d_i for the time period i to a key d_{i+1} for the next time period. However, for the sake of simplicity, we use the syntax in which the update algorithm allows the "direct update", so that $\mathtt{Upd}$ takes a key d_i for the time period i as input and outputs the secret key d_j as long as $i < j$. It is straightforward to see that the direct update functionality can be generally achieved by the sequential update algorithm of [8]. In addition, there are FSE schemes which support an efficient direct update algorithm (compared to running the "sequential update" algorithm many times), such as the FSE scheme instantiated with the HIBE scheme by Boneh et al. [5] via the HIBE-to-FSE transformation shown in [8].

Basic Forward-Secure Encryption from the Boneh-Boyen-Goh HIBE Scheme. In Fig. 1, we show the instantiation of an FSE scheme, which we call the basic BBG-FSE scheme, using the HIBE scheme by Boneh, Boyen, Goh (BBG-HIBE) [5] via the "chain"-style HIBE-to-FSE transformation. The basic version of our RE scheme in Sect. 4.2 is obtained from the TSE scheme based on the above basic BBG-FSE scheme.

In the chain-style HIBE-to-FSE transformation, we realize an FSE scheme which supports N time periods by interpreting a time period i in FSE as an identity-vector $(1,\dots,1)$ in HIBE whose length is i. To update a secret key for time period i to time period $j > i$, one can run the key derivation algorithm of the HIBE scheme to obtain a decryption key for the identity-vector $(1,\dots,1)$ of length j. In this section, we use this transformation.

Another more sophisticated HIBE-to-FSE transformation is the binary tree-based construction due to Canetti, Halevi, and Katz [8]. This construction has the advantage in that to instantiate an FSE scheme with N time periods, a building block HIBE only needs to support a hierarchy with depth $\log N$. In Appendix A, we use this binary tree-based transformation to obtain our full construction.

The common feature of these HIBE-to-FSE transformations is that multiple instances of FSE can virtually be instantiated so that they all share the same public parameters, by regarding the top-level identities as the indices for specifying an independent HIBE scheme, and then applying the HIBE-to-FSE transformations to each HIBE scheme instantiated in the second (and lower) level identity space. This trick will be used in our basic and main constructions.

For notational convenience, in Fig. 1, we describe the scheme so that the encryption and the update algorithms take an additional input $\sigma \in \mathbb{N}$. This value σ is used to instantiate the σ-th BBG-FSE schemes with N time periods under the same public parameter, in such a way that each time period of the instantiation does not cover others' time periods. An identity-vector used in the chain HIBE-to-FSE transformation is represented as $(1, \sigma, \dots \sigma)$, where the first "1" is the common prefix of the chains, obtained by using the above trick of sharing public parameter.

2.2 Decisional ℓ-wBDHI Assumption

We first recall bilinear groups. Let $\mathbb{G}$ and $\mathbb{G}_T$ be groups of order p for some large prime q (we assume that the size of q is implicitly determined by the security parameter λ), and let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be an efficiently computable mapping. We call a tuple $(\mathbb{G}, \mathbb{G}_T, e)$ *bilinear groups*, and e a *bilinear map*, if the following two conditions hold: (Bilinear:) for all generators $(g, h) \in \mathbb{G} \times \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, we have $e(g^a, h^b) = e(g, h)^{ab}$. (Non-degenerate:) for all generators $g, h \in \mathbb{G}$, we have $e(g, h) \neq 1_{\mathbb{G}_T}$.

Now we recall the decisional ℓ-wBDHI assumption (which is defined via the so-called decisional ℓ-wBDHI* problem [5, Sect. 2.3]). Let $\ell \in \mathbb{N}$. We say that the decisional ℓ-wBDHI assumption holds in $(\mathbb{G}, \mathbb{G}_T, e)$ if for any PPT algorithm $\mathcal{A}$ the following difference is negligible in the security parameter 1^λ:

$$|\Pr[\mathcal{A}(g, h, t_1, \dots, t_\ell, e(g, h)^{\alpha^{\ell+1}}) = 0] - \Pr[\mathcal{A}(g, h, t_1, \dots, t_\ell, W) = 0]|$$

where $g, h \xleftarrow{\$} \mathbb{G}$, $\alpha \xleftarrow{\$} \mathbb{Z}_q$, $t_i \leftarrow g^{(\alpha^i)}$, and $W \xleftarrow{\$} G_T$.

3 Our Basic Idea

In this section, we give an intuitive explanation of our strategy of the proposed construction and review an efficient construction of TSE due to [14].

Underlying Technique of Our Construction. As explained in Sect. 1.2, a TSE scheme is already a 1-out-of-1 dimensional RE scheme, and thus, it seems also possible to obtain a K-out-of-L dimensional RE scheme by extending TSE. A naive idea for such extension is to split a plaintext into L shares by using a K-out-of-L secret sharing, and individually encrypt these shares by using L different copies of the underlying 1-out-of-1 dimensional RE scheme. However, this construction is not secure as collusion attacks are quite effective. Namely, for example, assuming that there are two users who possess different decryption keys, even if any one of these keys can recover at most $K - 1$ shares, they could reconstruct the plaintext as they may obtain K shares in total by the collusion attack.

In this work, for constructing K-out-of-L dimensional RE, we basically follow the above (insecure) method but also introduce a countermeasure which prevents the above collusion attack. More specifically, we do not straightforwardly combine TSE and a secret sharing (as above) but modify TSE to have the *resplittable* threshold property [13]. Roughly speaking, we say that threshold encryption satisfies *resplittability* if for a given fixed public and decryption keys, it is possible to repeatedly generate shares of the decryption keys for multiple times. Furthermore, a common "label" is given to all the shares which are generated in the same share generation process, and the decryption algorithm correctly recovers the plaintext only when using the threshold number of shares with the same label. In our proposed scheme, each user is implicitly given a unique label, and therefore, even though two (or more) malicious users collude, they cannot recover the plaintext from their shares since their labels are different.

From the above discussion, we see that a promising approach for achieving our goal is to extend a basic TSE scheme to have the resplittable threshold property. As such a basic TSE scheme, we choose Kasamatsu et al.'s scheme [14] which is one of the most efficient constructions of TSE. In the remaining part of this section, we give a brief review and an overview of our idea to extend it to have the resplittable threshold property.

Kasamatsu et al.'s Time-Specific Encryption. Here, we review a TSE scheme which was proposed in [14]. In [14], Kasamatsu et al. pointed out the similarity between FSE and TSE, and showed that it is possible to efficiently construct TSE via a simple modification of FSE. Based on this observation, they actually construct a fairly practical TSE scheme from an FSE scheme which can be straightforwardly obtained from the BBG-HIBE scheme [4]. More specifically, Kasamatsu et al.'s scheme is essentially a double encryption by the BBG-HIBE scheme, and the algebraic structure of Kasamatsu et al.'s scheme is almost the same as that of the BBG-HIBE scheme. Furthermore, as pointed out in [13], natural constructions of threshold encryption from pairings usually have resplittability as it is. Thus, we can expect that it is not difficult to construct a resplittable threshold version of Kasamatsu et al.'s TSE (if thresholding Kasamatsu et al.'s TSE is not difficult), and that by using it, K-out-of-L dimensional RE can be obtained. Actually, our proposed K-out-of-L dimensional RE scheme is constructed in this way.

4 Our Basic Construction of Range Encryption

In this section, we give the definition of RE and our proposed construction of RE.

4.1 Definition of Ciphertext-Policy Multi-dimensional Range Encryption

In RE, secret keys are associated with a point $\mathtt{P} = (p_1, \ldots, p_L)$ in an L-dimensional space and ciphertexts are associated with a multi-dimensional range $\mathtt{F} = ([x_1, y_1], \ldots, [x_L, y_L])$. A receiver can decrypt a ciphertext only if the number of indices satisfying $p_i \in [x_i, y_i]$ is greater than or equal to the threshold K. For notational convenience, for a point $\mathtt{P} = (p_1, \ldots, p_L)$ and an L dimensional range $\mathtt{F} = ([x_1, y_1], \ldots, [x_L, y_L])$, we define $I_{\mathtt{P},\mathtt{F}} = \{i \in [L] \mid p_i \in [x_i, y_i]\}$ as the set consisting of indices i for which the coordinate p_i of the point $\mathtt{P}$ lies in the corresponding range $[x_i, y_i]$ of $\mathtt{F}$.

Now, we give the formal definition of RE as follows.

Definition 1. *Ciphertext-policy K-out-of-L dimensional range encryption is defined by the four algorithms (*$\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}$*):*

$\mathtt{Setup}(1^\lambda, L, N, K)$**:** The setup algorithm takes a security parameter 1^λ, the total number of dimensions L, the maximal size of the range N, and a threshold K as input, and outputs a public parameter PP and a master secret key MSK.

$\mathtt{KeyGen}(PP, MSK, \mathtt{P} = (p_1, \ldots, p_L))$**:** The key generation algorithm takes PP, MSK, and a point $\mathtt{P}$ as input, and outputs a secret key $SK_{\mathtt{P}}$.

$\mathtt{Enc}(PP, \mathtt{F} = ([x_1, y_1], \ldots, [x_L, y_L]), M)$**:** The encryption algorithm takes PP, an L dimensional range $\mathtt{F}$, and a plaintext M as input, and outputs a ciphertext $C_{\mathtt{F}}$.

$\mathtt{Dec}(PP, SK_{\mathtt{P}}, C_{\mathtt{F}})$**:** The decryption algorithm takes PP, $SK_{\mathtt{P}}$, and $C_{\mathtt{F}}$ as input, and outputs M or $\perp$.

We require that for all $\lambda \in \mathbb{N}$, all $N \in \mathbb{N}$, all points $\mathtt{P} = (p_1, \ldots, p_L) \in [N]^L$, all L dimensional ranges $\mathtt{F} = ([x_1, y_1], \ldots, [x_L, y_L])$ where $1 \leq x_i \leq y_i \leq N$ (for all $i \in [L]$), all $(PP, MSK) \leftarrow \mathtt{Setup}(1^\lambda, L, N, K)$, and all plaintexts M: if $|I_{\mathtt{P},\mathtt{F}}| \geq K$, then it holds that $\mathtt{Dec}(PP, \mathtt{KeyGen}(PP, MSK, \mathtt{P}), \mathtt{Enc}(PP, \mathtt{F}, M)) = M$.

Definition 2. *We say that a K-out-of-L dimensional RE scheme is selective IND-CPA secure if for any polynomial N, for any PPT adversary $\mathcal{A}$, the advantage function $Adv^{CPA}_{\mathcal{A},N}(\lambda)$ is negligible in the following game between a challenger $\mathcal{C}$ and $\mathcal{A}$:*

Init. *$\mathcal{A}$ chooses a challenge L-dimensional range $\mathtt{F}^* = ([x_1^*, y_1^*], \ldots, [x_L^*, y_L^*])$ and sends it to $\mathcal{C}$.*

Setup. *$\mathcal{C}$ runs $\mathtt{Setup}(1^\lambda, L, N, K)$ to generate (PP, MSK), and gives PP to $\mathcal{A}$.*

Phase 1. *$\mathcal{A}$ can adaptively issue the key generation query $\mathtt{P}_i$. If $|I_{\mathtt{P}_i,\mathtt{F}^*}| \geq K$, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ responds to the query by running $\mathtt{KeyGen}(PP, MSK, \mathtt{P}_i)$ to generate $SK_{\mathtt{P}_i}$ and sending $SK_{\mathtt{P}_i}$ to $\mathcal{A}$.*

Challenge. *$\mathcal{A}$ selects two challenge messages M_0, M_1, and sends them to $\mathcal{C}$. $\mathcal{C}$ chooses a random bit β, and runs $\mathtt{Enc}(PP, \mathtt{F}^*, M_\beta)$ to generate a challenge ciphertext $C^*_{\mathtt{F}^*}$. Then $\mathcal{C}$ gives $C^*_{\mathtt{F}^*}$ to $\mathcal{A}$.*

Phase 2. *$\mathcal{A}$ can adaptively issue key generation queries $\mathtt{P}_i$ satisfying $|I_{\mathtt{P}_i,\mathtt{F}^*}| < K$.*

Guess. *$\mathcal{A}$ outputs its guess β' for β.*

$\mathcal{A}$'s advantage in the above game is defined by $Adv^{CPA}_{\mathcal{A},N}(\lambda) = |\Pr[\beta' = \beta] - 1/2|$.

As usual, adaptive security is defined by allowing an adversary to choose a challenge L-dimensional range $\mathtt{F}^*$ in the challenge phase, instead of forcing the adversary to choose it in the initial phase.

4.2 Basic Construction

For the sake of simplicity, in this section, we give the basic version of our proposed construction whose public parameter size is $O(N) + O(L)$ and secret key size is $O(NL)$. Our full scheme, in which the public parameter size is $O(L) + O(\log N)$ and the secret key size is $O(L \log^2 N)$ by using binary tree structures inspired by the HIBE-to-FSE transformation of Canetti et al. [8], is given in Appendix A. We stress that those proposed schemes share the same idea as explained in Sect. 3, and we believe that the basic version of our proposed scheme is helpful for understanding the full construction.

Description of Our Basic Construction. We give the basic version of our proposed RE scheme. Let $(\mathbb{G}, \mathbb{G}_T, e)$ be bilinear groups, let $N \in \mathbb{N}$ be the size of the space, let $L \in \mathbb{N}$ be the number of the dimension, and let $(\mathtt{Gen}_{\mathtt{BBG}}, \mathtt{Upd}_{\mathtt{BBG}}, \mathtt{Enc}_{\mathtt{BBG}}, \mathtt{Dec}_{\mathtt{BBG}})$ be the BBG-FSE scheme given in Fig. 1. Then we construct the basic version of our RE scheme as in Fig 2.

As explained in Sect. 3, we consider L "related" instantiations of the 1-out-of-1 dimensional RE (i.e. TSE) scheme of [14] each of whose master secret key is a share obtained from a K-out-of-L secret-sharing of a top level master secret g^α, using the specific algebraic property of the TSE scheme. Furthermore, each 1-out-of-1 dimensional RE scheme is constructed by connecting two "restrictive" TSE schemes derived from the basic BBG-FSE scheme described in Fig. 1 in the same way as is done in [14]. More specifically, the secret key of the n-th 1-out-of-1 dimensional RE scheme consists of two secret keys $(d^{(1,n)}_{y_n}, d^{(2,n)}_{N-x_n+1})$ of the basic BBG-FSE schemes,

```
$\texttt{Setup}(1^{\lambda}, L, N, K)$ :
  $(PK, d_0) \leftarrow \texttt{Gen}_{\texttt{BBG}}(1^k, N)$;   $MSK \leftarrow d_0 = g^{\alpha}$
  Parse $PK$ as $(g, u, \boldsymbol{h} = (h_0, \ldots, h_N), R)$.
  $u_{1,i} \xleftarrow{\$} \mathbb{G}$ and $u_{2,i} \xleftarrow{\$} \mathbb{G}$ for all $i \in [L]$
  $PP \leftarrow (g, \{u_{1,i}, u_{2,i}\}_{i \in [L]}, \boldsymbol{h}, R, K)$.
  Return $(PP, MSK)$.
$\texttt{KeyGen}(PP, MSK, \texttt{P} = (p_1, \ldots, p_L))$ :
  Parse $PP$ as $(g, \{u_{1,i}, u_{2,i}\}_{i \in [L]}, \boldsymbol{h}, R, K)$.
  $\xi_i \xleftarrow{\$} \mathbb{Z}_q$ for all $i \in [L]$
  $a_j \xleftarrow{\$} \mathbb{Z}_q$ for all $j \in [K-1]$
  $f(x) := \alpha + \Sigma_{k=1}^{K-1} a_k x^k$
  For all $n \in [L]$:
    $PK_{1,n} \leftarrow (g, u_{1,n}, \boldsymbol{h}, R)$
    $PK_{2,n} \leftarrow (g, u_{2,n}, \boldsymbol{h}, R)$
    $n' \leftarrow 2(n-1)+1$
    $d_{p_n}^{(1,n)} \leftarrow \texttt{Upd}_{\texttt{BBG}}(PK_{1,n}, g^{f(n)+\xi_n}, p_n, n')$
    $d_{N-p_n+1}^{(2,n)}$
        $\leftarrow \texttt{Upd}_{\texttt{BBG}}(PK_{2,n}, g^{-\xi_n}, N - p_n + 1, 2n)$
  End For
  Return $SK_{\texttt{P}} \leftarrow (\{d_{p_i}^{(1,i)}, d_{N-p_i+1}^{(2,i)}\}_{i \in [L]}, \texttt{P})$.
$\texttt{Enc}(PP, \texttt{F} = ([x_1, y_1], \ldots, [x_L, y_L]), M)$ :
  Parse $PP$ as $(g, \{u_{1,i}, u_{2,i}\}_{i \in [L]}, \boldsymbol{h}, R, K)$.
  $s \xleftarrow{\$} \mathbb{Z}_q$
  For all $n \in [L]$:
    $PK_{1,n} \leftarrow (g, u_{1,n}, \boldsymbol{h}, R)$
    $PK_{2,n} \leftarrow (g, u_{2,n}, \boldsymbol{h}, R)$
    $n' \leftarrow 2(n-1)+1$
    $(C_1, C_2, C_{3,n}) \leftarrow \texttt{Enc}_{\texttt{BBG}}(PK_{1,n}, y_n, n', M; s)$
    $(C_1, C_2, C'_{3,n})$
        $\leftarrow \texttt{Enc}_{\texttt{BBG}}(PK_{2,n}, N - x_n + 1, 2n, M; s)$
  End For
  Return $C_{\texttt{F}} \leftarrow (C_1, C_2, \{C_{3,i}, C'_{3,i}\}_{i \in [L]}, \texttt{F})$.
```

```
$\texttt{Dec}(PP, SK_{\texttt{P}}, C_{\texttt{F}})$ :
  Parse $PP$ as $(g, \{u_{1,i}, u_{2,i}\}_{i \in [L]}, \boldsymbol{h}, R, K)$.
  Parse $SK_{\texttt{P}}$ as $(\{d_{p_i}^{(1,i)}, d_{N-p_i+1}^{(2,i)}\}_{i \in [L]}, \texttt{P})$.
  Parse $C_{\texttt{F}}$ as $(C_1, C_2, \{C_{3,i}, C'_{3,i}\}_{i \in [L]}, \texttt{F})$.
  Parse $\texttt{P}$ as $(p_1, \ldots, p_L)$.
  Parse $\texttt{F}$ as $([x_1, y_1], \ldots, [x_L, y_L])$.
  If $|I_{\texttt{P},\texttt{F}}| < K$ then return $\perp$.
  Let $I'$ be any set s.t. $I' \subseteq I_{\texttt{P},\texttt{F}} \wedge |I'| = K$
  Let $\widetilde{I}$ be any set s.t. $\widetilde{I} \subsetneq I' \wedge |\widetilde{I}| = K - 1$
  Let $\widetilde{n} := I' \backslash \widetilde{I}$
  For all $n \in I'$:
    $PK_{1,n} \leftarrow (g, u_{1,n}, \boldsymbol{h}, R)$
    $PK_{2,n} \leftarrow (g, u_{2,n}, \boldsymbol{h}, R)$
    $n' \leftarrow 2(n-1)+1$
    $d_{y_n}^{(1,n)} \leftarrow \texttt{Upd}_{\texttt{BBG}}(PK_{1,n}, d_{p_n}^{(1,n)}, y_n, n')$
    $d_{N-x_n+1}^{(2,n)}$
      $\leftarrow \texttt{Upd}_{\texttt{BBG}}(PK_{2,n}, d_{N-p_n+1}^{(2,n)}, N - x_n + 1, 2n)$
    Parse $d_{y_n}^{(1,n)}$ as $(d_{1,n}, d'_{1,n}, \ldots)$.
    Parse $d_{N-x_n+1}^{(2,n)}$ as $(d_{2,n}, d'_{2,n}, \ldots)$.
    $D_{1,n} \leftarrow \left( d_{1,n}^{\Delta_{n,I'}(0)}, (d'_{1,n})^{\Delta_{n,I'}(0)}, \ldots \right)$
    $D_{2,n} \leftarrow \left( d_{2,n}^{\Delta_{n,I'}(0)}, (d'_{2,n})^{\Delta_{n,I'}(0)}, \ldots \right)$
    If $n = \widetilde{n}$ then
      $C_n \leftarrow (C_1, C_2, C_{3,n})$
    Else (i.e. $n \neq \widetilde{n}$)
      $C_n \leftarrow (1_{\mathbb{G}_T}, C_2, C_{3,n})$
    End If
    $M_{1,n} \leftarrow \texttt{Dec}_{\texttt{BBG}}(D_{1,n}, C_n)$
    $C'_n \leftarrow (1_{\mathbb{G}_T}, C_2, C'_{3,n})$
    $M_{2,n} \leftarrow \texttt{Dec}_{\texttt{BBG}}(D_{2,n}, C'_n)$
  End For
  Return $M' \leftarrow \prod_{n \in I'} M_{1,n} \cdot M_{2,n}$.
```

Fig. 2. The basic version of our proposed K-out-of-L dimensional RE scheme. Let the Lagrange coefficient $\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}$ for $i \in \mathbb{Z}_q$ and a set S, of elements in $\mathbb{Z}_q$, and let $1_{\mathbb{G}_T}$ be an identity element in $\mathbb{G}_T$. For $K-1$ degree polynomial f and I which is sets of K elements of $\mathbb{Z}_q^*$, we have $f(x) = \Sigma_{n \in I} \Delta_{n,I}(x) f(n)$. For the notation regarding the Lagrange coefficients, we follow [19].

which are connected via a 2-out-of-2 secret sharing (using the "blinding factor" ξ_n), where $d_p^{(i,j)}$ indicates a secret key for the time period p of the i-th FSE scheme in the j-th 1-out-of-1 dimensional RE scheme. The secret key of our basic version of the K-out-of-L dimensional RE scheme consists of L secret keys of 1-out-of-1 dimensional RE schemes connected by shares $f(n)$ of a K-out-of-L secret sharing.

We would like the reader to notice that in Fig. 2, the scheme is described at the cost of efficiency, so that it is easy to see that the TSE scheme is extended to have the resplittable threshold property, as we explained above. For example, in the encryption scheme, the ciphertext components C_1 and C_2 are computed $2L$ times by running $\texttt{Enc}_{\texttt{BBG}}$ from the common randomness s. However, in practice, $\{C_{3,i}, C'_{3,i}\}_{i \in [L]}$ can be computed without calculating C_1 and C_2. The security is guaranteed by the following theorem.

Theorem 1. *If the decisional $(N+1)$-wBDHI assumption holds in $(\mathbb{G}, \mathbb{G}_T, e)$, then the K-out-of-L dimensional RE scheme constructed as in Fig 2 is selective IND-CPA secure.*

In the proof of the theorem 1, as usual, we will build an algorithm $\mathcal{B}$ that solves the decisional $(N+1)$-wBDHI problem in $(\mathbb{G}, \mathbb{G}_T, e)$ by using any selective IND-CPA adversary $\mathcal{A}$ that attacks our proposed scheme in Fig. 2. Intuitively, we simulate the IND-CPA game of $\mathcal{A}$ by combining the proof methodology of [14] and that of [19]. We can simulate a secret key of 1-out-of-1 dimensional RE by using the approach of [14]. Due to the technique of [19] by using Lagrange interpolation, we can also apply the approach of [14] to the way of simulating secret keys corresponding to each dimension for the proposed K-out-of-L RE scheme (recall that a secret key of the K-out-of-L RE scheme consists of the underlying L secret keys of 1-out-of-1 RE scheme). The detail of the proof is given in the full version of this paper.

Flexible Choice of Threshold. In Fig. 2, a threshold K is common to all secret keys. However, in fact our RE scheme can allow the choice of a threshold K for each secret key by setting a random $K-1$ degree polynomial f for each key generation, (We have to change the security game slightly so that an adversary can choose not only a point, but also a threshold.) We can straightforwardly prove the security of this flexible variant in essentially the same way as we did for our RE scheme in Fig. 2. A RE scheme which allows flexible choice of a threshold is useful in the cases in which available conditions need to be changed depending on users.

5 Discussion

In this section, we discuss applications of RE and the efficiency of our RE scheme.

5.1 Immediate Applications

We introduce some applications realized straightforwardly by using our RE. Our RE scheme is useful for services managed based on information of a membership card, e.g. expiration date, age, birth date, and utility time. For example, a shop does not want to sell products for customers who are minor or who have the out-of-date membership card. The shop can easily make such a management based on a secret key of a RE scheme stored in the membership card.

There are also needs to control available services based on information of a user in e-commerce. For example, an on-line rental video company could provide rental videos only for customers with age 15 or elder, and within the limited number of lent goods, and expiration date. Our RE scheme allows a seller to easily manage the privileges of users. The available services of a user are decided by a secret key based on information provided when registering the user.

5.2 Comparison with Functional Encryption

RE can be seen as a special case of functional encryption (FE) [16,3] for a class of policies which express range membership and attributes which express points in each dimension. Hence, we could construct RE by using CP-ABE [2] or Predicate Encryption (PE) [10] which are special cases of FE. However, RE schemes obtained straightforwardly by using FE tend to become less efficient. For example, Okamoto and Takashima's scheme in [15] is known as one of the most sophisticated and powerful CP-ABE schemes, but it has a linearly-increasing ciphertext overhead and the size of public parameter in $N \cdot L$ where L is the total number of dimensions and N is the maximum size of the range. Furthermore, to our knowledge, an RE scheme which allows flexible choice of threshold could not be constructed from existing schemes. [2]

On the other hand, parameter sizes of our full RE scheme in Appendix A depend on at most not $O(L \cdot N)$ but $O(L \cdot \log^2 N)$, since our scheme can efficiently express the range membership. More specifically, our full scheme has the public parameter size of $O(L) + O(\log N)$ group elements, the ciphertext overhead of $O(L)$ group elements, and the secret key size of $O(L \log^2 N)$.

References

1. Anderson, R.: Two remarks on public key cryptology. In: ACM CCS 1997 (1997) (Invited Lecture), `http://www.cyphernet.org/cyphernomicon/chapter14/14.5.html`
2. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-Policy Attribute-Based Encryption. In: IEEE Symposium on Security and Privacy 2007, pp. 321–334 (2007)
3. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 253–273. Springer, Heidelberg (2011)
4. Boneh, D., Boyen, X., Goh, E.-J.: Hierarchical identity based encryption with constant size ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005)
5. Boneh, D., Boyen, X., Goh, E.J.: Hierarchical identity based encryption with constant size ciphertext. Full version of [4]. Cryptology ePrint Archive: Report 2005/015 (2005)
6. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
7. Boneh, D., Gentry, C., Waters, B.: Collusion resistant broadcast encryption with short ciphertexts and private keys. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 258–275. Springer, Heidelberg (2005)
8. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 646–646. Springer, Heidelberg (2003)
9. Dodis, Y., Katz, J.: Chosen-ciphertext security of multiple encryption. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 188–209. Springer, Heidelberg (2005)

[2] One of the reviewers pointed out that applying the range method by Bethencourt, Sahai, and Waters [2] into CP-ABE by Waters [20] results in an efficient RE scheme. In fact, the RE scheme constructed in such a way seems to have each the public parameter size, ciphertext overhead, secret key size of $O(L \log N)$. However, compared to this RE scheme, our RE scheme still is superior in the public parameter size and ciphertext overhead.

10. Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 146–162. Springer, Heidelberg (2008)
11. Shi, E., Bethencourt, J., Chan, H., Song, D., Perrig, A.: Multi-Dimensional Range Query over Encrypted Data. In: IEEE Symposium on Security and Privacy 2007, pp. 350–364 (2007)
12. Fiat, A., Naor, M.: Broadcast encryption. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 480–491. Springer, Heidelberg (1994)
13. Hanaoka, G., Kawai, Y., Kunihiro, N., Matsuda, T., Weng, J., Zhang, R., Zhao, Y.: Generic Construction of Chosen Ciphertext Secure Proxy Re-Encryption. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 349–364. Springer, Heidelberg (2012)
14. Kasamatsu, K., Matsuda, T., Emura, K., Attrapadung, N., Hanaoka, G., Imai, H.: Time-Specific Encryption from Forward-Secure Encryption. In: Visconti, I., De Prisco, R. (eds.) SCN 2012. LNCS, vol. 7485, pp. 184–204. Springer, Heidelberg (2012)
15. Okamoto, T., Takashima, K.: Fully Secure Functional Encryption with General Relations from the Decisional Linear Assumption. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 191–208. Springer, Heidelberg (2010)
16. O'Neill, A.: Definitional Issues in Functional Encryption. Cryptology ePrint Archive: Report 2010/556 (2010)
17. Paterson, K.G., Quaglia, E.A.: Time-specific encryption. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 1–16. Springer, Heidelberg (2010)
18. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985)
19. Sahai, A., Waters, B.: Fuzzy Identity-Based Encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005)
20. Waters, B.: Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 53–70. Springer, Heidelberg (2011)
21. Zhang, R., Hanaoka, G., Shikata, J., Imai, H.: On the Security of Multiple Encryption or CCA-security+CCA-security=CCA-security? In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 360–374. Springer, Heidelberg (2004)

A Our Main Construction of Range Encryption

Here, we describe our main RE scheme obtained by using the binary tree structures for the basic version of our scheme presented in Sect. 4. As noted earlier, this construction is obtained by applying the technique from the HIBE-to-FSE transformation by Canetti et al. [8] to the basic version of the proposed scheme for reducing the sizes of the public parameter and decryption keys.

Let $\ell \in \mathbb{N}$. Consider a complete binary tree B with $N = 2^\ell - 1$ nodes, where N will be the maximal size of values in one dimension supported by our proposed RE scheme. The nodes in the binary tree B are numbered according to a pre-order traversal in an incremental order, with the root node of B being 1. Intuitively, the binary tree corresponds to one instantiation of FSE obtained via the HIBE-to-FSE transformation of Canetti et al. [8] to the BBG-HIBE scheme.

We need to introduce *point vectors* PV_p and *point vectors set* $PVSet_p$. PV_p is the vector consisting of the indices corresponding to the nodes included in the path from the node p to the root node (of B), where PV_0 is empty set. Intuitively, the point vector PV_i shows an identity-vector corresponding to the time period i of the BBG-FSE scheme. (Recall that a time period is interpreted as an identity-vector in HIBE-to-FSE transformation of Canetti et al. [8].) For an index $p \in [N]$ (of a node in the tree B), the set $PVSet_p$ is defined as follows: $PVSet_1 = \{PV_1\}$. Recursively, for $p \in [2, N]$, $PVSet_{p+1}$ is defined depending on $PVSet_p$ as follows: Let $s = \min\{j \mid PV_j \in PVSet_p\}$. If PV_s is a leaf node, then $PVSet_{p+1}$ is obtained by removing the vector PV_s from the set $PVSet_p$. Otherwise, let s_L (resp. s_R) be the index of the left (resp. right) node of the node s. $PVSet_{p+1}$ is the set obtained by removing PV_s from and adding PV_{s_L} and PV_{s_R} to the set $PVSet_p$.

Our proposed scheme is constructed from multiple BBG-FSE schemes. We virtually instantiate multiple BBG-FSE schemes by using a map $H_{i,\sigma}$ which takes as input point vector PV_p as well as additional inputs $i \in [2]$ and $\sigma \in [L]$. The map $H_{i,\sigma}$ generates a point vector corresponding to the time period p of a first or second BBG-FSE scheme of the σ-th dimension. (Recall that the 1-out-of-1 dimensional RE scheme is constructed from two BBG-FSE schemes.)

In the basic construction of RE given in Sect. 4.2, we implicitly use $PV_p = (1, 2(\sigma-1)+i, \ldots, 2(\sigma-1)+i)$ s.t. $|PV_p| = p$ as a point vector corresponding to the time period p of the i-th BBG-FSE scheme of the σ-th dimension. This is the difference between the main and the basic constructions of RE.

Description of Our Full Scheme. Let $(\mathbb{G}, \mathbb{G}_T, e)$ be bilinear groups (where $\mathbb{G}$ and $\mathbb{G}_T$ are of prime order q), let $\ell \in \mathbb{N}$, let $N = 2^\ell - 1$ be the size of the space, and let $L \in \mathbb{N}$ be the number of the dimension. Then we construct the main version of our RE scheme as in Fig. 3.

As explained Sect. 4.2, this construction is obtained by applying the technique from the HIBE-to-FSE transformation by Canetti et al. [8] to the basic version of the proposed scheme. In the technique, a secret key of FSE corresponding to time j consists of a set of the secret keys of basic FSE for $PV_v \in PVSet_j$. In Fig. 3, `KeyDerive` is used as the update algorithm of FSE.

Similarly to the basic version of our RE in Fig. 2, the secret key of the i-th 1-out-of-1 dimensional RE scheme consists of two secret keys $d_{1,PVSet_{p_i}}$, $d_{2,PVSet_{N-p_i+1}}$ of basic BBG-FSE connected by the shares ξ_i, $-\xi_i$ of a 2-out-of-2 secret sharing. The secret key of a K-out-of-L dimensional RE scheme consists of L secret keys of 1-out-of-1 dimensional RE schemes connected by the share $f(i)$ of a K-out-of-L secret sharing.

The security is guaranteed by the following theorem (the proof is given in the full version of this paper).

Theorem 2. *If the decisional $(\ell+1)$-wBDHI assumption holds in $(\mathbb{G}, \mathbb{G}_T, e)$, then the K-out-of-L constructed as in Fig. 3 is selective IND-CPA secure.*

```
$\texttt{Setup}(1^\lambda, L, N = 2^\ell - 1, K)$ :
  $\alpha \xleftarrow{\$} \mathbb{Z}_q$; $R \leftarrow e(g^\alpha, g)$; $MSK \leftarrow g^\alpha$;  $u_{1,i} \xleftarrow{\$} \mathbb{G}$ and $u_{2,i} \xleftarrow{\$} \mathbb{G}$ for all $i \in [\ell]$
  $h_j \xleftarrow{\$} \mathbb{G}$ for all $j \in [0, N]$
  Define $H_{i,\sigma}(PV = (pv_1, \ldots, pv_{|PV|})) := h_0^{2NL+1} \cdot \prod_{j=1}^{|PV|}(h_j^{\{2(\sigma-1)+i\}N+pv_j}) \cdot u_{i,\sigma}$
  $PP \leftarrow (g, \{u_{1,i}, u_{2,i}\}_{i\in[L]}, \{h_j\}_{j\in[0,\ell]}, R, H, K)$
  Return $(PP, MSK)$.
$\texttt{KeyGen}(PP, MSK, \texttt{P} = (p_1, \ldots, p_L))$ :
  Parse $PP$ as $(g, \{u_{1,i}, u_{2,i}\}_{i\in[L]}, \{h_j\}_{j\in[0,\ell]}, R, H, K)$.;  $\xi_i \xleftarrow{\$} \mathbb{Z}_q$ for all $i \in [L]$
  $b_j \xleftarrow{\$} \mathbb{Z}_q^*$ for all $j \in [K-1]$;  Let $f(x) := \alpha + b_1 x + b_2 x^2 + \cdots + b_{K-1} x^{K-1}$
  For all $n \in [L]$:
    For all $PV_v \in PVSet_{p_n}$:
      $r_v^{(1,n)} \xleftarrow{\$} \mathbb{Z}_q^*$; $D_1 \leftarrow g^{f(n)+\xi_n} \cdot H_{1,n}(PV_v)^{r_v^{(1,n)}}$
      $d_{PV_v}^{(1,n)} \leftarrow (D_1, g^{r_v^{(1,n)}}, \{h_{v'}^{r_v^{(1,n)}}\}_{v'\in[|PV_v|+1,N]})$
    End For
    For all $PV_{v'} \in PVSet_{N-p_n+1}$:
      $r_{v'}^{(2,n)} \xleftarrow{\$} \mathbb{Z}_q^*$; $D_2 \leftarrow g^{-\xi_n} \cdot H_{2,n}(PV_{v'})^{r_{v'}^{(2,n)}}$
      $d_{PV_{v'}}^{(2,n)} \leftarrow (D_2, g^{r_{v'}^{(2,n)}}, \{h_{v'}^{r_{v'}^{(2,n)}}\}_{v'\in[|PV_{v'}|+1,N]})$
    End For
    $d_{1,PVSet_{p_n}} \leftarrow \{d_{PV_v}^{(1,n)}\}_{PV_v \in PVSet_{p_n}}$
    $d_{2,PVSet_{N-p_n+1}} \leftarrow \{d_{PV_{v'}}^{(2,n)}\}_{PV_{v'} \in PVSet_{N-p_n+1}}$
  End For
  Return $SK_{\texttt{P}} \leftarrow (\{d_{1,PVSet_{p_i}}, d_{2,PVSet_{N-p_i+1}}\}_{i\in[L]}, \texttt{P})$.
$\texttt{Enc}(PP, \texttt{F} = ([x_1, y_1], \ldots, [x_L, y_L]), M)$ :
  $s \xleftarrow{\$} \mathbb{Z}_q$;  $C_{\texttt{F}} \leftarrow (R^s \cdot M, g^s, \{H_{1,i}(PV_{y_i})^s, H_{2,i}(PV_{N-x_i+1})^s\}_{i\in[L]}, \texttt{F})$
  Return $C_{\texttt{F}}$.
$\texttt{Dec}(PP, SK_{\texttt{P}}, C_{\texttt{F}})$ :
  Parse $PP$ as $(g, \{u_{1,i}, u_{2,i}\}_{i\in[L]}, \{h_j\}_{j\in[0,\ell]}, R, H, K)$.
  Parse $SK_{\texttt{P}}$ as $(\{d_{1,PVSet_{p_i}}, d_{2,PVSet_{N-p_i+1}}\}_{i\in[L]}, \texttt{P})$.
  Parse $C_{\texttt{F}}$ as $(C_1, C_2, \{C_{3,i}, C_{4,i}\}_{i\in[L]}, \texttt{F})$.
  Parse $\texttt{P}$ as $(p_1, \ldots, p_L)$.;  Parse $\texttt{F}$ as $([x_1, y_1], \ldots, [x_L, y_L])$.
  If $|I_{\texttt{P},\texttt{F}}| < K$ then return $\perp$.
  Let $I'$ be any set s.t. $I' \subseteq I_{\texttt{P},\texttt{F}} \wedge |I'| = K$
  For all $j \in I'$:
    $d_{PV_v}^{(1,j)} \in d_{1,PVSet_{p_j}}$ s.t. $PV_v \subseteq PV_{y_j}$
    $d_{PV_{v'}}^{(2,j)} \in d_{2,PVSet_{N-p_j+1}}$ s.t. $PV_{v'} \subseteq PV_{N-x_j+1}$
    $d_{PV_{y_j}}^{(1,j)} \leftarrow \texttt{KeyDerive}(PP, d_{PV_v}^{(1,j)}, PV_{y_j})$;  $d_{PV_{N-p_j+1}}^{(2,j)} \leftarrow \texttt{KeyDerive}(PP, d_{PV_{v'}}^{(2,j)}, PV_{N-x_j+1})$
    Parse $d_{PV_{y_j}}^{(1,j)}$ as $(D_{1,j}, D'_{1,j}, \ldots)$.;  Parse $d_{PV_{N-p_j+1}}^{(2,j)}$ as $(D_{2,j}, D'_{2,j}, \ldots)$.
  End For
Return $M' \leftarrow C_1 \cdot \prod_{i\in I'} \frac{e(C_{3,i}, {D'_{1,i}}^{\Delta_{i,I'}(0)}) \cdot e(C_{4,i}, {D'_{2,i}}^{\Delta_{i,I'}(0)})}{e(C_2^{\Delta_{i,I'}(0)}, D_{1,i}) \cdot e(C_2^{\Delta_{i,I'}(0)}, D_{2,i})}$.
```

```
$\texttt{KeyDerive}(PP, d_{PV}^{(i,j)}, PV' = (pv_1, \ldots, pv_{|PV'|}))$: where $PV \subset PV'$
  Parse $d_{PV}^{(i,j)}$ as $(D, D', D_{|PV|+1}, \ldots, D_N)$.; $r \xleftarrow{\$} \mathbb{Z}_q$
  $d_{PV'}^{(i,j)} \leftarrow \Big((D \cdot \prod_{v=|PV|+1}^{|PV'|} D_v^{\{2(j-1)+i\}N+pv_v} \cdot u_{i,j}) \cdot H_{i,j}(PV')^r,$
                                    $D' \cdot g^r,\ \{D_{v'} \cdot h_v^r\}_{v'\in[|PV'|+1,N]}\Big)$
  Return $d_{PV'}^{(i,j)}$.
```

Fig. 3. The main version of our proposed K-out-of-L multi-dimensional RE. Let the Lagrange coefficient $\Delta_{i,S}(x) = \prod_{j\in S, j\neq i} \frac{x-j}{i-j}$ for $i \in \mathbb{Z}_q$ and a set S, of elements in $\mathbb{Z}_q$. For $K-1$ degree polynomial f and I which is sets of K elements of $\mathbb{Z}_q^*$, we have $f(x) = \Sigma_{n\in I}\Delta_{n,I}(x)f(n)$. For the notation regarding the Lagrange coefficients, we follow [19].

Speeding Up Ate Pairing Computation in Affine Coordinates

Duc-Phong Le and Chik How Tan

Temasek Laboratories, National University of Singapore,
5A Engineering Drive 1, #09-02, Singapore 117411
{tslld,tsltch}@nus.edu.sg

Abstract. At Pairing 2010, Lauter et al's analysis showed that Ate pairing computation in affine coordinates may be much faster than projective coordinates at high security levels. In this paper, we further investigate techniques to speed up Ate pairing computation in affine coordinates. We first analyze Ate pairing computation using 4-ary Miller algorithm in affine coordinates. This technique allows us to trade one multiplication in the full extension field and one field inversion for several multiplications in a smaller field. Then, we focus on pairing computations over elliptic curves admitting a twist of degree 3. We propose new fast explicit formulas for Miller function that are comparable to formulas over even twisted curves. We further analyze pairing computation on cubic twisted curves by proposing efficient subfamilies of pairing-friendly elliptic curves with embedding degrees $k = 9$, and 15. These subfamilies allow us not only to obtain a very simple form of curve, but also lead to an efficient arithmetic and final exponentiation.

Keywords: Ate pairing, Pairing computation, final exponentiation, affine coordinates, cubic twisted curves, pairing-friendly elliptic curves.

1 Introduction

In recent years, the pairings have become extremely useful in public-key cryptography. Pairings used in cryptography are efficiently computable bilinear maps on torsion subgroups of points on a (hyper-)elliptic curve that map into the multiplicative group of a finite field. We call such a map a *cryptographic pairing*. Let $\mathbb{G}_1, \mathbb{G}_2$ be finite abelian groups written additively, and let $\mathbb{G}_3$ be a finite abelian group written multiplicatively. A cryptographic pairing is a map:

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3.$$

The first pairing application to cryptography was introduced in Joux' seminal paper [17] describing a one-round tripartite Diffie-Hellman key exchange protocol in 2000. Since then, the use of cryptographic protocols based on pairings has had a huge success with some notable breakthroughs such as practical Identity-based Encryption (IBE) schemes [7], and many other new cryptographic primitives.

Due to the high cost of pairing operations, the efficiency of pairing computation and the construction of pairing-friendly curves have become an active field of research. The former concerns many techniques having been exploited to dramatically improve the performance of the Miller algorithm, see [2][3][20][8][24]. The later focuses on constructing curves that are suitable for pairing-based cryptosystems. Whereas standard elliptic curve cryptography can be implemented using randomly generated elliptic curves, the elliptic curves required to implement pairing-based protocols must have a small embedding degree such that pairings can be efficiently computed in extension finite fields. Many works on constructing pairing-friendly elliptic curves have been presented in [27][9][4] and this research is collected and extended in the recent paper [13].

Projective coordinates are usually preferred than affine coordinates for implementing pairings. That is because point addition or doubling operations in affine coordinates involve a field modular inversion that is much expensive than one field multiplication in the base field $\mathbb{F}_p$. However, recent analysis in [22] showed that over $\mathbb{F}_{p^d}$, for larger d, the inversion-to-multiplication ratio is significant reduced. Ate pairing computation in affine coordinates is thus much faster than that in projective coordinates at high security levels.

This work presents our optimizations to Miller loop using a 4-ary algorithm with direct formulas to compute quadrupling of points and a multiplication of two line functions in affine coordinates. Our techniques make a trade-off between one multiplication in the full extension field $\mathbb{F}_{p^k}$, one inversion in the subfield $\mathbb{F}_{p^e}$ for some multiplications in $\mathbb{F}_{p^e}$, where k is the embedding degree of the elliptic curve E over the finite field $\mathbb{F}_p$, $e = k/d$, and d is the degree of the twist admitted during pairing computation.

This work also focuses on pairing computations over pairing-friendly elliptic curves admitting a cubic twist. Although, such a curve doesn't provide a full denominators elimination technique, but it allows a shorter Miller loop. We first present new fast formulas in affine coordinates for doubling/addition steps of Miller's algorithm over cubic twisted curves. Then, we give a finer choice for curves of embedding degrees $k = 9, 15$. By carefully choosing parameters, we point out that the desired curve is always of form $y^2 = x^3 + 1$. Finally, we present improvements for the hard part in final exponentiation for such curves.

The rest of the paper is organized as follows: Section 2 briefly recalls some basic knowledges about Ate pairing and its computation. Section 3 presents our improvements for the curves with even twisted degree. Section 4 presents new explicit formulas to speed up pairing computation on curves with cubic twisted degree. We conclude in Section 5.

2 Background on Pairings

For p prime and $p > 3$, an elliptic curve defined over a finite field $\mathbb{F}_p$ in short Weierstrass form is the set of solutions (x, y) to the equation

$$E : y^2 = x^3 + ax + b,$$

where $a, b \in \mathbb{F}_p$ such that the discriminant $\Delta = 4a^3 + 27b^2$ is non-zero. We denote by $\mathcal{O}$ the point at infinity on E, and by $\#E(\mathbb{F}_p)$ the number of points on E defined over $\mathbb{F}_p$. We have $n = \#E(\mathbb{F}_p) = p+1-t$, where t is the trace of Frobenius, which satisfies $|t| \leq 2\sqrt{p}$ (Hasse's theorem). Let r be a prime number that divides the number of points n and is co-prime to the characteristic p. Let k be the embedding degree of the elliptic curve E with respect to r, i.e., k the smallest positive integer such that $r|p^k - 1$. By this setting, we can define subgroups of points of prime order r on E and a multiplicative group of order r in the extension field $\mathbb{F}_{p^k}^* = \mathbb{F}_{p^k} \backslash \{0\}$, i.e., $\mathbb{F}_{p^k}^*$ contains the group μ_r of r-roots of unity.

2.1 The Ate Pairing

We denote subgroups of points of prime order r on $E(\mathbb{F}_{p^k})$ by $E[r]$. Let $m \in \mathbb{Z}$, $P \in E[r]$ and $f_{m,P}$ be a rational function on E with divisor $div(f_{m,P}) = m(P) - (mP) - (m-1)(\mathcal{O})$. Let π_p be the p-power Frobenius endomorphism on E, $\pi_p : E \to E$ given by $\pi_p(x, y) = (x^p, y^p)$. Let $T = t - 1$. We denote by $\mathbb{G}_1 = E[r] \cap Ker(\pi_p - [1]) = E(\mathbb{F}_p)[r]$, $\mathbb{G}_2 = E[r] \cap Ker(\pi_p - [p]) \subseteq E(\mathbb{F}_{p^k})[r]$. For $Q \in \mathbb{G}_2$ and $P \in \mathbb{G}_1$, the Ate pairing is defined as [16] (so with the arguments swapped in comparison to Tate pairing):

$$a_T = \mathbb{G}_2 \times \mathbb{G}_1 \to \mu_r, \qquad (Q, P) \mapsto f_{T,Q}(P)^{(p^k-1)/r}. \tag{1}$$

The length of Miller loop during Ate pairing computation is determined by the trace of Frobenius t. The Ate pairing is thus particularly suitable for pairing-friendly elliptic curves with small values of t. Usually, when implementing Tate pairing and its variants, instead of inputing the point Q on the curve $\mathbb{G}_2 \subseteq E(\mathbb{F}_{p^k})[r]$, one can take $Q' \in \mathbb{G}_2' \subseteq E'(\mathbb{F}_{p^{k/d}})[r]$, where E' is a twist of E, and $d|k$ is the degree of the twist. Points on the twisted curve are defined over a smaller field, and are thus obviously much faster for computation.

Let $\psi : E' \to E, Q' \mapsto Q$ be an isomorphism mapping points of the twisted curve to that of the original curve. The computation of $a_T(\psi(Q'), P)$ consists of two parts: evaluation of the function $f_{T,Q}$ at P and final exponentiation ensuring a unique result of the pairing. The first part is computed using Miller's algorithm [26] that is described in Algorithm 1.

```
Input: $T = \sum_{i=0}^{l-1} t_i 2^i$ (radix 2), $t_i \in \{0,1\}$, $Q' \in \mathbb{G}_2'$ not a multiple of $P \in \mathbb{G}_1$.
Output: $f_{T,\psi(Q')}(P)$ representing a class in $\mathbb{F}_{p^k}^*/(\mathbb{F}_{p^k}^*)^r$
$R' \leftarrow Q'$, $f \leftarrow 1$, ;
for $i = l-2$ to 0 do
    $f \leftarrow f^2 g_{\psi(R'),\psi(R')}(P)$ , $R' \leftarrow [2]R'$ ;
    if $r_i = 1$ then
        $f \leftarrow f g_{\psi(R'),\psi(Q')}(P)$ , $R' \leftarrow R' + Q'$ ;

end
return $f$
```

Algorithm 1. Miller's algorithm for Ate-like pairings

3 Improvements for the Even Twisted Curves

Pairing-friendly elliptic curves with an even embedding degree k are preferred in implementing Tate pairing and its variants. That is because the denominators elimination techniques can be used (see [2][14]) on such curves. Furthermore, such curves can admit a high-degree twist, e.g., twists of degree 4 or 6 such that the points on the twisted curve E' can be represented in a smaller finite field.

Lauter *et al.* analyzed the costs of Miller's algorithm in affine coordinates over curves with even embedding degrees [22, Table 1, 2]. They pointed out that when implementing one of the optimal Ate pairings [34] in high security levels, affine coordinates could be much faster than projective coordinates. This is because the ratio of the computational costs of inversions to multiplications for point doubling/addition operations is drastically reduced in extension fields.

3.1 4-ary Miller Algorithm

In this subsection, we present our optimizations of Miller loop using a 4-ary algorithm with direct formulas in affine coordinates. Usually, Miller's algorithm computes pairings using the *double-and-add* method. In [6], Blake *et al.* present the idea to compute the pairing using a 4-ary algorithm for the purpose of elimination vertical lines (i.e., denominators) in Miller's algorithm. Their algorithm can be applied on any curves (i.e., curves don't admit a twist and thus there isn't any denominators elimination technique), and has advantage if the binary expansion of the trace t has many zeros. Costello *et al.* [10] also addressed this problem by introducing a new algorithm so-called the Miller 2^n-tuple-and-add algorithm. They also presented explicit formulas in projective coordinates for cases of $n = 2, 3$.

Direct computation of $\ell_{R,R} \times \ell_{[2]R,[2]R}$*:* We assume that E', twisted curve of E defined in § 2, is given by an equation $E' : y^2 = x^3 + (a/\alpha^4)x + (b/\alpha^6)$ for some $\alpha \in \mathbb{F}_{p^k}$ with an isomorphism $\psi : E' \to E,\ (x, y) \mapsto (\alpha^2 x, \alpha^3 y)$. Furthermore, we assume that $\mathbb{F}_{p^k} = \mathbb{F}_{p^e}(\alpha)$, and we have $\alpha^d = \omega \in \mathbb{F}_{p^e}$, where d is the degree of the twist. Each element in $\mathbb{F}_{p^k}$ is given by a polynomial of degree $d-1$ in α with coefficients in $\mathbb{F}_{p^e}$.

Let $P \in E(\mathbb{F}_p)$, $R' \in E'(\mathbb{F}_{p^e})$ and $R = \psi(R')$. Let $R_3 = [2]R = (x_{R_3}, y_{R_3})$. Let $\ell_1 = \ell_{R,R}(P)$, and $\ell_2 = \ell_{R_3,R_3}(P)$. In the following computation, we use the abscissas of the point $-R_3$ instead of that of the point R in the line function ℓ_1 passing points R and $-R_3$. We also compute $\frac{\ell_1 \cdot \ell_2}{x_P - x_{R_3}}$ instead of $\ell_1 \cdot \ell_2$. Note that, for even twisted curves, the factor $x_P - x_{R_3}$ is in the proper subfield, thus we can make this division without changing the final result of Tate pairing. Two consecutive doubling steps are performed as follows:

$$\begin{aligned}
\ell = \frac{\ell_1 \cdot \ell_2}{x_P - x_{R_3}} &= \frac{(y_P + y_{R_3} - \lambda_1(x_P - x_{R_3}))(y_P - y_{R_3} - \lambda_2(x_P - x_{R_3}))}{x_P - x_{R_3}} \\
&= \frac{y_P^2 - y_{R_3}^2}{x_P - x_{R_3}} - \lambda_1(y_P - y_{R_3}) - \lambda_2(y_P + y_{R_3}) + \lambda_1\lambda_2(x_P - x_{R_3}) \\
&= x_P^2 + x_P x_{R_3} + x_{R_3}^2 + a - \lambda_1(y_P - y_{R_3}) - \lambda_2(y_P + y_{R_3}) + \lambda_1\lambda_2(x_P - x_{R_3}),
\end{aligned}$$

where λ_1, λ_2 are slopes when computing $[2]R$ and $[4]R$. Let $R'_3 = [2]R' = (x_{R'_3}, y_{R'_3})$. The details of computations is as follows:

$$\begin{aligned}\ell = \ell_1 \cdot \ell_2 &= \ell_{\psi(R'),\psi(R')}(P) \cdot \ell_{\psi([2]R'),\psi([2]R')}(P) = x_P^2 + \alpha^2 x_{R'_3} x_P + \alpha^4 x_{R'_3}^2 \\ &\quad + a - \alpha\lambda'_1(y_P - \alpha^3 y_{R'_3}) - \alpha\lambda'_2(y_P + \alpha^3 y_{R'_3}) + \alpha^2\lambda'_1\lambda'_2(x_P - \alpha^2 x_{R'_3}) \\ &= x_P^2 + a - \alpha(\lambda'_1 + \lambda'_2)y_P + \alpha^2(x_{R'_3} + \lambda'_1\lambda'_2)x_P + \alpha^4(x_{R'_3}^2 + (\lambda'_1 + \lambda'_2)y_{R'_3} + \lambda'_1\lambda'_2 x_{R'_3}),\end{aligned}$$

where $\lambda_1 = \alpha\lambda'_1$, and $\lambda_2 = \alpha\lambda'_2$. Since P is fixed throughout the computation, we assume that value x_P^2 is precomputed, the costs of updating ℓ are summarized in the following table. Note that, we use the same notations for field arithmetic costs as in [22]. Notations $\mathbf{I}_{p^e}$, $\mathbf{M}_{p^e}$, $\mathbf{S}_{p^e}$, $\mathbf{add}_{p^e}$, $\mathbf{sub}_{p^e}$, $\mathbf{neg}_{p^e}$ denote the costs for inversion, multiplication, squaring, addition, subtraction, and negation in the field $\mathbb{F}_{p^e}$, where $e = k/d$. The cost for a multiplication by a constant ω is denoted by $\mathbf{M}_{(\omega)}$.

Table 1. Number of operations for updating two consecutive line function values

	$\mathbf{M}_p$	$\mathbf{M}_{p^e}$	$\mathbf{S}_{p^e}$	$\mathbf{M}_{(\omega)}$	$\mathbf{add}_{p^e}$	$\mathbf{neg}_{p^e}$
$d = 2$	k	3	1	2	6	1
$d = 4$	$k/2$	3	1	1	5	1
$d = 6$	$k/3$	3	1	-	4	1

Fast quadrupling. Let $R'_4 = [4]R' = (x_{R'_4}, y_{R'_4})$. Traditionally, R'_4 can be obtained using two repeated doublings that require 2 field inversions. In [23], Le introduced fast algorithms for quadrupling a point on elliptic curves in affine coordinates. His algorithm requires $1\mathbf{I}_{p^e} + 8\mathbf{S}_{p^e} + 8\mathbf{M}_{p^e}$, and is better than two repeated doublings whenever $\mathbf{I}_{p^e} > 4\mathbf{M}_{p^e} + 4\mathbf{S}_{p^e}$. It performs even better for curves that allow "$a = 0$" speedup (found in pairing-friendly elliptic curves admitting twists of degrees 2, 3, or 6) as $[4]R'$ in affine coordinates can be computed just using only $1\mathbf{I}_{p^e} + 5\mathbf{S}_{p^e} + 6\mathbf{M}_{p^e}$. This section presents the revised formula for point quadrupling that requires fewer additions in comparison to that in [23] for pairing computation over curves with $a = 0$. Let $d = y_{R'}^4 + 18by_{R'}^2 - 27b^2$, $I = \frac{3x_{R'}^2}{2y_{R'}d}$, and ℓ is the product of two consecutive line function values as described above. One also can precompute and cache values $s = 18b$ and $t = 27b^2$.

This quadrupling formula only requires $1\mathbf{I}_{p^e} + 6\mathbf{M}_{p^e} + 5\mathbf{S}_{p^e} + e\mathbf{M}_p + 8\mathbf{add}_{p^e} + 9\mathbf{sub}_{p^e}$. If an inversion in $\mathbb{F}_{p^e}$ is more than $2\mathbf{M}_{p^e} + 1\mathbf{S}_{p^e} + e\mathbf{M}_p + 1\mathbf{sub}_{p^e}$, then the new quadrupling formula is faster than two repeated doublings. In the case of curves with a twist of degree 4 (i.e., $y^2 = x^3 + ax$), a similar quadrupling can be performed by $1\mathbf{I}_{p^e} + 9\mathbf{M}_{p^e} + 5\mathbf{S}_{p^e} + 14\mathbf{add}_{p^e} + 10\mathbf{sub}_{p^e}$.

Table 2 summarizes and compares the costs of our technique to those from [22] in affine coordinates and [10] in projective coordinates. Again, we assume that all values that depend only on fixed parameters, are precomputed and cached, and small multiples are computed by additions.

$$\begin{array}{c}
\lambda_1 = I \cdot d, \quad \lambda_2 = I(y_R^2 - 9b)^2/2 \\
x_{R'_3} = \lambda_1^2 - 2x_R, \quad y_{R'_3} = \lambda_1(x_R - x_{R'_3}) - y_R, \\
x_{R'_4} = \lambda_2^2 - 2x_{R'_3}, \quad y_{R'_4} = \lambda_2(x_{R'_3} - x_{R'_4}) - y_{R'_4}, \\
\ell = \ell_1 \cdot \ell_2 = \ell_{\psi(R'),\psi(R')}(P) \cdot \ell_{\psi(R'_3),\psi(R'_3)}(P),
\end{array}$$

$$\begin{array}{c}
A = y_{R'}^2, \quad B = A^2, \quad C = 3x_{R'}^2, \quad d = B + sA + t, \\
D = 2dy_{R'}, \quad E = D^{-1}, \quad I = C \cdot E, \quad \lambda_1 = I \cdot d, \\
x_{R'_3} = \lambda_1^2 - 2x_{R'}, \quad y_{R'_3} = \lambda_1(x_{R'} - x_{R'_3}) - y_{R'}, \quad \lambda_2 = \frac{(B - sA + 3t)I}{2}, \\
x_{R'_4} = \lambda_2^2 - 2x_{R'_3}, \quad y_{R'_4} = \lambda_2(x_{R'_3} - x_{R'_4}) - y_{R'_3}, \\
\ell = \ell_1 \cdot \ell_2 = \ell_{\psi(R'),\psi(R')}(P) \cdot \ell_{\psi(R'_3),\psi(R'_3)}(P)
\end{array}$$

Table 2. Operation counts for two doubling steps in the Ate-like Miller loop

d	Technique	$\mathbf{M}_p$	$\mathbf{M}_{p^k}$	$\mathbf{I}_{p^e}$	$\mathbf{M}_{p^e}$	$\mathbf{S}_{p^e}$	$\mathbf{M}_{(\cdot)}$	$\mathbf{add}_{p^e}$	$\mathbf{sub}_{p^e}$	$\mathbf{neg}_{p^e}$
2	Ours	$5k/2$	1	1	9	6	2	14	9	1
$(a = 0)$	Lauter *et al.* [22]	k	2	2	6	4	2	8	12	-
	Costello *et al.* [10]	$2k$	1	-	14	16	4	60	24	2
4	Ours	k	1	1	12	6	1	19	10	1
$(b = 0)$	Lauter *et al.* [22]	$k/2$	2	2	6	4	-	8	10	2
	Costello *et al.* [10]	k	1	-	11	20	3	55	27	2
6	Ours	$5k/6$	1	1	9	6	-	12	9	1
$(a = 0)$	Lauter *et al.* [22]	$k/3$	2	2	6	4	-	8	10	2
	Costello *et al.* [10]	$2k/3$	1	-	14	16	4	60	24	2

As showed in Table 2, the costs of two doubling steps on curves having a twist of degree $d = 2$ requires $\frac{5k}{2}\mathbf{M}_p + 1\mathbf{M}_{p^k} + 1\mathbf{I}_{p^{k/2}} + 9\mathbf{M}_{p^{k/2}} + 6\mathbf{S}_{p^{k/2}} + 2\mathbf{M}_{(\omega)} + 14\mathbf{add}_{p^{k/2}} + 9\mathbf{sub}_{p^{k/2}} + 1\mathbf{neg}_{p^{k/2}}$, while analysis in [22] require $k\mathbf{M}_p + 2\mathbf{M}_{p^k} + 2\mathbf{I}_{p^{k/2}} + 6\mathbf{M}_{p^{k/2}} + 4\mathbf{S}_{p^{k/2}} + 2\mathbf{M}_{(\omega)} + 8\mathbf{add}_{p^{k/2}} + 12\mathbf{sub}_{p^{k/2}}$. If $1\mathbf{M}_{p^k} + 1\mathbf{I}_{p^{k/2}} > 3\mathbf{M}_{p^{k/2}} + 2\mathbf{S}_{p^{k/2}} + 3\mathbf{add}_{p^{k/2}} + \mathbf{neg}_{p^{k/2}}$, then our technique is better.

4 Improvements for the Cubic Twisted Curves

4.1 Updating Miller Function

Pairing computation over cubic twisted curves with embedding degrees 9 or 15 were investigated in papers [25][12][11]. Although such curves only admit a cubic twist $d = 3$, and there exists no full denominators elimination technique, but they provide a shorter Miller loop. In [30], Scott pointed out that in the contexts of multi-pairings in conjunction with fixed arguments, these curves have more advantages than curves admitting a higher twist (i.e., 4 or 6). This section gives the first analysis about the costs of Miller's algorithm in such curves in affine coordinates.

Recall that cubic twisted curves have the form $y^2 = x^3 + b$. In [25], Lin *et al.* proposed a denominators elimination trick during Ate pairing computation on a $k = 9$ curve due to the following observation about the factor $1/v_{R+Q}(P)$:

$$\frac{1}{v_{R+Q}(P)} = \frac{1}{x_P - x_{R+Q}} = \frac{x_P^2 + x_P x_{R+Q} + x_{R+Q}^2}{(y_P - y_{R+Q})(y_P + y_{R+Q})}$$

Since $(y_P - y_{R+Q})(y_P + y_{R+Q})$ lies in a subfield when the curve admits a cubic twist, f function can be updated by multiplying by $x_P^2 + x_P x_{R+Q} + x_{R+Q}^2$ instead of dividing it by $v_{R+Q}(P)$. The updated factor is :

$$\ell'_{R,Q}(P) = (y_P - \lambda(x_P - x_{R+Q}) - y_{R+Q}) \cdot (x_P^2 + x_P x_{R+Q} + x_{R+Q}^2) \qquad (2)$$

where λ is the slope of the line function passing points R and Q. This formula needs one full extension field multiplication than the full denominators elimination technique of Barreto et al [2]. The following lemma allows us to save one multiplication in the full extension field in comparison to the analysis in [25].

Lemma 1. *For elliptic curves admitting a cubic twist, the rational function $g_{R,Q}(P)$ in Miller's algorithm can be rewritten as follows:*

$$g_{R,Q}(P) = \frac{\ell_{R,Q}(P)}{v_{R+Q}(P)} = \frac{x_{R+Q}^2 + x_{R+Q}x_P + x_P^2 - \lambda(y_P + y_{R+Q})}{y_P - y_{R+Q}} \qquad (3)$$

Proof. For the line function $\ell_{R,Q}(P)$, using the coordinates of the point $-(R+Q)$ instead of that of R, we have:

$$\begin{aligned}\frac{\ell_{R,Q}(P)}{v_{R+Q}(P)} &= \frac{y_P - \lambda(x_P - x_{R+Q}) + y_{R+Q}}{x_P - x_{R+Q}} \\ &= -\lambda + \frac{y_P^2 - y_{R+Q}^2}{(x_P - x_{R+Q})(y_P - y_{R+Q})} = -\lambda + \frac{x_P^3 - x_{R+Q}^3}{(x_P - x_{R+Q})(y_P - y_{R+Q})} \\ &= \frac{-\lambda(y_P - y_{R+Q}) + x_{R+Q}^2 + x_{R+Q}x_P + x_P^2}{y_P - y_{R+Q}}\end{aligned}$$

The factor $(y_P - y_{R+Q})$ lying in a proper subfield of $\mathbb{F}_{p^k}$ can be cancelled out. The actual updated factor is $x_{R+Q}^2 + x_{R+Q}x_P + x_P^2 - \lambda(y_P - y_{R+Q})$. The computation of this updated factor doesn't require one more multiplication in the full extension field and it is much faster than that given in [25].

Doubling step. Let the notations be described in Section 3. Let $e = k/3$, and let $\nu \in \mathbb{F}_{p^k}$ be not a cubic residue but a quadratic residue over $\mathbb{F}_{p^e}$, and $\nu^{1/2} = \omega \in \mathbb{F}_{p^e}$. Furthermore, we assume that $\mathbb{F}_{p^k} = \mathbb{F}_{p^e}(\nu^{1/6})$, i.e., each element in $\mathbb{F}_{p^k}$ can be represented by a polynomial $A + B\nu^{1/6} + C\nu^{1/3}$, where $A, B, C \in \mathbb{Z}_{p^e}$. Let the twisted curve is of the form $E' : y^2 = x^3 + b/\nu$. There exists an isomorphism $\psi : E'(\mathbb{F}_{p^e}) \to E(\mathbb{F}_{p^k})$, $(x, y) \mapsto (\nu^{1/3}x, \nu^{1/2}y)$. Let $P \in \mathbb{G}_1$, $R', Q' \in \mathbb{G}'_2$, and let $R = \psi(R')$, $Q = \psi(Q')$, where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}'_2$ are defined as in Section 2.1.

As showed in Lemma 1, the computation of the line functions need squarings of x-coordinates. This implies that a new coordinate (x, y, z), where $z = x^2$

matches the computation. Let $R'_3 = [2]R'$. Doubling steps can be computed as follows:

$$\begin{aligned}\ell_{R,R}(P) &= \nu^{2/3}x_{R'_3}^2 + \nu^{1/3}x_{R'_3}x_P + x_P^2 - \lambda(y_P - \nu^{1/2}y_{R'_3}) \\ &= z_P - \nu^{1/6}\lambda' y_p + \nu^{1/3}x_{R'_3}x_P + \nu^{2/3}(z_{R'_3} + \lambda' y_{R'_3}) \\ &= z_P + \nu^{1/6}(\omega(z_{R'_3} + \lambda' y_{R'_3}) - \lambda' y_p) + \nu^{1/3}x_{R'_3}x_P,\end{aligned}$$

where $x_{R'_3} = \lambda'^2 - 2x_{R'}$, $y_{R'_3} = \lambda'(x_{R'} - x_{R'_3}) - y_{R'}$ and $z_{R'_3} = x_{R'_3}^2$. We have $\lambda' = 3x_{R'}^2/2y_{R'} = 3z_{R'}/2y_{R'}$ and $\lambda = 3x_R^2/2y_R = \nu^{1/6}\lambda'$.

The double of R' needs $\mathbf{I}_{p^e} + 2\mathbf{M}_{p^e} + 2\mathbf{S}_{p^e} + 3\mathbf{add}_{p^e} + 4\mathbf{sub}_{p^e}$, where the computation of the slope λ' need $\mathbf{I}_{p^e} + \mathbf{M}_{p^e} + 3\mathbf{add}_{p^e}$. Assume that the multiplication of elements in $\mathbb{F}_{p^e}$ with a small constant (e.g., $3z_{R'}$, $2y_{R'}$) is computed by additions. Then, we need $2e\mathbf{M}_p + \mathbf{M}_{p^e} + \mathbf{M}_{(\omega)} + \mathbf{add}_{p^e} + \mathbf{sub}_{p^e}$ to compute the line function value. In total, our new formula requires $2e\mathbf{M}_p + \mathbf{I}_{p^e} + 3\mathbf{M}_{p^e} + 2\mathbf{S}_{p^e} + \mathbf{M}_{(\omega)} + 4\mathbf{add}_{p^e} + 5\mathbf{sub}_{p^e}$ for each doubling step.

Addition step. The line function is computed similarly as in doubling steps.

$$\ell_{R,R}(P) = z_P + \nu^{1/6}(\omega(z_{R'_3} + \lambda' y_{R'_3}) - \lambda' y_p) + \nu^{1/3}x_{R'_3}x_P,$$

where $R'_3 = R' + Q'$, and $x_{R'_3} = \lambda'^2 - x_{R'} - x_{Q'}$, $y_{R'_3} = \lambda'(x_{R'} - x_{R'_3}) - y_{R'}$ and $z_{R'_3} = x_{R'_3}^2$. The slope $\lambda' = (y_{R'} - y_{Q'})/(x_{R'} - x_{Q'})$. We have $\lambda = (y_R - y_Q)/(x_R - x_Q) = \nu^{1/6}\lambda'$.

Computation of the line function in addition steps has the same cost as in the doubling steps. It needs $\mathbf{I}_{p^e} + \mathbf{M}_{p^e} + 2\mathbf{sub}_{p^e}$ for computing the slope λ' and $\mathbf{M}_{p^e} + 2\mathbf{S}_{p^e} + 4\mathbf{sub}_{p^e}$ for computing the addition of R' and Q' from the slope λ'. In total, we need $2e\mathbf{M}_p + \mathbf{I}_{p^e} + 3\mathbf{M}_{p^e} + 2\mathbf{S}_{p^e} + \mathbf{M}_{(\omega)} + \mathbf{add}_{p^e} + 7\mathbf{sub}_{p^e}$ for each addition step.

We summarize the number of operations required by the Miller loop over cubic twisted curves in Table 3. We also make a comparison on the number of operations between affine coordinates and projective coordinates taken from [11].

Table 3. Number of operations in the Ate-like Miller loop over cubic twisted curves

	coord.	$\mathbf{M}_p$	$\mathbf{I}_{p^e}$	$\mathbf{M}_{p^e}$	$\mathbf{S}_{p^e}$	$\mathbf{M}_{(\cdot)}$	$\mathbf{add}_{p^e}$	$\mathbf{sub}_{p^e}$	$\mathbf{neg}_{p^e}$
DBL	affine	$2k/3$	1	3	2	$\mathbf{M}_{(\omega)}$	4	5	-
	proj. [11]	k	-	6	7	$\mathbf{M}_{(b/\omega)}$	11	10	1
ADD	affine	$2k/3$	1	3	2	$\mathbf{M}_{(\omega)}$	1	7	-
	proj. [11]	k	-	13	3	-	6	8	3

The above analysis showed that the number of operations in doubling steps over cubic twisted curves is similar to that over even twisted curves as analyzed in [22]. Addition steps require only $1\mathbf{S}_{p^e} + 1\mathbf{M}_{(\omega)}$ more than that for even twisted curves. Table 3 also showed that the doubling steps in affine coordinates are better than that in projective coordinates [11] if:

$$\mathbf{I}_{p^e} \leq e\mathbf{M}_p + 3\mathbf{M}_{p^e} + 5\mathbf{S}_{p^e} + 7\mathbf{add}_{p^e} + 5\mathbf{sub}_{p^e} + \mathbf{neg}_{p^e}, \tag{4}$$

where $e = k/3$.

Example 1. In the case of $k = 9$, we can obtain pairing-friendly elliptic curves of form $y^2 = x^3 + b$ admitting a cubic twist [25]. During Ate pairing computation, point operations are performed over $\mathbb{F}_{p^3}$ (i.e., $e = 3$). Analysis in [21, §5.1] showed that inversion over $\mathbb{F}_{p^3}$ needs 12 multiplications and one inversion over $\mathbb{F}_p$. If the inversion-to-multiplication ratio is around 13 as benchmarks in [22] and is used in this analysis, the cost of one inversion over $\mathbb{F}_{p^3}$ is around 25 multiplications over $\mathbb{F}_p$. Obviously, this cost is much less than $3\mathbf{M}_p + 3\mathbf{M}_{p^3} + 5\mathbf{S}_{p^3} \approx 21\mathbf{M}_p + 30\mathbf{S}_p$ (from Eq. 4). Note that using Karatsuba algorithm, $\mathbf{M}_{p^3} \approx 6\mathbf{M}_p$ and $\mathbf{S}_{p^3} \approx 6\mathbf{S}_p$.

4.2 Choice of Curves

In this section, we present efficient subfamilies of pairing-friendly elliptic curves with embedding degrees $k = 9, 15$ presented in [25] and [12].

The family of curves with $k = 9$ is described by the following polynomials:

$$\begin{aligned} p(x) &= ((x+1)^2 + ((x-1)^2(2x^3+1)^2)/3)/4, \\ r(x) &= (x^6 + x^3 + 1)/3, \\ n(x) &= (x-1)^2(x^6 + x^3 + 1)/3, \\ t(x) &= x + 1, \end{aligned} \tag{5}$$

where $t(x)$ is the trace of Frobenius, $p(x)$ represents the field size and $r(x)$ represents the pairing-friendly subgroup. In comparison to BN curves at 128-bit security level [4], this family supports a shorter Miller loop. But, BN curves provide a much more efficient tower extension field arithmetic.

El Mrabet *et al.* [12] introduced a family of pairing-friendly elliptic curve of embedding degree $k = 15$ and compared its performance with KSS curves [18] at 192-bit security level. Their family of curves is described as follows:

$$\begin{aligned} p(x) &= (x^{12} - 2x^{11} + x^{10} + x^7 - 2x^6 + x^5 + x^2 + x + 1)/3, \\ r(x) &= x^8 - x^7 + x^5 - x^4 + x^3 - x + 1, \\ n(x) &= (x-1)^2(x^2 + x + 1)(x^8 - x^7 + x^5 - x^4 + x^3 - x + 1)/3, \\ t(x) &= x + 1. \end{aligned} \tag{6}$$

For both families of curves, the ρ-value is equal to $4/3$ and the elliptic curves are of the form $y^2 = x^3 + b$. By using the above parameters when $x_0 \equiv 1 \pmod 3$, one is able to get all involved parameters being integers and construct a curve. The following theorem show that by choosing $x \equiv 1 \pmod 6$, we always choose the curve constant b equal to 1. That means that the multiplications with b is free.

Lemma 2. *Let $E : y^2 = x^3 + b$ be an elliptic curve defined over $\mathbb{F}_p$ where p prime and $p \equiv 1 \pmod 6$. Let $\#E(\mathbb{F}_p) = n$. If $2 \mid n$ and $3 \mid n$, then b is both a square and a cube in $\mathbb{F}_p$.*

Proof. The proof of Lemma 2 can be found in Appendix A.

Theorem 1. *By choosing $x_0 \equiv 1 \pmod 6$ for both above families of curves with embedding degrees $k = 9$ or 15, the desired curve is always of form $E(\mathbb{F}_p) : y^2 = x^3 + 1$.*

Proof. In [33, §X.5], Silverman showed that an curve defined over $\mathbb{F}_p$ with the *j invariant* $j(E) = 0$ (i.e. the curves of the form $y^2 = x^3 + b$) will only have six possible curve orders. More precisely, the CM construction only ensures that the order of a curve satisfying the norm equation $3y^2 = 4p - t^2$ has one of the six forms $\{p+1 \pm t, p+1 \pm (t \pm 3y)/2\}$. Moreover, assume that γ be both quadratic and cubic non-residue modulo p, these possible group orders occur as the order of one of the 6 twists with $b \in \{1, \gamma, \gamma^2, \gamma^3, \gamma^4, \gamma^5\}$.

For $x_0 \equiv 1 \pmod 6$ in (5) (and (6), resp.), is is easy to see that $n_0 = (x_0 - 1)^2(x_0^6 + x_0^3 + 1)/3$ ($n_0 = (x_0 - 1)^2(x_0^2 + x_0 + 1)(x_0^8 - x_0^7 + x_0^5 - x_0^4 + x_0^3 - x_0 + 1)/3$, resp.) is congruent to 0 modulo 6, i.e., $2|n_0$ and $3|n_0$. It is also easy to verify that $p(x_0) \equiv 1 \pmod 6$. From Lemma 2, b must be both a square and a cube in $\mathbb{F}_p$, it follows that $b = 1$ is the only option.

4.3 Final Exponentiation

After the main Miller loop, the Tate pairing (and its variants) must carry out the final exponentiation to ensure a unique result of the pairing. The output of the Miller loop f must be raised to be power of $(p^k - 1)/r$ to obtain a result of order r. Scott et al. [32] introduced an efficient algorithm to compute such a computation. Their algorithm splits the final exponentiation into two parts: the first part is "easy" as raising to the power of p is an almost free application of the Frobenius operator; the second so-called "hard" is to power of $\Phi_k(p)/r \in \mathbb{F}_p[x]$. The exponent of the hard part can be expanded to the base p as $a_{n-1}p^{n-1} + \cdots + a_1 p + a_0$, where $n = \varphi(k)$, the Euler-phi function. We refer readers to [32] for more details about this computation.

In this section, we give an efficient version of the hard part in final exponentiation for curves of embedding degrees $k = 9$. In the case of $k = 15$, readers can see in Appendix B.

In the case of $k = 9$. By setting $x = 6u + 1$, we obtain the new explicit polynomials as follows :

$$
\begin{aligned}
t(u) &= 6u + 2,\\
p(u) &= 559872u^8 + 559872u^7 + 233280u^6 + 54432u^5 + 7776u^4 + 648u^3 + 36u^2 + 6u + 1,\\
r(u) &= 15552u^6 + 15552u^5 + 6480u^4 + 1512u^3 + 216u^2 + 18x + 1.
\end{aligned}
$$

The cost of the final exponentiation for the Ate pairing on curves with $k = 9$ was analyzed by Lin *et al.* [25]. Let the hard part $\frac{p(u)^6+p(u)^3+1}{r(u)} = \sum_{i=0}^{5} a_i(u)p(u)^i$, where $a_i(u)$ are following explicit polynomials (see in [25, §6.1]):

$$
\begin{aligned}
a_5 &= 36u^2,\\
a_4 &= 216u^3 + 36u^2 = a_5(6u+1),\\
a_3 &= 1296u^4 + 432u^3 + 36u^2 = a_4(6u+1),\\
a_2 &= 7776u^5 + 3888u^4 + 648u^3 + 72u^2 = a_3(6u+1) + a_5,\\
a_1 &= 46656u^6 + 31104u^5 + 7776u^4 + 1080u^3 + 72u^2 = a_2(6u+1),\\
a_0 &= 279936u^7 + 233280u^6 + 77760u^5 + 14256u^4 + 1512u^3 + 72u^2 + 3 = a_1(6u+1) + 3.
\end{aligned}
\tag{7}
$$

Their calculation requires $65\mathbf{M}_{p^9} + 375\mathbf{S}_{p^9} + 45\mathbf{M}_p$ for computing this hard part (see [25, Section 6.2]). The following computation allows us to save $15\mathbf{M}_{p^9} + 66\mathbf{S}_{p^9} + 45\mathbf{M}_p$.

Let $T = t - 1$, where $t = 6u + 2$ is the trace of Frobenius. Furthermore, let f be the output of Miller algorithm, and $m = f^{p^3-1}$ (i.e., easy part). We compute the hard part as follows:

$$m^{\frac{p(u)^6+p(u)^3+1}{r(u)}} = \mu_0 \cdot \mu_1^p \cdot \mu_2^{p^2} \cdot \mu_3^{p^3} \cdot \mu_4^{p^4} \cdot \mu_5^{p^5},$$

where μ_i can be computed as follows:

$$\mu_5 = (m^{T-1})^{T-1},\ \mu_4 = (\mu_5)^T,\ \mu_3 = (\mu_4)^T,$$
$$\mu_2 = (\mu_3)^T \cdot \mu_5,\ \mu_1 = (\mu_2)^T,\ \mu_0 = (\mu_1)^T \cdot m^3.$$

This part requires 7 exponentiations by T or $T - 1$, 8 multiplications and one squaring in $\mathbb{F}_{p^9}$, and 5 p-power Frobenius operations. Let T be a number of 44 bits length and Hamming weight of T is 7 (as the example given in [25]). This part requires $2(44\mathbf{S}_{p^9} + 6\mathbf{M}_{p^9}) + 5(44\mathbf{S}_{p^9} + 6\mathbf{M}_{p^9}) + 8\mathbf{M}_{p^9} + 1\mathbf{S}_{p^9} = 309\mathbf{S}_{p^9} + 50\mathbf{M}_{p^9}$. We save $66\mathbf{S}_{p^9} + 15\mathbf{M}_{p^9} + 45\mathbf{M}_p$ in comparison to computations in [25].

4.4 Discussion

At 128-bit security level, the current public-key security recommendations, Barreto-Naehrig curves [4] lead a very efficient implementation. Many results have been reported in papers [28][5][29][1]. That is because BN curves can exploit a sextic twist and there exist efficient algorithms for squarings in $\mathbb{F}_{p^{12}}$ [15][19]. The former allows us to work on points of the twisted curve whose coordinates are in $\mathbb{F}_{p^2}$ instead of $\mathbb{F}_{p^{12}}$ during Miller loop computation. The later provides an efficient speedup for the final exponentiation step.

In [25] the authors consider curves with $k = 9$ at 128-bit security level. One advantage of such a curve compared with BN curve is that it will have an Ate pairing with 2/3 Miller loop length compared with the BN equivalent. With many optimizations in both Miller loop and the final exponentiation, BN curves are perfectly suited for implementing a single pairing. However, when we need to compute several pairings in parallel, where only one final exponentiation required to compute, curves with shorter Miller loop may offer a good choice. Our above analysis allowing to speed up pairing computation over cubic twisted curves in affine coordinates for both Miller loop and final exponentiation, are helpful for this case.

5 Conclusion

In this paper we further analyzed techniques to speed up Ate pairing computation in affine coordinates using 4-ary Miller algorithm. We focused on pairing computations over pairing-friendly elliptic curves admitting a cubic twist and presented the first and fast explicit formulas in affine coordinates for such curves. We also gave a finer choice for curves of embedding degrees $k = 9, 15$, and show that this choice leads to an efficient arithmetic and final exponentiation.

Acknowledgement. The authors thank the anonymous referees for their useful comments and suggestions.

References

1. Aranha, D.F., Karabina, K., Longa, P., Gebotys, C.H., López, J.: Faster explicit formulas for computing pairings over ordinary curves. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 48–68. Springer, Heidelberg (2011)
2. Barreto, P.S.L.M., Kim, H.Y., Lynn, B., Scott, M.: Efficient algorithms for pairing-based cryptosystems. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 354–368. Springer, Heidelberg (2002)
3. Barreto, P.S.L.M., Lynn, B., Scott, M.: On the selection of pairing-friendly groups. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 17–25. Springer, Heidelberg (2004)
4. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006)
5. Beuchat, J.-L., González-Díaz, J.E., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-speed software implementation of the optimal Ate pairing over barreto-naehrig curves. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 21–39. Springer, Heidelberg (2010)
6. Blake, I.F., Murty, V.K., Xu, G.: Refinements of Miller's algorithm for computing the Weil/Tate pairing. J. Algorithms 58(2), 134–149 (2006)
7. Boneh, D., Franklin, M.: Identity-Based Encryption from the Weil Pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
8. Boxall, J., El Mrabet, N., Laguillaumie, F., Le, D.-P.: A variant of miller's formula and algorithm. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 417–434. Springer, Heidelberg (2010)
9. Brezing, F., Weng, A.: Elliptic curves suitable for pairing based cryptography. Des. Codes Cryptography 37, 133–141 (2005)

10. Costello, C., Boyd, C., González Nieto, J.M., Wong, K.K.-H.: Avoiding full extension field arithmetic in pairing computations. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 203–224. Springer, Heidelberg (2010)
11. Costello, C., Lange, T., Naehrig, M.: Faster Pairing Computations on Curves with High-Degree Twists. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 224–242. Springer, Heidelberg (2010)
12. El Mrabet, N., Guillermin, N., Ionica, S.: A study of pairing computation for elliptic curves with embedding degree 15. Cryptology ePrint Archive, Report 2009/370 (2009), `http://eprint.iacr.org/`
13. Freeman, D., Scott, M., Teske, E.: A Taxonomy of Pairing-Friendly Elliptic Curves. J. Cryptol. 23, 224–280 (2010)
14. Galbraith, S.D., Harrison, K., Soldera, D.: Implementing the tate pairing. In: Fieker, C., Kohel, D.R. (eds.) ANTS 2002. LNCS, vol. 2369, pp. 324–337. Springer, Heidelberg (2002)
15. Granger, R., Scott, M.: Faster squaring in the cyclotomic subgroup of sixth degree extensions. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 209–223. Springer, Heidelberg (2010)
16. Hess, F., Smart, N.P., Vercauteren, F.: The eta pairing revisited. IEEE Transactions on Information Theory 52, 4595–4602 (2006)
17. Joux, A.: A One Round Protocol for Tripartite Diffie-Hellman. In: Bosma, W. (ed.) ANTS-IV. LNCS, vol. 1838, pp. 385–394. Springer, Heidelberg (2000)
18. Kachisa, E.J., Schaefer, E.F., Scott, M.: Constructing Brezing-Weng Pairing-Friendly Elliptic Curves Using Elements in the Cyclotomic Field. In: Galbraith, S.D., Paterson, K.G. (eds.) Pairing 2008. LNCS, vol. 5209, pp. 126–135. Springer, Heidelberg (2008)
19. Karabina, K.: Squaring in cyclotomic subgroups. Cryptology ePrint Archive, Report 2010/542 (2010), `http://eprint.iacr.org/`
20. Kobayashi, T., Aoki, K., Imai, H.: Efficient algorithms for tate pairing. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. E89-A(1), 134–143 (2006)
21. Kobayashi, T., Morita, H., Kobayashi, K., Hoshino, F.: Fast elliptic curve algorithm combining frobenius map and table reference to adapt to higher characteristic. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 176–189. Springer, Heidelberg (1999)
22. Lauter, K., Montgomery, P.L., Naehrig, M.: An analysis of affine coordinates for pairing computation. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 1–20. Springer, Heidelberg (2010)
23. Le, D.P.: Fast Quadrupling of a Point in Elliptic Curve Cryptography. Cryptology ePrint Archive, Report 2011/039 (2011), `http://eprint.iacr.org/`
24. Le, D.P., Liu, C.L.: Refinements of Miller's Algorithm over Weierstrass Curves Revisited. The Computer Journal 54(10), 1582–1591 (2011)
25. Lin, X., Zhao, C.A., Zhang, F., Wang, Y.: Computing the Ate pairing on elliptic curves with embedding degree k = 9. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. E91-A(9), 2387–2393 (2008)
26. Miller, V.S.: The Weil Pairing, and Its Efficient Calculation. Journal of Cryptology 17(4), 235–261 (2004)
27. Miyaji, A., Nakabayashi, M., Takano, S.: New Explicit Conditions of Elliptic Curve Traces for FR-Reduction. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 84(5), 1234–1243 (2001)
28. Naehrig, M., Niederhagen, R., Schwabe, P.: New software speed records for cryptographic pairings. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 109–123. Springer, Heidelberg (2010)

29. Pereira, G.C.C.F., Simplício, J.M.A., Naehrig, M., Barreto, P.S.L.M.: A family of implementation-friendly bn elliptic curves. J. Syst. Softw. 84, 1319–1326 (2011)
30. Scott, M.: On the efficient implementation of pairing-based protocols. In: Chen, L. (ed.) IMACC 2011. LNCS, vol. 7089, pp. 296–308. Springer, Heidelberg (2011)
31. Scott, M., Barreto, P.S.L.M.: Compressed pairings. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 140–156. Springer, Heidelberg (2004)
32. Scott, M., Benger, N., Charlemagne, M., Dominguez Perez, L.J., Kachisa, E.J.: On the final exponentiation for calculating pairings on ordinary elliptic curves. In: Shacham, H., Waters, B. (eds.) Pairing 2009. LNCS, vol. 5671, pp. 78–88. Springer, Heidelberg (2009)
33. Joseph, H.: Silverman: The Arithmetic of Elliptic Curves, 2nd edn. Springer (May 2009)
34. Vercauteren, F.: Optimal pairings. IEEE Transactions on Information Theory 56(1), 455–461 (2010)

A Proof of Lemma 2

Proof. Assume that $2 \mid n$, then the elliptic curve $E : y^2 = x^3 + b$ contains points of order 2. Let $P = (x_1, y_1) \in E$ be a point having order 2. The tangent at P meets $\mathcal{O}$ and hence $\left(\frac{dy}{dx}\right)_P = \infty$ or $y_1 = 0$. We have $y_1^2 = x_1^3 + b$, and hence $b = -x_1^3$ or b is a cube in $\mathbb{F}_p$.

When $3 \mid n$, E contains points of order 3. Assume that $P = (x_1, y_1)$ has order 3, that means $3P = \mathcal{O}$ or $2P = -P$. Let $Q = [2]P = (x_2, y_2)$. Then we have $x_2 = x_1$ which implies $\lambda^2 - 2x_1 = x_1,$ where $\lambda = 3x_1^2/2y_1$. Therefore, we obtain $9x_1(y_1^2 - b) = 12x_1y_1^2$, so that $x_1 = 0$ or $y_1^2 = -3b$.

In the former case $x_1 = 0$, it is easy to verify that the point $0, \delta$ has order 3 for some δ, and $b = \delta^2$ or b is a square in $\mathbb{F}_p$. For the later case $-3b = y_1^2$, to prove b square in $\mathbb{F}_p$ we need to show that -3 is a square in $\mathbb{F}_p$. We consider the Legendre symbol:

$$\left(\frac{-3}{p}\right) = \left(\frac{-1}{p}\right)\left(\frac{3}{p}\right) = (-1)^{\frac{p-1}{2}} \times (-1)^{\lfloor \frac{p+1}{6} \rfloor}.$$

– If $p \equiv 1 \pmod{12}$, we have

$$\left(\frac{-3}{p}\right) = 1 \times 1 = 1.$$

– If $p \equiv 7 \pmod{12}$, we have

$$\left(\frac{-3}{p}\right) = (-1) \times (-1) = 1.$$

In the other words, -3 is a square in $\mathbb{F}_p$.

B Final Exponentiation for Curves with $k = 15$

We present the first analysis for the hard part during Tate pairing computation over elliptic curves with embedding degree $k = 15$. By setting $x = 6u + 1$, we obtain the new explicit polynomials as follows :

$$\begin{aligned} t(u) &= 6u + 2, \\ p(u) &= 725594112u^{12} + 1209323520u^{11} + 906992640u^{10} + 403107840u^{9} + 117573120u^{8} \\ &\quad + 23607936u^{7} + 3343680u^{6} + 336960u^{5} + 23760u^{4} + 1080u^{3} + 36u^{2} + 6u + 1, \\ r(u) &= 1679616u^{8} + 1959552u^{7} + 979776u^{6} + 279936u^{5} + 50544u^{4} + 6048u^{3} \\ &\quad + 504u^{2} + 24x + 1. \end{aligned}$$

Once again, assume that the hard part $\frac{p(u)^{10}+p(u)^{5}+1}{r(u)}$ of the final exponentiation can be expanded as $\sum_{i=0}^{9} a_i(u)p(u)^i$. It is easy to verify $a_i(u)$ to be following explicit polynomials.

$$\begin{aligned} a_9 &= 432u^4 + 216u^3 + 36u^2, \\ a_8 &= 2592u^5 + 1728u^4 + 432u^3 + 36u^2 = a_9T, \\ a_7 &= 15552u^6 + 12960u^5 + 4320u^4 + 648u^3 + 36u^2 = a_8T, \\ a_6 &= 93312u^7 + 93312u^6 + 38880u^5 + 8208u^4 + 864u^3 + 36u^2 = a_7T, \\ a_5 &= 559872u^8 + 653184u^7 + 326592u^6 + 88128u^5 + 13392u^4 + 1080u^3 + 36u^2 = a_6T, \\ a_4 &= 3359232u^9 + 4478976u^8 + 2612736u^7 + 855360u^6 + 168480u^5 + 20304u^4 + 1512u^3 \\ &\quad + 72u^2 = a_5T + a_9, \\ a_3 &= 20155392u^{10} + 30233088u^9 + 20155392u^8 + 7744896u^7 + 1866240u^6 + 290304u^5 \\ &\quad + 29376u^4 + 1944u^3 + 72u^2 = a_4T, \qquad (8) \\ a_2 &= 120932352u^{11} + 201553920u^{10} + 151165440u^9 + 66624768u^8 + 18942336u^7 \\ &\quad + 3608064u^6 + 466560u^5 + 41040u^4 + 2376u^3 + 72u^2 + 1 = a_3T + 1, \\ a_1 &= 120932352u^{11} + 201553920u^{10} + 147806208u^9 + 62705664u^8 + 16982784u^7 + 3063744u^6 \\ &\quad + 375840u^5 + 31536u^4 + 1728u^3 + 36u^2 + 1 = a_2 - a_4 + a_5 - a_7 + a_8, \\ a_0 &= 120932352u^{11} + 181398528u^{10} + 120932352u^9 + 47029248u^8 + 11757312u^7 + 1975104u^6 \\ &\quad + 228096u^5 + 18144u^4 + 864u^3 + 1 = a_1 - a_3 + a_4 - a_6 + a_7 - a_9, \end{aligned}$$

where $T = 6u+1$. Similarly, we assume that f is the output of Miller algorithm, and $m = f^{p^3-1}$. The hard part can be performed as follows:

$$m^{\frac{p(u)^{10}+p(u)^5+1}{r(u)}} = \prod_{i=0}^{9} \mu_i^{p^i},$$

where μ_i can be computed as follows:

$$\mu_9 = ((m^{T-1})^{(T-1)/3} \cdot m^{T-1} \cdot m)^{(T-1)^2},\ \mu_i = (\mu_{i+1})^T \text{ for } i \in \{3,5,6,7,8\},\ \mu_4 = (\mu_5)^T \cdot \mu_9,$$

$$\mu_2 = (\mu_3)^T \cdot m,\ \mu_1 = \mu_2 \cdot \mu_5 \cdot \mu_8 \cdot (\mu_4 \cdot \mu_7)^{-1},\ \mu_0 = \mu_1 \cdot \mu_4 \cdot \mu_7 \cdot (\mu_3 \cdot \mu_6 \cdot \mu_9)^{-1}.$$

This part requires 11 exponentiations by T, $T-1$ or $(T-1)/3$, 22 multiplications, two inversions in $\mathbb{F}_{p^{15}}$, and 9 p-power Frobenius operations. Note that inversions in $\mathbb{F}_{p^{15}}$ can be computed for free using a simple conjugation [31]. Assume that we apply this family of curves for pairing computation at 192-bit security level. The sizes in bits of r, and T are 384 and 64, respectively. By carefully choosing parameters, we can get a value of T with low Hamming weight (*e.g.*, $H(T) = 7$). This final exponentiation will require $88\mathbf{M}_{p^{15}} + 528\mathbf{S}_{p^{15}}$.

An Improved Hardware Implementation of the Grain-128a Stream Cipher

Shohreh Sharif Mansouri and Elena Dubrova

Department of Electronic Systems, Royal Institute of Technology, Stockholm
{shsm,dubrova}@kth.se

Abstract. We study efficient high-throughput hardware implementations of the Grain-128a family of stream ciphers. To increase the throughput compared to the standard design, we apply five different techniques in combination: isolation of the authentication section, Fibonacci-to-Galois transformation of the feedback shift registers, multi-frequency implementation, simplification of the pre-outputs functions and internal pipelining. The combined effect of all these techniques enables an average 56% higher keystream generation throughput among all the ciphers, at the expense of an average 8% area penalty, an average 4% power overhead and a 21% slower keystream initialization phase. An alternative combination of techniques allows an average 23% throughput improvement in all phases.

1 Introduction

Feedback Shift Registers (FSR)-based stream ciphers, characterized by a low hardware footprint, are one of the most promising candidates for deployment in low-cost authentication devices [1]. Since FSR-based stream ciphers target highly-constrained environments, designing them efficiently is important. Hardware efficiency was one of the main parameters used for grading the ciphers during the eSTREAM project [2], that in 2008 identified a portfolio of three promising FSR-based stream ciphers: Grain [3], Mickey [4] and Trivium [5]. Until recently, however, only straightforward implementations of the ciphers were considered (standard syntheses of RTL models that are direct translation of the cipher algorithm) [6].

In 2010, special techniques to improve the hardware figures-of-merit of FSR-based stream ciphers were introduced in [7] and [8], and it was shown that the throughput of FSR-based stream ciphers can be considerably improved.

In 2011, Agren and co-workers introduced a new family of Grain ciphers that natively supports authentication, with a maximal tag length of 32 bits, called Grain-128a [9] (described in Section 2). To our best knowledge, no study on the hardware implementation of the Grain-128a ciphers has been conducted. In this work we aim on finding the best implementation of Grain-128a in terms of throughput. In Section 4 we implement the original Grain-128a ciphers using a straightforward design flow. Then, we apply five different techniques to improve their throughput: in Section 5 we isolate the authentication section of the ciphers, in Section 6 we transform the Fibonacci FSRs of the ciphers to Galois FSRs, in Section 7 we introduce dual-frequency implementations of the ciphers and we simplify and pipeline their pre-outputs functions.

The final results, reported in Section 8, show that among the six versions of the cipher we obtain an average 56% higher keystream generation throughput at the expense of an average 8% area penalty, a 4% power overhead and a 21% slower keystream initialization phase. We also introduce an alternative combination of techniques that improves throughput in both initialization and keystream generation phases, well suited for ciphers that process short bursts of data and spend a lot of time in the keystream initialization phase: in this case, the average throughput improvement is 23%.

2 The Grain-128a Cipher

The Grain-128a family of ciphers [9] are extensions of the Grain-128 stream cipher, that natively supports authentication with a variable tag size up to 32 bits. The non-linear functions of the Grain-128a ciphers are also slightly different compared to those of Grain-128.

The Grain-128a family of ciphers is constituted by one unparallelized cipher and five parallelized versions of the same cipher, which can have degree of parallelization 2, 4, 8, 16 or 32. We refer to a version of Grain-128a parallelized k times as Grain-128aXk (we refer to the unparallelized cipher as Grain-128a or Grain-128aX1). All members of the family are functionally equivalent, i.e. they have the same output when fed by the same input. In Grain-128Xk the FSR feedback functions and the pre-output functions, as well as some parts of the authentication section, are replicated k times compared to the unparallelized cipher. Grain-128aXk outputs $\frac{k}{2}$ keystream bits per cycle.

A complete schematic of the grain-128a cipher is shown in Figure 1. The cipher is divided into two parts: the *keystream generator*, which generates a pre-output stream, and the *authentication section*.

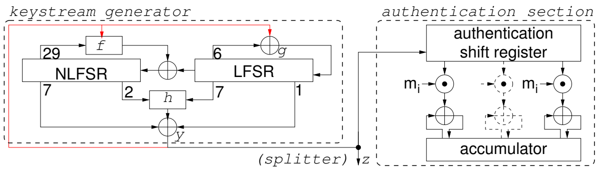


Fig. 1. The Grain-128a cipher

2.1 Keystream Generator

The keystream generator contains a 128-bit Linear FSR (LFSR) and a 128-bits Non-Linear FSR (NLFSR). The contents of the 128-bits LFSR are denoted as $s_0, s_1, ..., s_{127}$; the contents of the 128-bits NLFSR are denoted as $b_0, b_1, ..., b_{127}$.

All memory elements of the LFSR and the NLFSR are updated simultaneously. s_i is updated to s_{i+1} for $0 \leq i \leq 126$; s_{127} is updated to $f(s)$, where $f(s)$ is:

$$f(s) = s_0 + s_7 + s_{38} + s_{70} + s_{81} + s_{96}$$

b_i is updated to b_{i+1} for $0 \leq i \leq 126$; b_{127} is updated to $g(b, s_0)$, where $g(b, s_0)$ is:

$$\begin{aligned} g(b, s_0) = \; & s_0 + b_0 + b_{26} + b_{56} + b_{91} + b_{96} + b_3 b_{67} + b_{11} b_{13} + b_{17} b_{18} + b_{27} b_{59} \\ & + b_{40} b_{48} + b_{61} b_{65} + b_{68} b_{84} + b_{88} b_{92} b_{93} b_{95} + b_{22} b_{24} b_{25} + b_{70} b_{78} b_{82} \end{aligned}$$

The function $h(b, s)$ is:

$$h(b, s) = b_{12} s_8 + s_{13} s_{20} + b_{95} s_{42} + s_{60} s_{79} + b_{12} b_{95} s_{94}$$

The pre-output function $y(b, s)$ is:

$$y(b, s) = h(b, s) + s_{93} + b_2 + b_{15} + b_{36} + b_{45} + b_{64} + b_{73} + b_{89}$$

The sequence of pre-output bits output by the y function are denoted as $y_0, ..., y_i$. The output function $z(b, s) = y_{64+2i}$ outputs all pre-output bits of even index except the first 64. The first 64 pre-output bits and all bits of odd index are instead passed to the authentication section.

2.2 Authentication Section

Two authentication registers, the *accumulator* and the *authentication shift register*, both of size 32, are used. The content of the accumulator is denoted as $a_0, ..., a_{31}$. The content of the authentication shift register is denoted as $r_0, ..., r_{31}$. During an initial 64-cycles authentication initialization phase, the first 32 pre-output elements $y_0, ..., y_{31}$ of y_i are stored in the authentication shift register ($r_i = y_i$) while the following 32 elements $y_{32}, ..., y_{63}$ of y_i are stored in the accumulator ($a_i = y_{32+i}$).

In every cycle i, r_{31} is updated to the new pre-output bit $y_{64+2i+1}$ while all the other 31 elements r_j are updated to r_{j+1}. All bits a_j in the accumulator are updated to $a_j + m_i r_j$, where m_i is the bit of the message $m = m_0, ..., m_{L-1}$ that is being encrypted in cycle i. The final content of the accumulator once encryption is concluded is denoted as the tag t and can be used for authentication ($t_i = a_i$).

If the tag size is $w < 32$, only the part of the tag t with the w highest indexes is used as a tag; the other parts are discarded.

2.3 Cipher Phases

When the cipher starts operating, the 128-bit key $k_0, ..., k_{127}$ is loaded in the 128 NLFSR memory elements $b_0, ..., b_{127}$; the 96-bit Initial Value $IV_0, ..., IV_{95}$ is loaded in the first 96 LFSR memory elements $s_0, ..., s_{95}$; the last 32 bits of the LFSR are loaded with $s_{96}, ..., s_{126} = 1$ and $s_{127} = 0$. After having been loaded with the key and the initial value, the cipher goes through the following phases: (1) *keystream initialization phase*, 256 clock cycles in which the cipher does not produce any output bit and the output of the y function is fed back to the LFSR and the NLFSR (red lines in Figure 1); (2) *authentication initialization phase*, 64 clock cycles in which all the pre-output bits are stored in the accumulator and the authentication shift register; (3) *operational phase* in which half the pre-output bits are output as keystream and half are fed to the authentication section of the cipher. The *keystream generation phase* includes both phases (2) and (3). The phases of the cipher are summarized in Figure 2.

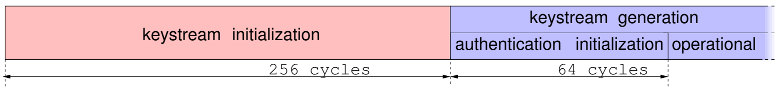


Fig. 2. Cipher phases

3 Implementation and Analysis Methodology

All timing, area and power figures reported in this paper are obtained by designing the ciphers at Register Transfer Level (RTL) in Verilog, and then synthesizing the code for best performances using Cadence RTL Compiler for the TSMC 90 nm ASIC technology.

To keep track of the phases of the cipher and decide when to change phase, the cipher uses an LFSR counter [10], the smallest and fastest type of counter. The FSRs are initialized serially with the key and the initial value.

Timing and area figures are obtained from the synthesis tool; power figures are obtained using the following procedure: the post-synthesis gate-level netlist is exported by the synthesis tool; a gate-level simulation is performed using the Cadence Incisive logic simulator with a set of random test vectors and a clock frequency of $10MHz$; the switching activity of all nets in the system is saved to a VCD file and read back by Cadence RTL Compiler, which then estimates the power consumption of the system.

4 Straightforward Implementation

We first implement Grain-128aXk using a standard design flow (applied to an RTL model that directly tranlates the algorithms), and optimizing the system for the highest throughput.

To improve the throughput of the different versions of the Grain-128a cipher, we study the location of the critical paths in the synthesized ciphers, i.e. the longest combinational propagation delays, which determine their throughput. We define the following delays:

- D_n: maximal propagation delay from any NLFSR flip-flop to any other NLFSR flip-flop. D_l is the LFSR counterpart.

Table 1. Timing in the original versions of Grain-128a

	X1	X2	X4	X8	X16	X32
Min_p (ps), CP	472, D_{hyn}	493, D_{hyn}	499, D_{hyn}	521, D_{hyn}	588, D_{hya}	665, D_{hya}

- D_{hy}: maximal propagation delay from any NLFSR or LFSR flip-flop through the h and y functions to the output of the cipher.
- D_{hya}: maximal propagation delay from any NLFSR or LFSR flip-flop through the h and y functions to any accumulator flip-flop.
- D_a: maximal propagation delay from any flip-flop in the authentication section of the cipher to any accumulator flip-flop.

Two additional delays, active only during the keystream initialization phase, are defined:

- D_{hyn}: maximal propagation delay from a flip-flop of the NLFSR or LFSR through the h and y functions to the first flip-flop of the NLFSR. D_{hyl} is the LFSR counterpart.

Table 1 reports the minimal clock period and the critical paths for all the versions of the cipher. The first observation is that Grain-128aX16 and Grain-128aX32 can benefit from breaking the h-y-accumulator path. This is discussed in the next section.

5 Isolating the Authentication Section

In a parallelized cipher Grain-128aXk with $k \geq 4$, the value of a_j must be updated to

$$a_j + \sum_{u=0}^{u<\frac{k}{2}} m_{i+u} \cdot r_{j+u}$$

in every cycle i. The implementation is straightforward for $j \leq 31 - \frac{k}{2}$. However, for $j > 31 - \frac{k}{2}$, the accumulator logic would need to access values of r_j with $j > 31$. These values can be seen as "future values" of the r bits. Since r shifts its elements by $\frac{k}{2}$ positions every clock cycle and loads $\frac{k}{2}$ new elements from $\frac{k}{2}$ outputs of the k parallel h/y functions, $\frac{k}{2}$ future values of the r elements can always be found on the output lines of the h/y functions, and can be accessed by the accumulator logic to implement the authentication functionality.

For high degrees of parallelism, this straightforward solution, used in Section 4, involves a long combinational path D_{hya} through the h/y functions and the accumulator logic that limits the performances of Grain-128aX16 and Grain-128aX32.

To break the D_{hya} path, flip-flops are inserted in the authentication section of the cipher on the outputs of the h/y functions, as shown in Figure 3. This solution adds one cycle latency in the production of the tag, but has no effect on cipher security.

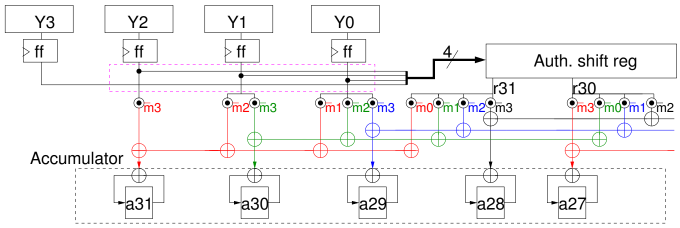


Fig. 3. Isolation of the authentication section for Grain-128aX8

Table 2. Timing in the versions of Grain-128a after the isolation of the authentication section

	X1	X2	X4	X8	X16	X32
Min_p (ps),CP	440, D_{hyn}	490, D_{hyn}	483, D_{hyn}	517, D_{hyn}	545, D_{hyn}	580, D_{hyn}

After applying this solution to all versions of Grain-128a, the minimal clock periods (Min_p) and critical paths (CPs) of the ciphers improve, as reported in Table 2. Timing improves for all versions of the cipher.

The critical paths of all versions of the cipher are now determined by D_{hyn}. Since h, y and g are optimized together by the synthesis tool, D_{hyn} depends on both the h/y functions (19 literals in total) and on the $g(b, s_0)$ feedback function of the NFLSR (30 literals).

In the next section we try to reduce D_{hyn} by reducing the maximal propagation delay of the paths going from the NLFSR bits to the first bit of the NLFSR through Fibonacci to Galois transformation [11].

6 Fibonacci to Galois Transformation

A Feedback Shift Register (FSR) consists of n binary storage elements, called *bits* [12]. Each bit i has an associated state variable x_i which represents the current value of bit i and a *feedback function* $f_i(x_0, ..., x_{n-1})$ which determines how the value of i is updated. All updates take place simultaneously.

The FSRs can be implemented in two configurations, *Fibonacci* or *Galois*. An FSR is in Fibonacci configuration if all feedback functions f_i except f_{n-1} take the form $f_i = x_{i+1}$. If some functions f_i with $i \neq n-1$ are not in this form, then the FSR is in *Galois* configuration. For LFSRs, this definition is more general than the traditional definition of Galois LFSRs, which corresponds to that of fully-shifted Galois LFSRs [11]. However, to keep the presentation simple, in this paper we use this definition for both NLFSRs and LFSRs.

As discussed in [11] a serially-initialized Fibonacci n-bit FSR can be transformed into an equivalent Galois FSR having the same output stream (i.e. the values of x_0 in the two FSRs are always identical). The transformation can be done by moving a set of product terms P from f_i to f_j while changing the indexes of each variable x_k of each product term in P to x_{k-i+j}. To guarantee the equivalence of a Fibonacci NLFSR to a Galois NLFSR, product terms cannot be shifted to positions lower that the *minimum terminal bit* τ_{min}, which is calculated as:

$$\tau_{min} = \max_{p_i \in P_T} \left(max\_index\,(p_i) - min\_index\,(p_i) \right)$$

where $min\_index(p_i)$ and $max\_index(p_i)$ denote respectively the minimum and maximum index of the variables in product term p_i and p_T is the set of all product terms. Proof of equivalence between the Fibonacci and the Galois FSRs can be found in [11].

When this transformation is applied to stream cipher's FSRs, as discussed in [7], the feedback functions in which a product term p_i can be moved are also limited by:

– *minimal index in the product term*: no product term p_i can be moved to a feedback function of grade lower than $n - 1 - min\_index(p_i)$.
– *combinational functions inputs*: to preserve the original encryption algorithm, no product term can be moved to a feedback function of grade lower than the highest state bit used as input of any combinational function.
– *degree of parallelization*: in an FSR parallelized k times, all feedback functions f_i except $n - j \cdot k - 1,\ \forall j = \{0, 1, \ldots, \lfloor (n-1)/k \rfloor - 1\}$ should have feedback functions of type $f_i = x_{i+1}$.

6.1 Throughput Optimization

Tranformation from Fibonacci to Galois for a single FSR can generally result into many different configurations. To choose the best design in terms of throughput, a heuristic algorithm was developed in [8]. This algorithm tries to find the fastest Galois FSR equivalent to a given Fibonacci FSR, i.e. the Galois FSR with the shortest critical path [8].

The algorithm associates every FSR to a cost, which is an estimation of its critical path, and tries to find the minimal-cost Galois FSR. Normally, given a Fibonacci FSR, the algorithm can choose among more than one minimal-cost Galois FSR. Although all of them have in principle similar throughput, they have slightly different area overheads.

In this paper we use the same algorithm suggested in [8] to find the best Fibonacci-to-Galois transformation; however, since the original algorithm does not consider area, we have introduced a final area optimization stage to it.

6.2 Area Optimization

The main idea of the area optimization stage is that area savings occur when two or more products in different feedback functions can be implemented using shared gates.

Once the algorithm in [8] has identified a minimal-cost FSR, its feedback functions are scanned to search for product terms in the form $x_i x_j \ldots$ and $x_{i+k} x_{j+k} \ldots$.

These product terms will be encountered if the original Fibonacci FSR also contained product terms in the form $x_{i^*} x_{j^*} \ldots$ and $x_{i^*+k^*} x_{j^*+k^*} \ldots$, a common occurrence for cryptographic FSRs.

The algorithm removes the two product terms from the feedback functions and tries to place them exactly at distance k from each other, with the first above the second. All available positions are scanned to find a suitable placement. The product-term movement takes place only if it does not increase the cost of the FSR, i.e. its estimated critical path. If a suitable placement for the two product terms is found, the products $x_i x_j$ and $x_{i+k} x_{j+k}$ are transformed to the same product $x_{i^{**}} x_{j^{**}}$, and both products can be implemented using a single shared AND gate. The main idea of this optimization is shown in Figure 4: the Galois configuration allows to implement the two product terms of the Fibonacci FSR using a single AND gate. The algorithm continues until all suitable product terms have been placed and no further area optimization is possible.

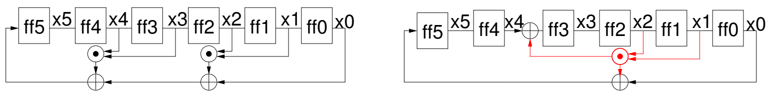


Fig. 4. Area savings obtained through shared AND gates

7 Final Optimization

By transforming the FSRs of Grain-128aXk from a Fibonacci to a Galois configuration, Galois FSRs (both LFSR and NLFSR) run faster compared to the original Fibonacci FSRs. The highest improvement in timing is 67% for Grain-128aX1's NLFSR; the average timing improvement is 34%.

However, after implementing the Grain128a ciphers with Galois FSRs, the minimal clock periods of the complete ciphers improve only by 9% on average (compare Tables 3 and 2). The reason is that for all versions of Grain-128a, the critical path is given by D_{hyn}, i.e. performances are limited by the initialization path from the FSRs bits through the h, y and g functions to the first NLFSR bit. To increase further the throughput of the cipher, we suggest two alternative approaches: the first improves cipher performance during the keystream generation phase but sacrifices performances during the keystream initialization phase; the second tries to optimize throughput during both phases.

Table 3. Timing in the versions of Grain-128a after the Fibonacci to Galois transformation

	X1	X2	X4	X8	X16	X32
Min_p (ps), CP	440, D_{hyn}	445, D_{hyn}	472, D_{hyn}	482, D_{hyn}	573, D_{hyn}	580, D_{hyn}

7.1 First Approach: Dual Frequency Implementation

Similarly to the work in [7], to increase throughput during the keystream generation phase we realize a dual-frequency implementation of Grain-80 in which the cipher works with a slower clock clk_{ki} during the keystream initialization phase (the only phase in which the path D_{hyn} is active) and with a faster clock clk_{kg} during the keystream generation phase. The clock clk_{ki} can be generated externally or generated locally from clk_{kg} by a clock divider block.

We synthesize the ciphers optimizing them for operation during the keystream generation phase by defining the paths from the outputs of the y functions to the inputs of the LFSR and NLFSR as false paths during synthesis, i.e. we instruct the synthesis tool not to optimize any combinational path going from the outputs of the y functions to the inputs of the LFSR and NLFSR. This makes D_{hyn} larger, but reduces D_n, D_a and D_{hy} because it pushes the tool to optimize them as much as possible. Timing figures are reported in Table 4, row N/A, for both keystream initialization and keystream generation phases. Based on the results in Table 4, to ensure correct operation for all degrees of parallelization, it is sufficient that clk_{ki} be twice slower than clk_{kg}, i.e. clock division by a factor two is sufficient to guarantee correct operation if the keystream generation clock period is defined by the keystream generation critical path.

For high-parallelism versions of the cipher, the critical path during the operational phase is given by one of the feedback functions of the NLFSR, which are hard to optimize further. However, for $k \leq 8$, the performances of the cipher are limited by the propagation delay through the h and y functions. To keep the presentation simple, we consider the cascade of the h and y functions as a single non-linear function hy. To improve throughput, this hy function is pipelined using a 2-levels or 3-levels pipeline (see Figure 5-A).

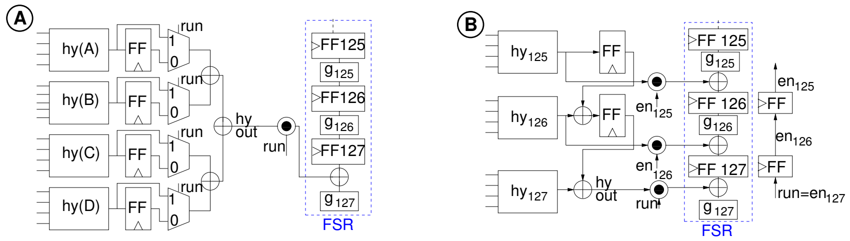


Fig. 5. Pipelined hy function in Section 7.1 (left) and Section 7.2 (right)

During the keystream generation phase, the output of the hy function goes to the output and the authentication section of the cipher. There is no feedback to the keystream generator; therefore pipelining the hy function does not alter the functionality of the cipher but only introduces a latency delay in the generation of the output stream and the authentication tag, which does not have any effect on the security of the cipher.

Table 4. Timing (Min_p (ps), CP) in the versions of Grain-128a during the keystream initialization (K.I.) and generation (K.G.) phases without pipelining (N/A) and with pipelining (P.L.) of the hy function (2 or 3 pipeline levels considered)

P.L.	Phase	X1	X2	X4	X8	X16	X32
N/A	K.I.	561, D_{hyn}	598, D_{hyn}	607, D_{hyn}	639, D_{hyn}	751, D_{hyn}	840, D_{hyn}
	K.G.	389, D_{hy}	372, D_{hy}	389, D_{hy}	403, D_{hy}	446, D_n	498, D_n
2	K.I.	579, D_{hyn}	604, D_{hyn}	629, D_{hyn}	657, D_{hyn}	830, D_{hyn}	904, D_{hyn}
	K.G.	303, D_{hy}	286, D_{hy}	328, D_{hy}	350, D_n	417, D_a	488, D_a
3	K.I.	569, D_{hyn}	623, D_{hyn}	668, D_{hyn}	685, D_{hyn}	827, D_{hyn}	881, D_{hyn}
	K.G.	283, D_{hy}	280, D_{hy}	305, D_{hy}	350, D_n	410, D_a	475, D_a

During the keystream initialization phase, the output of the hy function is fed back to the first bits of the NLFSR and the LFSR; if the hy function is pipelined during this phase, the functionality of the cipher is altered. Therefore, multiplexers are implemented in the pipeline to bypass the flip-flops and deactivate them during the keystream initialization phase. The initialization path through the multiplexers is defined as a false path during synthesis to push the tool to optimize for the keystream generation phase.

Pipelining the hy function has no effect on cipher security but has some drawbacks: flip-flops have to be inserted to implement the pipeline; An L level pipeline adds a latency of L cycles in the production of the tag. In general. the drawbacks increase with the number of pipeline levels.

Table 4 shows timing figures and critical paths for versions of the cipher with and without (N/A) pipelined hy function. Pipelining the hy function improves the timing of all versions of Grain-128a. The best improvement is obtained for Grain-128aX2.

7.2 Second Approach: Transformed *hy* Function

The solution presented in Section 7.1 is not well-suited for ciphers that spend a lot of time in keystream initialization phase because it makes this phase slower. In this section we introduce a new approach which decreases the delay of the path D_{hyn} by breaking down the hy function into several smaller functions. The approach is based on the idea that it is possible to "move product terms" of the hy function similarly to a FSR Fibonacci-to-Galois transformation.

The hy function is indicated as hy_{127} because it is fed to state bits s_{127} and b_{127}. The transformation is done by moving a set of product terms P of hy_{127} to hy_{127-i} while changing the index of each variable x_j of each product term $p_i \in P$ to x_{j-i}. Similarly to the Fibonacci-to-Galois transformation in Section 6, $127-i$ can not be smaller that the minimum terminal bit τ_{min} for the hy function, i.e. the maximal difference between variable indexes across all the product terms of hy. As an example, the hy function can be broken into three parts with:

Table 5. Timing (Min_p (ps), CP) in the versions of Grain-128a with transformed hy function

X1	X2	X4	X8	X16	X32
382, D_{hyn}	389, D_{hyn}	397, D_{hyn}	439, D_{hyn}	468, D_{hyn}	580, D_{hyn}

$$\begin{aligned}
hy_{127}(b,s) &= b_{12}s_8 + s_{13}s_{20} + b_{95}s_{42}\\
hy_{126}(b,s) &= b_{11}b_{94}s_{93} + b_{72} + b_1 + s_{59}s_{78}\\
hy_{125}(b,s) &= s_{91} + b_{87} + b_{13} + b_{34} + b_{43} + b_{62}
\end{aligned}$$

As shown in Figure 5-B, during the keystream initialization phase all hy_{127-i} functions are fed back to states bits b_{127-i} and s_{127-i} of the NLFSR and the LFSR. During the keystream generation phase, the feedback loop is disconnected and the hy_i functions form a 3 levels pipeline.

To preserve functional equivalence with the original cipher, care should be taken in determining the moment in which the hy feedback loop is activated and removed: before feeding the FSRs, the output of each hy_{127-i} function is ANDed with a signal en_{127-i} ($en_{127} = run$). This guarantees that each hy_{127-i} feedback becomes activated/de-activated i cycles after the hy_{127} feedback at the start/end of the keystream initialization phase. The en_{127-i} signals can be generated by delaying the run signal using flipflops (as shown in Figure 5) or can alternatively be driven by the internal counter.

Table 5 reports the minimal clock period and the critical paths for all the versions of the Grain-128aXk with a three-stage hy function when $k \leq 4$ and for a two-stage hy function when $k = 8, 16$.

8 Final Comparison

In this Section, we report the final throughputs of Grain-128a after applying all the techniques introduced in Sections 5, 6, 7.1 and 7.2. For every value, we report also the improvement over the original cipher. We denote as (ORG) the original ciphers (see Section 4); as (F2G) the original ciphers after the isolation of the authentication section and the Fibonacci to Galois transformation of the FSRs (see Sections 5 and 6); as (2F) the F2G ciphers modified with the implementation of the dual frequency solution (see Section 7.1) and 2-levels (for degree of parallelism $k > 4$) or 3-levels (for $k \leq 4$) internal pipelining; (1F) are the F2G ciphers with transformed hy function introduced in Section 7.2. For Grain-128aXk with $k \leq 4$, the hy function is divided into three functions ($hy_{127}, hy_{126}, hy_{125}$); for $k > 4$, it is divided into two functions (hy_{127}, hy_{126}).

The results are reported in terms of maximal frequency, throughput ($f_{max} \cdot \frac{k}{2}$), area and power. for 2F designs, the frequency reported in Table 6 is the frequency

Table 6. Implementation results

k	data	Freq. (GHz)			Through.(Gb/s)			Area(μm^2)			Power (μW)		
	imp.	ORG	2F	1F	ORG	2F	1F	ORG	2F	1F	ORG	2F	1F
X1	d.	2.1	3.5	2.6	1.1	1.5	1.3	5876	5856	5884	96.9	94.1	93.9
	%	-	67	24	-	67	24	-	0	0	-	3	3
X2	d.	2	3.6	2.6	2	3.6	2.6	6972	7314	7345	106.1	113.1	102.6
	%	-	80	30	-	80	30	-	0	0	-	-7	3
X4	d.	2	3.3	2.5	4	6.6	2.5	8299	9145	8614	120.6	136.1	125.1
	%	-	65	25	-	65	25	-	-10	-4	-	-13	4
X8	d.	1.9	2.9	2.3	7.6	11.6	9.2	10778	11087	10729	176.4	174.6	164.8
	%	-	53	21	-	53	21	-	-3	0	-	1	6
X16	d.	1.7	2.4	2.1	13.6	19.2	17.1	15709	17653	14585	247.8	275.4	205.4
	%	-	41	24	-	41	24	-	-12	7	-	-11	17
X32	d.	1.5	2	1.7	24	32	27.2	23430	28917	25554	417.9	415.1	403.5
	%	-	33	13	-	33	13	-	-23	-9	-	1	3

during the keystream generation phase. The keystream initialization frequency is twice lower than this frequency.

As shown in Table 6, the highest improvement in keystream generation throughput is achieved by the 2F implementation of Grain-128a, with an average 56% improvement in throughput among all the versions of the cipher. The highest improvement is 80% for Grain-128aX2 and the minimal improvement is 33% for Grain-128aX32. The 2F implementation can be used in applications which encode/decode large data sets. On the other hand, with the 1F implementation we achieve on average a 23% throughput improvement in all phases. Although the 1F implementation has a lower throughput improvement compared to the 2F implementation, it does not use a double clock and is therefore simpler and with a lower area overhead. It is well-suited for applications which encode/decode short data sets and switch often between operational phases.

9 Conclusion

In conclusion, we have shown that it is possible to considerably improve the hardware timing figures of merit of the different versions of the Grain-128a cipher by applying a combination of different techniques. With a two-frequencies implementation, the keystream generation throughput improved on average 56% at the expense of a 21% slowing of the keystream initialization phase an reasonable overheads. An alternative single-clock solution allowed us to obtain an average 23% higher throughput in all phases.

Acknowledgment. This work was supported in part by a project No 621-2010-4388 from Swedish Research Council.

References

1. Good, T., Benaissa, M.: ASIC hardware performance. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 267–293. Springer, Heidelberg (2008)
2. Robshaw, M., Billet, O. (eds.): New Stream Cipher Designs: The eSTREAM Finalists. LNCS, vol. 4986. Springer, Heidelberg (2008)
3. Hell, M., Johansson, T., Maximov, A., Meier, W.: The Grain family of stream ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 179–190. Springer, Heidelberg (2008)
4. Babbage, S., Dodd, M.: The mickey stream ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 191–209. Springer, Heidelberg (2008)
5. De Cannière, C., Preneel, B.: Trivium. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 244–266. Springer, Heidelberg (2008)
6. Good, T., Benaissa, M.: Hardware results for selected stream cipher candidates. In: Workshop Record of Stream Ciphers 2007 (SASC 2007), pp. 191–204 (2007)
7. Mansouri, S., Dubrova, E.: An improved hardware implementation of the grain stream cipher. In: 2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD), pp. 433–440 (September 2010)
8. Chabloz, J.-M., Mansouri, S.S., Dubrova, E.: An algorithm for constructing a fastest galois nlfsr generating a given sequence. In: Carlet, C., Pott, A. (eds.) SETA 2010. LNCS, vol. 6338, pp. 41–54. Springer, Heidelberg (2010)
9. Agren, M., Hell, M., Johansson, T., Meier, W.: Grain-128a: a new version of grain-128 with optional authentication. Int. J. Wire. Mob. Comput. 5, 48–59 (2011)
10. Balph, T.: Lfsr counters implement binary polynomial generators. Motorola Semiconductor, EDN 43, 155–156 (1998)
11. Dubrova, E.: A transformation from the fibonacci to the galois nlfsrs. IEEE Transactions on Information Theory 55(11), 5263–5271 (2009)
12. Golomb, S.: Shift Register Sequences. Aegean Park Press (1982)

Appendix A: Fibonacci to Galois Transformation of FSRs in Grain-128a

NLFSR Fibonacci to Galois Transformation

We use the algorithm described in [8] and the area optimization algorithm from Subsection 6.2 to transform the NLFSRs of Grain-128aXk from a Fibonacci to a Galois configuration.

For Grain-128a, the product term with the maximal difference in variable indexes is $b_3 b_{67}$, i.e. $\tau_{min} = 64$ (see Section 6). Product terms cannot be allocated to feedback functions g_i of grade $i < 95$ because bit b_{95} is used in function h (see Section 6).

The area optimization algorithm (see Subsection 6.2) places the Fibonacci product terms $b_{88}b_{92}b_{93}b_{95}$, $b_{22}b_{24}b_{25}$ and $b_{70}b_{78}b_{82}$ respectively 27, 11 and 30 feedback functions downer than the product terms $b_{61}b_{65}$, $b_{11}b_{13}$ and $b_{40}b_{48}$.

For Grain-128aX1, the following Galois NLFSR is obtained:

$$\begin{aligned}
&g_{127} = s_0 \oplus b_0\\
&g_{126} = b_{127} \oplus b_{39}b_{47}\\
&g_{125} = b_{126} \oplus b_{59}b_{63}\\
&g_{124} = b_{125} \oplus b_0b_{64}\\
&g_{123} = b_{124} \oplus b_{52}\\
&g_{116} = b_{117} \oplus b_0b_2\\
&g_{105} = b_{106} \oplus b_0b_2b_3\\
&g_{110} = b_{111} \oplus b_0b_1\\
&g_{102} = b_{103} \oplus b_{71}\\
&g_{101} = b_{102} \oplus b_0\\
&g_{100} = b_{101} \oplus b_0b_{32}\\
&g_{99} = b_{100} \oplus b_{63}\\
&g_{98} = b_{99} \oplus b_{59}b_{63}b_{64}b_{66}\\
&g_{97} = b_{98} \oplus b_{38}b_{54}\\
&g_{96} = b_{97} \oplus b_{39}b_{47}b_{51}
\end{aligned}$$

Here and in the remainder of the paper, unspecified feedback functions are in the form $g_i = x_{i+1}$.

For Grain-128aX2 the Fibonacci product terms of the original NLFSR can be moved only to feedback functions g_{127}, g_{125}, g_{123}, g_{121}, g_{119}, g_{117}, g_{115}, g_{113}, g_{111}, g_{109}, g_{107}, g_{105}, g_{103}, g_{101} and g_{99}. The following Galois NLFSR is obtained after application of the timing and area optimization algorithms:

$$\begin{aligned}
&g_{127} = b_0 \oplus s_0\\
&g_{125} = b_{126} \oplus b_1b_{65}\\
&g_{123} = b_{124} \oplus b_{57}b_{61}\\
&g_{121} = b_{122} \oplus b_5b_7\\
&g_{119} = b_{120} \oplus b_9b_{10}\\
&g_{115} = b_{116} \oplus b_{15}b_{47}\\
&g_{113} = b_{114} \oplus b_{12}\\
&g_{111} = b_{112} \oplus b_6b_8b_9\\
&g_{109} = b_{110} \oplus b_{73}\\
&g_{107} = b_{108} \oplus b_{62}b_{58}b_{50}\\
&g_{105} = b_{106} \oplus b_{18}b_{26}\\
&g_{103} = b_{104} \oplus b_{72}\\
&g_{101} = b_{102} \oplus b_{30}\\
&g_{99} = b_{100} \oplus b_{40}b_{56}\\
&g_{97} = b_{98} \oplus b_{58}b_{62}b_{63}b_{65}
\end{aligned}$$

The area optimization algorithm places the Fibonacci product terms $b_{88}b_{92}b_{93}b_{95}$ and $b_{70}b_{78}b_{82}$ respectively 26 and 16 feedback functions downer than the Fibonacci product term $b_{61}b_{65}$. Also, the $b_{22}b_{24}b_{25}$ product term is placed 10 feedback functions downer than the $b_{11}b_{13}$ product term. This allows sharing gates among some of the parallelized feedback functions.

For Grain-128aX4 the Fibonacci product terms of the original NLFSR can be moved only to feedback functions g_{127}, g_{123}, g_{119}, g_{115}, g_{111}, g_{107}, g_{103} and g_{99}.

The NLFSR is transformed to:

$$\begin{aligned}
g_{127} &= s_0 \oplus b_0 \oplus b_3 b_{67} \\
g_{123} &= b_{124} \oplus b_{22} \oplus b_{52} \oplus b_{23} b_{55} \\
g_{119} &= b_{120} \oplus b_9 b_{10} \oplus b_3 b_5 \\
g_{115} &= b_{116} \oplus b_{70} b_{66} b_{58} \\
g_{111} &= b_{112} \oplus b_6 b_8 b_9 \\
g_{107} &= b_{108} \oplus b_{68} b_{72} b_{73} b_{75} \\
g_{103} &= b_{104} \oplus b_{72} \oplus b_{37} b_{41} \\
g_{99} &= b_{100} \oplus b_{40} b_{56} \oplus b_{63} \oplus b_{12} b_{20}
\end{aligned}$$

The area optimization algorithm places the Fibonacci $b_{88}b_{92}b_{93}b_{95}$ product term 8 feedback functions downer than the product term $b_{70}b_{78}b_{82}$; the product term $b_{22}b_{24}b_{25}$ is placed 8 feedback functions downer than the product term $b_{11}b_{13}$.

For Grain-128aX8 the Fibonacci product terms of the original NLFSR can be moved only to feedback functions g_{127}, g_{119}, g_{111} and g_{103}. The NLFSR is transformed to:

$$\begin{aligned}
g_{127} &= s_0 \oplus b_0 \oplus b_3 b_{67} \oplus b_{88} b_{92} b_{93} b_{95} \\
g_{119} &= b_{120} \oplus b_9 b_{10} \oplus b_3 b_5 \oplus b_{32} b_{40} \oplus b_{60} b_{76} \\
g_{111} &= b_{112} \oplus b_{10} \oplus b_{40} \oplus b_{11} b_{43} \oplus b_{75} \oplus b_6 b_8 b_9 \\
g_{103} &= b_{104} \oplus b_{72} \oplus b_{37} b_{41} \oplus b_{46} b_{54} b_{58}
\end{aligned}$$

For Grain-128aX16 the Fibonacci product terms of the original NLFSR can be moved only to feedback functions g_{127} and g_{111}. The NLFSR is transformed to:

$$\begin{aligned}
g_{127} &= s_0 \oplus b_0 \oplus b_{56} \oplus b_3 b_{67} \oplus b_{11} b_{13} \oplus b_{40} b_{48} \oplus b_{22} b_{24} b_{25} \oplus b_{70} b_{78} b_{82} \\
g_{111} &= b_{112} \oplus b_{10} \oplus b_{75} \oplus b_{80} \oplus b_1 b_2 \oplus b_{11} b_{43} \oplus b_{45} b_{49} \oplus b_{72} b_{76} b_{77} b_{79} \oplus b_{68} b_{52}
\end{aligned}$$

For Grain-128aX32 the Fibonacci product terms of the original NLFSR can be moved only to feedback functions g_{127}, i.e. the NLFSR cannot be transformed into a Galois NLFSR.

Optimized GPU Implementation and Performance Analysis of HC Series of Stream Ciphers*

Ayesha Khalid[1], Deblin Bagchi[2], Goutam Paul[2], and Anupam Chattopadhyay[1]

[1] Institute for Communication Technologies and Embedded Systems, RWTH Aachen University, Aachen 52074, Germany
{ayesha.khalid,anupam.chattopadhyay}@ice.rwth-aachen.de

[2] Department of Computer Science and Engineering, Jadavpur University, Kolkata 700 032, India
deblinbagchi@gmail.com, goutam.paul@ieee.org

Abstract. The ease of programming offered by the CUDA programming model attracted a lot of programmers to try the platform for acceleration of many non-graphics applications. Cryptography, being no exception, also found its share of exploration efforts, especially block ciphers. In this contribution we present a detailed walk-through of effective mapping of HC-128 and HC-256 stream ciphers on GPUs. Due to inherent inter-S-Box dependencies, intra-S-Box dependencies and a high number of memory accesses per keystream word generation, parallelization of HC series of stream ciphers remains challenging. For the first time, we present various optimization strategies for HC-128 and HC-256 speedup in tune with CUDA device architecture. The peak performance achieved with a single data-stream for HC-128 and HC-256 is 0.95 Gbps and 0.41 Gbps respectively. Although these throughput figures do not beat the CPU performance (10.9 Gbps for HC-128 and 7.5 Gbps for HC-256), our multiple parallel data-stream implementation is benchmarked to reach approximately 31 Gbps for HC-128 and 14 Gbps for HC-256 (with 32768 parallel data-streams). To the best of our knowledge, this is the first reported effort of mapping HC-Series of stream ciphers on GPUs.

Keywords: CUDA, eSTREAM, GPU, HC-128, HC-256, stream cipher.

1 Introduction

The eSTREAM [12] Portfolio (revision 1 in September 2008) contains the stream cipher HC-128 [21] in Profile 1 (SW) which is a lighter version of HC-256 [22] stream cipher born as an outcome of 128-bit key limitation imposed in the competition. Several research contributions exist on the cryptanalysis of HC-128 [14,15,13,18,20]. However, HC-256 has undergone fewer cryptanalytic attempts [16,19]. For algorithmic details of HC-128 and HC-256, the reader may refer to Appendix A.

* This work was done in part while the second author was a summer intern and the third author was an Alexander von Humboldt Fellow at RWTH Aachen, Germany.

After NVIDIA introduced a general purpose parallel computing platform namely Compute Unified Device Architecture (CUDA) in November 2006 [24], many cryptographers harnessed GPUs for acceleration. The earliest successful effort of AES acceleration on GPUs, that outperformed CPU in throughput, was presented by Manavski [1] who reported a throughput of 8.28 Gbps for AES-128 encryption on NVIDIA GeForce 8800. His work was later criticized for having half of the throughput rates that it could achieve by using shared memory instead of constant memory for T-boxes [2]. A more recent work by Iwai *et al.* [3] reported 35 Gbps of throughput for AES encoding on NVIDIA GeForce GTX285 by exploiting memory granularity for independent threads.

Several endeavors undertook more than one cipher to present a suite of CUDA based crypto accelerator application. Liu *et al.* [4] studied the effect of number of parallel threads, size of shared memory for lookup tables and data coalescing in device memories for several block encryption algorithms (AES, TRI-DES, RC5, TWOFISH) processing on GPU using CUDA. Nishikawa *et al.* [5] targeted five 128-bit symmetric block ciphers from an e-government recommended ciphers list by CRYPTREC in Japan and achieved substantial speedup.

The block ciphers, when subjected to parallelism offered by CUDA, generally show high speedups compared to CPUs because of the absence of data dependency between the consecutive data blocks. Generally, the plaintext is broken into n-many blocks of same size and subjected to independent threads of GPUs. Higher sizes of plaintext give more data blocks and hence result in better throughput by achieving more data parallelism, till the device limit is reached.

Unlike block ciphers, stream ciphers in general cannot be subjected to this *'divide and rule'* strategy. The reason is the dependencies in the states/S-boxes that are used for keystream generation. The only endeavor of mapping eSTREAM (including HC-128) and SHA-3 cryptographic algorithms on GPUs was presented by D. Stefan in his masters thesis [7]. He reported a throughput of 2.26 Gbps (4.39 cycles/byte) for HC-128 implementation using multiple parallel data-streams on NVIDIA GTX 295 GPU device[7]. This effort, however, lacks any optimization opportunity exploiting the structure of the algorithm and is, therefore, easily surpassed by our implementation in throughput.

This work presents a novel implementation of HC series of stream ciphers on recent graphics hardware. To the best of our knowledge, this is the first publication employing CUDA framework for GPU acceleration of any stream cipher.

2 Limitations in Parallelization of HC Ciphers

The *keystream generation* for HC series of stream ciphers has two steps, we name them as *self-update step* (SUS) of P/Q array and *keystream word generation step* (KWGS). In a serial implementation, each 32-bit word of P array SUS is followed by one KWGS. This goes on for 512 iterations in HC-128 and 1024 iterations for HC-256. The same follows for Q array for exactly the same number of iterations. Ideally, a fast GPU-based implementation would be able to run all these steps in parallel by independent *threads* as long as the device capacity is not over-budgeted. However, ciphers like HC have highly iterative structures, prohibiting parallelization beyond a limit.

2.1 Intra-S-Box Dependency in Self Update Step of S-Boxes

The gain of parallelization offered by CUDA programming model can be exploited easily if each iteration of a given iterative code block is independent of its past execution. Such loops can be converted to parallel kernels by complete unrolling where each loop iteration is executed by an independent *thread.* If an array value being computed by a loop iteration has an intra-array-dependency, such parallelism cannot be harnessed.

The SUS of HC-128 has a data dependency, the update of element $P[j]$ depends on its current and past values, i.e., $P[j]$, $P[j \boxminus 3]$, $P[j \boxminus 10]$ and $P[j \boxminus 511]$. Since the nearest dependency in the SUS of $P[j]$ is on $P[j \boxminus 3]$, one cannot unroll the loop more than 3 times.

```
//Three times unrolled version of $P$ array SUS
$for(j = 0; j < 512; j = j + 3)$
{
    $P[j] = P[j] + g_1(P[j \boxminus 3], P[j \boxminus 10], P[j \boxminus 511]);$
    $P[j + 1] = P[j + 1] + g_1(P[j \boxminus 2], P[j \boxminus 9], P[j \boxminus 510]);$
    $P[j + 2] = P[j + 2] + g_1(P[j \boxminus 1], P[j \boxminus 8], P[j \boxminus 509]);$
}
```

Fig. 1 describes the data dependencies for calculating the values at the i^{th}, $(i + 1)^{th}$ and $(i + 2)^{th}$ indices of P array pictorially. Calculation of $(i + 3)^{th}$ index value requires the value at i^{th} index of the array, making a simultaneous update of values at indices i and $(i + 3)$ impossible. This dependency limits the number of *threads* carrying out the SUS of P/Q array to no more than 3. The same arguments can be extended for HC-256 SUS. Moreover, due to similar limitations, we cannot harness more than 2 and 3 simultaneous *threads* for Step 1 and 3 respectively of *initialization* phase in HC series of stream ciphers.

2.2 Inter-S-Box Dependency in Keystream Generation

For exploiting parallelism we try to investigate if it is possible to carry out SUS P and Q arrays simultaneously (no spatial data dependency) or their current and future copies simultaneously (no temporal data dependency).

Inter-S-Box Spatial Data Dependency. Consider the *keystream generation* phase of HC-128 as given in Appendix A. The SUS of P and Q arrays does not require values from each other. However, KWGS after SUS of P array has a dependency on Q array and vice versa. Hence a naive implementation with simultaneous update of P and Q arrays will not bear correct results for KWGS. In HC-256, even the SUS of the two S-Boxes is dependent on each other. Moreover, the KWGS dependency after SUS in HC-256 is the same as in HC-128.

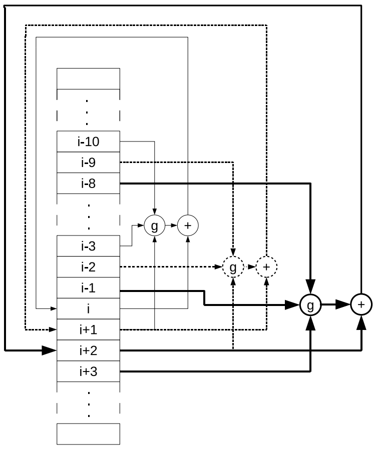


Fig. 1. Dependency in SUS at indices i, $i+1$ and $i+2$ in S-Boxes

Inter-S-Box Temporal Data Dependency. Temporal data dependency between the current instance of S-Boxes and their future instance is investigated to exploit the possibility of simultaneous *keystream generation* from these arrays for multiple data blocks. Consider two temporal instances of P array. Let $P_{current}$ contain the expanded values after *initialization* phase and P_{future} be the one that will have the future values of P array after SUS. Note that SUS of P_{future} has a dependency on $P_{current}$, hence making it impossible to simultaneously update multiple temporal instances of P/Q arrays. Arguing along the same lines, its evident to see data dependency of P/Q arrays on their past instances in HC-256 too.

2.3 Data-Intensiveness

When comparing the computational nature of stream ciphers with block ciphers, a striking trend can be seen. Stream ciphers are predominantly *data intensive* while block ciphers are *computation intensive.* HC series of stream ciphers are no exception. Appendix B gives the list and frequency of various 32-bit binary operations required by the SUS and KWGS of HC-128 and HC-256. The high ratio of memory accesses to the arithmetic operations can be seen to be quite high.

3 Optimization Strategies for GPU Implementation of HC Series of Stream Ciphers

Kernels in CUDA compatible devices are assigned a small budget of thread-local registers. Shared memory is local to a block of threads and is comparatively bigger. The biggest memory in size is the grid-local global memory whose access incurs a 100x penalty as compared to register access [9]. Our device NVIDIA GeForce GTX 590 has 3 GB of global memory, 48 KB of shared memory per MP and a maximum of 64 registers per thread. Considering the memory hierarchy, the fastest single data-stream implementation of the algorithm should use the fastest memory, i.e., the registers. However, the S-boxes of HC-128 (4 KB) and HC-256 (8 KB) are far too big to fit in them. The next best possibility is to put the P and Q arrays in the shared memory and let the registers hold their smaller 16-element snapshot as suggested for the optimized implementation in [21,22]. However, this single thread implementation of *keystream generation* does not lead to significant throughput. For example, HC-128 on our device yielded a throughput of only *0.24* Gbps.

For exploiting parallelism, we strive to launch multiple threads simultaneously. As registers are local to one kernel, we use shared memory instead and discuss various optimization strategies for single data-stream implementation in Section 3.1. For multiple data-streams implementation, the use of on-chip block-local shared memory instead of off-chip grid-local global memory can boost the speedup significantly. However, each data-stream requires a memory budget m for P and Q arrays, where m = 4 KB for HC-128 and m = 8 KB for HC-256 and hence the number of parallel data-streams per MP is restricted to s/m, where s = 48 KB is the shared memory size. Therefore, we perform the multiple data-streams implementation using global memory, as discussed in detail in Section 3.2. A brief overview of the CUDA programming model for GPUs is given in Appendix C.

3.1 Single Data-Stream Optimizations

Program listing of a simple implementation of *keystream generation* code for HC-128 with the degree of parallelism that is straightforward to manipulate is given in Table 1. Since the *initialization* phase is similar and simpler, its explanation is skipped. The intra-dependency of S-Box arrays does not allow more than 3 parallel threads to update P/Q arrays as described in Section 2.1. The CUDA kernel is called with 1 block of 512 threads. The code is divided into *four* parts. The first and third parts give SUS for P and Q arrays respectively while part two and four perform KWGS. Only 3 out of 512 threads update P array in part one, requiring 171 (512/3) times execution for completely updating P array. In part 2, the S-Boxes are employed to generate 512 words of keystream using 512 threads simultaneously. Part 3 updates the Q array followed by 512 words of KWGS in part 4. This implementation yields a throughput of *0.37* Gbps for *keystream generation* in HC-128.

Next we discuss the optimization strategies undertaken to improve the parallelism and consequently the throughput of this simple parallel CUDA based implementation of HC-128. In case the strategies are applicable only to one of the ciphers in HC series of stream ciphers, it has been explicitly mentioned.

Parallelization of P/Q Array SUS with Key Generation(512 words). One way of increasing the degree of parallelism in HC-128 algorithm was

Table 1. *Keystream generation* implementation of HC-128 using three threads

```
$if(threadIdx.x <= 2)$
    $for(i = threadIdx.x; i < 512; i = i + 3)$
        $P\_s[i] = P\_s[i] + g1(P\_s[i \boxminus 3], P\_s[i \boxminus 10], P\_s[i \boxminus 511];$
---
$i = threadIdx.x;$
$s[i] = h1(Q\_s, P\_s[i \boxminus 12]) \oplus P\_s[i];$
---
$if(threadIdx.x <= 2)$
    $for(i = threadIdx.x; i < 512; i = i + 3)$
        $Q\_s[i] = Q\_s[i] + g2(Q\_s[i \boxminus 3], Q\_s[(i \boxminus 10)], Q\_s[i \boxminus 511]);$
---
$i = threadIdx.x;$
$s[i + 512] = h2(P\_s, Q\_s[i \boxminus 12]) \oplus Q\_s[i];$
```

suggested by Chattopadhyay *et al.* [23]. The authors proposed carrying out SUS of either of the S-Boxes along with a simultaneous KWGS from the other S-Box. The parallelism can be employed ensuring correct results by keeping multiple temporal copies of S-Boxes (say $P0$, $Q0$, $P1$, $Q1$). If the shared memory of the GPU device used for S-Box instances is not over-budgeted, this strategy can be employed for achieving parallelism. As seen from Appendix A, each round of HC-128 *keystream generation* for 1024 words has a P-SUS and P-KWGS for 512 words, followed by a similar Q-SUS and Q-KWGS for 512 words. With two copies of S-Boxes, we can parallelize the P-SUS with Q-KWGS and vice versa. The series of steps as proposed in [23] are summarized in Table 2. After *initialization* routine, arrays $P0$, $Q0$ contain the expanded key and IV. SUS of P array starts by reading values from $P0$ (past values) and updating $P1$ (current values). No more than 3 parallel threads (due to intra-data-dependency) execute iteratively updating the entire 512 words array. In step 1 the Q array is updated reading values from $Q0$ (past values) and updating $Q1$ (current values). KWGS using $P1$ and $Q0$ is done by 512 parallel threads simultaneously - we denote this by Keygen($Q0$,$P1$). Similar notations describe the other steps.

Table 2. Parallelizing one SUS warp with one KWGS block

Step #	KWGS	SUS	Comments
Step 0	-	P1	3 threads for SUS
Step 1	Keygen($Q0$,$P1$)	Q1	3 active threads (out of a warp) for SUS + 512 threads for KWGS
Step 2	Keygen($Q1$,$P1$)	$P0$	
Step 3	Keygen($Q1$,$P0$)	$Q0$	
Step 4	Keygen($Q0$,$P0$)	$P1$	

After the initial step, $Q1$, $P0$, $Q0$, $P1$ are updated in successive steps, each time simultaneously generating keystream words from the S-Box updated in the previous step. This goes on by repetition of step 1 till 4 for as many keystream values as required. CUDA framework for HC-128 parallel implementation employs 544 threads for *keystream generation* in total. Out of these, 512 threads carry out KWGS from an entire array of S-Box words simultaneously. One thread warp with three active threads carry out the SUS of the S-Box. Here parallelism is achieved at the cost of extra resources, since only multiple copies of the S-Boxes guarantee correct results for parallel implementation. This strategy is applied to HC-256 as well. Similarly, one warp with 3 active threads remains under-utilized; however KWGS is carried out by 1024 parallel threads for larger S-Boxes in HC-256.

Parallelization of P and Q SUS with Key Generation (1024 words). Further parallelization of HC-128 is possible by simultaneous P-SUS and P-KWGS of 512 words as well as the Q-SUS and Q-KWGS of 512 words in *keystream generation* phase as described in Appendix A. Thus both the S-Boxes can be updated in parallel along with simultaneous generation of 1024 words of keystream. However, step 1 and 3 of *keystream generation* in Table 2 reveal a data dependency. $Q0$ is needed for generating key from $P1$, and $Q1$ for generating key from $P0$. Hence, update of $P0$, $Q0$ and generating 1024 keystream words using Keygen($Q0$, $P1$) and Keygen($P1$, $Q1$) gives rise to a *race condition*, commonly called a *Read After Write (RAW) hazard* where the keystream values would depend upon which statement gets executed first. This can be successfully avoided by using 2 more copies of Q arrays, namely Q_{Buff0} and Q_{Buff1} for keeping backups of $Q0$ and $Q1$ respectively. For preserving correctness, these buffers need to be updated at every alternate step. All arrays are stored in the shared memory for fast access.

Table 3 describes a step by step execution. After *initialization*, the expanded key and IV reside in $P0$, $Q0$. All other temporal S-Box copies i.e., $P1$, $Q1$, Q_{Buff0} and Q_{Buff1} are left un-initialized. Simultaneous SUS of P and Q arrays is carried out by reading values from $P0$, $Q0$ (past values) and updating $P1$, $Q1$ (current values) respectively. A copy of $Q0$ is backed up in Q_{Buff0} simultaneously. In this step, 6 threads of 2 warps carry out the SUS for $P1$ and $Q1$. For $Q0$ backup, 512 parallel threads make a copy.

Table 3. Parallelizing 2 S-Box SUS warps with 2 KWGS blocks

Q_{Buff} copy	KWGS		SUS		Comments
Q_{Buff0} copy	-	-	$P1$	$Q1$	3 + 3 threads for SUS, 512 threads for copying $Q0$ to Q_{Buff0}
Q_{Buff1} copy	Keygen ($Q1$,$P1$)	Keygen (Q_{Buff0},$P1$)	$P0$	$Q0$	3 + 3 threads for SUS, 512 threads for Keygen($Q1$,$P1$), 512 threads for copying $Q1$ to Q_{Buff1} and Keygen(Q_{Buff0},$P1$)
Q_{Buff0} copy	Keygen ($Q0$,$P0$)	Keygen (Q_{Buff1},$P0$)	$P1$	$Q1$	3 + 3 threads for SUS, 512 threads for Keygen($Q0$,$P0$), 512 threads for copying $Q0$ to Q_{Buff0} and Keygen(Q_{Buff1},$P0$)

In step 1, we employ a block of 1024 threads for generating 1024 words of keystream, each thread generates one word of keystream. Out of these, 512 threads are used to execute the extra step of copying values to the buffers. Alternate updates of $P0$, $Q0$ and $P1$, $Q1$ follows, simultaneously generating 1024 words of keystream. Hence step 1 and 2 are repeated as long as the *keystream generation* is required.

A single kernel cannot be invoked with more than 1024 threads in a block. We break the thread budget in two blocks, each having 544 threads. The two blocks run concurrently, one warp in each carrying out SUS and 512 threads generating keystream. GPUs with compute capability 2.0 or more have the capability of calling concurrent kernels at the same time as well.

This strategy of achieving parallelism cannot be extended for HC-256 since its SUS of the S-Boxes is dependent on each other.

3.2 Multiple Data-Streams Optimization

The GPU clock is slower than the CPU clock speed. Thus speedup in GPU devices can be achieved in two ways. One way is by employing parallel threads respecting data dependencies in a single stream of data as investigated in Section 3.1. A better alternative in terms of resource utilization and throughput is to employ all the SPs (stream processors) of the CUDA device by employing ciphers of multiple data-streams in parallel. Due to the limited size of shared memory, we employ the larger albeit slower global memory for ciphering multiple parallel streams of data.

Performance tuning on the GPU requires understanding device specifications and accordingly finding and exposing enough parallelism to populate all the multiprocessors (MPs). NVIDIA GeForce GTX 590 can accommodate up to 8 blocks (or 48 warps) per MP. Since each warp can have 32 homogeneous threads, an MP can process up to 1536 threads (48×32). To fully utilize each MP, the number of threads it should get assigned should be no more than 192 per block (1536/8). This limit is kept in mind when assigning the thread budget to each MP for HC series of stream ciphers.

For HC-128, the 3 threads for SUS of each of the S-Boxes constitute one warp. Since these threads execute a total of 171 times (512/3) for complete update of either of the S-Boxes, the number of parallel threads employed for KWGS can be adjusted so that the budget of total number of 192 threads per block is never exceeded. We employ 128 threads for KWGS and 2 warps for S-Box update in case of HC-128. Hence 2 warps of S-Box SUS and 4 warps of KWGS are kept in the same block of 192 threads. For HC-256, however, only one warp is used for SUS and 4 for KWGS, making the total thread budget equal to 160 per block. This strategy ensures maximum number of parallel data-streams the device can encrypt simultaneously, showing noticeable increase in the throughput of both HC-128 and HC-256.

4 Experimental Results

Throughput performances of HC ciphers for single and multiple parallel data-streams were benchmarked on NVIDIA GeForce GTX 590. We used an AMD Phenom$^{\text{TM}}$II X6 1100T Processor with 8 GBs of RAM as host CPU. Each test was conducted 1000 times and the results were averaged. Appendix D summarizes the hardware specifications of the two computation platforms.

4.1 Encryption of Single Data-Stream

Initialization phase of HC ciphers has been implemented using shared memory and global memory in two separate experiments. The last step of *initialization* phase is similar to SUS phase, consequently 3 parallel *threads* are employed for it. In the second step of *initialization* phase, intra-dependency for W is even more severe, limiting the number of simultaneous *threads* to 2. Using faster shared memory instead of global memory accelerates *initialization* phase as shown in Table 4. It however, incorporates the overhead of copying P, Q and W arrays on shared memory that can be done simultaneously using 512 and 1024 parallel threads in case of HC-128 and HC-256 respectively.

Table 4. Duration and throughput of *initialization* phase of HC series of stream ciphers

	NVIDIA GeForce GTX 590		AMD Phenom$^{\mathrm{TM}}$II
	Global memory	Shared memory	X6 1100T
HC-128	1.386 ms 22.53 Mbps	1.078 ms 28.98 Mbps	27 μs 1.15 Gbps
HC-256	1.930 ms 32.35 Mbps	1.666 ms 53.56 Mbps	60 μs 1.04 Gbps

The performance results of *keystream generation* phase are presented in Fig. 2 and Fig. 3 for HC-128 and HC-256 respectively. The throughput shows an increasing trend, till it saturates for higher data sizes considered. The maximum throughput when using the global memory for storing S-Boxes of HC-128 is *0.41* Gbps. Using shared memory gives a boost to performance because of its smaller access time. A similar trend is observed for HC-256. The size of the S-Boxes is double compared to that of HC-128, the amount of shared memory used by the optimized version of our algorithm is 16 KB (two copies of each S-Box). A GPU

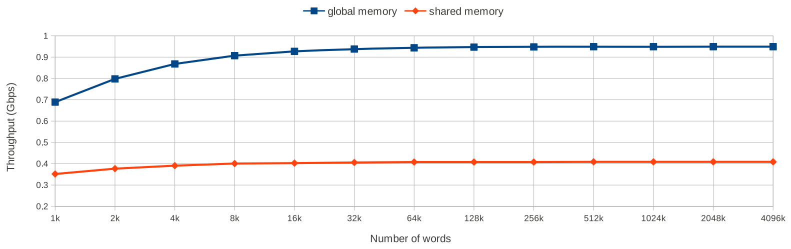


Fig. 2. HC-128 *keystream generation* throughput using shared and global memory

device with lower compute capability has no more than 16 KB of shared memory per MP. Hence, this optimized implementation of HC-256 on one thread block of such devices is not possible. The maximum throughput from the global memory implementation of HC-256 is 0.15 Gbps and for shared memory implementation is 0.41 Gbps.

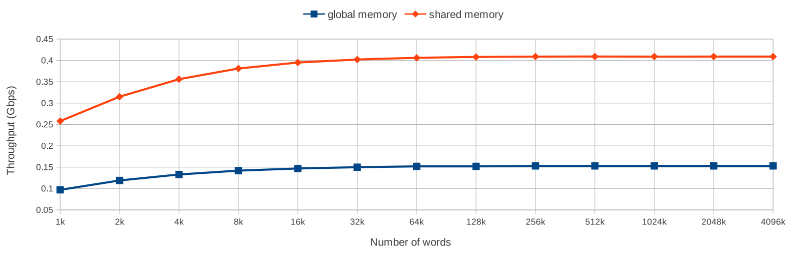


Fig. 3. HC-256 *keystream generation* throughput using shared and global memory

4.2 Encryption of Multiple Data-Streams in Parallel

The parallelism offered by the CUDA device can be well exploited using multiple parallel streams of data. For simulation purposes we start with a single stream of data and double them up to 32K parallel streams. Fig. 4 gives the throughput of HC-128 and HC-256 for increasing number of parallel data-streams on our CUDA device. The trend of throughput rise shown by the two ciphers is similar, having an apparent peak for 64 parallel streams. The CUDA device used has a total of 16 MPs and each MP can accommodate 8 blocks at most. Maximum utilization of MPs is achieved for 128 parallel streams of data (16×8). Further increase in the number of parallel data-streams shows a slight improvement in the throughput. The reason is that the parallel streams in excess of 128 are waiting in instruction queue and are launched with negligible context switch time. The maximum throughput achieved is 31 Gbps for HC-128 and 14 Gbps for HC-256 employing 32768 parallel streams.

4.3 Throughput Comparison of HC Series of Stream Ciphers on Various Platforms

We compare our acceleration results with the only available figures for HC-128 acceleration on GPUs by D. Stefan in his masters thesis [7]. Without employing parallelism within a single data-stream for HC-128, he assigned one thread to one data-stream. For supporting multiple data-streams, he employed global memory for S-boxes. The highest throughput achieved is reported and compared with our implementation in Table 5. For the same number of blocks, our throughput is approximately 14 times higher. Comparing the cycles/byte performance also shows a significant decrease. Results for *initialization* phase are not available for comparison.

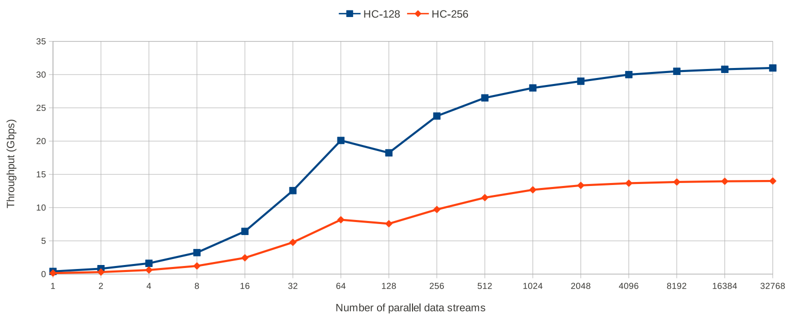


Fig. 4. *Keystream generation* throughput for varying number of multiple data-streams

Table 5. Comparison of our HC-128 acceleration with D. Stefan [7]

	Implementation by D. Stefan[7]	Our Implementation
NVIDIA device	GeForce GTX 295	GeForce GTX 590
Release date	January 8, 2009	March 24, 2011
Compute Capability	1.3	2.0
Memory Used	Global Memory	Global Memory
Threads / data-stream	1	192
data-stream / Block	256	1
Total blocks used	680	680
Total data-streams	680×256	680
Total threads used	680×256	192×680
Performance(Cycles/byte)	4.39	0.279
Throughput(Gbps)	2.26	31

The HC-128 performance evaluation on CPU was done using the *eSTREAM testing framework* [6]. The C implementation of the testing framework was installed in the CPU machine (specs given in Appendix D) on CentOs 5.8 (Linux version 2.6.18-308.11.1.el5xen). For the benchmark implementation of HC-128 and HC-256 the highest *keystream generation* speeds were found to be 2.36 cycles/byte and 3.63 cycles/byte respectively. Table 6 gives a comparison of throughput of HC series of stream ciphers on various platform. The throughput obtained on an AMD Phenom^{TM} II X6 1100T Processor is 10.94 Gbps and 7.5 Gbps for *keystream generation* phase of HC-128 and HC-256 respectively. The high speed rendered by CPU is primarily because it has to incur no memory overhead for RAM located contents unlike the GPU memory accesses. Moreover, the limitation of SIMD architecture of GPUs requires homogeneity of warp threads which is not a limitation in CPUs. Consequently the CUDA mapping of the HC family of ciphers is 11-18 times slower. The ASIC based implementation proposed by Chattopadhyay *et al.* is so far the fastest reported implementation of HC-128 claiming a throughput of 22.88 Gbps [23]. The throughput results of HC-256 are however not reported.

Table 6. Throughput (Gbps), Cycles/Byte (C/B) of a single data-stream HC ciphers

	AMD Phenom^{TM} II X6 1100T		NVIDIA GeForce GTX 590		ASIC [23] (65nm Technology)	
HC-128	10.9 Gbps	2.36 C/B	0.95 Gbps	9.27 C/B	22.88 Gbps	0.5 C/B
HC-256	7.5 Gbps	3.63 C/B	0.41 Gbps	21.82 C/B	Not reported	Not reported

For multiple data-streams we get promising results which for CPUs is not straightforward to implement. For 32768 parallel data-streams, our GPU gives a throughput of 31 Gbps for HC-128 and 14 Gbps for HC-256. Hence we conclude that HC-series of stream ciphers is unfit to be off-loaded to GPUs in case of a single data-stream application. In contrast, an application exploiting multiple parallel data-streams can achieve GPU acceleration up to 2.8 times faster in case of HC-128 and 1.87 times faster for HC-256 (with 32768 parallel data-streams).

5 Conclusion and Future Work

This work presents the first detailed study of algorithmic acceleration limitations in HC series of stream ciphers for mapping on a GPU device. The high degree of data dependency in their S-box update procedures puts strict limitations on exploiting the inherent parallelism that a graphics device offers. Moreover these ciphers are primarily data intensive in nature. These limitations explain the absence of relevant scientific publications in this arena. We present various strategies to improve the throughput of the HC-128 and HC-256 ciphers at the cost of replicated copies of S-Boxes. However, for a single data-stream acceleration, our throughput does not go beyond 0.95 Gbps and 0.41 Gbps for HC-128 and HC-256 respectively on a GeForce GTX 590 (leaving it 11-18 times slower than a standard CPU in throughput).

For multiple data-streams, however, we beat the CPU performance. We did a thorough tuning on the GPU for optimizing all the architectural features that the device could offer. Thread and warp grouping is done so as to expose enough parallelism to the device to keep all the MP cores busy all the time. Our GPU based acceleration resulted in being 2.8 times faster than CPU in case of HC-128 and 1.87 times faster for HC-256 (with 32,768 parallel data-streams). Hence we conclude that GPUs can successfully be employed as a co-processor with a CPU host to accelerate HC series of stream ciphers using multiple parallel streams of data. As future work, we plan to investigate the parallelism opportunities offered by the entire eSTREAM portfolio [12] of software stream ciphers and compare the performance against today's CPUs.

References

1. Manavski, S.A.: CUDA compatible GPU as an efficient hardware accelerator for AES cryptography. In: International Signal Processing and Communications (ICSPC), pp. 65–68. IEEE (2007)

2. Biagio, A., Barenghi, A., Agosta, G., Pelosi, G.: Design of a parallel AES for graphics hardware using the CUDA framework. In: International Symposium on Parallel & Distributed Processing (IPDPS), pp. 1–8. IEEE (2009)
3. Iwai, K., Nishikawa, N., Kurokawa, T.: Acceleration of AES encryption on CUDA GPU. International Journal of Networking and Computing 2(1), 131–145 (2012)
4. Liu, G., An, H., Han, W., Xu, G., Yao, P., Xu, M., Hao, X., Wang, Y.: A Program Behavior Study of Block Cryptography Algorithms on GPGPU. In: Fourth International Conference on Frontier of Computer Science and Technology 2009, FCST 2009, pp. 33–39. IEEE (2009)
5. Nishikawa, N., Iwai, K., Kurokawa, T.: High-Performance Symmetric Block Ciphers on Multicore CPU and GPUs. International Journal of Networking and Computing 2(2), 251–268 (2012)
6. Cannire, C.D.: eSTREAM testing framework, `http://www.ecrypt.eu.org/stream/perf`
7. Stefan, D.: Analysis and Implementation of eSTREAM and SHA-3 Cryptographic Algorithms (2011), `http://hgpu.org/?p=5972`
8. Bauer, M., Cook, H., Khailany, B.: CudaDMA: Optimizing GPU memory bandwidth via warp specialization. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, New York (2011); Article 12
9. `http://stanford-cs193g-sp2010.googlecode.com/svn/trunk/lectures/lecture_4/cuda_memories.pdf`
10. Bernstein, D.: Cache-timing attacks on AES (2005), `http://cr.yp.to/papers.html#cachetiming`
11. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997)
12. eSTREAM: the ECRYPT Stream Cipher Project, `http://www.ecrypt.eu.org/stream`
13. Kircanski, A., Youssef, A.M.: Differential Fault Analysis of HC-128. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 261–278. Springer, Heidelberg (2010)
14. Liu, Y., Qin, T.: The key and IV setup of the stream ciphers HC-256 and HC-128. In: International Conference on Networks Security, Wireless Communications and Trusted Computing, pp. 430–433. IEEE (2009)
15. Maitra, S., Paul, G., Raizada, S., Sen, S., Sengupta, R.: Some observations on HC-128. Designs, Codes and Cryptography 59(1-3), 231–245 (2011)
16. Zenner, E.: A Cache Timing Analysis of HC-256. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 199–213. Springer, Heidelberg (2009)
17. Osvik, D.A., Shamir, A., Tromer, E.: Cache Attacks and Countermeasures: The Case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)
18. Paul, G., Maitra, S., Raizada, S.: A Theoretical Analysis of the Structure of HC-128. In: Iwata, T., Nishigaki, M. (eds.) IWSEC 2011. LNCS, vol. 7038, pp. 161–177. Springer, Heidelberg (2011)
19. Sekar, G., Preneel, B.: Improved Distinguishing Attacks on HC-256. In: Takagi, T., Mambo, M. (eds.) IWSEC 2009. LNCS, vol. 5824, pp. 38–52. Springer, Heidelberg (2009)
20. Stankovski, P., Ruj, S., Hell, M., Johansson, T.: Improved distinguishers for HC-128. Designs, Codes and Cryptography 63(2), 225–240 (2012)

21. Wu, H.: The Stream Cipher HC-128, http://www.ecrypt.eu.org/stream/hcp3.html
22. Wu, H.: A New Stream Cipher HC-256. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 226–244. Springer, Heidelberg (2004), http://eprint.iacr.org/2004/092.pdf
23. Chattopadhyay, A., Khalid, A., Maitra, S., Raizada, S.: Designing High-Throughput Hardware Accelerator for Stream Cipher HC-128. In: International Symposium on Circuits and systems (ISCAS), pp. 1448–1451. IEEE (2012)
24. NVIDIA CUDA, http://developer.NVidia.com/object/CUDA.html

Appendix A: Description of HC-128 and HC-256 Keystream Generation

HC-t uses t-bit secret key and IV, and 32-bit element internal arrays P and Q each of length $4t$, where t is either 128 or 256. We briefly sketch the *keystream generation* phase of the algorithms here. For details of key and IV setup, one may refer to [21,22]. The operators used are $+$ (addition modulo 2^{32}), $\boxminus$ (subtraction modulo 512), $\oplus$ (bit-wise exclusive OR), $\gg, \ll$ (32-bit shifts) and $\ggg, \lll$ (32-bit rotations). Let s_r denote the keystream word generated at the r-th step, $r = 0, 1, 2, \ldots$. The functions g_1 and g_2 (3 inputs for HC-128 and 2 inputs for HC-256) are used for self-update of P and Q and functions h_1 and h_2 are used in the *keystream generation*, as follows.

	HC-128	HC-256
t	128	256
g_1	$((x \ggg 10) \oplus (z \ggg 23)) + (y \ggg 8)$	$((x \ggg 10) \oplus (y \ggg 23)) + Q[(x \oplus y) \bmod 4t]$
g_2	$((x \lll 10) \oplus (z \lll 23)) + (y \lll 8)$	$((x \ggg 10) \oplus (y \ggg 23)) + P[(x \oplus y) \bmod 4t]$
h_1	$Q[x_{(0)}] + Q[2t + x_{(2)}]$	$Q[x_{(0)}] + Q[t + x_{(1)}] + Q[2t + x_{(2)}] + Q[3t + x_{(3)}]$
h_2	$P[x_{(0)}] + P[2t + x_{(2)}]$	$P[x_{(0)}] + P[t + x_{(1)}] + P[2t + x_{(2)}] + P[3t + x_{(3)}]$
$P[j]+=$	$g_1(P[j \boxminus 3], P[j \boxminus 10], P[j \boxminus 511])$	$P[j \boxminus 10] + g_1(P[j \boxminus 3], P[j \boxminus 1023])$
$Q[j]+=$	$g_2(Q[j \boxminus 3], Q[j \boxminus 10], Q[j \boxminus 511])$	$Q[j \boxminus 10] + g_2(Q[j \boxminus 3], Q[j \boxminus 1023])$

The last two rows of the above table show the self-update steps (SUS) for the arrays P and Q. Here $x = x_{(3)} \| x_{(2)} \| x_{(1)} \| x_{(0)}$ is a 32-bit word, with $x_{(0)}, x_{(1)}, x_{(2)}$ and $x_{(3)}$ being the four bytes from right to left. The *keystream generation* phase happens in cycles of $8t$ rounds, in the first $4t$ of which the array P is updated followed by a keystream word generation step (KWGS) $s_i = h_1(P[j \boxminus 12]) \oplus P[j]$. In the next $4t$ rounds, the array Q is updated and the corresponding KWGS is given by $s_i = h_2(Q[j \boxminus 12]) \oplus Q[j]$.

Appendix B: List of Operations for Keystream Generation in HC-128 and HC-256

	HC-128 SUS	HC-128 KWGS	HC-256 SUS	HC-256 KWGS
Modulo Additions	2	2	3	7
Xor	1	1	2	1
Modulo Subtractions	3	1	3	1
Rotations	3	0	2	0
Shifts	0	1	0	3
Total operations	**9**	**5**	**10**	**12**
Memory Reads	4	4	5	6
Memory Writes	1	1	1	1
Total memory accesses	**5**	**5**	**6**	**7**

Appendix C: Overview of CUDA Programming Model

CUDA exposes the device as a repository of thousands of parallely executable threads as shown in Fig. 5. The GPU chip is organized as a collection of multiprocessors (MPs). Each MP has a number of Stream Processors (SPs), each handling one thread. Each MP is responsible for handling one or more thread blocks. Since thread blocks have no dependencies among themselves, their assignment is independent of MPs allowing transparent scaling of programs across different GPUs. Here are some technical terms relevant to the CUDA execution model.

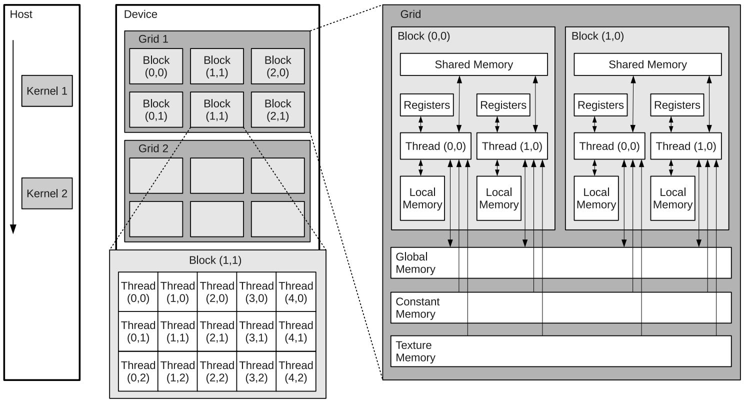


Fig. 5. CUDA GPU execution model

1. **Thread**: the smallest unit of execution in CUDA.
2. **Warp**: the threads are forwarded to the CUDA MPs in groups (*warps*) of 32 for execution. If all thread kernels in a warp are homogeneous, all the SPs in an MP execute the same instruction in a true SIMD fashion.
3. **Block**: a group of threads each with exclusive local memories and registers and a single shared memory as shown in Fig. 5.

4. **Grid**: one or more thread blocks being executed by a kernel in memory form a grid. Each MP handles one or more blocks in a grid. Threads in a block are not divided across multiple MPs.
5. **Kernel**: a block of code called from the host CPU, and then sent to the device with a grid of thread blocks. CUDA gives the freedom of choosing the threads and block structure and dimension to the coder.

Appendix D: Hardware Specifications of CPU and GPU used for Throughput Comparison

	AMD Phenom^{TM}II X6 1100T	NVIDIA GeForce TX 590
Transistors	904 million	6 billion
Processor Frequency (GHz)	3.31	1.2
Cores/SPs	6	1024
Cache/shared Memory	L2-512 KB, L3-6 MB$\times$6	48 KB$\times$32
Threads executed per cycle	6	1024
Active Hardware threads	6	49152 (maximum)

Trusted Launch of Virtual Machine Instances in Public IaaS Environments

Nicolae Paladi[1], Christian Gehrmann[1], Mudassar Aslam[1], and Fredric Morenius[2]

[1] Swedish Institute of Computer Science, Stockholm, Sweden
[2] Ericsson Research, Stockholm, Sweden
{nicolae,chrisg,mudassar.aslam}@sics.se, fredric.morenius@ericsson.com

Abstract. Cloud computing and Infrastructure-as-a-Service (IaaS) are emerging and promising technologies, however their adoption is hampered by data security concerns. At the same time, Trusted Computing (TC) is experiencing an increasing interest as a security mechanism for IaaS. In this paper we present a protocol to ensure the launch of a virtual machine (VM) instance on a trusted remote compute host. Relying on Trusted Platform Module operations such as *binding* and *sealing* to provide integrity guarantees for clients that require a trusted VM launch, we have designed a trusted launch protocol for VM instances in public IaaS environments. We also present a proof-of-concept implementation of the protocol based on OpenStack, an open-source IaaS platform. The results provide a basis for the use of TC mechanisms within IaaS platforms and pave the way for a wider applicability of TC to IaaS security.

Keywords: IaaS, security, trusted computing, trusted virtual machine launch, OpenStack.

1 Introduction

One of the distinguished trends in IT operations today is the consolidation of IT systems onto common platforms. A key technology in realizing this is system virtualization [1]. System virtualization makes it possible to streamline IT operations, save energy and obtain better utilization of hardware resources. A virtualized computing infrastructure allows clients to run own services in form of Virtual Machines (VM) on shared computing resources. This approach however introduces new challenges, as it means that information previously controlled by one administrative domain and organization, is now under the control of a third party provider and that the information owner loses direct control over how data and services are used and protected. IaaS [2] is one of the business models based on system virtualization and security aspects are among the main identified obstacles for its adoption[1]. The problems with securing IaaS are evident not least

[1] AFCEA Cyber Committee – October, 2011, http://www.afcea.org/mission/intel/documents/cloudcomputingsecuritylessonslearned_final.pdf

through the fact that widely known platforms such as Amazon EC2, Microsoft Azure, services provided by RackSpace and other IaaS services are plagued by vulnerabilities at several levels of the software stack, from the web based cloud management console [3] to VM side-channel attacks, to information leakage, to collocation with malicious virtual machine instances [4].

A promising approach towards handling IaaS security threats and a mean to provide service confidence is the use of Trusted Computing technologies as defined by the Trusted Computing Group (TCG) [5]. The core component in the TCG-defined security architecture is the Trusted Platform Module (TPM), a hardware module that can be used as a trust anchor for software integrity verification in open platforms that also offers protected storage for sensitive parameters. TPM usage and deployment models for IaaS clouds are currently an active research area [6,7,8,9,10,11]. Earlier research has introduced principles of a trusted IaaS platform [9], later extended to cover both trusted VM launch [10] and VM migration [11]. These research results demonstrate principles of combining basic TPM attestation mechanisms with standard cryptographic techniques to design an infrastructure for VM protection. However, such solutions have limitations with respect to security, complexity and target compute host selection procedures.

In this paper we describe a trusted VM launch process that addresses these limitations and present a trusted launch protocol that does not require secure pre-packaging of the VM image on the client side. Furthermore, in order to be usable in a significant proportion of IaaS deployment scenarios and to provide full scheduling flexibility on the IaaS side, the protocol allows the IaaS provider to select a target trusted compute host without directly involving the client. The main contributions of this paper are:

1. Description of a trusted launch protocol for VM instances in public IaaS environments.
2. Implementation of the proposed protocol based on a widely-known IaaS platform.

The paper is further organized as follows: In section 2 we define the trust and attack models, formulate the problem area and define requirements for a scheme to address the identified issues; section 3 presents the main contribution of the paper, namely a platform-agnostic protocol for trusted virtual machine launching. In section 4 we perform a security analysis of the proposed protocol and continue with a description of the prototype implementation based on the OpenStack IaaS platform in section 5. In section 6 we provide summaries of significant related work within trusted computing in IaaS environments. We conclude in section 7 and provide a set of further research suggestions.

2 Trust and Attack Models, Problem Description and Requirements

Next we describe the trust and attack models we assume in this paper, list the top security and general design requirements applicable given the defined trust and attack models and revisit virtual machine images in the context of a trusted VM instance launch. We also discuss the characteristics that can be expected from a well-designed VM instance launch.

2.1 Trust and Attack Models

In the trust model and consequently the attack model used in this paper, the client does not implicitly trust any aspect of the IaaS provider. Although potentially true for many IaaS environment types, this trust model should be particularly relevant to public IaaS environments (according to the definition in [12]), where the relationship between the client and the IaaS provider is often formal and lacks prerequisites for implicit trust.

We share the attack model with [9,10,11] which assume that privileged access rights can be maliciously used by IaaS provider remote system administrators ($\mathcal{A}_r$). This scenario assumes that $\mathcal{A}_r$ can log in remotely to any host maintained by the IaaS provider and obtain root access. However, in this model $\mathcal{A}_r$ does not have physical access to the hosts. The only possibility for $\mathcal{A}_r$ to circumvent this constraint is by succeeding to force a client to launch their VM instance on an $\mathcal{A}_r$-controlled compute host outside the physically secured IaaS provider perimeter. Furthermore, we assume that an $\mathcal{A}_r$ obtaining remote root access to the compute host will not be able to access the volatile memory of any VM residing on the compute host at that time, i.e. the compute host offers VMs a closed box execution environment[2]. This assumption is required in order to ensure that $\mathcal{A}_r$ can not access the nonce provided by $\mathcal{C}$ and its implementation is out of the scope of this paper.

In a trusted VM launch context this means that we consider both the attack where $\mathcal{A}_r$ attempts to launch a VM instance on a non-trusted compute host instead of on a trusted one and the attack where $\mathcal{A}_r$ attempts to substitute the VM image requested by the client with a maliciously modified VM image.

In the current attack model, a VM instance is considered trusted if and only if it fulfils the following criteria:

1. The VM image used for the instance is itself trusted;
2. The VM instance is started on a trusted compute host;
3. The VM instance has the client-generated verification token injected (see section 3.1)

2.2 Virtual Machine Images

As an implication of the above trust and attack models, we consider the following two properties of virtual machines in the context of trusted computing:

– No VM instance, or any entity communicating with the VM instance, can determine whether the hypervisor the VM instance is running on is trusted or not.

[2] This does not include any VMs which are part of the hosting infrastructrure, such as Xen dom0 VM).

– A VM instance cannot be trusted to reliably determine if it has the configuration originally requested by the client.

To overcome these issues, we suggest a launch protocol where we use standard TPM v1.2 functionality to first ensure that the client can detect the situation when it is communicating with a VM instance that is not launched on a trusted platform and subsequently utilize the trusted platform to verify the integrity of the VM image prior to VM launch.

It is essential, in the scope of the protocol, that no modifications or customizations of the VM image to be launched are performed by the IaaS provider without the client's knowledge.

2.3 Requirements for a Trusted VM Launch Protocol

Considering the trust and attack models above, it is important for the client to be able to obtain reasonable security guarantees from the IaaS provider. These include both trustworthiness of the computing resources, as well as guarantees regarding VM integrity and confidentiality. In order to also be cost and implementation efficient, the underlying infrastructure should provide such guarantees with a minimal operational overhead without increasing structural complexity. The expectations can be summarized as a set of basic requirements towards a trustworthy VM launch process:

– R1: The client shall have the mechanisms to ensure that the VM instance has been launched on a trusted compute host.
– R2: The client should have the possibility to reliably determine that it is communicating with a VM instance launched on a trusted compute host, and not with a different VM instance.
– R3: The integrity of the VM image scheduled to be launched must be verifiable by the target trusted compute host.
– R4: The trusted VM launch procedure should be scalable and have a minimum impact on the performance of the IaaS platform.
– R5: Clients should have a transparent view of the trusted launch procedure.

3 A Trusted Launch Protocol for Virtual Machine Images in IaaS Environments

Based on the above requirements for a trusted launch protocol for VM instances in IaaS environments, we present a platform-agnostic protocol that shows principles of using TPM functionality to ensure the integrity of the compute host and of the VM image requested to be launched by the client. The below protocol addresses the security concerns presented above by focusing on simplicity, transparency, scalability and minimal interference with the currently known setup of the IaaS implementations. Furthermore, the protocol is based on widely-used and verified techniques, such as hashing and asymmetric cryptography in combination with TPM functionality.

The protocol requires the participation of four entities, three of which are typically involved in VM launch procedures in IaaS architectures:

1. *Client* ($\mathcal{C}$) is a IaaS user and intends to launch a VM instance. In this paper, $\mathcal{C}$ is considered to be a *non-expert*, i.e. one not capable of assessing the security of platform configurations based on values contained in the measurement list. $\mathcal{C}$ requires a VM instance to be launched on a trusted platform. Furthermore, it is important for $\mathcal{C}$ to be able to either verify or trust the security of VM images provided for launch.
2. *Scheduler* ($\mathcal{S}$) is responsible for receiving requests for VM instance launches from $\mathcal{C}$, as well as scheduling and rescheduling of VM instances on available compute hosts at the IaaS provider. $\mathcal{S}$ should be able to function with minimal involvement in the security-specific message passing.
3. The *compute host* ($\mathcal{CH}$) is the target resource that will be chosen by $\mathcal{S}$ to run the particular VM instance. $\mathcal{CH}$ represents a physical or virtual server that is able to host one or more VM instances (however, this paper considers exclusively the case when the $\mathcal{CH}$ is a physical server). For the purposes of the proposed protocol, a $\mathcal{CH}$ must also be equipped with a TCG-compliant TPM as well as be immune to modification attempts when in a trusted state.
4. The *Trusted third party ($\mathcal{TTP}$)* is, as the name implies, trusted by both the *Client* and the *IaaS provider* and can not be controlled or manipulated by the IaaS provider. The recent breaches of Certificate Authorities have emphasized the drawbacks of centralized security models and their susceptibility to attacks [13]. The more complex the operations performed by the $\mathcal{TTP}$, the higher the probability of it having exploitable vulnerabilities. It is therefore important to keep the implementation of the $\mathcal{TTP}$ as simple as possible. The main task of the $\mathcal{TTP}$ is to attest the configuration of the $\mathcal{CH}$ that will host the VM instance and assess its security profile according to predefined policies. Within the current trust model, $\mathcal{TTP}$s could be implemented by an *expert* $\mathcal{C}$, as long as the IaaS provider agrees to that and $\mathcal{C}$ has the capability to set up and operate an attestation and evaluation engine.

For the purpose of the protocol, we also introduce the concept '*security profile* of a $\mathcal{CH}$':

Definition 1. *A security profile ($\mathcal{SP}$) is a verified setup of an OS including underlying libraries and configuration files, which is considered to be* trusted *by all parties. $\mathcal{SP}$ can range on an ascending integer scale which reflects the level of verification, from least to most strict (and hence more restrictive).*

The information needed to calculate the $\mathcal{SP}$ and also to compare the setup of two $\mathcal{CH}$s is stored in the *integrity measurement log* (IML), as the IML contains hashes of the components that were loaded or used during the boot sequence of the $\mathcal{CH}$. The validity of the IML is confirmed through a signature using the attestation identity keys (AIK) of a TPM. The AIK are persistent, non-migratable keys that are used to sign and authenticate by the means of an AIK certificate (denoted by $AIK - cert$) the validity of the information provided by the TPM in case of an external attestation [14]. We thus assume that the $\mathcal{SP}$ of any given $\mathcal{CH}$ can be deterministically calculated by each of the parties involved in the protocol.[3]

3.1 Platform-Agnostic Protocol Description

The following steps are required in order to perform a trusted VM launch (Fig. 1, the steps of the protocol correspond to the steps in the figure[4]).

1. Before initiating the launch procedure, $\mathcal{C}$ generates a sufficiently long nonce $\mathcal{N}$, to be used as a proof token in communications between $\mathcal{C}$ and the VM instance and must be kept confidential to untrusted parties throughout the launch process.
2. $\mathcal{C}$ creates a token which we denote by $\mathcal{T}$, representing a data structure with information necessary for the trusted VM launch. $\mathcal{T}$ contains $\mathcal{N}$, the minimum $\mathcal{SP}$ and the hash of the VM image used for launch, denoted by $H_{VMimage}$[5]. Finally, the token is encrypted with the public key of $\mathcal{TTP}$, represented by PK_{TTP}, while the encrypted token is represented by $\mathcal{T}_{PK_{TTP}}$.
3. $\mathcal{C}$ provides the *scheduler (S)* the following parameters:
   - VM image identifier and optionally the VM image to be launched;
   - $\mathcal{SP}$;
   - URL of the $\mathcal{TTP}$;
   - Encrypted token $\mathcal{T}_{PK_{TTP}}$ generated in step **(2)**;

   $\mathcal{SP}$ will determine the lower bound of trust level required from $\mathcal{CH}$ on which the VM will run, with stricter security profiles accepted.
4. S schedules a VM on the appropriate $\mathcal{CH}$, depending on its membership in the respective security profile group and sends the $\mathcal{CH}$ a request to generate a bind key PK_{Bind}, also providing the URL of the $\mathcal{TTP}$.
5. Once the destination $\mathcal{CH}$ receives the bind key request, it retrieves a PCR-locked non-migratable TPM-based bind key PK_{Bind}. This key can be periodically regenerated by $\mathcal{CH}$ according to a administrator-defined policy, using the current platform state represented by the TPM PCRs. It is important to note that the values of the PCRs should not necessarily be in a trusted state in order to create a trusted state bind key.
6. In order to prove that the bind key is a non-migratable, PCR-locked, asymmetric TPM key, $\mathcal{CH}$ uses the `TPM_CERTIFY_KEY` TPM command in order to retrieve the `TPM_CERTIFY_INFO` structure signed with the TPM attestation indentity key [14], which we denote by PK_{AIK}; we also denote the signed structure by
$H_{\texttt{TPM\_CERTIFY\_INFO}}{}^{AIK}$. The `TPM_CERTIFY_INFO` data structure contains the hash of the bind key and the PCR value required for the key usage.

[3] The methodology for calculating the $\mathcal{SP}$ of a $\mathcal{CH}$ is out of the scope of this paper.

[4] Due to space limitations, "Attestation data" was chosen as the condensed notation for: $\mathcal{T}_{PK_{TTP}}, PK_{Bind}, \texttt{TPM\_CERTIFY\_INFO}, H_{\texttt{TPM\_CERTIFY\_INFO}}{}^{AIK}, \ IML, AIK - cert$

[5] If non-repudiation of VM launch is required, the client should also sign the VM image hash and include the signature and corresponding client certificate into the token.

7. $\mathcal{CH}$ sends an attestation request to the $\mathcal{TTP}$ through an HTTPS session using the URL supplied by $\mathcal{C}$. The following arguments are sent in the request to $\mathcal{TTP}$:
   - Client-provided token $\mathcal{T}_{PK_{TTP}}$
   - *Attestation data*, which includes the public bind key, the `TPM_CERTIFY_INFO` structure, the hash of `TPM_CERTIFY_INFO` signed with the AIK[6], the IML and the AIK-certificate collectively represented as:
     PK_{Bind}, `TPM_CERTIFY_INFO`, $H_{\texttt{TPM\_CERTIFY\_INFO}}{}^{AIK}$, IML, AIK-cert .
8. $\mathcal{TTP}$ uses its private key PrK_{TTP}, which corresponds to the public PK_{TTP} to attempt to decrypt the token $\mathcal{T}_{PK_{TTP}}$.
9. $\mathcal{TTP}$ validates the attestation information obtained from $\mathcal{CH}$ as follows:
   - Validates the AIK certificate;
   - Validates the structure of the AIK-signed `TPM_CERTIFY_INFO`;
   - Validates the key PK_{Bind} by comparing its digest with the digest received in `TPM_CERTIFY_INFO`;
   - Calculates the hash of the PCR values H_{PCR} based on the information in the IML and compares it with the hash of `PCR_INFO`, which is a component of `TPM_CERTIFY_INFO`
10. $\mathcal{TTP}$ examines the entries in the IML in order to determine the trustworthiness of the $\mathcal{CH}$ and decides whether $\mathcal{SP}$ is satisfied.
11. If step **10** is true, $\mathcal{TTP}$ encrypts $\mathcal{N}$ and the hash $H_{VMimage}$ with the bind key PK_{Bind} obtained from $\mathcal{CH}$, to ensure that $\mathcal{N}$ is only available to $\mathcal{CH}$ in a trusted state. By sending $\mathcal{N}$ and $H_{VMimage}$ encrypted with the public key PK_{Bind} available to the trusted configuration of $\mathcal{CH}$, the security perimeter expands to include three parties: $\mathcal{C}$ itself, $\mathcal{TTP}$ and $\mathcal{CH}$ in its trusted configuration. This implies that all actions performed by $\mathcal{CH}$ in its trusted configuration are trusted by default.
12. Prior to launching the VM, $\mathcal{CH}$ decrypts $\mathcal{N}$ and $H_{VMimage}$ using the TPM-issued PrK_{Bind}, which is available to it in its trusted configuration but stored in the TPM; next, $\mathcal{CH}$ compares $H_{VMimage}$ obtained from the $\mathcal{TTP}$ with the hash of the VM image to be used for launch and accepts the image only in case the values are equal.
13. $\mathcal{CH}$ injects $\mathcal{N}$ into the VM image prior to launching the VM instance.
14. $\mathcal{CH}$ returns an acknowledgement to S to confirm a successful launch.
15. To verify that the VM instance has been launched on a trusted platform, $\mathcal{C}$ challenges the VM instance to prove its knowledge of $\mathcal{N}$.

The fact that $\mathcal{N}$ is kept confidential allows it to be used as an authentication token while establishing a secure communication channel between $\mathcal{C}$ and the launched *VM* instance. $\mathcal{N}$ can be used as the pre-shared secret in order to add protection against man-in-the-middle attacks when using Diffie-Hellman key exchange, as specified in the password-authenticated key-exchange protocol [15].

Some of the operations can be optimized taking into account the operational environment. For example, the validity period of PK_{Bind} created in step **(5)**

[6] Expressed as $H_{\texttt{TPM\_CERTIFY\_INFO}}{}^{AIK}$.

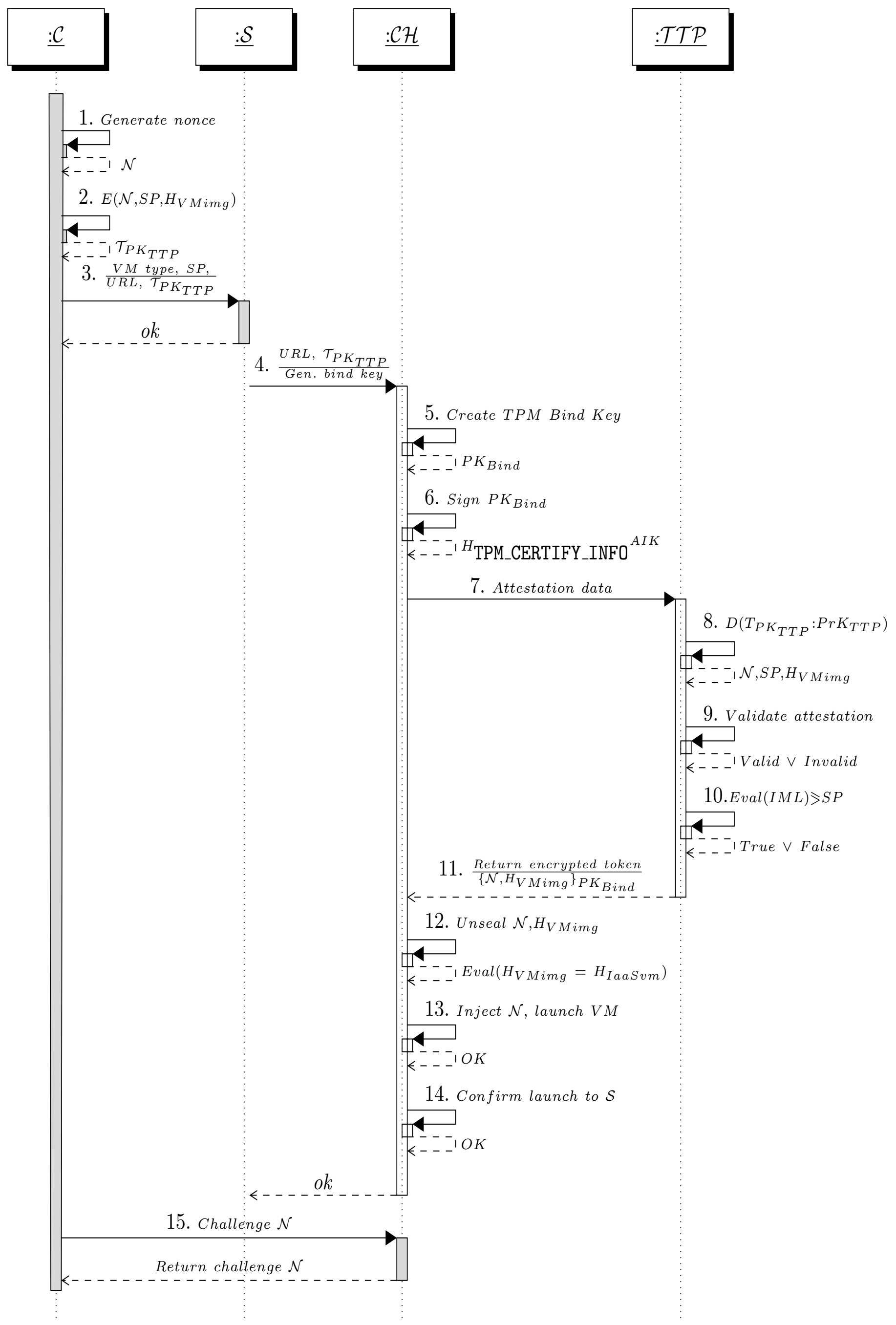
can be adjusted. In a similar way, $\mathcal{TTP}$ can have a cache of the PK_{Bind} keys created by $\mathcal{CH}$s with verified trusted configurations. In this case, steps **(9)** and **(10)** can be skipped for a certain number of cases, which can also be regulated by an administrative policy. However, it is important to remember that the use of such a cache introduces further complexity to $\mathcal{TTP}$, the analysis of which is out of the scope of this paper.

4 Protocol Security Analysis

In this section we present a critical review of the protocol and highlight improvement areas that were left as future work. We begin with a security analysis of the protocol, in order to outline its strengths and weaknesses.

Returning to the security concerns expressed in the requirements on the trusted launch protocol formulated in section 2.3, they are addressed as follows. Let φ be the guest VM instance launched on $\mathcal{CH}$, then:

- R1: Following above protocol, $\mathcal{C}$ and φ have a shared secret $\mathcal{N}$. The fact that φ is running on a *trusted platform* is ensured by the properties of the bind key used to seal the shared secret $\mathcal{N}$ to the trusted configuration of $\mathcal{CH}$;
- R2: The fact that $\mathcal{C}$ is communicating with φ and not any other unexpected VM instance φ' is ensured through the combination of: **a.** verification of $\mathcal{CH}$ by the $\mathcal{TTP}$, **b.** presence of the token $\mathcal{N}$ injected into φ where $\mathcal{N}$ is only available to $\mathcal{CH}$ in a trusted state; **c.** the VM image integrity verification performed by the $\mathcal{CH}$ prior to the launch. A failure at any of the steps of the above sequence would prevent the trusted VM launch, a fact that would be verifiable by $\mathcal{C}$.
- R3: Integrity of the VM image is ensured through the verification performed by $\mathcal{CH}$ in a trusted state, prior to the trusted VM launch. Thus, the VM image is verified using the hash value obtained from the TTP. By comparing the hash of the VM image with the expected $H_{VMimage}$ provided by $\mathcal{C}$, $\mathcal{CH}$ ensures a one-to-one correspondence between the VM image to be used for launch and the VM image expected by $\mathcal{C}$. The chain is completed once $\mathcal{C}$ verifies the presence of $\mathcal{N}$ injected into φ. The presence of the correct token $\mathcal{N}$ guarantees the integrity of φ requested by $\mathcal{C}$.
- R4: Scalability of the protocol is ensured by the lightweight nature of operations that must be performed by both $\mathcal{TTP}$ and $\mathcal{CH}$ and the flexibility in the choice of $\mathcal{TTP}$. While a challenging topic, especially in the case of high-availability and heavy load IaaS setups, the design of a scalable $\mathcal{TTP}$ architecture is out of the scope of this paper.
- R5: Transparency of the trusted VM launch procedure is ensured by the introduction of client parameters, such as the URL of the $\mathcal{TTP}$, the trust level of $\mathcal{CH}$ and the secret token generated by $\mathcal{C}$. The ability to choose $\mathcal{TTP}$ opens the possibility for $\mathcal{C}$ to ensure the trustworthiness of the $\mathcal{CH}$ attestation procedure, either through audit controls of the $\mathcal{TTP}$ or by itself serving the role of $\mathcal{TTP}$.



:$\mathcal{C}$
:$\mathcal{S}$
:$\mathcal{CH}$
:$\mathcal{TTP}$
1. Generate nonce
$\mathcal{N}$
2. $E(\mathcal{N}, SP, H_{VMimg})$
$\mathcal{T}_{PK_{TTP}}$
3. $\frac{VM\ type,\ SP,}{URL,\ \mathcal{T}_{PK_{TTP}}}$
ok
4. $\frac{URL,\ \mathcal{T}_{PK_{TTP}}}{Gen.\ bind\ key}$
5. Create TPM Bind Key
PK_{Bind}
6. Sign PK_{Bind}
$H_{\mathtt{TPM\_CERTIFY\_INFO}}^{AIK}$
7. Attestation data
8. $D(T_{PK_{TTP}}:PrK_{TTP})$
$\mathcal{N}, SP, H_{VMimg}$
9. Validate attestation
$Valid \vee Invalid$
10. $Eval(IML) \geqslant SP$
$True \vee False$
11. $\frac{Return\ encrypted\ token}{\{\mathcal{N}, H_{VMimg}\}PK_{Bind}}$
12. Unseal $\mathcal{N}, H_{VMimg}$
$Eval(H_{VMimg} = H_{IaaSvm})$
13. Inject $\mathcal{N}$, launch VM
OK
14. Confirm launch to $\mathcal{S}$
OK
ok
15. Challenge $\mathcal{N}$
Return challenge $\mathcal{N}$

Fig. 1. Trusted VM launch protocol: $\mathcal{C}$: Client; $\mathcal{S}$: Scheduler; $\mathcal{CH}$: Compute Host; $\mathcal{TTP}$: Trusted Third Party

4.1 $\mathcal{TTP}$ Verification Model

The stateless architecture of the $\mathcal{TTP}$ implies that it does not maintain knowledge of $\mathcal{N}$ except for at the moment of sealing it to $\mathcal{CH}$ and does not maintain any session state at any point of the protocol. As a result, an $\mathcal{A}_r$ can only obtain $\mathcal{N}$ from $\mathcal{TTP}$ if they obtain $\mathcal{TTP}$'s private key PrK_{TTP}. Furthermore, assessment of the trust level of a $\mathcal{CH}$ according to a deterministic algorithm which only takes two inputs (in the form of static set of reference measurement data and dynamic attestation calls from any $\mathcal{CH}$) will be easily traceable and reproducible based on the original input data, without the need to recreate or rely on a certain state of the TPP's internal data. Finally, a stateless architecture of the $\mathcal{TTP}$ contributes indirectly towards requirement R4.

4.2 Protocol Caveats

One aspect that requires more attention is the possibility of a post-launch modification of the software stack of $\mathcal{CH}$. The runtime process infection method, which is a method for infecting binaries during runtime[7] is one of the malicious approaches that could be used to this end. This scenario is in fact a common threat to all TCG-based systems, also touched upon in [16], described in detail in [17] and should thus be prevented using means within the platform which is part of the trusted computing base verified at boot time, the presence of which is verified by the above protocol.

5 Protocol Implementation

In order to validate the assumptions made during the protocol design phase, we have implemented it as an extension to OpenStack, an open source IaaS platform chosen given the open access to its codebase, its large community and the traction it has gained. This section briefly introduces the OpenStack architectural model and changes made for the prototype implementation.

5.1 OpenStack IaaS Platform

The Essex release of OpenStack comprises five core components (projects), namely Compute (Nova), Image Service (Glance), Object Storage (Swift), Identity Service (Keystone) and Dashboard (Horizon). Nova has several sub-components: nova-api, nova-compute, nova-schedule, nova-network, nova-volume, plus an SQL database and message queue functionality to pass messages between sub-components. OpenStack components affected by the protocol implementation are mentioned here in more detail:

[7] Runtime process infection, `http://www.phrack.org/issues.html?issue=59&id=8&mode=txt`

- Nova-api is the interface for nova- compute and volume API calls. It is through this interface most of the cloud orchestration operations are performed. The interface supports both the OpenStack and Amazon EC2 API.
- Nova-compute handles VM instance life cycle tasks through hypervisor API calls. Notably the libvirt and XenAPI hypervisor APIs are supported.
- Nova-schedule is responsible for selecting $\mathcal{CH}$(s) to run VM instances on. The $\mathcal{CH}$ selection process is determined by which scheduling policy/algorithm is employed.
- The nova SQL database holds tables and relations to describe the state of nova, such as launched instances and network configurations.
- The Dashboard is a web based GUI for OpenStack operation and administration. It interfaces nova-api.

5.2 Prototype Implementation

Below are the main additions to OpenStack required for the prototype implementation.

Nova SQL database The nova SQL database has been extended to include tables to hold the available $\mathcal{CH}$s and their $\mathcal{SP}$s:

- An $\mathcal{SP}$ is an integer in the range 1-10, with a higher number being more trusted than a lower number.
- The security profile of a $\mathcal{CH}$ is global, rather than specific per e.g. tenant.

Dashboard and nova-api The Dashboard web based GUI has been extended to include the option to request $\mathcal{CH}$ attestation, minimum $\mathcal{SP}$ selection, token $\mathcal{T}_{PK_{TTP}}$ entry and $\mathcal{TTP}$ URL provision **(3)** into the *"Launch Instance"* dialog. This information is included in the OpenStack API HTTP payload to nova-api, which propagates the information to the scheduler.

In the prototype implementation, steps **(1)** and **(2)** are performed by a script which outputs $\mathcal{T}_{PK_{TTP}}$, which then can be manually input into the Dashboard dialog. Note that it is not an option to let Dashboard provide functionality for generating $\mathcal{T}_{PK_{TTP}}$, since Dashboard is not trusted by $\mathcal{C}$.

Scheduler, compute host and virtualization driver The nova scheduler $\mathcal{S}$ is a central component as it decides on which $\mathcal{CH}$ a certain VM instance will be launched. Each $\mathcal{S}$ works according to a specific configurable algorithm and several $\mathcal{S}$ implementations are available in OpenStack by default. In the SimpleScheduler implementation, $\mathcal{S}$ identifies the least loaded $\mathcal{CH}$ and schedules the VM instance to be launched on that $\mathcal{CH}$.

We extend the behaviour of the SimpleScheduler to include the policy that a $\mathcal{CH}$ must belong to a certain $\mathcal{SP}$ or stricter in order to be acceptable for hosting the VM instance. This policy is realized as follows: first $\mathcal{S}$ looks up the recorded $\mathcal{SP}$ of $\mathcal{CH}$ in the nova database and proceeds if $\mathcal{SP}$ is sufficient compared to the requirements of $\mathcal{C}$ (corresponds to **(4)**). The second step is to request $\mathcal{CH}$ to attest itself with $\mathcal{TTP}$. If $\mathcal{SP}$ was not sufficient, the next eligible $\mathcal{CH}$ is selected.

Steps **(5)**-**(7)** are perfomed by $\mathcal{CH}$, followed by $\mathcal{TTP}$ in steps **(8)**-**(11)**. Token $T_{CH} = \{\mathcal{N}, H_{VMimage}\}_{PK_{Bind}}$ is returned from $\mathcal{TTP}$ to $\mathcal{CH}$ after which $\mathcal{CH}$ includes the token in the return message to $\mathcal{S}$. If the attestation was successful, $\mathcal{S}$ requests the now trusted $\mathcal{CH}$ to launch the VM instance and includes $\mathcal{T}_{CH}$ in the request.

Next, $\mathcal{CH}$ decrypts $\mathcal{T}_{CH}$ and obtains $\mathcal{N}$ and $H_{VMimage}$. To verify the integrity of the VM image, $H_{VMimage}$ is included in the call to the virtualization driver (`libvirt` is used by the prototype), which fetches the VM image from Glance and caches it locally on $\mathcal{CH}$. The hash of the cached image is calculated and compared to $H_{VMimage}$. If the hashes do not match, an exception is raised. Otherwise, the launch procedure continues **(12)** and the file injection capability of Nova is used to inject $\mathcal{N}$ into the file system of the VM image **(13)**. The VM image is then used to launch the VM instance on $\mathcal{CH}$ and steps **(14)** and **(15)** are completed.

6 Related Work

Application of trusted computing principles within IaaS environments has been the focus of several research papers examined below.

Santos et al propose the design of a "trusted cloud compute platform" (TCCP) that ensures VMs are running on a trusted hardware and software stack with a remote and initially untrusted $\mathcal{CH}$ [9]. The authors propose a remote attestation process where a trusted coordinator ($\mathcal{TC}$) stores the list of attested $\mathcal{CH}$s that run a "trusted virtual machine monitor" which can securely run the client's VM. A trusted $\mathcal{CH}$ maintains in its memory an individual *trusted key* (TK) used for identification each time the client $\mathcal{C}$ instantiates a VM on the trusted $\mathcal{CH}$. The paper presents a good initial set of ideas for trusted VM launch and migration, in particular the use of a $\mathcal{TC}$. A limitation of this solution is that the TK resides in the memory of the trusted $\mathcal{CH}$, which leaves the solution vulnerable to cold boot attacks [18] with keys extractable from memory. Furthermore, the authors require that the $\mathcal{TC}$ maintains information about all $\mathcal{CH}$ deployed on the IaaS platform, but do not mention mechanisms for anonymizing this information, making it valuable to an attacker and unacceptable for a public IaaS provider. Finally, the solution lacks both mechanisms for revocation of the TK and considerations for the re-generation of TK outside of $\mathcal{CH}$ reboot.

A decentralized approach to integrity attestation is adopted by Schiffman et al in [19]. The primary concerns addressed by this approach are the limited transparency of IaaS platforms and the limits to scalability imposed by third party integrity attestation mechanisms, as described in [9]. The authors examine a trusted cloud architecture where the integrity of the IaaS $\mathcal{CH}$ is verified by the IaaS client through a "cloud verifier" ($\mathcal{CV}$) proxy that resides in the application domain of the IaaS platform provider and is accessible by the client. Thus, in the first step of the protocol the client evaluates the integrity of the $\mathcal{CV}$ in order to include the $\mathcal{CV}$ into its trust perimeter if the integrity level of the $\mathcal{CV}$ is considered satisfactory. Next, the $\mathcal{CV}$ sends attestation requests from $\mathcal{CH}$, i.e. the

$\mathcal{CH}$ where the guest VM instance can potentially be deployed, thus extending the trust chain to the $\mathcal{CH}$. Finally, $\mathcal{CH}$ verifies the integrity of the VM image, which is countersigned by the $\mathcal{CV}$ and returned to the client which evaluates the VM image integrity data and allows or disallows the VM launch on the $\mathcal{CH}$. While the idea of increasing the transparency of the IaaS platform for the client is indeed supported in industry [20,21], the authors do not clarify how the introduction of an additional integrity attestation component in the architecture of the IaaS platform has positive effects on the transparency of the IaaS platform. Furthermore, the proposed protocol increases the complexity model for the IaaS client both by introducing the evaluation of integrity attestation reports of the $\mathcal{CV}$ and $\mathcal{CH}$ and introduction of additional steps in the trusted VM launch, where the client has to take actions based on the data returned from the $\mathcal{CV}$. This requires either human interaction or a fairly complex integrity attestation evaluation component (or a combination thereof) on the client side, making a wide-scale adoption of the solution difficult.

In [10], Aslam et al proposed principles for trusted VM launch on public cloud platforms using trusted computing techniques. In order to ensure that the requested VM instance is launched on a $\mathcal{CH}$ with verifiable integrity, the client encrypts the VM image (along with all injected data) with a symmetric key sealed to a particular configuration of $\mathcal{CH}$, which is reflected through the values in the platform configuration registers (PCR) of the TPM deployed on the $\mathcal{CH}$. The solution proposed by Aslam et al presents a suitable model in the case of trusted VM launch scenarios for enterprise clients. It requires that the VM image is pre-packaged and encrypted by $\mathcal{C}$ prior to IaaS launch. However the proposed model does not cover the very common scenario of launching an unmodified VM image made available by the IaaS provider or uploaded by $\mathcal{C}$. Furthermore, we believe that reducing the number of steps required from $\mathcal{C}$ will facilitate the adoption of the trusted IaaS model. Likewise, direct communication between $\mathcal{C}$ and $\mathcal{CH}$, as well as significant changes to the existing VM launch implementations in IaaS platforms hamper the implementation of this protocol. This paper reuses some of the ideas proposed in [10] and directly addresses the above limitations, namely actions to be performed by $\mathcal{C}$, also touching upon the requirements towards the launched VM instance and required changes to the IaaS platform.

7 Conclusion

In this paper we have presented a detailed trusted launch protocol for VM instance launch in public IaaS environments. Furthermore, we have provided a prototype implementation of the launch protocol in OpenStack. Detailed performance measurement and evaluation, as well as alternative implementation choices have been left for future work.

The presented results make a case for broadening the range of use cases for trusted computing by applying it to IaaS environments, especially within the security model of an untrusted IaaS provider. Trusted computing offers capabilities to securely perform data manipulations on remote hardware owned and maintained by another party by potentially preventing the use of untrusted software on that hardware for such manipulations. The presented design is directly applicable to the process of developing a trusted virtualized environment, e.g. a public IaaS service.

Future research recommendations can be grouped into three categories:

First is the extension of the trust chain to other operations on VM instances (migration, suspension, updates, etc.), as well as data storage and virtual network communication security.

The second category includes addressing certain assumptions of the proposed launch protocol, e.g. the assumption that the $\mathcal{CH}$ configuration is not changed after the trusted launch of the VM instance, since even in the case of a bona fide IaaS provider the $\mathcal{CH}$ can be compromised through runtime process infection. A technique to enable $\mathcal{C}$ to either directly or through mediated access discover such events and protect the data used by the VM instance is a promising research topic.

The third category focuses on the design and implementation of the evaluation policies of the TTP. The current assumption is that the TTP has access to information regarding "secure" configurations and the PCR values, something which needs to be directly addressed as evaluating exactly how secure a certain software stack is, is a challenge. Furthermore, taking into account the diversity of available libraries as well as the different combinations in which they can be loaded during the boot process, verification of PCR values (such as values stored in `PCR10` and reference values in `binary_runtime_measurements`) becomes a less trivial task.

References

1. Smith, J., Nair, R.: Virtual Machines: Versatile Platforms for Systems and Processes. Morgan Kaufmann (June 2005)
2. Krutz, R.L., Vines, R.D.: Cloud Security: A Comprehensive Guide to Secure Cloud Computing. John Wiley & Sons (August 2010)
3. Somorovsky, J., Heiderich, M., Jensen, M., Schwenk, J., Gruschka, N., Lo Iacono, L.: All Your Clouds are Belong to us: Security Analysis of Cloud Management Interfaces. In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security, CCSW 2011, pp. 3–14. ACM, New York (2011)
4. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS 2009, pp. 199–212. ACM, New York (2009)
5. Pohlmann, N., Reimer, H.: Trusted Computing - eine Einführung. In: Pohlmann, N., Reimer, H. (eds.) Trusted Computing, pp. 3–12. Vieweg+Teubner (2008), doi:10.1007/978-3-8348-9452-6_1
6. Neisse, R., Holling, D., Pretschner, A.: Implementing Trust in Cloud Infrastructures. In: 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 524–533 (May 2011)
7. Sadeghi, A.-R., Stüble, C., Winandy, M.: Property-Based TPM Virtualization. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 1–16. Springer, Heidelberg (2008)

8. Danev, B., Masti, R.J., Karame, G.O., Capkun, S.: Enabling Secure VM-vTPM Migration in Private Clouds. In: Proceedings of the 27th Annual Computer Security Applications Conference, ACSAC 2011, pp. 187–196. ACM, New York (2011)
9. Santos, N., Gummadi, K.P., Rodrigues, R.: Towards Trusted Cloud Computing. In: Proceedings of the 2009 Conference on Hot Topics in Cloud Computing, HotCloud 2009. USENIX Association, Berkeley (2009)
10. Aslam, M., Gehrmann, C., Rasmusson, L., Björkman, M.: Securely Launching Virtual Machines on Trustworthy Platforms in a Public Cloud - An Enterprise's Perspective. In: Leymann, F., Ivanov, I., van Sinderen, M., Shan, T. (eds.) CLOSER, pp. 511–521. SciTePress (2012)
11. Aslam, M., Gehrmann, C., Björkman, M.: Security and Trust Preserving VM Migrations in Public Clouds. In: 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Liverpool (2012)
12. Mell, P., Gance, T.: The nist definition of cloud computing. Technical report, National Institute of Standards and Technology (September 2011)
13. Goyal, P.: Application of a Distributed Security Method to End-2-End Services Security in Independent Heterogeneous Cloud Computing Environments. In: 2011 IEEE World Congress on Services, pp. 379–384 (July 2011)
14. Trusted Computing Group: TCG Specification, Architecture Overview, revision 1.4. Technical report, Trusted Computing Group (2007)
15. Boyko, V., MacKenzie, P., Patel, S.: Provably secure password-authenticated key exchange using diffie-hellman. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer, Heidelberg (2000)
16. Price, M.: The Paradox of Security in Virtual Environments. Computer 41(11), 22–28 (2008)
17. Wojtczuk, R., Rutkowska, J., Tereshkin, A.: Attacking intel trusted execution technology. In: Black Hat USA 2008, Las Vegas, NV, August 7 (2008)
18. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest We Remember: Cold-Boot Attacks on Encryption Keys. Commun. ACM 52, 91–98 (2009)
19. Schiffman, J., Moyer, T., Vijayakumar, H., Jaeger, T., McDaniel, P.: Seeding Clouds With Trust Anchors. In: Proceedings of the, ACM Workshop on Cloud Computing Security, CCSW 2010, pp. 43–46. ACM, New York (2010)
20. Molnar, D., Schechter, S.: Self Hosting vs. Cloud Hosting: Accounting for the Security Impact of Hosting in the Cloud. In: Workshop of the Economics of Cloud Security, pp. 1–18 (2010)
21. Chen, Y., Paxson, V., Katz, R.: The Hybrex Model for Confidentiality and Privacy in Cloud Computing. Technical Report UCB/EECS-2010-5, EECS Department, University of California, Berkeley (January 2010)

Secure and Privacy-Aware Multiplexing of Hardware-Protected TPM Integrity Measurements among Virtual Machines

Michael Velten and Frederic Stumpf

Fraunhofer Research Institution AISEC, Munich, Germany
{michael.velten,frederic.stumpf}@aisec.fraunhofer.de

Abstract. Measuring the integrity of critical operating system components and securely storing these measurements in a hardware-protected Trusted Platform Module (TPM) is a well-known approach for improving system security. However, currently it is not possible to securely extend this approach to TPMs used in virtualized environments. In this paper, we show how to multiplex integrity measurements of arbitrarily many Virtual Machines (VMs) with just a single standard TPM. In contrast to existing approaches such as vTPM, our approach achieves a higher level of security since measurements will never be held in software but are fully hardware-protected by the TPM at all times. We establish an integrity-protected mapping between each measurement and its respective VM such that it is not possible for an attacker to alter this mapping during remote attestation without being detected. Furthermore, all measurements will be stored in the TPM in a concealed manner in order to prevent information leakage of other VMs during remote attestation. The experimental results of our proof of concept implementation show the feasibility of our approach.

Keywords: Integrity Measurement, Attestation, Trusted Platform Module, Trusted Computing, Virtualization.

1 Introduction

Virtualization and the utilization of a Hardware Security Module (HSM) are two well-known approaches for improving system security. Virtualization can be used to partition a system into several Virtual Machines (VMs) such that critical system components are isolated from one another and to allow for a reduced Trusted Computing Base (TCB) of the overall system. Virtualization is also heavily used in the context of cloud computing where multiple VMs of different customers run concurrently on the same system platform. In this context, it is crucial that one VM cannot access or manipulate data of another VM.

An HSM is a hardware device usually capable of securely managing cryptographic keys and storing data such that it is not possible for an attacker to extract or manipulate these keys and data. A very prominent and widespread HSM is the Trusted Platform Module (TPM) [1] as specified by the Trusted Computing Group (TCG) [2]. In particular, the TPM can be used to securely store integrity measurements in special Platform Configuration Registers (PCRs) that reflect a system's configuration. An *authenticated boot* is used to establish a *chain of trust* by measuring each component in the booting sequence, starting with an inherently trusted component called the Core Root of Trust for Measurement (CRTM). Developments such as the Integrity Measurement Architecture (IMA) [3] extend this chain of trust to the application layer by measuring programs executed in the OS and storing the measurements in a PCR of the TPM. Finally, the integrity measurements are used in the course of a *remote attestation* to prove to a remote party that the system platform is in a trusted state.

Unfortunately, the TPM was not designed to be used in virtualized environments and thus the advantages of virtualization and HSMs cannot be easily combined. In particular, the TPM was not designed to store integrity measurements on a per-VM basis. Furthermore, it is impossible to perform remote attestations only for particular VMs. Researchers have proposed several ideas to tackle these problems. The emulation of PCRs in software for each VM was proposed in [4,5]. However, on a compromised system these PCRs can be manipulated by an attacker, allowing him to forge remote attestations. There also exist proposals that describe next-generation TPMs with hardware-based virtualization support that do not suffer from the aforementioned security vulnerability [6,7,8]. However, such TPMs are not available yet.

In this paper, we show how to multiplex integrity measurements of arbitrarily many VMs with just a single standard TPM and only requiring one PCR. In contrast to [4,5], which emulate PCRs in software, our approach achieves a higher level of security since measurements are always stored in the hardware-protected PCRs of the TPM. We show how to establish an integrity-protected mapping between each measurement and its respective VM such that it is not possible for an attacker to alter this mapping (e.g., hiding malicious programs by mapping their measurements to other VMs) without being detected. Furthermore, we develop a remote attestation protocol for attesting the integrity of individual VMs. A crucial problem we have to solve in the context of remote attestation is that our approach of sharing PCRs among VMs, inherently requires the disclosure of all measurements of all VMs. This entails security and privacy issues as even a legitimate challenger in the remote attestation protocol is then able to determine exactly which software is running in all other VMs. This information might be used to exploit (known) vulnerabilities of that software. We overcome this problem by storing all measurements in the multiplexed PCR in a concealed manner. This enables us to fully disclose the (concealed) contents of the PCR and to selectively reveal non-concealed measurements on a per-VM basis. Finally, the experimental results of our proof of concept implementation show the feasibility of our approach.

The rest of this paper is organized as follows. Sect. 2 first gives an overview of our concept, states our assumptions, describes the threat model, and then explains in detail the multiplexing technique used for storing measurements and attesting individual VMs. Sect. 3 gives the security analysis. Sect. 4 describes our proof of concept implementation. Sect. 5 presents our evaluation results. Sect. 6 discusses related work. Sect. 7 concludes this paper.

2 Concept

Our concept is based on a virtualized platform consisting of a single hardware TPM, a hypervisor, and arbitrarily many VMs. A *multiplexing agent* (MPA) is located in the hypervisor (or in a privileged VM) and processes integrity measurements received from the VMs and stores them in the TPM. In each VM runs a *measurement agent* (MA) (e.g., IMA [3]) that monitors the execution of supervised files, calculates integrity measurements, and forwards them to the MPA. Alternatively, the files could be monitored by using passive monitoring techniques where the monitoring is implemented in the hypervisor (e.g., HIMA [9] or Patagonix [10]). Our proposed multiplexing concept is compatible with either approach.

2.1 Overview of Multiplexed Storage and Attestation

The MPA stores integrity measurements in a single shared PCR of the TPM. Each of the 24 PCRs of a TPM may (conceptually) hold arbitrarily many measurements by *extending* them as a hash chain, i.e., $PCR[i] \leftarrow SHA1(PCR[i]||m)$, for a measurement m and PCR i (where $||$ denotes concatenation). However, to retain the (integrity-protected) information in which VM a measurement m took place, the MPA not only extends m but also the corresponding VM's unique virtual machine identifier (VM-ID) in the PCR (cf. Sect. 2.3).

Furthermore, the MPA is able to attest the integrity of individual VMs to a verifier (cf. Sect. 2.4). However, without further precautions, this requires the disclosure of *all* measurements of *all* VMs sharing the PCR. This entails security and privacy related problems as described in the introduction. Therefore, before extending the PCR, the MPA first conceals each measurement with a special value called *concealment.* A concealment is a non-predictable random or pseudo-random value that is at least the size of the output of the $SHA1$ hash function in which we will use it (cf. Sect 2.3). The reason for this size is to adequately protect against lookup attacks trying to extract the plain measurements. Note that the concept of a concealment is related to the concept of a salt. However, in contrast, a concealment is unknown to a verifier and will only be disclosed to him when attesting a particular VM. The MPA maintains one *base concealment* for each VM and derives further concealments from it. In addition to concealing measurements, we also conceal the measurement's associated VM-ID to prevent a verifier from gathering information about how many measurements have been conducted in other VMs. This information might otherwise be misused to detect usage patterns (e.g., activity level of VMs of competitors).

Finally, this enables the MPA to disclose all measurements of all VMs in a concealed manner to a verifier. For the attested VM, the non-concealed measurements, along with the attested VM's base concealment, are additionally revealed. The base concealment is used by the verifier to derive the same concealments as the MPA, which are then used to link the non-concealed measurements to their corresponding concealed measurements. This, in turn, allows the verifier to recalculate the proper hash chain (consisting of concealed measurements only) and to match it against the (signed) PCR value, thus ensuring the measurements' integrity and authenticity (cf. Sect. 2.5).

2.2 Assumptions and Threat Model

Assumptions. In the remote attestation protocol, we focus on the measurement data of the multiplexed PCR as we assume the rest of the system can be attested with general remote attestation techniques. We assume the MPA utilizes some form of VM-based (one-way) authentication of parties requesting a remote attestation for a particular VM. Note that even for authenticated parties, we still need to privacy-protect the measurement data of other (non-authenticated) VMs as done by our approach.

Threat Model. We consider all man-in-the-middle (MITM) attacks on the remote attestation protocol. The MITM is located between the prover and the verifier and is able to intercept and manipulate the transmitted data (e.g., discarding or forging measurements). We also consider attacks on the attesting platform in which an attacker tampers with software (e.g., by forging remote code updates). The malicious software (or the attacker) may try to hide its execution by removing or manipulating its respective integrity measurement. We do not consider direct physical attacks on the TPM.

2.3 Integrity Measurement Transformation and Storage

We let $id_{vm_1}, \dots, id_{vm_n}$ denote unique and publicly known VM-IDs w.r.t. the set of all n VMs on a particular system. The MPA maintains for each VM id_{vm} one non-predictable *base concealment* $c_{vm} \in \{0,1\}^k$, with $k \geq 160$ (i.e., at least the size of the output of $SHA1$). For the i'th measurement transformation (counting from zero) of a VM id_{vm}, the MPA derives a new *concealment* c^i_{vm} by incrementing c_{vm} i times, that is, $c^i_{vm} := c_{vm} + i,\ i \geq 0$. Note that c^0_{vm} denotes the base concealment c_{vm}. For brevity, we define $H := SHA1$ in the remainder of this document.

Each time MA measures (the content of) a monitored file f executed in VM id_{vm} by calculating $m := H(f)$ and forwards it to the MPA, the MPA associates m with id_{vm}, conceals both m and id_{vm}, and extends the result to the shared PCR p. In particular, for the i'th measurement of VM id_{vm}, the MPA does the following five steps (called a *round* in the following):

1. Derive new VM-specific concealment c^i_{vm} from base concealment c_{vm}
2. Conceal measurement m by hashing it with c^i_{vm}, i.e., $\mu := H(m||c^i_{vm})$
3. Conceal VM-ID id_{vm} with same concealment c^i_{vm}, i.e., $\delta_{vm} := H(id_{vm}||c^i_{vm})$
4. Hash over the concealed measurement value μ combined with the concealed VM-ID δ_{vm}, i.e., $\varphi := H(\mu||\delta_{vm})$
5. Extend the TPM's shared PCR p with φ

Note that it is not possible to defer this measurement transformation (e.g., to the point in time where a remote attestation is requested) because the measurement must immediately be stored in the TPM in order to prevent an attacker from removing or manipulating previous integrity measurements once the system gets compromised.

Step one guarantees that we use a new concealment for each round. It is important that a verifier is able to produce the exact same sequence of concealments $c_{vm}^1, c_{vm}^2, \ldots$ from the base concealment $c_{vm} = c_{vm}^0$ (cf. Sect. 2.5). Note that simple incrementation is sufficient for deriving the concealments (in terms of confidentiality of the concealed values in steps two and three) since two consecutive (and thus similar) concealments c_{vm}^i and c_{vm}^{i+1} result in two completely different hash values $H(c_{vm}^i)$ and $H(c_{vm}^{i+1})$ due to the avalanche effect [11]. Step two makes sure that it is sufficient to only disclose concealed measurements to a verifier V in order to reconstruct the hash chain represented by the shared PCR p. V can easily verify that a measurement m of the attested VM corresponds to the concealed hash value μ by checking whether $\mu = H(m||c_{vm}^i)$ holds. Note that it is infeasible to find some other preimage $x \neq m||c_{vm}^i$ such that $H(x) = H(m||c_{vm}^i)$ because of the second-preimage resistance property of H. In step three, we conceal the VM-ID to prevent V from gathering usage patterns of other VMs. Note that the usage of a static (VM-based) concealment c_{vm} would always map a VM-ID id_{vm} to the same concealed VM-ID $\delta_{vm} = H(id_{vm}||c_{vm})$, thus allowing to link (concealed) VM-IDs and measurements. We use different concealments for each round in order to prevent this. Step four establishes the mapping between μ and δ_{vm} and thus implicitly also between m and id_{vm}. In step five, the concealed hash value φ gets finally extended to the PCR p by using φ as the incoming operand `TPM_DIGEST` of the `TPM_Extend` command [1]. Note that it is sufficient to use the standard, non-modified `TPM_Extend` operation. Also note that storing the just described mapping between measurement and VM-ID directly in the integrity protected PCR (`PCR_Quote` may be used to sign the value of the PCR) makes it redundant to maintain an external integrity protected mapping.

Multiplexed Measurement List (MML). The final hash chain value contained in PCR p is not sufficient to reconstruct the actual measurement data. Therefore, the MPA separately stores all measurement data in chronological order w.r.t. their corresponding `TPM_Extend` operations in the *multiplexed measurement list* (MML). The MML is an ordered list of pairs of the form (m, id_{vm}), where m is a (non-concealed) measurement and id_{vm} the corresponding (non-concealed) VM-ID id_{vm}, denoting the VM in which the measurement took place, that is:

$$MML := \big\langle (m_0, id_{vm_{i_0}}), (m_1, id_{vm_{i_1}}), \ldots, (m_n, id_{vm_{i_n}}) \big\rangle$$

2.4 Integrity Reporting

Fig. 1 shows our adapted remote attestation protocol enabling the integrity reporting of individual VMs. Note that the remote attestation process actually

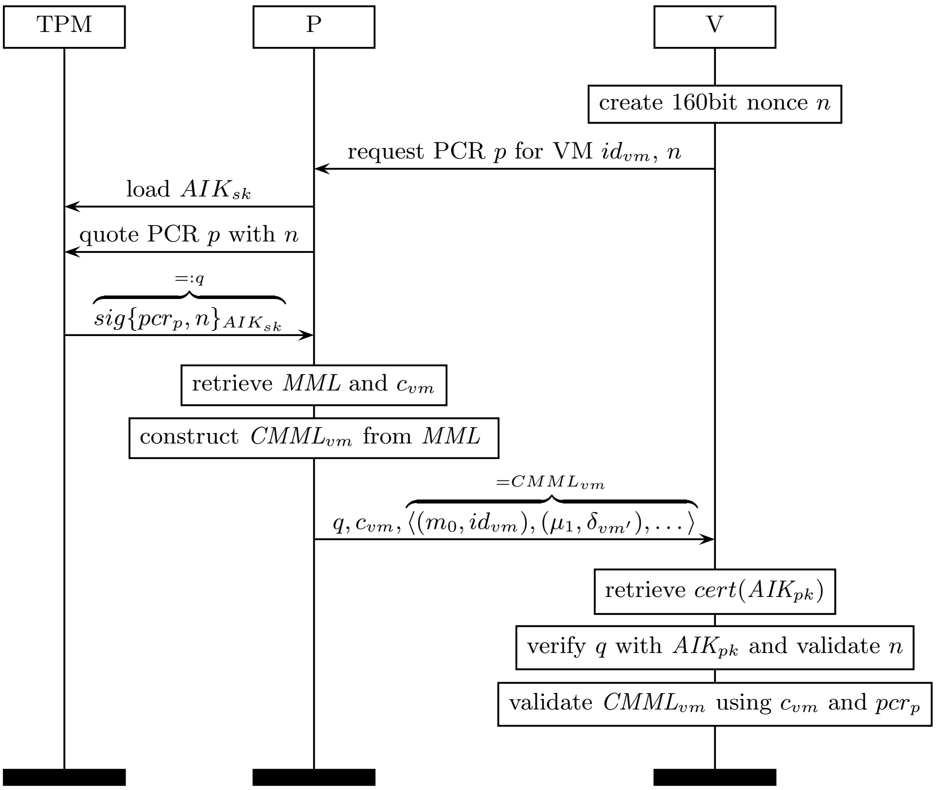

TPM
P
V
create 160bit nonce n
request PCR p for VM id_{vm}, n
load AIK_{sk}
quote PCR p with n
$\overbrace{sig\{pcr_p, n\}_{AIK_{sk}}}^{=:q}$
retrieve MML and c_{vm}
construct $CMML_{vm}$ from MML
$q, c_{vm}, \overbrace{\langle(m_0, id_{vm}), (\mu_1, \delta_{vm'}), \ldots\rangle}^{=CMML_{vm}}$
retrieve $cert(AIK_{pk})$
verify q with AIK_{pk} and validate n
validate $CMML_{vm}$ using c_{vm} and pcr_p

Fig. 1. Multiplexed remote attestation protocol

consists of the integrity reporting phase as explained in the following as well as of the integrity validation phase as explained in Sect. 2.5.

First, the verifier V requests integrity measurement data for a particular VM and PCR p by providing the VM's unique and publicly known VM-ID id_{vm}. The prover P (the MPA in our case) then signs the content pcr_p of the requested PCR p with a special key of the TPM, a so-called Attestation Identity Key (AIK). This proves to V the content of the requested PCR. In the next step, the VM-specific concealed multiplexed measurement list (CMML) $CMML_{vm}$ is constructed from the MML. Recall that all pairs of the MML are non-concealed.

The construction is done by sequentially processing all pairs of the MML from left to right. Pairs not belonging to the attested VM id_{vm} are substituted with their concealed counterparts. In particular, the i'th occurrence (counting from zero) of a pair $(m, id_{vm'}) \in MML$, for some measurement m and some VM-ID $id_{vm'} \neq id_{vm}$, gets substituted with $(H(m||c^i_{vm'}), H(id_{vm'}||c^i_{vm'}))$. Pairs belonging to the attested VM remain non-concealed. Finally, P sends $CMML_{vm}$, the base concealment c_{vm}, and the signature data q of content pcr_p to V. Note that the MPA may cache the concealed pairs to avoid recalculating them for each remote attestation.

Algorithm 1. Validation of CMML

```
1: procedure VALIDATE_CMML($id_{vm}, cmml_{vm}, c_{vm}, pcr$)
2:     $pcr' := 0$
3:     $used_c := false$
4:     for $(a, b)$ in $cmml_{vm}$ do
5:         if $b = id_{vm}$ then                         ▷ does pair belong to attested VM?
6:             $\mu \leftarrow H(a||c_{vm})$                  ▷ construct concealed measurement value
7:             $\delta \leftarrow H(b||c_{vm})$                  ▷ construct concealed VM-ID
8:             $c_{vm} \leftarrow c_{vm} + 1$                    ▷ set concealment for next round
9:             $used_c \leftarrow true$          ▷ exhaustive blinding attack not possible anymore
10:        else
11:            $\mu \leftarrow a$                             ▷ measurement already concealed
12:            $\delta \leftarrow b$                             ▷ VM-ID already concealed
13:            if $\delta = H(id_{vm}||c_{vm}) \lor \delta = H(id_{vm}||c_{vm} - 1)$ then
14:                return $false$                              ▷ blinding attack
15:            end if
16:        end if
17:        $\varphi \leftarrow H(\mu||\delta)$
18:        $pcr' \leftarrow H(pcr'||\varphi)$                        ▷ simulate PCR_Extend
19:    end for
20:    if $pcr' = pcr \land used_c = true$ then
21:        return $true$                            ▷ confirm integrity of $cmml_{vm}$
22:    else
23:        return $false$                            ▷ integrity violation detected
24:    end if
25: end procedure
```

2.5 Integrity Validation

In the following, we will describe how to validate the CMML and its contained measurements which were transmitted in an attestation protocol run as described before. We will show that a validation always fails if a MITM manipulates combinations of $CMML_{vm}$, c_{vm}, and q. The verification process done by V is twofold.

In the first phase, the CMML is validated to make sure that a MITM did not tamper with it and that consequently all contained measurements are correct. The validation process is shown in Algorithm 1 and will be explained in the following. In the second phase, V inspects these measurements to determine the trustworthiness of the attested VM. This might be done by a whitelist or blacklist approach that checks for good measurements (e.g., legitimate programs) or bad measurements (e.g., known malware), respectively. However, the second phase is outside the scope of this paper.

V first uses AIK_{pk} on q to verify the authenticity and integrity of content pcr_p of the requested PCR p. This detects all manipulations of q by a MITM as well as replay attacks due to the included nonce n. V then validates the CMML with the help of c_{vm} and pcr_p. The validation process is shown in Algorithm 1. It simulates all `PCR_Extend` operations that have (allegedly) been done by P and compares the result with the signed PCR value $pcr_p =: pcr$. This is done by inspecting each pair (a, b) of the CMML. Each pair (a, b) with $b = id_{vm}$ contains non-concealed measurement data a for the attested VM id_{vm}. However, since all measurements have been extended to the PCR by P in a concealed manner (cf. Sect. 2.3), V needs to reconstruct the corresponding concealed value $\varphi := H(\mu||\delta)$, where $\mu := H(a||c^i_{vm})$ and $\delta := H(b||c^i_{vm})$ (for round i), in order to correctly simulate all `PCR_Extend` operations. All other pairs (a, b) with $b \neq id_{vm}$ do not belong to the attested VM id_{vm} and have already been concealed by P, that is, $a = \mu$ and $b = \delta$. Thus, the concealed values μ and δ can be directly used to construct $\varphi := H(\mu||\delta)$. Finally, φ is used to simulate the `PCR_Extend` operation. These steps are repeated for each pair of the CMML. If the final simulated PCR value pcr' matches the signed PCR value pcr, the measurements of the CMML correctly reflect the actual measurements of the attested VM.

The check in line 13 of Algorithm 1 detects *blinding attacks* where a MITM tries to hide non-concealed pairs $(a, b) = (a, id_{vm})$ belonging to the attested VM id_{vm}. The *blinding* is done by substituting pairs (a, id_{vm}) with their corresponding concealed pairs $(\mu, \delta) := (H(a||c^i_{vm}), H(id_{vm}||c^i_{vm}))$, with the intention of misleading V into thinking that the concealed pairs (μ, δ) do not belong to VM id_{vm}. Note that in this case the recalculated pcr' would still match pcr since (μ, δ) has indeed been extended to the TPM. Note also that in our concept we intentionally conceal a measurement m and its corresponding VM-ID id_{vm} separately instead of concealing m and id_{vm} combined, e.g., $\varphi := H(m||id_{vm}||c^i_{vm})$. In the latter case, it would be impossible for V to check whether a non-concealed pair (m, id_{vm}) has been blinded (i.e., checking whether $\varphi = H(m||id_{vm}||c^i_{vm})$ holds) because the measurement m is unknown to V. We will come back to blinding attacks in the security analysis in Sect. 3.

A special case of the described blinding attack is to blind *all* non-concealed pairs and to additionally substitute the base concealment $c_{vm} = c^0_{vm}$ with some $c'_{vm} \neq c^0_{vm} \wedge c'_{vm} \neq c^1_{vm}$. Note that in this case the check for blinding attacks in line 13 fails since the original base concealment c_{vm} used for the (first) blinding operation now differs from the concealment c'_{vm} used in the check. Furthermore, the substitution of c_{vm} with c'_{vm} will not be detected since c'_{vm} is never used to calculate a concealed pair out of a non-concealed one (because there are no non-concealed pairs left) and thus pcr' matches pcr. Therefore, in order to detect such *exhaustive blinding attacks*, we explicitly check in lines 9 and 20 of Algorithm 1 that V used the base concealment in the calculation of pcr'.

3 Security Analysis

An attacker might try to remove or manipulate previous measurements on a compromised system. However, all measurements, along with the mapping to their respective VM-IDs, are stored in the hardware-protected PCR of the TPM and thus it is impossible to remove or manipulate them.

In the context of a remote attestation, a MITM might try to simply discard or substitute measurements (e.g., malicious programs) contained in the transmitted CMML or to manipulate a measurement's associated VM-ID. However, in each case the hash chain value *pcr′* calculated by V from the CMML (cf. Algorithm 1) will not match the TPM's quoted PCR value *pcr* anymore and the attestation will fail.

In a blinding attack, a MITM substitutes measurements *and* VM-IDs with their corresponding concealed pairs. There exist four types of blinding attacks (and combinations thereof) w.r.t. the position of the blinded pairs within the CMML. In the following, we will show that our concept protects against each type. Note that since concealed pairs in the CMML do not influence the state of the concealment c_{vm} in Algorithm 1, we consider, w.l.o.g., only pairs of the attested VM. In particular, we consider the following *CMML* (along with signature data q and base concealment c_{vm}) is sent from P to V in the course of a remote attestation protocol run as described in Sect. 2.4:

$$CMML_{vm} = \langle (m_0, id_{vm}), (m_1, id_{vm}), (m_2, id_{vm}) \rangle$$

Intermediate Blinded Pairs. In this attack, a MITM blinds a pair (or several consecutive pairs) which is neither the first pair nor the last pair of the CMML. In other words, there exists at least one pair before and after the blinded pair, respectively:

$$q, c_{vm}^0, \Big\langle (m_0, id_{vm}), \underline{(H(m_1||c_{vm}^1), H(id_{vm}||c_{vm}^1))}, (m_2, id_{vm}) \Big\rangle$$

In this case, even without the explicit check for blinded pairs in line 13 of Algorithm 1, the attestation fails because the wrong concealment c_{vm}^1 is used by the algorithm to conceal the third pair (since the concealment will not be incremented when processing the intermediate blinded pair (cf. Algorithm 1, lines 11 to 15)).

Trailing Blinded Pairs. In this attack, a MITM blinds one or more consecutive trailing pairs:

$$q, c_{vm}^0, \Big\langle (m_0, id_{vm}), \underline{(H(m_1||c_{vm}^1), H(id_{vm}||c_{vm}^1))}, \underline{(H(m_2||c_{vm}^2), H(id_{vm}||c_{vm}^2))} \Big\rangle$$

Note that in contrast to the previous scenario, in this case the attestation would actually succeed if there was not the explicit check for blinded pairs. The reason is that the "out of sync" concealment will not be used anymore after concealing the first pair (as was the case above). With the explicit check, the algorithm detects $H(id_{vm}||c_{vm}^1)$ in the second pair and the attestation fails. In general, the check always matches the leftmost trailing blinded pair.

Leading Blinded Pairs. In this attack, a MITM blinds one or more consecutive leading pairs:

$$q, c_{vm}^2, \Big\langle \underline{(H(m_0||c_{vm}^0), H(id_{vm}||c_{vm}^0))}, \underline{(H(m_1||c_{vm}^1), H(id_{vm}||c_{vm}^1))}, (m_2, id_{vm}) \Big\rangle$$

Note that in this type of attack, the MITM needs to manipulate the transferred base concealment such that it correctly blinds the first non-concealed pair in the

CMML. In particular, since the base concealment is now c_{vm}^2, the explicit check for the first pair on whether δ equals $H(id_{vm}||c_{vm}^2) \vee H(id_{vm}||c_{vm}^1)$ fails because c_{vm}^0 was used for the blinding by the MITM. However, the check matches the second pair and the attestation fails. In general, the check always matches the rightmost leading blinded pair.

Exhaustively Blinded Pairs. In this attack, a MITM blinds *all* pairs and substitutes the base concealment c_{vm}^0 with some $c'_{vm} \neq c_{vm}^0 \wedge c'_{vm} \neq c_{vm}^1$. Since in this case the above checks fail (as explained in Sect. 2.5), we enforce the usage of the base concealment in the calculation of pcr' in order to detect the base concealment's manipulation, thus preventing such attacks.

4 Implementation

We have implemented a proof of concept using the QEMU emulator [12] (version 1.0.50) with enabled KVM [13] full virtualization support. The host system runs the Ubuntu OS (version 11.04). Each guest VM runs Ubuntu 10.04 with an IMA-enabled Linux kernel (2.6.35). We patched the IMA kernel code so that measurements are not directly extended to the TPM but instead are forwarded to the MPA running in the host system. The MPA has exclusive access to the TPM (using TrouSerS [14], version 0.3.5-2) and implements the multiplexing concept as described in Sect. 2.

The communication between the MPA and the VMs is done over Virtual LAN (VLAN). The MPA listens on a dedicated range of ports for incoming connections. Whenever a new VM is started, QEMU connects the VM to a free port in that range with a guest forward (*guestfwd*) rule. The so established socket is then used by our patched IMA to forward measurements to the MPA; all communication over other ports is blocked. Furthermore, the MPA uses this port number to derive the unique VM-ID of the connected VM. This mapping cannot be changed from within the VM in an attempt to forge the VM-ID since it is maintained solely by QEMU and the MPA.

Note that our patched IMA does not block until the measurements have actually been extended to the TPM. It rather just forwards them to the MPA and is immediately ready for further tasks. The MPA asynchronously processes and extends the measurements in a round-robin fashion as soon as they arrive. This significantly increases response times and overall performance in the VMs.

5 Evaluation

We use our implementation to determine whether the MPA might constitute a possible performance bottleneck since it represents the centralized location where all measurements from all VMs are collected, processed, and extended to the TPM. The testing hardware consists of a PC with an Intel Core2 Duo 3 GHz CPU, 4,096 MB RAM, and a TPM 1.2.7.40 from STM. Each VM gets assigned

Table 1. Average processing time for 10,000 measured files in each VM (in seconds)

VMs	No IMA	Patched IMA		Ratio VM	Ratio total
		VM only	Total		
1	48.87	86.73	200.84	1.77	4.11
2	50.91	104.96	400.29	2.06	7.86
3	79.61	171.85	601.02	2.16	7.55
4	108.73	229.32	825.06	2.11	7.59
5	146.32	295.27	1318.17	2.02	9.01

512 MB RAM and contains 10,000 distinct testing binaries which, on execution, just return. Furthermore, each VM runs our patched IMA that we additionally modified for the evaluation such that only the testing binaries get processed. To start the testing, we simultaneously trigger in all VMs the execution of the testing binaries in successive order.

Table 1 shows the testing results. Column four lists the total time needed from measuring all files to extending the measurements in the TPM. Note that the TPM requires most of the computation time. It takes about 200ms for 10,000 operations. Column three shows the fraction spent in a VM (on average) for measuring and forwarding. Column two lists the time needed by a VM (on average) running no IMA at all. The latter system allows us to better compare how the parallel execution of multiple VMs naturally slows down program execution time in the VMs because of shared hardware resources. In fact, the parallel execution of more than three VMs exhibits such behavior for both our approach and the system running no IMA at all, as indicated by our results. The time ratio in column five indicates an overhead of factor ≈ 2 for our approach considering the time spent in the VMs. This is due to forwarding the measurements over VLAN to the MPA. Techniques like shared memory may be used to further reduce this overhead. The total ratio in column six reflects mainly the time needed for the TPM extend operations as noted above. We can see that our approach scales roughly linearly with the number of VMs. The increased slowdown with more than three VMs is mainly due to the rather limited hardware resources of our testing system as it occurs also with the system running no IMA at all. Hence, the results indicate that the MPA does not constitute a performance bottleneck.

For the remote attestation, in order to attest a single VM, we need to send the measurement data (CMML) of all VMs (cf. Sect. 2.4). Thus, the size of the transferred data increases linearly with the number of VMs. This is a disadvantage compared to other approaches that emulate a set of PCRs for each VM in software [4,5] or maintain them in hardware [6,8,7], where it is sufficient to only send measurement data of the attested VM.

6 Related Work

Berger et al. describe a virtualized TPM emulating TPM functionality in software, called vTPM [4]. In particular, each VM is provided its own vTPM with its own instance of (upper) PCRs. All upper PCRs are held in software and their contents may be signed by the hardware TPM. However, this does not provide the same level of security as storing measurements in hardware-protected PCRs since measurements held in software can be manipulated by an attacker once the system is compromised.

In [5], England et al. try to reduce the complexity of approaches such as vTPM by not emulating the entire TPM interface in software. They utilize a para-virtualized approach that will pass through most of the functionality of a real TPM, but changes some aspects of the device interface. However, this approach suffers from the same problem as vTPM since (upper) PCRs are emulated in software and thus can be manipulated on a compromised system.

Feller et al. propose dcTPM [8], an architecture to multiplex several software-based TPMs, hardware TPMs, or a combination thereof. By multiplexing only hardware TPMs, the above issue of software-emulated PCRs can be solved. However, their approach does not scale very well. In fact, multiplexing cloud systems consisting of hundreds of VMs becomes infeasible in terms of technology (e.g., limited number of FPGA pins needed for multiplexing TPMs) and in terms of economy (e.g., hardware must be especially built with as many hardware TPMs as the (maximum) number of associated VMs).

In [6], Stumpf et al. propose a concept for enhancing a TPM to support hardware-based virtualization without the above scaling issues. This is achieved by employing a root-data structure that is only accessible by the hypervisor and a TPM-control structure that is used to dynamically swap TPM-context information of each VM in and out in an encrypted manner. Unfortunately, such a TPM is not available for production use.

The TCG also defines a specification for a virtualized trusted platform architecture [7]. However, such an envisioned TPM 2.0 is not available yet.

7 Conclusion

We have shown that it is possible to multiplex integrity measurements of arbitrarily many VMs with just a single standard TPM and only requiring one PCR. In contrast to existing approaches that emulate PCRs in software, our approach achieves a higher level of security since measurements, along with the mapping to their respective VMs, will always be stored in the hardware-protected PCRs of the TPM. We presented a remote attestation protocol for attesting the integrity of individual VMs. A crucial problem we had to solve in this context, was that our approach of sharing PCRs among VMs, inherently requires the disclosure of all measurements of all VMs. We overcame this by storing measurements in the PCR in a concealed manner. We additionally conceal a measurement's associated VM-ID, so as to prevent the collection of usage patterns. This enables us to fully disclose the (concealed) contents of the PCR and to selectively reveal non-concealed measurements of individual VMs. Finally, the experimental results of our proof of concept implementation show the practicality of our approach.

Acknowledgements. We would like to thank our colleagues Julian Horsch, Dominik Merli, Steffen Wagner, Sascha Wessel, and Philipp Zieris for fruitful discussions and valuable comments. This work was partly supported by the Federal Ministry of Economics and Technology (BMWi) through grant 01MD11012.

References

1. Trusted Platform Module, Main Specification, Level 2, Version 1.2, Revision 116 (2011), `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`
2. Trusted Computing Group, `https://www.trustedcomputinggroup.org/`
3. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and implementation of a tcg-based integrity measurement architecture. In: Proceedings of the 13th Conference on USENIX Security Symposium, SSYM 2004, vol. 13, p. 16. USENIX Association, Berkeley (2004)
4. Berger, S., Cáceres, R., Goldman, K.A., Perez, R., Sailer, R., van Doorn, L.: vtpm: virtualizing the trusted platform module. In: Proceedings of the 15th Conference on USENIX Security Symposium, USENIX-SS 2006, vol. 15, USENIX Association, Berkeley (2006)
5. England, P., Loeser, J.: Para-virtualized tpm sharing. In: Lipp, P., Sadeghi, A.-R., Koch, K.-M. (eds.) Trust 2008. LNCS, vol. 4968, pp. 119–132. Springer, Heidelberg (2008)
6. Stumpf, F., Eckert, C.: Enhancing trusted platform modules with hardware-based virtualization techniques. In: The International Conference on Emerging Security Information, Systems, and Technologies, pp. 1–9 (2008)
7. Virtualized Trusted Platform Architecture Specification, Version 1.0, Revision 26 (2011), `http://www.trustedcomputinggroup.org/resources/virtualized_trusted_platform_architecture_specification`
8. Feller, T., Malipatlolla, S., Kasper, M., Huss, S.A.: dctpm: A generic architecture for dynamic context management. In: 2011 International Conference on Reconfigurable Computing and FPGAs (ReConFig), November 30-December 2, pp. 211–216 (2011)
9. Azab, A.M., Ning, P., Sezer, E.C., Zhang, X.: Hima: A hypervisor-based integrity measurement agent. In: ACSAC, pp. 461–470. IEEE Computer Society (2009)
10. Litty, L., Lagar-Cavilla, H.A., Lie, D.: Hypervisor support for identifying covertly executing binaries. In: Proceedings of the 17th Conference on Security Symposium, SS 2008, pp. 243–258. USENIX Association, Berkeley (2008)
11. National Institute of Standards and Technology. Secure Hash Standard (SHA-1). Federal Information Processing Standards Publication 180-1 (1993)
12. Bellard, F.: Qemu, a fast and portable dynamic translator. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC 2005, p. 41. USENIX Association, Berkeley (2005)
13. Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: kvm: the Linux virtual machine monitor. In: OLS 2007: Proceedings of the Linux Symposium, vol. 1, pp. 225–230 (June 2007)
14. TrouSerS – The open-source TCG Software Stack, `http://trousers.sourceforge.net`

Improved Key Recovery Attacks on Reduced-Round Salsa20 and ChaCha*

Zhenqing Shi[1], Bin Zhang[2], Dengguo Feng[1], and Wenling Wu[1]

[1] Institute of Software, Chinese Academy of Sciences, Beijing, 100190, China
[2] Institute of Information Engineering,
Chinese Academy of Sciences, Beijing, 100195, China
zhenqingshi@gmail.com, {zhangbin,feng,wwl}@is.iscas.ac.cn

Abstract. Salsa20 is a stream cipher designed by Bernstein in 2005 and Salsa20/12 has been selected into the final portfolio of the eSTREAM Project. ChaCha is a variant of Salsa20 with faster diffusion for similar performance. The previous best results on Salsa20 and ChaCha proposed by Aumasson et al. exploits the differential properties combined with the probabilistic neutral bits (PNB). In this paper, we extend their approach by considering a new type of distinguishers, named (column and row) chaining distinguishers. Besides, we exhibit new high probability second-order differential trails not covered by the previous methods, generalize the notion of PNB to probabilistic neutral vectors (PNV) and show that the set of PNV is no smaller than that of PNB. Based on these findings, we present improved key recovery attacks on reduced-round Salsa20 and ChaCha. Both time and data complexities of our attacks are smaller than those of the best former results.

Keywords: Stream ciphers, Salsa20, ChaCha, Neutral bits, Distinguisher.

1 Introduction

Salsa20 [1] is a stream cipher designed by Bernstein in 2005 for the eSTREAM project [2]. The 8- and 12-round variants, Salsa20/8 and Salsa20/12 [3], were also published. ChaCha [4] is a variant of Salsa20 with faster diffusion in the core function for similar performance. After three evaluation phases, Salsa20/12 was selected into the final portfolio of the eSTREAM Project in 2008.

Since their publication, Salsa20 and ChaCha have undergone significant cryptographic analysis. In 2006, Crowley presented a truncated differential cryptanalysis on Salsa20/5 [5]. The result revealed that the diffusion of Salsa20 was not ideal, though it did not threaten the full round security of Salsa20. For the same variant, V. Velichkov et al. mounted a key recovery attack using UNAF [6]

* This work was supported by the programs of the National Natural Science Foundation of China (Grant No. 60833008, 60603018, 61173134, 91118006, 61272476), the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDA06010701) and the National Grand Fundamental Research 973 Program of China(Grant No. 2013CB338002).

Table 1. Comparisons of our attacks with other well known attacks on Salsa20 and ChaCha

Cipher	Round/Key length	Time	Data	Reference
Salsa20	5/256	2^{165}	2^{6}	[5]
		2^{167}	2^{7}	[6]
		2^{55}	2^{10}	This work
	6/256	2^{177}	2^{16}	[7]
		2^{73}	2^{16}	This work
	7/256	2^{151}	2^{26}	[9]
		2^{148}	2^{24}	This work
	8/256	2^{251}	2^{31}	[9]
		2^{250}	2^{27}	This work
	7/128	2^{111}	2^{21}	[9]
		2^{109}	2^{19}	This work
ChaCha	6/256	2^{139}	2^{30}	[9]
		2^{136}	2^{28}	This work
	7/256	2^{248}	2^{27}	[9]
		$2^{246.5}$	2^{27}	This work
	6/128	2^{107}	2^{30}	[9]
		2^{105}	2^{28}	This work

at FSE 2012. At Indocrypt 2006, Fischer et al. described some non-randomness properties of Salsa20/5 and used this observation to construct a key-recovery attack on Salsa20/6 [7]. At SASC 2007, Tsunoo et al. exploited a bias of Salsa20/4 to construct an attack on Salsa20/7 [8]. So far, the best key recovery attacks on variants of Salsa20 and ChaCha were proposed by Aumasson et al [9] at FSE 2008, exploiting first-order differential properties combined with the probabilistic neutral bits (PNB) technique.

In this paper, we extend the approach of Aumasson et al. by considering a new type of distinguishers, named (column and row) chaining distinguishers, which can efficiently make use of the biases of multiple differential trails and the matrix structure of the cipher. Besides, we find new high probability second-order differential trails that are not covered by the previous results, some of which are employed in our attack. The notion of PNB is generalized to that of probabilistic neutral vectors (PNV), which investigate the properties of the underlying function when more than one input bit are flipped simultaneously and include the PNB as a special case. It is shown that the set of PNV is no smaller than that of PNB. Based on these findings, we construct improved key recovery attacks on reduced-round Salsa20 and ChaCha, repectively. Both time and data complexities of our new attacks are smaller than those of the best former results. Table 1 presents our results together with comparisons with other well known attacks.

The rest of the paper is organized as follows. A brief description of Salsa20 and ChaCha is presented in Section 2. The attacks of Aumasson et al. are recalled in Section 3. In Section 4, our new attacks are described in details with the introduction of the (column and row) chaining distinguishers, high probability differential trails and the notion of PNV. Finally, we conclude the paper in Section 5.

2 Description of Salsa20 and ChaCha

2.1 Salsa20

Salsa20 operates on 32-bit words and supports keys of 128 bits and 256 bits. During its operation, the key, a 64-bit nonce (unique message number), a 64-bit counter and four 32-bit constants are used to construct the 512-bit initial state.

Denote the 256-bit key by $k = (k_0, k_1, ..., k_7)$, the 64-bit nonce by $v = (v_0, v_1)$ and the 64-bit counter corresponding to the integer i by $t = (t_0, t_1)$. The Salsa20 function acts on the following 4×4 matrix of 32-bit words.

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} = \begin{pmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ t_0 & t_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{pmatrix}.$$

The c_i's are predefined constants. For the 128-bit key k', just fill the 256 key bits in the matrix as $k = k'||k'$. If not mentioned otherwise, we focus on the 256-bit version. A keystream block Z is then defined as $Z = X + X^{20}$, where $+$ is the wordwise integer addition and $X^r = \mathsf{Round}^r(X)$ with the round function Round of Salsa20. This round function is based on the following nonlinear operation (also called the quarterround function), which transforms a vector (x_0, x_1, x_2, x_3) to (z_0, z_1, z_2, z_3) by sequentially computing

$$\begin{aligned} z_1 &= x_1 \oplus [(x_3 + x_0) \lll 7] \\ z_2 &= x_2 \oplus [(x_0 + z_1) \lll 9] \\ z_3 &= x_3 \oplus [(z_1 + z_2) \lll 13] \\ z_0 &= x_0 \oplus [(z_2 + z_3) \lll 18]. \end{aligned}$$

In odd numbers of rounds (which are called columnrounds in the original specification of Salsa20), the nonlinear operation is applied to the columns (x_0, x_4, x_8, x_{12}), (x_5, x_9, x_{13}, x_1), $(x_{10}, x_{14}, x_2, x_6)$, $(x_{15}, x_3, x_7, x_{11})$. In even numbers of rounds (which are also called the rowrounds), the nonlinear operation is applied to the rows (x_0, x_1, x_2, x_3), (x_5, x_6, x_7, x_4), $(x_{10}, x_{11}, x_8, x_9)$, $(x_{15}, x_{12}, x_{13}, x_{14})$. We write Salsa20/$R$ for R-round variants, i.e., with $Z = X + X^R$. After R iterations of the Salsa20/R round function, the updated state is used as a 512-bit keystream output. Each such output block is an independent combination of the key, nonce, and counter.

2.2 ChaCha

ChaCha is similar to Salsa20 except the following three differences: the composition of the initial matrix, the composition of the quarterround function and the round function. See [4] for details.

3 Aumasson et al's Attacks on Salsa20 and ChaCha

In this section, we briefly recall the first-order differential attacks of Aumasson et al. These attacks are based on the concept of probabilistic neutral bits (PNB). Let us first list some notations used hereafter.

- Let x_i be the i-th word of the initial matrix X and the j-th least significant bit of x_i is denoted by $[x_i]_j$.
- Let x_i' be an associated word with the difference $\Delta_i^0 = x_i \oplus x_i'$.
- The first-order differential for the input X, with a single-bit input-difference $[\Delta_i^0]_j = 1$ and a single-bit output-difference $[\Delta_p^r]_q$ after r rounds, is denoted by $([\Delta_p^r]_q|[\Delta_i^0]_j)$.

3.1 First-Order Differential Analysis of Salsa20 and ChaCha

First note that the round function of Salsa20 and ChaCha is invertible. Define the r-round inverse $X^{-r} = \mathsf{Round}^{-r}(X)$, which is different depending on whether it inverts after an odd or an even number of rounds.

For a fixed differential $([\Delta_p^r]_q|[\Delta_i^0]_j)$, the corresponding output Z and Z' can be observed for nonce v, counter t and key k. Based on $Z = X + X^R$, if we have v, t and k, we can get $X^r = (Z - X)^{r-R}$ and $(X')^r = (Z' - X')^{r-R} (r < R)$. Then $[\Delta_p^r]_q = X^r \oplus (X')^r = (Z - X)^{r-R} \oplus (Z' - X')^{r-R}$ can be observed. Define the Boolean function $f(k, v, t, Z, Z') = [\Delta_p^r]_q$. Then, for a fixed key, the bias ε_d of f is defined by $Pr\{f(k, v, t, Z, Z') = 1\} = Pr\{[\Delta_p^r]_q = 1|[\Delta_i^0]_j\} = \frac{1}{2}(1 + \varepsilon_d)$, where the probability holds over all nonces and counters.

Given enough output block pairs with the presumed differential, we have $Pr\{f(\bar{k}, v, t, Z, Z') = 1\} = \frac{1}{2}(1 + \varepsilon_d)$ conditioned on $\bar{k} = k$, whereas for (almost all) $\bar{k} \neq k$ we expect f be unbiased, i.e., $Pr\{f(\bar{k}, v, t, Z, Z') = 1\} = \frac{1}{2}$. Note that the complexity of the above way is 2^{256}. Instead, Aumasson et al. proposed to find some approximation g of f which effectively depends on s key bits ($s < 256$). Let f be correlated to g with the bias ε_a, i.e., $Pr\{g(k', v, t, Z, Z') = f(k, v, t, Z, Z')\} = \frac{1}{2}(1+\varepsilon_a)$, where k' is the s bits subkey of k and the probability holds over all nonces and counters. Assume $Pr\{g(k', v, t, Z, Z') = 1\} = \frac{1}{2}(1+\varepsilon)$, then under some reasonable independency assumptions, we have $\varepsilon = \varepsilon_d \cdot \varepsilon_a$. Hence, given enough output block pairs with the presumed differential, we can verify the correctness of a guessed candidate subkey $\bar{k}'$ for the subkey k' by evaluating the bias of the function g. More precisely, we have $Pr\{g(\bar{k}', v, t, Z, Z') = 1\} = \frac{1}{2}(1+\varepsilon)$ if $\bar{k}' = k'$, whereas for (almost all) $\bar{k}' \neq k'$ we expect g be unbiased, i.e. $Pr\{g(\bar{k}', v, t, Z, Z') = 1\} = \frac{1}{2}$. In this way, the complexity is reduced from 2^{256} to 2^s. Next, we will give a concise description of how the approximation g of f is found.

3.2 Key Recovery Attacks on Salsa20 and ChaCha

In [9], Aumasson et al. gave an efficient way of finding suitable approximations $g(k', v, t, Z, Z')$ of the function $f(k, v, t, Z, Z')$ using the concept of PNB.

Definition 1. *The neutrality measure of the key bit k_i with respect to the function $f(k, v, t, Z, Z')$ is defined as γ_i, where $Pr = \frac{1}{2}(1+\gamma_i)$ is the probability (over all k, v and t) that complementing the key bit k_i does not change the output of $f(k, v, t, Z, Z')$.*

When the threshold γ is set, all the key bits can be divided into two sets: significant key bits with neutrality measure $\gamma_i < \gamma$ (size of s) and non-significant key bits with neutrality measure $\gamma_i \geq \gamma$ (size of ns). Then we can define $g(k', v, t, Z, Z')$ as an approximation of $f(k, v, t, Z, Z')$ just simply making k' be the significant key bits and the non-significant key bits be set to fixed values (e.g. all zero). More details of the construction of the distinguisher g can be found in [9]. Then, the whole attack can be carried out as follows:

Attack Online

```
Parameters: $N$, $s$, $\varepsilon$
1: Collect $N$ pairs of keystream blocks where each pair is produced by states
   with a random nonce and counter (satisfying the relevant input-difference).
2: For each choice of the subkey (i.e. the $s$ significant key bits) do:
   (2.1) Compute the bias of $g$ using the $N$ keystream block pairs.
   (2.2) If the optimal distinguisher legitimates the subkeys candidate as a
         (possibly) correct one, perform an additional exhaustive search over
         the $ns$ non-significant key bits in order to check the correctness of this
         filtered subkey and to find the non-significant key bits.
   (2.3) Stop if the right key is found and output the recovered key.
```

In this attack, there is a set of 2^s sequences of random variables by guessing s significant key bits. Actually, $2^s - 1$ of them should verify the null hypothesis H_0, and a single one verify the alternative hypothesis H_1. For a realization a of the corresponding random variable A, the decision rule $D(a) = i$ to accept H_i can lead to two types of errors:

1. Non-detection: $D(a) = 0$ and $A \in H_1$. The probability of this event is p_{nd}.
2. False alarm: $D(a) = 1$ and $A \in H_0$. The probability of this event is p_{fa}.

The Neyman-Pearson decision theory gives results to estimate the number of samples N required to get some bounds on the probabilities. It can be shown that

$$N \approx \left(\frac{\sqrt{\alpha \log 4} + 3\sqrt{1-\varepsilon^2}}{\varepsilon} \right)^2 \tag{1}$$

samples suffices to achieve $p_{nd} = 1.3 \times 10^{-3}$ and $p_{fa} = 2^{-\alpha}$. Calculus details and the construction of the optimal distinguisher can be found in [10].

The time complexity of this attack can be estimated as follows. Step 2 is repeated 2^s times. For each subkey, step (2.1) is always executed which has complexity of N. The search part of step (2.2) is performed only with probability $p_{fa} = 2^{-\alpha}$ which brings an additional cost of $2^{ns} (= 2^{256-s})$ in case a subkey passes the optimal distinguisher's filter. Therefore, the complexity of step (2.2) is $2^{256-s} \cdot p_{fa}$, showing a total complexity of $2^s(N + 2^{256-s} \cdot p_{fa}) = 2^s \cdot N + 2^{256-\alpha}$.

4 Our Attacks

In the above attack, α (and hence N) is chosen such that it minimizes $2^s \cdot N + 2^{256-\alpha}$ based on the single distinguisher g. In this section, our new attacks are described in details with the introduction of the (column and row) chaining distinguishers, high probability differential trails and the notion of PNV.

4.1 Chaining Distinguishers

For a subkey K', define the set $S(K') = \{k_i | k_i \text{ is involved in the subkey } K'\}$. Note that, all the single distinguishers we mentioned below are constructed with the method in Section 3.

Definition 2. *Column Chaining Distinguishers(CCD for short): For a collection of subkey $\{K'_i\}_{i\in A}$ with $S(K'_i) \subset S(K'_j)$ ($\forall i, j \in A$ and $i < j$), if there exists a collection of distinguishers $\{D_i\}_{i\in A}$, and $\forall i \in A$ the distinguisher D_i effectively depends on the subkey K'_i, we call $\{D_i\}_{i\in A}$ the Column Chaining Distinguishers of $\{K'_i\}_{i\in A}$.*

What's the advantage of CCD? Here is an example: Suppose $A = \{1, 2, 3\}$, and $\{D_i\}_{i\in A}$ are CCD of $\{K'_i\}_{i\in A}$ with $S(K'_1) \subset S(K'_2) \subset S(K'_3)$. For each $i \in A$, there is a relation between the data N_i and $(p_{fa})_i = 2^{-\alpha_i}$ with the $(p_{nd})_i$ fixed (see Eq. 1). Let $s_i = |S(K'_i)|$, so we have $s_1 < s_2 < s_3$. Then the execution of CCD is described as follows:

Execution of CCD

```
Parameters: $\{N_i\}$, $\{s_i\}$
1: For each subkey candidate $\bar{K}'_1$ by guessing the $s_1$ key bits of set $S(K'_1)$,
   verify $\bar{K}'_1$ with the distinguisher $D_1$ of $N_1$ pairs of keystream blocks, if this
   step succeeds, do Step 2;
2: Guess the $s_2 - s_1$ key bits of set $S(K'_2) - S(K'_1)$, then for the subkey candidate
   $\bar{K}'_2$ ($\bar{K}'_1$ plus $s_2 - s_1$ guessed key bits), verify $\bar{K}'_2$ with the distinguisher $D_2$
   of $N_2$ pairs of keystream blocks, if this step succeeds, do Step 3;
3: Guess the $s_3 - s_2$ key bits of set $S(K'_3) - S(K'_2)$, then for the subkey candidate
   $\bar{K}'_3$ ($\bar{K}'_2$ plus $s_3 - s_2$ guessed key bits), verify $\bar{K}'_3$ with the distinguisher $D_3$
   of $N_3$ pairs of keystream blocks, if this step succeeds, do Step 4;
4: Perform an additional exhaustive search over the $256 - s_3$ key bits (i.e. not
   in the set $S(K'_3)$) in order to check the correctness of this filtered subkey
   and to find the remaining key bits.
```

Let us discuss the time complexity of such an attack. Step 1 is repeated for all 2^{s_1} subkey candidates, and each incorrect subkey candidate passes the test of distinguisher D_1 with probability $(p_{fa})_1 = 2^{-\alpha_1}$ (according to N_1 pairs of keystream blocks). Step 2 needs to search over $s_2 - s_1$ key bits of set $S(K'_2) - S(K'_1)$, and each incorrect subkey candidate passes the test of distinguisher D_2 with probability $(p_{fa})_2 = 2^{-\alpha_2}$ (according to N_2 pairs of keystream blocks). Step 3 needs to search over $s_3 - s_2$ key bits of set $S(K'_3) - S(K'_2)$, and each incorrect subkey candidate passes the test of distinguisher D_3 with probability $(p_{fa})_3 = 2^{-\alpha_3}$ (according to N_3 pairs of keystream blocks). And step 4 needs to search over the remaining $256 - s_3$ key bits. So the total complexity is

$$\begin{aligned}
& 2^{s_1} \cdot N_1 + 2^{s_1} \cdot (p_{fa})_1 \cdot 2^{s_2 - s_1} \cdot N_2 + \\
& 2^{s_1} \cdot (p_{fa})_1 \cdot 2^{s_2 - s_1} \cdot (p_{fa})_2 \cdot 2^{s_3 - s_2} \cdot N_3 + \\
& 2^{s_1} \cdot (p_{fa})_1 \cdot 2^{s_2 - s_1} \cdot (p_{fa})_2 \cdot 2^{s_3 - s_2} \cdot (p_{fa})_3 \cdot 2^{256 - s_3} \\
= \ & 2^{s_1} \cdot N_1 + 2^{s_2 - \alpha_1} \cdot N_2 + 2^{s_3 - \alpha_1 - \alpha_2} \cdot N_3 + 2^{256 - \alpha_1 - \alpha_2 - \alpha_3}
\end{aligned}$$

If we use single distinguisher D_1 to recover the key, the time complexity is $2^{s_1} \cdot N_1 + 2^{256 - \alpha_1}$. Furthermore, we can easily get:

$$\min_{N_1}\{2^{l_1} \cdot N_1 + 2^{256-\alpha_1}\} \geq \min_{N_1, N_2, N_3}\{2^{l_1} \cdot N_1 + 2^{l_2 - \alpha_1} \cdot N_2 + 2^{l_3 - \alpha_1 - \alpha_2} \cdot N_3 + 2^{256 - \alpha_1 - \alpha_2 - \alpha_3}\}.$$

Two ordinary methods of constructing CCD are as follows:

First method of constructing CCD

```
1: (a) Find a highly biased differential and set a threshold $\gamma^{(1)}$;
   (b) Estimate the measure $\gamma_i$ of all the key bits and put all those key bits
       with $\gamma_i \geq \gamma^{(1)}$ into the set $S_1$;
   (c) Construct a single distinguisher $D_1$ with the key bits in $S_1$ being set to
       a fixed value, if the bias of $D_1$ is non-negligible, do Step 2;
2: (a) Find another highly biased differential and set a threshold $\gamma^{(2)}$;
   (b) Estimate the measure $\gamma_i$ of all the key bits in set $S_1$ and put all those
       key bits in set $S_1$ with $\gamma_i \geq \gamma^{(2)}$ into the set $S_2$;
   (c) Construct a single distinguisher $D_2$ with the key bits in $S_2$ being set to
       a fixed value(the same as in Step1), if the bias of $D_2$ is non-negligible,
       do Step 3;
3: (a) Find another highly biased differential and set a threshold $\gamma^{(3)}$;
   (b) Estimate the measure $\gamma_i$ of all the key bits in set $S_2$ and put all those
       key bits in set $S_2$ with $\gamma_i \geq \gamma^{(3)}$ into the set $S_3$;
   (c) Construct a single distinguisher $D_3$ with the key bits in $S_3$ being set to
       a fixed value(the same as in Step1), if the bias of $D_3$ is non-negligible,
       do Step 4;
4: Continue the work until only a few key bits are left to be guessed.
```

Second method of constructing CCD

```
1: Find a highly biased differential and set a threshold $\gamma$;
2: Construct a distinguisher $g_\gamma(K'_\gamma, v, t, Z, Z')$ based on the subkey $K'_\gamma =$
   $\{k_i | \gamma_i < \gamma\}$;
3: If the bias of $g_\gamma(K'_\gamma, v, t, Z, Z')$ is non-negligible, set a series of thresholds
   $\gamma^{(1)} < \gamma^{(2)} < ... < \gamma^{(e)}$ with $\gamma^{(i)} \geq \gamma$, and for each $\gamma^{(i)}$, construct the dis-
   tinguisher $g_{\gamma^{(i)}}(K'_{\gamma^{(i)}}, v, t, Z, Z')$ effectively depending on the subkey $K'_{\gamma^{(i)}}$.
```

The first method is feasible for lower rounds of Salsa20 and ChaCha because of sufficient numbers of PNB's (usually more than half of the key bits with high γ_i). And the second method of constructing CCD is based on a single distinguisher, which is more feasible when the numbers of PNB's are insufficient.

Definition 3. *Row Chaining Distinguishers(RCD for short): For a fixed subkey K', if there exists a collection of distinguishers $\{D_i\}_{i \in A}$ which effectively depend on the subkey K', we call the $\{D_i\}_{i \in A}$ the Row Chaining Distinguishers for K'.*

The advantage of RCD is obvious: firstly, some incorrect subkey candidate $\bar{K}'$ may verify the alternative hypothesis of distinguisher D_i, while the probability that it verifies all the alternative hypothesis of distinguishers $\{D_i\}_{i \in A}$ is much lower; secondly, RCD can be used as a CCD[1] . We will show how to construct RCD based on a second-order differential in the next subsection.

4.2 Second-Order Differential Analysis on Salsa20 and ChaCha

First, we recall the second-order differential: let X be the initial matrix, X_1, X_2 and X_3 be associated initial matrices with a single-bit input-difference $[\Delta_i^0]_j = 1$, a single-bit input-difference $[\Delta_m^0]_n = 1$ and the double-bit input-differences $[\Delta_i^0]_j = 1$ and $[\Delta_m^0]_n = 1$ respectively. Note that $(i-m)^2 + (j-n)^2 = 0$ should not hold. We consider a single-bit output-difference $[\Delta_p^r]_q = [X_p^r]_q \oplus [(X_1)_p]_q^r \oplus [(X_2)_p^r]_q \oplus [(X_3)_p^r]_q$ after r rounds. Then the second-order differential for the input X is denoted by $([\Delta_p^r]_q | [\Delta_i^0]_j, [\Delta_m^0]_n)$. The bias ε_d of the output-difference is defined by $Pr\{([\Delta_p^r]_q = 1 | [\Delta_i^0]_j, [\Delta_m^0]_n)\} = \frac{1}{2}(1+\varepsilon_d)$, where the probability holds over all keys, nonces and counters. We found many highly biased differentials for Salsa20 and ChaCha (see Table 2).

For a fixed differential $([\Delta_p^r]_q | [\Delta_i^0]_j, [\Delta_m^0]_n)$ with bias ε_d, let $Z = X + X^R$, $Z_1 = X_1 + (X_1)^R$, $Z_2 = X_2 + (X_2)^R$, and $Z_3 = X_3 + (X_3)^R$, so Z, Z_1, Z_2, and Z_3 can be observed for nonce v, counter t and key k. As mentioned in section 3, the round functions of Salsa20 and ChaCha is invertible, i.e. $X^r = (Z - X)^{r-R} (r < R)$, so $[\Delta_p^r]_q = [((Z-X)^{r-R} \oplus (Z_1 - X_1)^{r-R} \oplus (Z_2 - X_2)^{r-R} \oplus (Z_3 - X_3)^{r-R})_p]_q$. Define $F_{p,q,i,j,m,n}(k, v, t, Z, Z_1, Z_2, Z_3) = [\Delta_p^r]_q$. For short, we define $F_{p,q,i,j,m,n}(k, W) = [\Delta_p^r]_q$ where $W = (v, t, Z, Z_1, Z_2, Z_3)$. The next work is finding suitable approximations $G_{p,q,i,j,m,n}(k', W)$ of the function $F_{p,q,i,j,m,n}(k, W)$. Here, we also use the PNB's mentioned in Section 3.

After all the neutrality measure γ_l's of each key bit k_l be estimated, we set a threshold γ and put all the key bits with $\gamma_l < \gamma$ into a set denoted by $K_{p,q,i,j,m,n}(\gamma) = \{k_l | \gamma_l < \gamma\}$. Having found the set $K_{p,q,i,j,m,n}(\gamma)$, we simply let k' be subkey with the key bits in the set $K_{p,q,i,j,m,n}(\gamma)$ and define $G_{\gamma,p,q,i,j,m,n}(k', W)$ as $F_{p,q,i,j,m,n}(k, W)$ with the remaining key bits (i.e. not in the set $K_{p,q,i,j,m,n}(\gamma)$) with a fixed value. The bias ε_a of the correlation between F and G is defined by $Pr\{G_{\gamma,p,q,i,j,m,n}(k', W) = F_{p,q,i,j,m,n}(k, W)\} = \frac{1}{2}(1+\varepsilon_a)$, where the probability holds over all keys, nonces and counters. Denote the bias

[1] Actually, RCD are special CCD with the subkey unchanged.

Table 2. Some highly biased differentials for Salsa20/4 and ChaCha3

Type	$[\Delta_i^0]_j, [\Delta_m^0]_n$	$[\Delta_p^r]_q$	ε_d
	$[\Delta_7^0]_{24}, [\Delta_8^0]_{17}$	$[\Delta_1^4]_7$	0.67
	$[\Delta_7^0]_{24}, [\Delta_8^0]_{18}$	$[\Delta_1^4]_7$	0.64
Salsa20/4	$[\Delta_7^0]_{24}, [\Delta_8^0]_{19}$	$[\Delta_1^4]_7$	0.62
	$[\Delta_7^0]_{24}, [\Delta_8^0]_{20}$	$[\Delta_1^4]_7$	0.58
	$[\Delta_7^0]_{25}, [\Delta_8^0]_{17}$	$[\Delta_1^4]_7$	0.59
	$[\Delta_{12}^0]_{15}, [\Delta_{13}^0]_{20}$	$[\Delta_3^3]_0$	0.43
	$[\Delta_{12}^0]_{20}, [\Delta_{15}^0]_{15}$	$[\Delta_2^3]_0$	0.43
ChaCha3	$[\Delta_{13}^0]_{15}, [\Delta_{14}^0]_{20}$	$[\Delta_0^3]_0$	0.43
	$[\Delta_{14}^0]_{15}, [\Delta_{15}^0]_{20}$	$[\Delta_1^3]_0$	0.43
	$[\Delta_{13}^0]_{16}, [\Delta_{14}^0]_{20}$	$[\Delta_0^3]_0$	0.41

of G by ε, i.e. $Pr\{G_{\gamma,p,q,i,j,m,n}(k', W) = 1\} = \frac{1}{2}(1+\varepsilon)$. Under some reasonable independency assumptions, the equality $\varepsilon = \varepsilon_d \cdot \varepsilon_a$ holds. Hence, given enough output block pairs with the presumed differential, we can verify the correctness of a guessed candidate subkey $\bar{k}'$ for the subkey k' by evaluating the bias of the function G. More precisely, we have $Pr\{G_{\gamma,p,q,i,j,m,n}(k', W) = 1\} = \frac{1}{2}(1+\varepsilon)$ conditioned on $\bar{k}' = k'$, whereas for (almost all) $\bar{k}' \neq k'$ we expect G be unbiased, i.e. $Pr\{G_{\gamma,p,q,i,j,m,n}(k', W) = 1\} = \frac{1}{2}$. The way for searching such a distinguisher is similar to that of the first-order differentials.

Now, we show how to use the second-order differentials to construct RCD. For a second-order differential $([\Delta_p^r]_q | [\Delta_i^0]_j, [\Delta_m^0]_n)$, we choose a threshold γ empirically and construct a single distinguisher $G_{\gamma,p,q,i,j,m,n}(k', W)$ using the method above, where k' is the subkey of all key bits in the set $K_{p,q,i,j,m,n}(\gamma)$. In order to construct RCD, the subkey should stay the same, i.e. the set $K_{p,q,i,j,m,n}(\gamma)$ should stay the same. Now, we consider the factors of the set $K_{p,q,i,j,m,n}(\gamma)$: p, q, i, j, m, n, γ. By tests, we find: if the value of p changes, the set $K_{p,q,i,j,m,n}(\gamma)$ will change with a high probability, so do the factors q, i, m, γ; while, if only the factor j changes, $K_{p,q,i,j,m,n}(\gamma)$ will stay the same with a high probability, so does the factor n. Hence, for the distinguisher $G_{\gamma,p,q,i,j,m,n}(k', W)$, we search over all j's only or all m's only to construct a new distinguisher with subkey unchanged. Here we give an example of the RCD on 256-bit ChaCha7. We construct 4-Step RCD $\{G_{\gamma,9,0,14,j,15,12}(k, W)\}_{j\in\{0,1,2,28\}}$. Let $\gamma = 0.3$, and we get $K_{9,0,14,j,15,12}(\gamma) =\{$ 3, 7, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 39, 40, 63, 67, 79, 80, 95, 99, 159, 160, 184, 185, 186, 187, 188, 189, 190, 191, 200, 255$\}$ for any $j \in \{0, 1, 2, 28\}$.

Actually, we can easily find RCD for Salsa20 and ChaCha using the method above. However, we did not find enough PNB's to improve our attacks[2] on Salsa20 and ChaCha. And such a reality limits us to display the use of RCD. We believe the concept of RCD can be used in other ciphers.

4.3 Probabilistic Neutral Vectors

Note that, contrary to the mutual interaction between neutral bits, we have directly combined several PNB's without altering their probabilistic quality, so do J. Aumasson et al. In order to justify the reasonableness, we introduce a generalized concept of PNB's called *probabilistic neutral vectors*(PNV's).

Definition 4. *The neutrality measure of the two-dimension key bit vector (k_i, k_j) with respect to the function $f(k, v, t, Z, Z')$ is defined as $\gamma_{(i,j)}$, where $Pr = \frac{1}{2}(1 + \gamma_{(i,j)})$ is the probability (over all k, v and t) that complementing the key bit k_i and k_j does not change the output of $f(k, v, t, Z, Z')$.*

Simulations shows that: for key bit vector (k_i, k_j), we have $\gamma_i \cdot \gamma_j \leq \gamma_{(i,j)} \leq max\{\gamma_i, \gamma_j\}$. Furthermore, for a fixed differential, denote the set $H_1(\gamma) = \{k_i | \gamma_i \geq \gamma\}$ and $H_2(\gamma) = \{k_i | \gamma_{(i,j)} \geq \gamma$ for at least one $k_j\}$. Then we have the following lemma:

Lemma 1. *For a fixed differential of Salsa20 or ChaCha, $H_1(\gamma) \subseteq H_2(\gamma)$, and hence $| H_1(\gamma) | \leq | H_2(\gamma) |$.*

For Salsa 20/7 with the differential $([\Delta_1^4]_{14} | [\Delta_7^0]_{31})$, we have $H_2(0.4) - H_1(0.4) = \{k_1, k_{78}\}$. So it's reasonable to combine several PNB's directly in our attacks. Actually, if we want to construct a distinguisher with s key bits fixed, we should use the concept of s-dimension PNV's. However, when s is too big, the cost of finding the most significant PNV's is too high to search over all C_{256}^s cases.

4.4 Experimental Results with CCD

We present attacks on 256-bit Salsa20/5 and Salsa20/6 with the CCD constructed by the first method. And the rest improved attacks are based on the CCD constructed by the second method. In order to compare our method with that in [9], we use the same differentials and the same threshold γ as used in [9]. And we believe there exists other choices that lead better results.

Attack on 256-bit Salsa20/5. The output differential is observed after working two rounds backward from a 5-rounds keystream block. We use five differentials: $([\Delta_3^3]_{15} | [\Delta_6^0]_0)$, $([\Delta_3^3]_9 | [\Delta_6^0]_0)$, $([\Delta_8^3]_{11} | [\Delta_7^0]_2)$, $([\Delta_8^3]_{20} | [\Delta_7^0]_0)$ and $([\Delta_{12}^3]_{23} | [\Delta_7^0]_1)$. We set the threshold $\gamma = 0.9$ and the subkeys for each distinguisher are listed in the Appendix A. The parameters of our attacks are listed in Table 3 (see

[2] We only test the second-order differential with single bit input and single bit(and double bits) output, and for other second-order differential, there maybe exist enough PNB's to improve the attacks.

Appendix B). And the total attack runs in time 2^{55} and data 2^{10}.

Attack on 256-bit Salsa20/6. The output differential is observed after working two rounds backward from a 6-rounds keystream block. We construct a CCD using four differentials: $([\Delta_6^4]_{26}|[\Delta_7^0]_{31})$, $([\Delta_1^4]_3|[\Delta_7^0]_{29})$, $([\Delta_1^4]_{26}|[\Delta_7^0]_{13})$ and $([\Delta_1^4]_{12}|[\Delta_7^0]_{13})$. For each difference, we use the same threshold $\gamma = 0.9$ and the subkeys for each distinguisher are listed in the Appendix A. The parameters of our attacks are listed in Table 4 (see Appendix B). And the total attack runs in time 2^{73} and data 2^{16}.

Attack on 256-bit Salsa20/7. We use the differential $([\Delta_1^4]_{14}|[\Delta_7^0]_{31})$ with $|\varepsilon_d| = 0.131$. The parameters and results of Aumasson's attacks are listed in Table 5 (see Appendix B). We construct 2-step CCD using $\gamma^{(1)} = 0.5$ and $\gamma^{(2)} = 0.6$ with $\varepsilon^{(1)} = 0.0022$ and $\varepsilon^{(2)} = 0.0050$ respectively. Note that, $\varepsilon^{(2)} = 0.0050 < 0.0064$. That because we test and find that such a value leads a result: if the correct key passes the distinguisher of $\gamma^{(1)} = 0.5$ (with success probability 50%[3]), then it can pass the distinguisher of $\gamma^{(2)} = 0.6$ with success probability more than 90% (we define this probability by step success probability). The time complexity is $2^{125} \cdot N_1 + 2^{132-\alpha_1} \cdot N_2 + 2^{256-\alpha_1-\alpha_2}$. We choose $\alpha_1 = 10$ and $\alpha_2 = 104$, then get $N_1 = 2^{23}$ and $N_2 = 2^{23}$ respectively by Eq.1. So the time complexity is $2^{125} \cdot N_1 + 2^{132-\alpha_1} \cdot N_2 + 2^{256-\alpha_1-\alpha_2} \approx 2^{148}$, the data complexity is $2^{23} + 2^{23} = 2^{24}$, and the success probability is $50\% \times 90\% = 45\%$.

Attack on 256-bit Salsa20/8. For the differential $([\Delta_1^4]_{14}|[\Delta_7^0]_{31})$ with $|\varepsilon_d| = 0.131$, we construct 2-step CCD. using $\gamma^{(1)} = 0.15$ [4] and $\gamma^{(2)} = 0.20$ with $\varepsilon^{(1)} = 0.00047$ and $\varepsilon^{(2)} = 0.00102$ respectively. For the threshold $\gamma^{(1)} = 0.15$, we find $ns_1 = 33$ non-significant key bits, and for the threshold $\gamma^{(2)} = 0.20$, we find $ns_2 = 30$ non-significant key bits. Note that, the value $\varepsilon^{(2)} = 0.00102$ is chosen with the step success probability 90%. The time complexity is $2^{223} \cdot N_1 + 2^{226-\alpha_1} \cdot N_2 + 2^{256-\alpha_1-\alpha_2}$. We choose $\alpha_1 = 2$ and $\alpha_2 = 7$, then get $N_1 = 2^{26.5}$ and $N_2 = 2^{25}$ respectively by Eq.1. So the time complexity is 2^{250}, the data complexity is $2^{26.5} + 2^{25} = 2^{27}$, and the success probability is $50\% \times 90\% = 45\%$.

Attack on 128-bit Salsa20/7. For the differential $([\Delta_1^4]_{14}|[\Delta_7^0]_{31})$, we construct 4-step CCD. The parameters of our attacks are listed in Table 6 (see Appendix B). Note that, the value $\varepsilon^{(i)}(i = 2, 3, 4)$ is chosen with the step success probability 95%. The time complexity is $2^{90} \cdot N_1 + 2^{92-\alpha_1} \cdot N_2 + 2^{94-\alpha_1-\alpha_2} \cdot N_3 + 2^{98-\alpha_1-\alpha_2-\alpha_3} \cdot N_4 + 2^{128-\alpha_1-\alpha_2-\alpha_3-\alpha_4} \approx 2^{109}$, the data complexity is $2^{19} + 2^{17.5} + 2^{16.5} + 2^{15.5} \approx 2^{19}$, and the success probability is $50\% \times (95\%)^3 \approx 43\%$.

[3] In [9], they use the median bias in their attack, which leads in a success probability of at least $\frac{1}{2}(1 - p_{nd}) \approx 50\%$.

[4] In [9], the threshold γ is set to 0.12, and they get $\varepsilon_a = 0.0011$. However, $\varepsilon_a = 0.0011$ is not reasonable as they say: we can only measure a bias of about $|\varepsilon_a| > c \cdot 2^{-12}$(where $c \approx 10$ for a reasonable estimation error).

Attack on 256-bit Chacha6. For the differential $([\Delta^3_{11}]_0|[\Delta^0_{13}]_{13})$ with $|\varepsilon_d| = 0.026$, we construct 3-step CCD. The parameters of our attacks are listed in Table 7 (see Appendix B). Note that, the value $\varepsilon^{(i)}(i = 2, 3)$ is chosen with the step success probability 95%. The time complexity is $2^{209} \cdot N_1 + 2^{213-\alpha_1} \cdot N_2 + 2^{256-\alpha_1-\alpha_2} \cdot N_3 + 2^{214-\alpha_1-\alpha_2-\alpha_3} \approx 2^{136}$, the data complexity is $2^{27} + 2^{25.5} + 2^{26.3} \approx 2^{28}$, and the success probability is $50\% \times (95\%)^2 \approx 45\%$.

Attack on 256-bit Chacha7. For the differential $([\Delta^3_{11}]_0|[\Delta^0_{13}]_{13})$, we construct 4-step CCD. The parameters of our attacks are listed in Table 8 (see Appendix B). Note that, $\varepsilon^{(i)}(i = 2, 3, 4)$ is chosen with the step success probability 95%. The time complexity is $2^{221} \cdot N_1 + 2^{222-\alpha_1} \cdot N_2 + 2^{224-\alpha_1-\alpha_2} \cdot N_3 + 2^{228-\alpha_1-\alpha_2-\alpha_3} \cdot N_4 + 2^{256-\alpha_1-\alpha_2-\alpha_3-\alpha_4} \approx 2^{246.5}$, the data complexity is $2^{26.3} + 2^{25.3} + 2^{24.2} + 2^{22.4} \approx 2^{27}$, and the success probability is $50\% \times (95\%)^3 \approx 43\%$.

Attack on 128-bit Chacha6. For the differential $([\Delta^3_{11}]_0|[\Delta^0_{13}]_{13})$ with $|\varepsilon_d| = 0.026$, we construct 3-step CCD. The parameters of our attacks are listed in Table 9 (see Appendix B). Note that, the value $\varepsilon^{(i)}(i = 2, 3)$ is chosen with the step success probability 95%. The time complexity is $2^{77} \cdot N_1 + 2^{81-\alpha_1} \cdot N_2 + 2^{85-\alpha_1-\alpha_2} \cdot N_3 + 2^{128-\alpha_1-\alpha_2-\alpha_3} \approx 2^{105}$, the data complexity is $2^{27.9} + 2^{24.6} + 2^{23.3} \approx 2^{28}$, and the success probability is $50\% \times (95\%)^2 \approx 45\%$.

5 Conclusions

In this paper, we extend the approach of Aumasson et al. by considering a new type of distinguishers, named (column and row) chaining distinguishers, which can efficiently make use of the biases of multiple differential trails and the matrix structure of the cipher. Besides, we find new high probability second-order differential trails that are not covered by the previous results, some of which are employed in our attack. The notion of PNB is generalized to that of probabilistic neutral vectors (PNV), which investigate the properties of the underlying function when more than one input bit are flipped simultaneously and include the PNB as a special case. It is shown that the set of PNV is no smaller than that of PNB. Based on these findings, we construct improved key recovery attacks on reduced-round Salsa20 and ChaCha, repectively. Both time and data complexities of our new attacks are smaller than those of the best former results.

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

References

1. Bernstein, D.J.: Salsa20. Technical Report 2005/025, eSTREAM, ECRYPT Stream Cipher Project, `http://cr.yp.to/snuffle.html`

2. The eSTREAM project, `http://www.ecrypt.eu.org/stream/`
3. Bernstein, D.J.: Salsa20/8 and Salsa20/12. Technical Report 2006/007, eSTREAM, ECRYPT Stream Cipher Project, `http://cr.yp.to/snuffle/812.pdf`
4. Bernstein, D.J.: ChaCha, a variant of Salsa20, `http://cr.yp.to/chacha.html`
5. Crowley, P.: Truncated differential cryptanalysis of five rounds of Salsa20. In: Stream Ciphers Revisited - SASC 2006 (2006)
6. Velichkov, V., Mouha, N., De Cannière, C., Preneel, B.: UNAF: A Special Set of Additive Differences with Application to the Differential Analysis of ARX. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 287–305. Springer, Heidelberg (2012)
7. Fischer, S., Meier, W., Berbain, C., Biasse, J.-F., Robshaw, M.J.B.: Non-randomness in eSTREAM Candidates Salsa20 and TSC-4. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 2–16. Springer, Heidelberg (2006)
8. Tsunoo, Y., Saito, T., Kubo, H., Suzaki, T., Nakashima, H.: Differential cryptanalysis of Salsa20/8. In: The State of the Art of Stream Ciphers - SASC 2007 (2007)
9. Aumasson, J.-P., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New features of Latin dances: analysis of Salsa, ChaCha, and Rumba. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 470–488. Springer, Heidelberg (2008)
10. Siegenthaler, T.: Decrypting a class of stream ciphers using ciphertext only. IEEE Transactions on Computers 34(1), 81–85 (1985)

Appendix A: CCD of Salsa20/5 and Salsa20/6

For Salsa20/5, we get the significant key bits sets:
$A_1 = \{0, 1, 32, 33, 34, 35, 36, 37, 38, 74, 75, 76, 77, 78, 84, 85, 86, 87, 88, 89, 90, 129, 130 131, 132, 133, 134, 135, 143, 144, 145, 146, 147, 148, 149, 208, 209, 210, 211, 212 244, 245, 246, 247, 248 \}$,
$A_2 = \{21, 22, 23, 24, 25, 26, 27, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 68, 69, 70, 71, 72, 80, 81, 82, 83, 128, 139, 140, 141, 142, 200, 201, 202, 203, 204, 205, 206, 236, 237, 238, 239, 240, 241, 242 \}$,
$A_3 = \{ 2, 3, 4, 5, 6, 7, 8, 14, 15, 16, 17, 18, 19, 20, 43, 44, 45, 46, 47, 96, 97, 98, 99, 100, 101, 102, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 207, 213, 214, 215, 216, 217, 218, 219\}$,
$A_4 = \{ 9, 10, 11, 12, 13, 28, 29, 30, 31, 50, 51, 52, 91, 92, 93, 103, 104, 105, 106, 107, 108, 109, 110, 111, 136, 137, 138, 181, 182, 183, 184, 185, 186, 187, 188, 189, 192, 193, 194, 195, 196, 220, 221, 222, 223 \}$,
$A_5 = \{ 39, 40, 41, 48, 49, 112, 113, 114, 115, 116, 117, 151, 152, 153, 154, 155, 156, 157, 160, 161, 162, 190, 191, 197, 198, 199, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 243, 249, 250, 251, 252, 253 \}$.
For each differential $([\Delta_3^3]_{15}|[\Delta_6^0]_0)$, $([\Delta_3^3]_9|[\Delta_6^0]_0)$, $([\Delta_8^3]_{11}|[\Delta_7^0]_2)$, $([\Delta_8^3]_{20}|[\Delta_7^0]_0)$ and $([\Delta_{12}^3]_{23}|[\Delta_7^0]_1)$, construct the single distinguisher $D_j(j = 1, 2, ..., 5)$ with the non-significant key bits being set to a fixed value. And the D_j effectively depends on subkey $K'_j = \{k_i | i \in \cup_{l=1}^{j} A_l\}$.

For Salsa20/6, we get the significant key bits sets:

A_1 = {0, 1, 2, 3, 31, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 74, 75, 76, 77, 78, 79, 80, 81, 96 108, 109, 110, 111, 112, 113, 114, 122, 123, 124, 125, 126, 127, 185, 186, 187, 188 189, 190, 191, 217, 218, 219, 220, 221, 222, 223, 230, 231, 232, 233, 234, 235, 236 },
A_2 = { 8, 9, 10, 11, 12, 13, 14, 35, 36, 37, 67, 68, 69, 70, 71, 72, 73, 85, 86, 87, 88, 89, 90, 91, 130, 131, 132, 133, 134, 135, 136, 166, 167, 168, 169, 170, 171, 172, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 243, 244, 245, 246, 247, 248, 249, 250},
A_3 = { 4, 5, 21, 22, 23, 24, 25, 32, 33, 34, 58, 59, 60, 61, 62, 63, 64, 82, 92, 93, 94, 95, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 198, 199, 200, 201, 202, 203, 204, 205, 206, 237, 238, 239, 240, 241},
A_4 = { 7, 15, 18, 19, 20, 50, 51, 52, 53, 54, 55, 65, 66, 139, 140, 141, 142, 143, 144, 145, 175, 176, 177, 178, 179, 180, 181, 192, 193, 194, 195, 224, 225, 226, 227, 252, 253, 254, 255}.

For each differential $([\Delta_6^4]_{26}|[\Delta_7^0]_{31})$, $([\Delta_1^4]_3|[\Delta_7^0]_{29})$, $([\Delta_1^4]_{26}|[\Delta_7^0]_{13})$, $([\Delta_1^4]_{12}|[\Delta_7^0]_{13})$, construct the single distinguisher $D_j(j = 1, 2, 3, 4)$ with the non-significant key bits being set to a fixed value. And the D_j effectively depends on subkey $K'_j = \{k_i | i \in \cup_{l=1}^{j} A_l\}$.

Appendix B: Parameters for Our Attacks

Table 3. Different parameters for our attack on 256-bit Salsa 20/5

i	Differential	ns	ε_d	ε_a	ε	α	Data
1	$([\Delta_3^3]_{15}\|[\Delta_6^0]_0)$	211	0.995	0.677	0.674	45	2^8
2	$([\Delta_3^3]_9\|[\Delta_6^0]_0)$	165	0.929	0.670	0.622	46	2^8
3	$([\Delta_8^3]_{11}\|[\Delta_7^0]_2)$	121	0.999	0.737	0.736	44	2^8
4	$([\Delta_8^3]_{20}\|[\Delta_7^0]_0)$	76	0.971	0.947	0.921	45	2^7
5	$([\Delta_{12}^3]_{23}\|[\Delta_7^0]_1)$	32	0.918	0.943	0.866	44	2^7

Table 4. Different parameters for our attack on 256-bit Salsa 20/6

i	Differential	ns	ε_d	ε_a	ε	α	Data
1	$([\Delta_6^4]_{26}\|[\Delta_7^0]_{31})$	196	0.201	0.680	0.137	60	2^{13}
2	$([\Delta_1^4]_3\|[\Delta_7^0]_{29})$	140	0.1113	0.664	0.075	56	2^{15}
3	$([\Delta_1^4]_{26}\|[\Delta_7^0]_{13})$	93	0.110	0.771	0.085	47	2^{14}
4	$([\Delta_1^4]_{12}\|[\Delta_7^0]_{13})$	54	0.183	0.801	0.147	39	2^{13}

Table 5. Different parameters for Aumasson's attack on 256-bit Salsa 20/7

γ	ns	ε_a	ε	α	Time	Data
1.0	39	1.000	0.1310	31	2^{230}	2^{13}
0.9	97	0.655	0.0860	88	2^{174}	2^{15}
0.8	103	0.482	0.0634	93	2^{169}	2^{16}
0.7	113	0.202	0.0265	101	2^{162}	2^{19}
0.6	124	0.049	0.0064	108	2^{155}	2^{23}
0.5	131	0.017	0.0022	112	2^{151}	2^{26}

Table 6. Parameters for our attack on 128-bit Salsa 20/7

i	$\gamma^{(i)}$	ns_i	$\varepsilon^{(i)}$	α_i	N_i
1	0.40	38	0.0059	2	2^{19}
2	0.42	36	0.0105	4	$2^{17.5}$
3	0.45	34	0.0165	6	$2^{16.5}$
4	0.60	30	0.0359	18	$2^{15.5}$

Table 7. Parameters for our attack on 256-bit Chacha6

i	$\gamma^{(i)}$	ns_i	$\varepsilon^{(i)}$	α_i	N_i
1	0.60	147	0.00048	4	2^{27}
2	0.66	143	0.00091	8	$2^{25.5}$
3	0.75	139	0.00171	120	$2^{26.3}$

Table 8. Parameters for our attack on 256-bit Chacha7

i	$\gamma^{(i)}$	ns_i	$\varepsilon^{(i)}$	α_i	N_i
1	0.50	35	0.00059	3.8	$2^{26.3}$
2	0.53	34	0.00080	3.5	$2^{25.3}$
3	0.55	32	0.00127	5	$2^{24.2}$
4	0.58	28	0.00280	9	$2^{22.4}$

Table 9. Parameters for our attack on 128-bit Chacha6

i	$\gamma^{(i)}$	ns_i	$\varepsilon^{(i)}$	α_i	N_i
1	0.50	51	0.00034	4	$2^{27.9}$
2	0.56	47	0.00114	5.5	$2^{24.6}$
3	0.65	43	0.00281	25	$2^{23.3}$

Multi-differential Cryptanalysis on Reduced DM-PRESENT-80: Collisions and Other Differential Properties

Takuma Koyama[1], Yu Sasaki[2], and Noboru Kunihiro[1]

[1] The University of Tokyo
5-1-5 Kashiwanoha, Kashiwa-shi, Chiba 277-8561 Japan
{t-koyama@it.,kunihiro@}k.u-tokyo.ac.jp
[2] NTT Secure Platform Laboratories, NTT Corporation
3-9-11 Midori-cho, Musashino-shi, Tokyo 180-8585 Japan
sasaki.yu@lab.ntt.co.jp

Abstract. The current paper studies differential properties of the compression function of reduced-round DM-PRESENT-80, which was proposed at CHES 2008 as a lightweight hash function with 64-bit digests. Our main result is a collision attack on 12 rounds with a complexity of $2^{29.18}$ 12-round DM-PRESENT computations. Then, the attack is extended to an 18-round distinguisher and an 12-round second preimage attack. In our analysis, the differential characteristic is satisfied by the start-from-the-middle approach. Our success lies in the detailed analysis of the data transition, where the internal state and message values are carefully chosen so that a differential characteristic for 5 rounds can be satisfied with complexity 1 on average. In order to reduce the attack complexity, we consider as many techniques as possible; multi-inbound technique, early aborting technique, precomputation of look-up tables, multi-differential characteristics.

Keywords: DM-PRESENT-80, Collision, Second preimage, Multi-differential cryptanalysis, Rebound attack.

1 Introduction

Recently, demand on the secure communication in a resource constraint environment has been increased, e.g., sensor network with RFID tags. From this background, block-ciphers and hash functions suitable for a resource constraint environment are actively discussed. They are called lightweight block-ciphers and hash functions. One of the remarkable designs for lightweight block-ciphers is PRESENT, which was proposed by Bogdanov *et al.* at CHES 2007 [4]. The block size of PRESENT is 64 bits, and it supports 80- and 128-bit keys. It adopts an SPN structure and consists of 31 rounds for both key sizes. Recently, PRESENT has been adopted by ISO as one of the international standards in lightweight cryptography [12]. Several cryptanalytic results were published against reduced-round PRESENT [2, 7, 8, 11, 14, 21–23, 25]. The current best key recovery attack is up to 26 rounds with the assumption that the full codebook is available to the attacker. Without the full codebook, the best attack is up to 25 rounds.

Hash functions are usually constructed by using a block-cipher or permutation as a building block. Hence, it is natural to design lightweight hash functions based on lightweight block-ciphers or a permutation inside lightweight block-ciphers. In fact, there are several lightweight hash functions based on PRESENT or the permutation inside PRESENT. DM-PRESENT was proposed by Bogdanov *et al.* at CHES 2008 [5], where the compression function is simply constructed by using the PRESENT block-cipher in the Davies-Meyer mode [20, Algorithm 9.42]. Another compression function called H-PRESENT was proposed in [5], which consists of a double-block-length mode-of-operation instantiating the PRESENT block-cipher. SPONGENT was proposed by Bogdanov *et al.* at CHES 2011 [3], which adopts the sponge construction [1], and its internal permutation is based on PRESENT.

Attack scenarios for block-ciphers and hash functions are very different. Intuitively, the complexity for the attack on block-ciphers is bounded by the key size, while, for collision attacks agains hash function, the attack complexity is bounded by only a half of the digest size. Thus, attacks on block-ciphers cannot be converted to the attack on hash functions directly. Hence the security of PRESENT-based hash functions must be evaluated independently of the attacks on the PRESENT block-cipher.

In this paper, we study the security of DM-PRESENT. As far as we know, no result is published about it. Regarding H-PRESENT, two results have been announced. One is by Ferguson at the rump session of CRYPTO 2011 [9]. It reported a weakness of the mode-of-operation of H-PRESENT-128 leading to the pseudo-preimage attack whose complexity is a half of the brute force attack. The other is by Kobayashi and Hirose at SCIS 2012 [13], which reports differential attacks for H-PRESENT-80 reduced to 10 rounds.

Our Contributions. This paper studies differential properties of the compression function of reduced-round DM-PRESENT-80. Our main result is a collision attack on 12 rounds with a complexity of $2^{29.18}$ 12-round DM-PRESENT computations. Attacks on block-ciphers and hash functions are different. Therefore, we need to construct a differential characteristic from scratch with considering the following properties; (1) for attacks on hash functions, the attacker can choose internal state and message values so that differential characteristics for several rounds can be satisfied with low complexity. Hence, characteristics must be chosen to take into account such impact. (2) To generate collisions, the differential form of the plaintext and ciphertext must be identical so that they can be canceled each other with the feed-forward operation.

As a result, we construct a 12-round differential characteristic that produces a collision of the compression function with probability of 2^{-70} for a randomly chosen message and chaining variable. We then search for paired values satisfying the characteristic much more efficiently with the rebound attack [19]. The characteristic is divided into inbound part (Round 3 to 7) and outbound part

(Round 0 to 2 and Round 8 to 11). With several techniques such as multi-inbound technique [15, 16] and precomputation of a look-up table, the inbound part is satisfied with a very low complexity, which reduces the attack complexity from 2^{70} to 2^{37}. Moreover, with several techniques such as early aborting technique [6, 26] and multi-differential characteristics [18], the outbound part is satisfied with the complexity of $2^{29.18}$ 12-round DM-PRESENT computations. Finally, the attack becomes faster than the birthday attack.

The 12-round differential characteristic can be extended for other attack scenarios. With respect to a distinguisher, the differential form of the plaintext and ciphertext can be different and the attacker may be allowed to spend more than 2^{32} computations. By extending the 12-round characteristic in forward and backward, we can construct an 18-round distinguisher. Furthermore, with the approach by Yu *et al.* [27], this can be used to mount a second preimage attack for 12 rounds of the compression function. The attack results are summarized in Table 1.

Table 1. Summary of our attacks

Attacks	♯Rounds	Time	Memory
Collision	12	$2^{29.18}$	2^{12}
2nd Preimage	12	$2^{61.91}$	Negl.
Distinguisher	18	$2^{57.18}$	2^{12}

Paper Outline. The organization of this paper is as follows. Sect. 2 summarizes related work. Sect. 3 describes a new collision attack against 12 rounds. In Sect. 4, we extend the attack to several different scenarios. Finally, we conclude the paper in Sect. 5. We postpone the specification of DM-PRESENT-80 in App. A.

2 Previous Work

2.1 Iterative Linear Characteristic of Key Recovery Attack

Linear and multi-linear analyses are the best approach for the key recovery attack on PRESENT. They use linear characteristics of an iterative form. The base of the iterative characteristic is as follows. The linear form of `0x02` can be transformed into `0x06` during the S-box transformation, and the opposite also can be transformed. The idea is also useful to construct an iterative differential characteristic in our attack.

2.2 Rebound Attack

Rebound attack, which was proposed by Mendel *et al.* at FSE 2009, is an approach to satisfy a truncated differential characteristic when the key value is known to the attacker [19]. The technique is useful to analyze hash functions.

Suppose that the round function adopts the SPN structure, where S-layer adopts the S-box transformations, and P-layer introduces a diffusion. In truncated differential cryptanalysis, the only probabilistic part is the transformation in the P-layer. The basic rebound attack can satisfy the differential characteristic with two P-layers $\sharp$IN $\rightarrow$ P $\rightarrow$ S $\rightarrow$ P $\rightarrow$ $\sharp$OUT. The attacker chooses the input difference Δ^{IN} and compute $\mathrm{P}(\Delta^{\mathrm{IN}})$. This can be computed without determining actual values. Similarly, the attacker chooses the output difference Δ^{OUT} and compute $\mathrm{P}^{-1}(\Delta^{\mathrm{OUT}})$. Finally, paired values are determined so that the differential transformation through the middle S-layer is satisfied. Several improved techniques have been proposed after the publication of [19]. In this paper, we particularly use the start-from-the-middle technique [17] and the multi-inbound technique [15, 16]. We stress that our attack is the differential attack, not the truncated one. Thus, these techniques cannot be applied straight-forward, but the ideas of determining internal state values and bypassing several rounds are also useful for our attack.

2.3 Second Preimage Attack on MD4

In 2005, Yu *et al.* proposed a second preimage attack on MD4 [27]. In the second preimage attack, the attacker is given a message M and its digest $H(M)$. For a random oracle, the probability that $H(M) = H(M')$ is satisfied for $M \neq M'$ is 2^{-n}, where n is the size of the hash value. Therefore, finding a way to choose M' satisfying the above equation with a higher probability than 2^{-n} can be regarded as the second preimage attack. MD4 generates 128-bit hash values. Yu *et al.*, against full MD4, found the message difference ΔM that would result in $H(M) = H(M \oplus \Delta M)$ with a probability of 2^{-61}. In a later section, we propose the second preimage attack on the reduced-round compression function of DM-PRESENT-80 with a similar idea.

3 12-Round Collision Attack on Compression Function

This section shows a collision attack against the 12-round DM-PRESENT-80 compression function with a complexity of $2^{29.18}$. We choose the differential approach to find a collision. At first, we give the detailed analysis of the differential propagation within 1 round. Then, the differential characteristic for 12 rounds is introduced.

3.1 Analysis of Differential Properties of S-box

The S-box used in PRESENT is a 4-bit to 4-bit S-box $S(\cdot)$. The following table gives the detailed specification of the S-box in hexadecimal notation.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

We search for pairs of input/output differences of the S-box $(\varDelta x, \varDelta y)$, where $\varDelta x, \varDelta y \in \mathbb{F}_2^4$, satisfying the following two conditions.

$$\mathcal{HW}(\varDelta x) + \mathcal{HW}(\varDelta y) = 3, \tag{1}$$

$$\exists x, y \in \mathbb{F}_2^4 : \Big(S(x) \oplus S(x \oplus \varDelta x) = \varDelta y\Big) \wedge \Big(S(y) \oplus S(y \oplus \varDelta y) = \varDelta x\Big). \tag{2}$$

Note that $\mathcal{HW}(x)$ indicates a Hamming weight of x. Let $\Pr(\varDelta a, \varDelta b)$ be the probability that the input difference $\varDelta a$ is transformed into $\varDelta b$ with an S-box transformation. More strictly, $\Pr(\varDelta a, \varDelta b)$ is defined as $\sharp\{a | S(a) \oplus S(a \oplus \varDelta a) = \varDelta b\}/2^4$. Then, we identify $(\varDelta x, \varDelta y)$ which achieves the maximum value for the following probability;

$$\Pr(\varDelta x, \varDelta y) \times \Pr(\varDelta y, \varDelta x). \tag{3}$$

- (1): Slower differential propagations lead to longer differential characteristics. Thus, we need to minimize the number of bits with differences. In the S-box of PRESENT, any input difference with a single bit always produces output differences with at least two bits. Thus, the minimum number of (1) is 3.
- (3): For a fixed $(\varDelta x, \varDelta y)$, we get an input/output pair with the probability $\Pr(\varDelta x, \varDelta y)$. The $\Pr(\varDelta\cdot, \varDelta\cdot)$ is either 2^{-2}, 2^{-3}, or zero.

These can be verified by enumerating through all $2^4 \times 2^4$ input/output pairs. It is remarkable that there is no differential pairs satisfied with the total probability of $2^{-2-2} = 2^{-4}$ in the condition (3). In other words, the maximum value is 2^{-2-3} or 2^{-3-2}, which is 2^{-5}. As a result, we found only three pairs $(\varDelta\texttt{0x4}, \varDelta\texttt{0x9})$, $(\varDelta\texttt{0x4}, \varDelta\texttt{0x5})$, and $(\varDelta\texttt{0x1}, \varDelta\texttt{0x3})$ that satisfy the all conditions. A detailed description of the pairs is given in App. B.

3.2 Entire Differential Characteristic

We construct a 12-round differential characteristic by using the good 1-round characteristics observed in Sect. 3.1. Each three pair can be used to construct a 6-round iterative characteristic with the same probability. However, using $(\varDelta\texttt{0x4}, \varDelta\texttt{0x9})$ and $(\varDelta\texttt{0x4}, \varDelta\texttt{0x5})$ is more advantageous than using $(\varDelta\texttt{0x1}, \varDelta\texttt{0x3})$ because multi-differential characteristics can be constructed. Hereafter, we mainly use the pair $(\varDelta\texttt{0x4}, \varDelta\texttt{0x9})$ to construct 12-round characteristic, and use $(\varDelta\texttt{0x4}, \varDelta\texttt{0x5})$ for the multi-differential characteristics.

Fig. 1 shows our 12-round differential characteristic. Hereafter, we call the bits with differences *active* or *active bits*. In Fig. 1, the blue and black bits represent active bits for the characteristic for $(\varDelta\texttt{0x4}, \varDelta\texttt{0x9})$, and the red and black bits in the first and last 3 rounds represent the ones for $(\varDelta\texttt{0x4}, \varDelta\texttt{0x5})$. We describe the 12-round characteristic for $(\varDelta\texttt{0x4}, \varDelta\texttt{0x9})$ right now, and mention the characteristic for $(\varDelta\texttt{0x4}, \varDelta\texttt{0x5})$ later in Sect. 3.3. Note that $\Pr(\varDelta\texttt{0x4}, \varDelta\texttt{0x9})$ and $\Pr(\varDelta\texttt{0x9}, \varDelta\texttt{0x4})$ are 2^{-3} and 2^{-2}, respectively. First of all, by using the pair $(\varDelta\texttt{0x4}, \varDelta\texttt{0x9})$, we construct a 6-round iterative differential characteristic where the number of active bits transits $8 \to 4 \to 2 \to 1 \to 2 \to 4 \to 8$ with a probability of $2^{-8-4-2-3-6-12} = 2^{-35}$. Second, we repeat the iterative

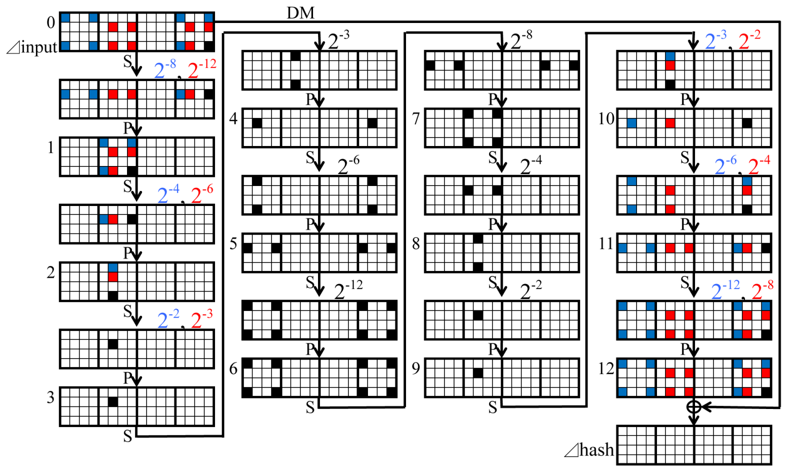


Fig. 1. Differential characteristics on 12-round DM-PRESENT. Black bits denote active bits. White bits denote zero difference. Red and blue bits represent two variants.

characteristic twice and construct the 12-round differential characteristic. The total probability that a randomly chosen input/output pair satisfies the 12-round differential characteristic is $2^{-35-35} = 2^{-70}$.

3.3 Multi-differential Characteristics for Collisions

As later discussed, we use the rebound approach to satisfy the characteristic. For this purpose, we fix the differential characteristic for the middle 5 rounds (Round 3 to Round 8) and choose internal state and message values so that the characteristic can be satisfied. Then, the first 3 and last 4 rounds belong to the outbound phase, where the characteristic is satisfied probabilistically.

We consider reducing the complexity of our collision attack by introducing multi-differential characteristics for the outbound phase. Because both of $(\varDelta\texttt{0x4}, \varDelta\texttt{0x9})$ and $(\varDelta\texttt{0x4}, \varDelta\texttt{0x5})$ contain $\varDelta\texttt{0x4}$, from the fixed middle 6-round characteristic, we can construct two differential characteristics which have the same active-bit patterns at the plaintext and ciphertext. In Fig. 1, two characteristics are denoted by blue and red. $\Pr(\varDelta\texttt{0x4}, \varDelta\texttt{0x5})$ and $\Pr(\varDelta\texttt{0x5}, \varDelta\texttt{0x4})$ are 2^{-2} and 2^{-3}, respectively. Hence, the probability of satisfying the red characteristic is the same as the one for the blue characteristic. As a result, we can obtain a collision pair that follows either two differential characteristics with 2^{-69}, which is double of the single-characteristic case.

3.4 Attack Overview

This section gives an overview of our collision attack procedure. The procedure mainly consists of an inbound phase and an outbound phase. These names derive from the rebound attack described in Section 2.2. We start searching for a colliding pair from a middle round. In the inbound phase, we aim to obtain many of internal-state values and round-message values that satisfy the differential characteristic for the middle five rounds; state ♯3 to state ♯8 of Fig. 2. The paired values satisfying the differential characteristic for the inbound phase are called *start points*. We need to generate many start points so that the differential propagation of the outbound phase can be satisfied. The inbound phase is further divided into five 1-round procedures. In each procedure, several bits of internal states are fixed to satisfy the differential characteristic. We independently perform the procedures, and then choose several bits of round-message values that connect the results of procedures without any contradiction. After we choose several bits of the internal states and round messages that satisfy the middle five rounds, 63 bits of a round message remain unfixed. We use those bits as the available degrees of freedom for the outbound phase. Therefore, we can prepare enough start points with a very low complexity.

In the outbound phase, we compute each start point in outward until plaintext and ciphertext with checking whether or not the differential propagation conforms one of the multi-differential characteristics. Because the differential propagation is probabilistic, we need to generate enough start points. Due to the DM-mode, the output of the compression function is derived from the exclusive-or of plaintext and ciphertext, and they have the identical differential form. Therefore, the plaintext and ciphertext differences cancel each other surely. In the following part, we describe the procedure of inbound phase and outbound phase in more details.

The Inbound Phase. This phase consists of five 1-round inbound procedures. In each 1-round inbound procedure, the goal is finding paired values satisfying the differential characteristic between the state just before the sBoxlayer and immediately after the pLayer. Note that our inbound phase is the (single) differential attack, not truncated differential attack. Therefore, the differential characteristic is already fixed uniquely. In the inbound phase of i-th round, we firstly fix active column values of state ♯i.5 to satisfy the differential characteristic. For instance, if 1 active column transits from $\Delta 4$ to $\Delta 9$ through the S-box, the output of the S-box must be either `0x7` or `0xE` due to the S-box characteristic. For more details of possible output values of the active column, please refer to App. B. After we fix the paired values of active columns either (`0x7`, `0xE`) or (`0xE`, `0x7`), we compute these values in backward through one S^{-1} function and in forward by one pLayer. The inbound procedure is applied to another adjacent round independently. Then, we merge two inbound procedures by fixing several bits of a round-message value. By iterating the above, we can merge five inbound procedures by fixing several bits of round messages. Choosing several bits of round-message values in different rounds sometimes causes the contradiction in the key schedule function. Using the precomputed look-up table, we can merge five inbound procedures. That is to say, a start point for the outbound phase is generated with a low complexity.

The Outbound Phase. The first and last three rounds belong to the outbound phase, where the differential transition of each S-box is satisfied probabilistically. The probability that one start point satisfies the differential characteristic of seven outbound rounds is $2^{-2-4-8-2-3-6-12} = 2^{-37}$. By using the multi-differential characteristics described in Section 3.3, the probability increases to 2^{-36}. In other words, one of the 2^{36} start points can yield the collision. We check the differential propagation of each of 2^{36} start points round by round with the early aborting technique [6, 26]. Namely, we first check whether or not the start points satisfy the differential transition from round 8 to round 9. If it is not satisfied, we stop the computation of this start point, and choose different one. If it is satisfied, we continue the computation for the next round. Due to this effort, the complexity for examining 2^{36} start points can be reduced into about $\frac{1}{12\times 16} \times 2^{36}$, which is faster than the birthday attack on 64-bit values, 2^{32}.

3.5 Attack Procedure

Our attack procedure against the 12-round compression function is as follows.

1. In the precomputation step, we generate a lookup table that are used in Step 11 of inbound phase. We compute $2^{4+4+4=12}$ tuples of $\{(x, y, m)|x, y, m \in \mathbb{F}_2^4; y = S(x \oplus m)\}$ and store them in the look-up table. x and y indicate the input/output of a single S-box, and m is 4 bits of a round message.
2. Inbound phase consists of 11 steps as follows. In this phase, we look for a pair of internal state values and one message value that satisfy the differential transition from state $\sharp 3$ to $\sharp 8$. We fix the internal states and round messages bit by bit. In Fig. 2, the colored bits in the inbound phase are classified into two types, simple-colored bits and shaded-colored bits. The simple-colored bits of internal states are fixed solely by the S-box characteristic. The shaded-colored bits are fixed after all bits of round message are determined.

Step 1. fix the values of the four columns indicated by black and red (active columns) in state $\sharp 5.5$ so that the differential transition from Δ0x4 to Δ0x9 is satisfied through the sBoxlayer in round 5. The probability of this transition is 2^{-3} per column, which means that only one pair of values can satisfy the transition. 0x7 and 0xE are the values of the S-box output satisfying the transition. For the eight black bits, we have 2^4 choices (2 choices per column). We pick any 1 from these 2^4 patterns. We then compute these 4 columns in forward until state $\sharp 6$.

Step 2. fix similarly the black and blue active columns in state $\sharp 4.5$. We need to fix the values of each S-box output only either 0x7 or 0xE for the same reason as described in Step 1. We pick any 1 from these 2^2 patterns, and then compute these 2 columns in backward until state $\sharp 5$. The 8 bits of the state just before the sBoxlayer of the fourth round are also uniquely computed.

Step 3. merge the fixed values in Step 1 and 2 to satisfy the transition from state $\sharp 5$ to $\sharp 5.5$ by fixing the four red bits of M_6. Depending on un-fixed bits of M_6, there are several possibilities for the shaded-red bits of state ♯5 in Fig. 2. The following Steps from 4 to 9 are similar to Step 2 and 3. Only positions of the fixed bits and the number of choices are different from Step 2 and 3. Step 8 and 9 are, however, reverse of Step 2 and 3.

Step 4. fix the black and yellow active columns in state ♯3.5. For the four black bits, we have 2 choices. After we pick one choice for two black bits, we compute the fixed bits in forward until state ♯4 and in backward until state ♯3.

Step 5. merge the fixed values in Step 2 and 4 to satisfy the transition from state ♯4 to ♯4.5. We can fix two blue bis of M_5.

Step 6. fix the black and green active columns in state ♯6.5. In fact, we can fix the each column either `0x0`, `0x4`, `0xB`, or `0xF` due to the differential characteristic of S-box. For the eight black bits, we have 2^8 choices (4 choices per column).

Step 7. merge the fixed values in Step 1 and 6 to satisfy the transition from state ♯6 to ♯6.5. We can fix eight green bis of M_7.

Step 8. fix two orange bis of M_8 to merge the fixed values in Step 6 and 9 to satisfy the transition. Due to the message schedule, the values of two bits of fixed M_7 and two bits of M_8 are overlapped. Thus we cannot fix both adjacent inbound procedures independently. Because of the characteristic of S-box, we can still merge them by reducing the choices from 4 to 2 per column.

Step 9. fix the two black and orange active columns in state ♯7.5. We already fixed some bits of M_8, thus we have 2^2 choices (2 choices per column, not 4 choices) for the two black bits.

Step 10. fix all remaining 63 bits of round messages randomly.

Step 11. merge whole inbound procedures. After Step 10, we can compute the shaded-colored bits in Fig. 2. In ♯5, each of 12 columns including fixed yellow or fixed blue bit must transits compatibly to ♯5.5. Thus, we fix the white bits of the internal states to satisfy the transition referring the look-up table. For each of 12 columns, we have two choices of values on averages. In other words, 2^{12} start points for one round message can be constructed with a low complexity. And we can construct more $2^{4+2+1+8+2+63+12} = 2^{92}$ start points because of the forementioned freedom degree.

3. Outbound phase consists of two steps as follows.

Step 1. compute start points in both forward and backward. Then, the total probability of the outbound phase is 2^{-37}. We check the differential propagation round by round by using the early aborting technique.

Step 2. link the input and output values of the internal cipher by exclusive-or of the DM-mode. The input and output differences match with probability of exactly 1, because both of these differences are Δ`0x9009000000009009`[1].

[1] Note that our attack is a differential attack, not a truncated differential attack. Hence, the probability of the match is 1, not 2^{-8}.

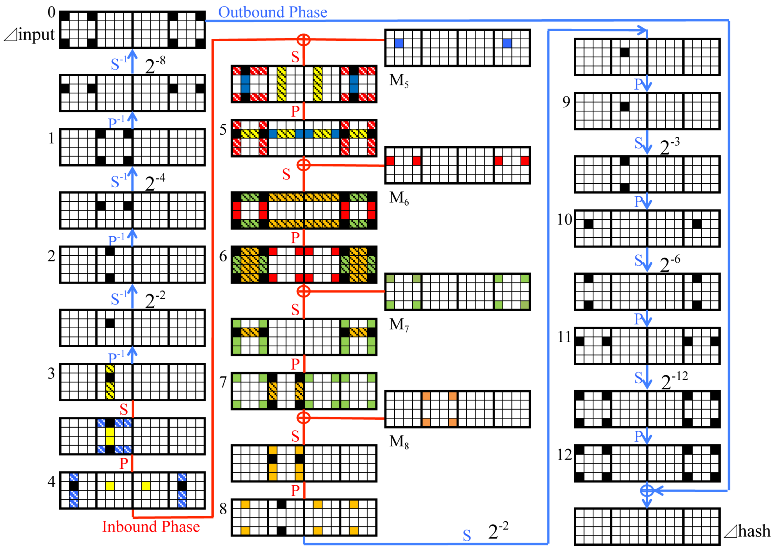


Fig. 2. Differential characteristic focused the inbound and outbound phases. Black denotes active bits. The rest of colored bits is fixed in the inbound phase.

3.6 Complexity Evaluation

We can generate 2^{12} start points for the outbound phase that satisfies the inbound phase with the complexity of about 1 on average and the 2^{12} bits memory requirement. The probability that a pair satisfies the whole outbound phase is 2^{-37}. Utilizing the multi-differential characteristics, the probability that a pair satisfies outbound phase is 2^{-36}. Then, we have to generate 2^{36} start points to find a collision pair. Remember that we can generate enough start points for the outbound phase because of the freedom degrees of internal state and message values. At a glance, a rough evaluation of our attack complexity to find a collision is 2^{36} 12-round DM-PRESENT-80 computations. However, considering the early aborting technique [6, 26], our attack complexity to find a collision is in fact much smaller. Let the complexity of 1-round function is $\frac{1}{12}$ of the 12-round DM-PRESENT-80 function, and the complexity of a column is $\frac{1}{16}$ of 1-round function. We examine all 2^{36} start points for round 8. It is only necessary to compute a column whether or not the start points satisfy the differential characteristics. Hence, the complexity for round 8 is $\frac{1}{12} \times \frac{1}{16} \times 2^{36}$. After round 8, only $2^{36-2} = 2^{34}$ pairs follow the characteristics in Fig. 1. Similarly, we examine 2^{34} start points for round 3. And then, $2^{34-3} = 2^{31}$ pairs follow the red characteristic, and $2^{34} \times 2^{-2} = 2^{32}$ pairs follow the blue one. Hence, we examine $2^{31} + 2^{32}$ pairs for round 9, and thus the complexity for round 9 is $\frac{1}{12} \times \frac{4}{16} \times (2^{31} + 2^{32})$. After round 9, $2^{36-2-3-2} = 2^{29}$ pairs satisfy the red characteristic and $2^{36-2-2-3} = 2^{29}$ pairs satisfy the blue one. After all, the attacker computes $\frac{1}{12\times 16} \times (2^{36} + 2^{34}) + \frac{4}{12\times 16} \times (2^{31} + 2^{32} + 2^{29} + 2^{29}) + \frac{1}{12} \times (2^{23} + 2^{25} + 2^{11} + 2^{17} + 2^{7} + 2^{11}) \approx 2^{29.18}$

12-round DM-PRESENT-80 computations. Finally, we can find collisions of the 12-round DM-PRESENT-80 compression function faster than the birthday attack, which requires 2^{32} computations.

4 Application for other Attacks

The differential characteristic discussed in the previous section can be used to construct other kinds of attacks. In this section, we discuss an 18-round distinguisher and 12-round second preimage attack on the compression function.

4.1 18-Round Distinguisher

We construct an 18-round differential distinguisher that finds a pair of messages with specific input and output differences. We show that the attack on 18-round DM-PRESENT-80 is faster than the attack on an ideal compression function.

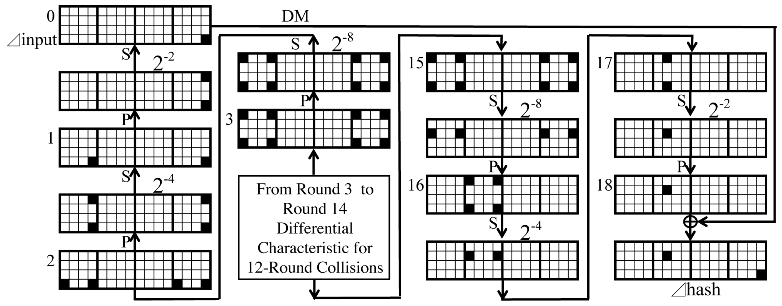

0
⊿input
DM
S
2^{-2}
P
1
2^{-4}
2
2^{-8}
3
From Round 3 to Round 14 Differential Characteristic for 12-Round Collisions
15
16
17
18
2^{-8}
2^{-4}
2^{-2}
⊿hash

Fig. 3. Differential characteristic for 18-round distinguisher

For this attack, we extend the differential characteristic by 3 rounds in backward and 3 rounds in forward. The procedure of the multi-inbound phase is the same as the one in Sect 3, where middle 5 rounds can be satisfied with average complexity 1. Hence, only the outbound phase is extended and satisfying the entire differential characteristic becomes harder for these extended outbound phase. Note that we do not have to match the differential forms of the plaintext and ciphertext. This enables the attacker to use the differential propagation with probability 2^{-2} for each S-box transformation in both directions, i.e., $\varDelta 1 \rightarrow \varDelta 9$ with probability of 2^{-2} instead of $\varDelta 4 \rightarrow \varDelta 9$ with probability of 2^{-3}.

The differential propagations for the first and last 3 rounds are shown in Fig. 3. The input value has 1-bit difference in the chaining variable and no difference in the message. The output values has 1-bit difference. As discussed in Sect. 3, satisfying the middle 12 rounds (round 3 to round 14) requires the complexity of $2^{29.18}$. Then, extending the characteristic from Round 3 to 2, 2 to 1, and 1 to 0 requires the complexity of $2^{2\times 4} = 2^8$, $2^{2\times 2} = 2^4$, and 2^2, respectively. Similarly extending the characteristic from Round 14 to 17 requires $2^{8+4+2} + 2^{14}$. Finally, the entire characteristic is satisfied with the complexity of $2^{2+4+8+29.18+14} = 2^{57.18}$ 18-round DM-PRESENT computations. We then show that finding such pairs in an ideal compression function requires more complexity. In the truncated differential analysis, this complexity is evaluated by the limited birthday attack proposed Gilbert and Peyrin [10]. However, our attack only allows a 1-bit difference on the input. Therefore, the *structure* technique (constructing 2^{2x-1} pairs with 2^x queries) cannot be applied. The best way is randomly generating (h_{i-1}, M) and check whether or not $\mathrm{CF}(h_{i-1}, M) \oplus \mathrm{CF}(h_{i-1} \oplus \Delta^{\mathrm{IN}}, M) = \Delta^{\mathrm{OUT}}$. The relation holds with a probability of 2^{-64}. Thus, our attack is faster than the ideal case.

4.2 12-Round Second Preimage Attack on Compression Function

Our differential attack can be converted into a second preimage attack on the compression function by using the conversion proposed at CANS 2005 by Yu *et al.* [27]. In the second preimage attack on the compression function, the attacker is given a message M, input chaining variable h, and the output of the compression function $\mathrm{CF}(h, M)$. For an ideal compression function, the probability that $\mathrm{CF}(h, M) = \mathrm{CF}(h', M')$ is satisfied for $(h, M) \neq (h', M')$ is 2^{-n}. Therefore, finding a way to choose (h', M') satisfying the above equation with a higher probability than 2^{-n} can be regarded as the second preimage attack. Note that the second preimage attack on the compression function is also discussed by Rechberger [24]. Finally, if we can find the differences $(\Delta h, \Delta M)$ where $\Pr[\mathrm{CF}(h, M) = \mathrm{CF}(h \oplus \Delta h, M \oplus \Delta M)] > 2^{-n}$, for a randomly chosen h, M, we can succeed in constructing the second preimage attack. The differential characteristic for 12-round collisions in Sect. 3 satisfies the above equation with probability 2^{-70}. At a glance, we need 2^{70} 12-round DM-PRESENT-80 computations to the second preimage attack. However, considering the early aborting technique again, our attack complexity can be much smaller. In fact, we can construct the second preimage attack on the compression function with a complexity of $2^{61.91}$ 12-round DM-PRESENT-80 computations. Due to the regulation of pages, we omit the detail of the complexity evaluation.

5 Concluding Remarks

In this paper, we presented the first third-party security analysis of reduced-round DM-PRESENT-80. The main result is a collision attack on the 12-round compression function. We constructed a differential characteristic suitable for collisions, and efficiently found paired values with the multi-inbound technique. Based on this attack, we also presented the 18-round distinguisher and 12-round second preimage attack on the compression function. Because PRESENT is one of the most successful designs for the lightweight cryptography, we believe that our results contribute to deeper understanding of lightweight designs.

Acknowledgments. The authors would like to thank anonymous reviewers for their helpful comments to improve the quality of this paper.

References

1. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the indifferentiability of the sponge construction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 181–197. Springer, Heidelberg (2008)
2. Blondeau, C., Gérard, B.: Multiple differential cryptanalysis: Theory and practice. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 35–54. Springer, Heidelberg (2011)
3. Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: SPONGENT: A lightweight hash function. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 312–325. Springer, Heidelberg (2011)
4. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
5. Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash functions and RFID tags: Mind the gap. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 283–299. Springer, Heidelberg (2008)
6. De Cannière, C., Rechberger, C.: Finding SHA-1 characteristics: General results and applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer, Heidelberg (2006)
7. Cho, J.Y.: Linear cryptanalysis of reduced-round PRESENT. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 302–317. Springer, Heidelberg (2010)
8. Dai, Z., Wang, M., Sun, Y.: Effect of the dependent paths in linear hull. Cryptology ePrint Archive: Report 2010/325 (2010)
9. Ferguson, N.: Observations on H-PRESENT-128. Rump Session of CRYPTO 2011 (2011)
10. Gilbert, H., Peyrin, T.: Super-Sbox cryptanalysis: Improved attacks for AES-like permutations. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 365–383. Springer, Heidelberg (2010)
11. Hermelin, M., Nyberg, K.: Linear cryptanalysis using multiple linear approximations. Cryptology ePrint Archive: Report 2011/093 (2011)
12. ISO/IEC 29192-2:2011: Information technology–Security techniques–Lightweight cryptography–Part 2: Block ciphers (2011)
13. Kobayashi, T., Hirose, S.: Collision attack on double-block length compression function using round-reduced PRESENT. In: SCIS 2012 (2012) (in Japanese)
14. Kumar, M., Yadav, P., Kumari, M.: Flaws in differential cryptanalysis of reduced round PRESENT. Cryptology ePrint Archive: Report 2010/407 (2010)
15. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound distinguishers: Results on the full whirlpool compression function. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 126–143. Springer, Heidelberg (2009)
16. Matusiewicz, K., Naya-Plasencia, M., Nikolić, I., Sasaki, Y., Schläffer, M.: Rebound attack on the full LANE compression function. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 106–125. Springer, Heidelberg (2009)
17. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved cryptanalysis of the reduced Grøstl compression function, ECHO permutation and AES block cipher. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 16–35. Springer, Heidelberg (2009)
18. Mendel, F., Pramstaller, N., Rechberger, C., Rijmen, V.: The impact of carries on the complexity of collision attacks on SHA-1. In: Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 278–292. Springer, Heidelberg (2006)

19. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and `Grøstl`. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 260–276. Springer, Heidelberg (2009)
20. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of applied cryptography. CRC Press (1997)
21. Nakahara Jr., J., Sepehrdad, P., Zhang, B., Wang, M.: Linear (hull) and algebraic cryptanalysis of the block cipher PRESENT. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 58–75. Springer, Heidelberg (2009)
22. Ohkuma, K.: Weak keys of reduced-round PRESENT for linear cryptanalysis. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 249–265. Springer, Heidelberg (2009)
23. Özen, O., Varıcı, K., Tezcan, C., Kocair, Ç.: Lightweight block ciphers revisited: Cryptanalysis of reduced round PRESENT and HIGHT. In: Boyd, C., González Nieto, J. (eds.) ACISP 2009. LNCS, vol. 5594, pp. 90–107. Springer, Heidelberg (2009)
24. Rechberger, C.: Second-preimage analysis of reduced SHA-1. In: Steinfeld, R., Hawkes, P. (eds.) ACISP 2010. LNCS, vol. 6168, pp. 104–116. Springer, Heidelberg (2010)
25. Wang, M.: Differential cryptanalysis of reduced-round PRESENT. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 40–49. Springer, Heidelberg (2008)
26. Wang, X.: Cryptanalysis of hash functions and potential dangers. Invited Talk at CT-RSA 2006 (2006)
27. Yu, H., Wang, G., Zhang, G., Wang, X.: The second-preimage attack on MD4. In: Desmedt, Y.G., Wang, H., Mu, Y., Li, Y. (eds.) CANS 2005. LNCS, vol. 3810, pp. 1–12. Springer, Heidelberg (2005)

A Specification of DM-PRESENT-80

The lightweight block cipher PRESENT was proposed by Bogdanov *et al.* in CHES 2007 [4]. PRESENT has a 31-round SPN (Substitution and Permutation Network) construction. The block length is 64 bits and two key lengths of 80 and 128 bits are supported. The lightweight hash functions DM-PRESENT-80/-128 were also proposed by Bogdanov *et al.* at CHES 2008 [5]. The compression function of DM-PRESENT is constructed by PRESENT with the Davies-Meyer (DM) mode [20, Algorithm 9.42]. Thus, the j-th 64-bit chaining variable H_j of DM-PRESENT is updated using a 80 or 128 bits message M_j to $H_{j+1} = E(H_j, M) \oplus H_j$. $E(\cdots, K)$ indicates the encryption operation of PRESENT under a secret key K. The proposers of PRESENT recommended the 80-bit key version for applications in resource constraint environments. So we show attacks on DM-PRESENT-80 in this paper, and the details of PRESENT-80 in this section. We denote a state of 64 bits data block $X = (x_{63}, x_{62}, ..., x_0)$ by 4-by-16 matrix as;

$$\text{X} = \begin{pmatrix} x_{63}, x_{59}, x_{55}, x_{51}, x_{47}, x_{43}, x_{39}, x_{35}, x_{31}, x_{27}, x_{23}, x_{19}, x_{15}, x_{11}, x_7, x_3 \\ x_{62}, x_{58}, x_{54}, x_{50}, x_{46}, x_{42}, x_{38}, x_{34}, x_{30}, x_{26}, x_{22}, x_{18}, x_{14}, x_{10}, x_6, x_2 \\ x_{61}, x_{57}, x_{53}, x_{49}, x_{45}, x_{41}, x_{37}, x_{33}, x_{29}, x_{25}, x_{21}, x_{17}, x_{13}, x_9, x_5, x_1 \\ x_{60}, x_{56}, x_{52}, x_{48}, x_{44}, x_{40}, x_{36}, x_{32}, x_{28}, x_{24}, x_{20}, x_{16}, x_{12}, x_8, x_4, x_0 \end{pmatrix}. \tag{4}$$

The round transformations of PRESENT are as follows.

– addRoundkey (AK) adds the the 64 bits round key.
– sBoxlayer (S) is a 4-to-4 bits S-box of PRESENT and applies to each vertical 4 bits. The transition by sBoxlayer is illustrated at the right side of Fig. 4.
– pLayer (P) permutes the horizontal 4 bits to the vertical 4 bits. The left side of Fig. 4 illustrates where every 4 bits group is permuted by pLayer.

Fig. 4 illustrates the transition by S and P. Then, the i-th round function F of PRESENT can be denoted by

$$X_{i+1} = F(X_i) \equiv P \circ S \circ AK(X_i). \tag{5}$$

The round keys are generated as follows. The 80 bits secret key is stored in a key register K and represented as $k_{79}k_{78}...k_0$. The i-th round key K_i $(1 \leq i \leq 32)$ consists of leftmost 64-bit of the actual content of register K. Thus the first round key K_1 is $K_1 = k_{79}k_{78}...k_{16}$. To generate next round key, the key resister K is updated as follows.

– 61 bits rotation: $[k_{79}k_{78}...k_0] = [k_{18}k_{17}...k_0k_{79}...k_{19}]$
– partial sBoxlayer: $[k_{79}k_{78}k_{77}k_{76}] = sBoxlayer[k_{79}k_{78}k_{77}k_{76}]$
– addRound_counter: $[[k_{19}k_{18}k_{17}k_{16}k_{15}] = [k_{19}k_{18}k_{17}k_{16}k_{15}] \oplus round\_counter(i)$

The $round\_counter(i)$ is the 5-bit binary representation of i. K_{32} is used for post-whitening. It is similar to the round keys that we call M_i as i-th round message from the message M.

We use the following notation to the internal states to describe our attacks: $\sharp x.y$ denotes the number of intermediate states. See Fig. 1, state $\sharp 0$, for example, indicates an input differential value, and state $\sharp 5.5$ indicates the internal state after the S transformation of the fifth round and before the P transformation of the fifth round, and state $\sharp 5.5$, for example, indicates the internal state after the S transformation of the fifth round and before the P transformation of the fifth round.

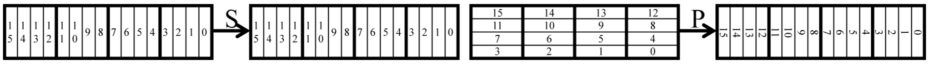


Fig. 4. The sBoxlayer and pLayer of PRESENT. Each rectangle contains 4 bits. Both transformations are operated per 4 bits.

B Differential Characteristics of S-Box

This section shows the input/output pairs of S-box that are used in our differential characteristic in detail. We note that following values of the transition are represented in hexadecimal notation. For a fixed differences $(\Delta x, \Delta y)$, where $\Delta x = x \oplus x'$ and $\Delta y = S(x) \oplus S(x')$, we searched for the input pairs (x, x') that satisfy $(\Delta x, \Delta y)$. These pairs can be searched by computing all 16×16 pairs of (x, x') and $(S(x), S(x'))$. As a result the number of input/output pairs that satisfy one $(\Delta x, \Delta y)$ is either two, four or zero. Obviously, the number of input pairs (x, x') that satisfy $\Delta x = x \oplus x'$ is 16. Table 2 indicates all the possible input pairs (x, x') that satisfy either $(\Delta 4, \Delta 9)$, $(\Delta 9, \Delta 4)$, $(\Delta 4, \Delta 5)$, or $(\Delta 5, \Delta 4)$. The input pair $(9, D)$, for instance, satisfies the differential transition of $(\Delta 4, \Delta 9)$. The order of x and x' can be exchanged against the S-box operation. Then the symmetric input pairs $(9, D)$ and $(D, 9)$ satisfy $(\Delta x, \Delta y)$. Hence, there are two pairs that satisfy $(\Delta 4, \Delta 9)$. And there are four pairs that satisfy $(\Delta 9, \Delta 4)$ similarly. The same analysis is applied to $(\Delta 4, \Delta 5)$ and $(\Delta 5, \Delta 4)$.

Table 2. The input pairs (x, x') of S-box used in our differential characteristic

(x, x')	$\Delta input$	$\Delta output$
(9, D)	9⊕D = 4	S(9)⊕S(D) = E⊕7 = 9
(3, A)	3⊕A = 9	S(3)⊕S(A) = B⊕F = 4
(5, C)	5⊕C = 9	S(5)⊕S(C) = 0⊕4 = 4
(0, 4)	0⊕4 = 4	S(0)⊕S(4) = C⊕9 = 5
(1, 5)	1⊕5 = 4	S(1)⊕S(5) = 5⊕0 = 5
(8, D)	8⊕D = 5	S(8)⊕S(D) = 3⊕7 = 4

Estimating the Probabilities of Low-Weight Differential and Linear Approximations on PRESENT-Like Ciphers

Mohamed Ahmed Abdelraheem

Department of Mathematics
Technical University of Denmark, Lyngby, Denmark

Abstract. We use large but sparse correlation and transition-difference-probability submatrices to find the best linear and differential approximations respectively on PRESENT-like ciphers. This outperforms the branch and bound algorithm when the number of low-weight differential and linear characteristics grows exponentially which is the case in PRESENT-like ciphers. We found linear distinguishers on 23 rounds of the SPONGENT permutation. We also found better linear approximations on PRESENT using trails covering at most 4 active Sboxes which give us 24-round statistical saturation distinguishers which could be used to break 26 rounds of PRESENT.

Keywords: block cipher, differential, difference matrix, linear hull, correlation matrix, statistical saturation attack, PRESENT, SPONGENT.

1 Introduction

In recent years, the need for lightweight encryption systems has been increasing as many applications use RFID and sensor networks which have a very low computational power and thus incapable of performing standard cryptographic operations. In response to this problem, the cryptographic community designed a number of lightweight cryptographic primitives that varies from stream ciphers such as Grain [15], Trivium [8], and block ciphers such as PRESENT [5], KATAN/KTANTAN [7] and recently to hash functions such as QUARK [1], PHOTON [14] and SPONGENT [4].

Out of these many lightweight primitives, the block cipher PRESENT gets a lot of attention from the cryptographic community and it has been recently adopted as an ISO standard (ISO/IEC 29192) [18]. In this paper, we focus on the differential and linear cryptanalysis of the following two ciphers: PRESENT and SPONGENT. We mainly discuss how to estimate the probabilities of low-weight differential and linear approximations on these kind of ciphers.

Our Contribution: We estimate the probability of low-weight linear and differential approximations in PRESENT-like ciphers. By using large but sparse correlation and difference submatrices, we overcome the memory and time problems that appears in the branch and bound algorithm [22] when the number of good linear and differential trails grows exponentially. For instance, there would be memory and time problems in the branch and bound method when the number of differential and linear trails grows exponentially but using sparse correlation and difference submatrices we can handle any number of trails and investigate many differential and linear approximations with a negligible cost. For instance, all the linear approximations of PRESENT at Table 4 described in the Appendix come from a number of good and bad trails exceeding 2^{89} which clearly shows a convincing advantage of using sparse submatrices over the branch and bound algorithm when the number of trails grows exponentially.

Using sparse correlation and difference submatrices of PRESENT and SPONGENT: We report the first improved analysis on SPONGENT [4], specifically we improve the linear cryptanalysis on the SPONGENT permutation presented by the designers by one more round. We present better linear approximations on PRESENT and present a 24-round statistical saturation distinguisher that uses better input and output subspaces compared to the original attack paper [10]. These approximations also show that the assumption, made by all the previous analyses on PRESENT [9, 21, 25], that the linear approximations consisting of trails with one active Sbox at each round, yield the highest bias is not valid. We also found many 16-round differential approximations activating at most 4 Sboxes per round with probability larger than 2^{-64}, which could be used to mount a differential attack on 18-round PRESENT similar to the ones in [2, 29].

Outline of the paper: In Section 2, we give a brief description of PRESENT and SPONGENT. Section 3 defines the basic concepts about linear and differential cryptanalysis. Section 4 describes our sparse matrices approach for finding tight linear and differential approximations and its time complexity. Section 5 shows the linear and differential approximations in PRESENT and SPONGENT that were found using our method. Finally, we conclude on Section 6.

2 A Short Description of PRESENT and SPONGENT

PRESENT is a 64-bit iterated block cipher. It consists of 31 rounds and supports 80-bit and 128-bit key lengths. It was mainly designed for hardware constrained devices [5]. It has a very simple design as it consists of only three layers: the key addition layer, the 4-bit Sbox layer (SboxLayer) and the bitwise permutation layer (PLayer). This simple linear layer consisting of the bitwise permutation only allows the existence of low-weight differential and linear characteristics.

SPONGENT is a new lightweight hash function [4], its core permutation is inspired by PRESENT as it inherits its three layers. There are many variants of SPONGENT here we are concerned with the permutation of SPONGENT-88 which runs for 45 rounds. The SPONGENT permutation can be seen as a cipher using identical round keys (the key is almost the zero key xored with few bits at the leftmost and the rightmost ends generated from a counter that is meant to prevent sliding properties and invariant subspaces). In other words, the core permutation of SPONGENT can be seen as a PRESENT-like cipher which is a definition we borrowed from [6]. For more details about the description of PRESENT and SPONGENT, we refer to [4,5].

3 Preliminaries

Suppose we have a symmetric cipher defined by the permutation F, under a key $K \in \mathbb{F}_2^n$, $F_K : \mathbb{F}_2^n \to \mathbb{F}_2^n$.

Differential cryptanalysis exploits the difference distribution under some algebraic group operation (usually $\oplus$) of a pair of plaintexts and their corresponding ciphertexts, i.e. attacker finds a difference α in the plaintext pairs and a difference β in their corresponding ciphertexts such that $\Pr(F_K(X \oplus \alpha) \oplus F_K(X) = \beta)$ is higher than 2^{1-n}.

Linear cryptanalysis finds a linear relation between some plaintext bits and ciphertext bits (and also some secret key bits in the case of block ciphers) and then exploits the bias or the correlation of this linear relation, i.e., attacker finds an input mask α and an output mask β that yield a higher absolute *bias* $\epsilon_F(\alpha, \beta) \in [-\frac{1}{2}, \frac{1}{2}]$. In other words $\Pr(\langle \alpha, X \rangle + \langle \beta, F_K(X) \rangle = \langle \gamma, K \rangle) = \frac{1}{2} + \epsilon_F(\alpha, \beta)$ is deviated from half, where $\langle \cdot, \cdot \rangle$ denotes an inner product. The correlation of a linear approximation is defined as $C_F(\alpha, \beta) := 2\epsilon_F(\alpha, \beta)$.

Linear and differential attacks are based on the so-called linear and differential characteristics (aka trails or paths) respectively, each characteristic is a sequence of intermediate linear or difference relations for all rounds where the probability of each element in this sequence determine the probability of a differential characteristic and the bias of a linear characteristic. The collection of all the r-round differential characteristics with input $\alpha = \alpha_0$ and output $\beta = \alpha_r$ is called the *differential* of the difference approximation (α_0, α_r), each r-round differential characteristic can be seen as a sequence $(\alpha_0, \alpha_{1i}, \cdots, \alpha_{(r-1)i}, \alpha_r)$, where i defines the i-th differential trail between α_0 to α_r. Similarly, the collection of all the linear characteristics with input mask $\alpha = \alpha_0$ and output mask $\beta = \alpha_r$ is often called the *linear hull* of the linear approximation (α_0, α_r), also each r-round linear characteristic could be seen as a sequence $(\alpha_0, \alpha_{1i}, \cdots, \alpha_{(r-1)i}, \alpha_r)$, where i defines the i-th linear trail between α_0 to α_r.

One class of these iterated ciphers defined in [20], is called Markov ciphers. For such ciphers, under well established independence assumptions, the probability of a differential and the correlation of a linear relation can be computed using a difference transition matrix and a correlation matrix respectively. In the following, we briefly describe how to construct and use these matrices.

Difference Transition Matrix [20]: Given an r round Markov cipher and assuming independent and uniformly random round keys. We estimate the probability of an r-round differential (α_0, α_r) by considering the probability of each differential characteristic between α_0 and α_r. Thus, the probability of the i-th differential characteristic $(\alpha_0 = \alpha_{0i}, \alpha_{1i}, \cdots, \alpha_{(r-1)i}, \alpha_r = \alpha_{ri})$ is $p_i = \prod_{j=1}^{r} \Pr(F_K(X) \oplus F_K(X') = \alpha_{ji} | X \oplus X' = \alpha_{(j-1)i})$. Consequently the probability of an r-round differential (α_0, α_r) is the sum of all the probabilities of all the possible differential characteristics between (α_0, α_r), that is $\sum_{i=1}^{N_d} p_i$, where N_d is the number of all the possible differential characteristics between (α_0, α_r). Let D denote the transition difference-probability matrix of an n-bit Markov cipher. D has size $(2^n - 1) \times (2^n - 1)$, the (i, j) entry in D corresponds to the probability of an output difference, say β_j, when we have an input difference, say β_i, i.e., $\Pr(\Delta(F_K(X)) = \beta_j | \Delta(X) = \beta_i)$, where F_K is the round function of the Markov cipher. Now, for any r, the (i, j) entry of the matrix D^r, $p_{ij}^{(r)}$ is equivalent to the probability of the r-round differential (β_i, β_j).

Correlation Matrix [11, 12]: Given a composite function $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ such that $F = F_r \circ \cdots \circ F_2 \circ F_1$. We estimate the correlation of an r-round linear approximation (α_0, α_r) by considering the correlation of each linear characteristic between α_0 and α_r, the correlation of i-th linear characteristic $(\alpha_0 = \alpha_{0i}, \alpha_{1i}, \cdots, \alpha_{(r-1)i}, \alpha_r = \alpha_{ri})$ is $C_i = \prod_{j=1}^{r} C_{F_j}(\alpha_{(j-1)i}, \alpha_{ji})$. It is well known, see e.g., [12], that the correlation of a linear approximation is the sum of all correlations of linear trails starting with the same mask α and ending with the same mask β, i.e., $C_F(\alpha_0, \alpha_r) = \sum_{i=1}^{N_l} C_i$, where N_l is the number of all the possible linear characteristics between (α_0, α_r).

The sign of the correlation of a linear trail depends on the round keys. In [12] the following formulas were proven under the assumption that we have a key-alternating cipher[1]: $C_i = (-1)^{s_i} \prod_{j=1}^{r} C_{F_j}(\alpha_{(j-1)i}, \alpha_{ji})$, where $s_i \in \mathbb{F}_2$ depends on the i-th linear characteristic and the round keys. Therefore, the correlation of the linear hull (α, β) is $C_F(\alpha, \beta) = \sum_{i=1|\alpha_i=(\alpha=\alpha_{0i},\cdots,\alpha_{(r-1)i},\beta=\alpha_{ri})}^{N_l} (-1)^{s_i \oplus d_i} |C_i|$, where $d_i \in \mathbb{F}_2$ refers to the sign of the correlation, C_i.

Let C denote the correlation matrix of an n-bit key-alternating cipher. C has size $(2^n - 1) \times (2^n - 1)$, the (i, j) entry in C corresponds to the correlation of an input mask, say β_i, and output mask, say β_j, i.e. $C_F(\beta_i, \beta_j) = 2\Pr(\langle \beta_i, x \rangle = \langle \beta_j, F(x) \rangle) - 1$, where F is the un-keyed composite function of the key-alternating cipher and 'x' is its input. Now the correlation matrix for the keyed round function is obtained by changing the signs of each row in C according to the round subkey bits or the round constant bits involved.

Statistical Saturation Attacks: They are the first attacks proposed on the block cipher PRESENT. Briefly the idea behind these attacks is to fix some input bits to a certain value and study the distribution of some output bits, for more details we refer to [10]. In [21] it was shown that it is closely related to the linear multidimensional attack and especially the one on PRESENT [9]. The following proposition formulated at [21] estimates the capacity of statistical saturation attacks which is used to estimate the data complexity required to mount the attack.

Proposition 1. *Let $F : \mathbb{F}_2^r \times \mathbb{F}_2^s \to \mathbb{F}_2^t \times \mathbb{F}_2^u$ be an n-bit encryption function where $r + s = t + u = n$, F is restricted by fixing s bits in the input and*

[1] Key-alternating ciphers are a subclass of Markov ciphers that alternate key addition with key-independent rounds.

only t bits of the output are considered. Let $U = \mathbb{F}_2^s \subseteq \mathbb{F}_2^n$ and $V = \mathbb{F}_2^t \subseteq \mathbb{F}_2^n$ be the two subspaces corresponding to the input and output masks respectively. Then the average capacity over all the possible s-bit fixations is estimated by $\overline{C_F} = \sum_{u \in U, v \in V} (C_F(u,v))^2$.

The data complexity of statistical saturation attacks is determined by the squared Euclidean distance which is equivalent to $2^t\overline{C_F}$ where t is the number of the output bits considered in the distribution. Statistical saturation attacks perform well when we identify subspaces U and V that make the sum $\sum_{u \in U, v \in V} (C_F(u,v))^2$ big.

4 Description of Our Estimation Approach

Assuming that PRESENT-like ciphers are Markov ciphers [20, 24], we make use of submatrices of the correlation and the transition probability matrices of the target ciphers to find the best linear and differential approximations. We focus only on describing how to find better linear approximations.

4.1 Large Sparse Correlation and Difference Matrices

In [3, 4], a submatrix of the correlation matrix of size $4n_s \times 4n_s$ was used to estimate the correlation of a linear approximation, where n_s is the number of the 4-bit Sboxes used in the permutation where only input and output masks of Hamming weight one are considered. We extend this approach by adding input and output masks with Hamming weight ≤ 4. This results in having a large correlation submatrix whose entries activate at most 4 active Sboxes. Suppose that we use a correlation submatrix with input and output masks with Hamming weight less than or equal to m. Then the size of the submatrix will be $\sum_{i=1}^{m} \binom{n}{i} \times \sum_{i=1}^{m} \binom{n}{i}$, where n is the block size of the cipher in bits. The submatrix size is large but most of its entries are zeros. For instance, we see that for any input element activating 's' Sboxes ($1 \leq s \leq m$), all the other output elements corresponding to the other Sboxes yield a zero correlation. Thus, there are more than $\sum_{i=1}^{m} \binom{n}{i} - 15^s$ zero output elements for any input activating 's' Sboxes. Thus, this submatrix has few non zero elements and therefore it can easily fit in memory using a sparse matrix storage format (See Section 5).

The construction of the correlation submatrix is straightforward. For instance to fill the submatrix entries from an input with Hamming weight 5, we proceed as follows: for each possible input, we determine the number of activated Sboxes which is in this case at least 2. Suppose we have i active Sboxes, then all the possible ordered solutions of the inequality $x_1 + x_2 + \cdots + x_i \leq m$ determine the Hamming weight of the output bits of each of the i active Sboxes. Then we fill the submatrix entries corresponding to the specified input by considering all the possible output bits of the specified Hamming weight. To estimate the cost of filling these entries, we consider the simple case where we have two active Sboxes. The time cost for filling the corresponding submatrix entries is $\sum_{2 \leq i+j \leq 5} \binom{4}{i}\binom{4}{j}$

and the number of all the possible inputs with Hamming weight 5 activating 2 Sboxes is $N_2 = \sum_{w_1+w_2=5} \binom{\frac{n}{4}}{2}\binom{4}{w_1}\binom{4}{w_2}$. Now by symmetry, the cost of filling the corresponding entries that have outputs with Hamming weight 5 activating 2 Sboxes is similar but we only exclude the duplicated cases where $i+j=5$, so the cost is $\sum_{2\leq i+j<5}\binom{4}{i}\binom{4}{j}$. Therefore, the total construction time of input and output with Hamming weight 5 activating 2 Sboxes is $2N_2\sum_{2\leq i+j\leq 5}\binom{4}{i}\binom{4}{j} - N_2\sum_{i+j=5}\binom{4}{i}\binom{4}{j}$. One can see that the construction time can be generalized as follows.

Proposition 2. *The time cost for computing the correlations corresponding to inputs and outputs of Hamming weight 'w', $1 \leq w \leq m$ is in the order of:*
$2(N_1\sum_{1\leq i\leq w}\binom{4}{i} + N_2\sum_{2\leq i+j\leq w}\binom{4}{i}\binom{4}{j} + \cdots + N_{w-1}\sum_{w-1\leq i+\cdots+z\leq w}\binom{4}{i}\cdots\binom{4}{z}) + N_w4^w - N_2\sum_{i+j=w}\binom{4}{i}\binom{4}{j} - \cdots - N_{w-1}\sum_{i+\cdots+z=w}\binom{4}{i}\cdots\binom{4}{z}$, *where* $N_1+\cdots+N_w = \binom{n}{w}$ *and* $N_i = \sum_{w_1+\cdots+w_i=w}\binom{\frac{n}{4}}{i}\binom{4}{w_1}\cdots\binom{4}{w_i}$ *is the number of elements with Hamming weight 'w' activating 'i' number of Sboxes.*

Note that $N_1 = 0$ when $w \geq 5$ as in this case we have at least two active Sboxes. The total construction time is in the order of the sum of construction times of all the possible input and output weights (w), i.e. $1 \leq w \leq m$, where the dominant term is when $w = m$.

After constructing the correlation submatrix, C. The correlation approximations after r rounds is computed by $C^r = \prod_{i=1}^{r} M_i$, where M_i is the correlation submatrix at round i formed by changing the signs of C according to the round key and the round constant used in the cipher. The maximum correlation after r rounds is thus given by $c^r_{max} := \max|C^r_{ij}|$. This works in the case of SPONGENT since we know that it uses an almost zero key at each round and thus we can compute the actual correlation of each approximation but in the case of PRESENT, we need to compute the average squared correlation (aka potential linear approximation [23] or expected linear probability [13]) of a linear approximation (the sum of the squares of the correlations of all trails) in order to compute the capacity of the statistical saturation attack.

As the size of the correlation submatrix gets bigger when considering masks with Hamming weight equal to 4, the matrix-matrix multiplications might not be always possible for high number of rounds especially when there are many trails for most of the approximations as these make the resulted submatrix C^r very dense and consequently we might run out of memory. Thus, instead we use successive matrix-vector multiplications as described by Algorithm 1 in the Appendix.

Note that before Step 3 in Algorithm 1 when we are computing the maximum absolute correlation (for example in SPONGENT), we have to change the signs at some entries of the correlation submatrix M at each round according to the corresponding round constant. The time complexity of Algorithm 1 is the order of $l \times (r-1)$ matrix-vector multiplications where l is the number of rows or columns of the submatrix. If l is a large number, then the most convenient way is to consider correlation matrices with Hamming weight up to 2 or

3 bits depending on the size of the block cipher. Then, try to perform matrix-matrix multiplications and find the active input and output Sboxes that yield the maximum absolute value as they would probably be the Sboxes that yield the maximum value when considering matrices using Hamming weight more than 3. For instance, experiments on the PRESENT correlation submatrix with Hamming weight up to 3, where we are able to perform matrix-matrix multiplication and thus determine the correlations of all the approximations, showed us that the best linear approximations often come from an input mask activating only one Sbox and also an output mask activating only one Sbox (which get permuted afterwards).

5 Improved Linear and Differential Approximations

We use the approach described in section 4.1 and report the best linear and differential approximations we found in PRESENT and SPONGENT. Using the time complexity formula, given at section 4.1, we show the times taken in constructing the sparse matrices in PRESENT and SPONGENT.

Table 1. $n \equiv$ Cipher's block size, $m \equiv$ maximum Hamming weight used, $size \equiv$ submatrix size, $nnzC \equiv$ non zero elements of the correlation submatrix, $nnzD \equiv$ non zero elements of the difference submatrix, Time complexity of correlation or difference submatrix construction $\equiv O(t)$, $- \equiv$ bounded by t since each step in t fills an entry in the correlation or difference submatrix. The time complexity unit is simple arithmetic operations.

Cipher	n	m	$\log_2(size)$	$\log_2(nnzC)$	$\log_2(nnzD)$	$\log_2(t)$
PRESENT	64	4	19.37×19.37	23.26	18.41	27.83
PRESENT	64	5	22.99×22.99	-	-	33.85
PRESENT	64	6	26.31×26.31	-	-	39.61
SPONGENT	88	4	21.22×21.22	23.63	19.18	29.58
SPONGENT	88	5	25.31×25.31	-	-	36.04

As shown in Table 1, the number of elements in the correlation and differences submatrices of both PRESENT and SPONGENT is huge. A standard matrix representation would cost $2^{41.74}$ and $2^{45.44}$ bytes for the difference matrix of PRESENT and SPONGENT respectively. This is more than 1 TB. Therefore, we need to avoid running out of memory by using a sparse matrix representation which reduces memory by only allocating space for the nonzero elements. This will also speed the matrix-vector or matrix-matrix multiplications which we perform to find the best linear and difference approximations.

Table 1 shows us that our submatrices are very sparse, for instance the first table entry indicates that the density $(=\frac{nnz}{Size \times Size})$ of the difference transition submatrix of PRESENT with input and output differences of Hamming weight up to 4 is 7.56×10^{-7}. This confirms that these large submatrices are considerably sparse. Therefore, using a sparse matrix storage format where we only allocate storage for the nonzero elements, our large correlation submatrix could easily fit in memory. The very general and simple format for storing sparse matrices is called Compressed Column Storage (CCS). Using this format, the storage cost of a sparse matrix depends on the number of its nonzero elements (nnz) and its column size ($ncol$). More specifically, the cost of a real-valued sparse matrix in CCS format is equivalent to the cost of nnz real-valued numbers and $(nnz+ncol+1)$ integers [27]. Thus, on a 64-bit machine, where we have 8 bytes for both real and integer numbers, the total memory cost would be $8nnz + 8(nnz + ncol + 1)$ bytes. Using the numbers on Table 1, we see that the total memory cost for a CCS sparse representation of PRESENT and SPONGENT difference submatrices is 11014024 ($\approx 2^{23.4}$) and 29085768 ($\approx 2^{24.8}$) bytes respectively. Also the memory cost for a CCS representation of PRESENT and SPONGENT correlation submatrices is 165990920 ($\approx 2^{27.3}$) and 226974888 ($\approx 2^{27.8}$) bytes respectively. Each of these submatrices costs less than 1 GB and thus would easily fit in memory.

5.1 Application on SPONGENT

Here we find linear approximations that can be used to distinguish 23 rounds of the SPONGENT-88 permutation using the whole code book and this is one more round than what has been provided in [4]. We also give the maximum differential characteristic probability we found on 16-round SPONGENT-88.

Differential Approximations: We constructed a difference transition submatrix for SPONGENT-88 with input and output differences having Hamming weight at most 4 bits. The maximum differential probability obtained by powering the transition submatrix[2] is a 16-round differential and it has probability $2^{-77.83}$. One of the differentials having this probability is $e_1 \oplus e_4 \oplus e_{17} \oplus e_{20} \to e_9 \oplus e_{33} \oplus e_{75} \oplus e_{77}$ and it consists of only one differential trail. This is one round less than the best differential provided by the designers as their differential include characteristics with differences having Hamming weight more than 4. It would be interesting to see whether input and output differences with Hamming weight at most 5 bits would yield better estimations. However, as noted in Table 1 the time complexity is 2^{36} arithmetic operations which have not tried due to the lack of computing resources.

Linear Approximations: The SPONGENT Sbox was chosen carefully to avoid the many linear trails with one active Sbox in each round existing on PRESENT [25]. For instance in SPONGENT-88, there is only one trail that have one active Sbox at each round, which makes a linear distinguisher possible for not more than 22 rounds. Now we use a correlation submatrix with input and output masks of Hamming weight up to 4 to activate at most 4 Sboxes. As a result,

[2] This is possible for the difference submatrices of PRESENT and SPONGENT but not for their correlation matrices as they are dense.

we found many linear approximations with correlations larger than 2^{-44} for 23-round of SPONGENT-88. Thus, we improved the linear distinguishers provided by the designers one more round. Table 2 shows the correlations obtained along with the corresponding number of trails written between parentheses. The table shows that correlations obtained from using correlation matrices with masks of Hamming weight at most 2 bits, 3 bits and 4 bits do not vary significantly and this might indicate that linear characteristics covering more than 4 active Sboxes per round do not have a significant effect in the total correlation. The table also shows that it is difficult to accurately estimate the total correlation of some linear approximations. For instance for 22 rounds, $|C^{\leq 2}(e_{70} \oplus e_{71}, e_{56} \oplus e_{78})|$ is smaller than $|C^{\leq 3}(e_{70} \oplus e_{71}, e_{56} \oplus e_{78})|$ but bigger than $|C^{\leq 4}(e_{70} \oplus e_{71}, e_{56} \oplus e_{78})|$. This suggests that the characteristics with Hamming weight 4 bits contributed negatively to the total correlation.

Table 2. $r \equiv$ number of rounds, $\alpha \equiv$ input mask, $\beta \equiv$ output mask, $|C^{\leq i}(\alpha, \beta)| \equiv$ correlation using a submatrix with input and output masks with Hamming weight at most i bits, $- \equiv$ not applicable. $e_i \equiv$ the unit vector with single 1 at position i whose length is 88 (SPONGENT-88's block size). The values between parentheses represent the $\log_2$ of the corresponding number of trails which are easily calculated by replacing each nonzero entry with 1 in the correlation submatrix and then powering it to r.

r	α	β	$\log_2(\lvert C^{\leq 2}(\alpha,\beta)\rvert)$	$\log_2(\lvert C^{\leq 3}(\alpha,\beta)\rvert)$	$\log_2(\lvert C^{\leq 4}(\alpha,\beta)\rvert)$
22	$e_6 \oplus e_7$	$e_3 \oplus e_{25} \oplus e_{47}$	-	-43.82 (20.52)	-43.83 (36.04)
23	$e_6 \oplus e_7$	$e_3 \oplus e_{25} \oplus e_{47}$	-	-43.81 (21.91)	-43.74 (38.44)
22	$e_6 \oplus e_7$	$e_3 \oplus e_{25} \oplus e_{47} \oplus e_{69}$	-	-	-43.83 (36.17)
23	$e_6 \oplus e_7$	$e_3 \oplus e_{25} \oplus e_{47} \oplus e_{69}$	-	-	-43.75 (38.56)
22	$e_{70} \oplus e_{71}$	$e_7 \oplus e_{51}$	-42.06 (7.67)	-42.05 (22.75)	-42.05 (38.25)
23	$e_{70} \oplus e_{71}$	$e_7 \oplus e_{51}$	-44.02 (7.95)	-44.01 (24.13)	-43.95 (40.65)
22	$e_{70} \oplus e_{71}$	$e_{56} \oplus e_{78}$	-42.03 (7.12)	-42.03 (22.51)	-42.04 (38.29)
23	$e_{70} \oplus e_{71}$	$e_{56} \oplus e_{78}$	-43.99 (6.94)	-43.99 (23.89)	-43.96 (40.69)
22	$e_{70} \oplus e_{71}$	$e_7 \oplus e_{29} \oplus e_{51} \oplus e_{73}$	-	-	-42.04 (37.76)
23	$e_{70} \oplus e_{71}$	$e_7 \oplus e_{29} \oplus e_{51} \oplus e_{73}$	-	-	-43.94 (40.16)
22	$e_9 \oplus e_{10} \oplus e_{11}$	$e_3 \oplus e_{25} \oplus e_{47}$	-	-43.99 (19.73)	-43.93 (35.35)
23	$e_9 \oplus e_{10} \oplus e_{11}$	$e_3 \oplus e_{25} \oplus e_{47}$	-	-43.97 (21.14)	-43.88 (37.75)
22	$e_{46} \oplus e_{47} \oplus e_{48}$	$e_{34} \oplus e_{56} \oplus e_{78}$	-	-42.72 (22.49)	-42.69 (39.23)
23	$e_{46} \oplus e_{47} \oplus e_{48}$	$e_{34} \oplus e_{56} \oplus e_{78}$	-	-43.95 (23.86)	-43.89 (41.62)

5.2 Application on PRESENT

The PRESENT block cipher has been analyzed in several publications. In [29], a 16-round differential attack was mounted. Later a statistical saturation attack was mounted and claimed to break 24 rounds [10]. In [25], the author showed the existence of 32% of weak keys which have a higher bias that makes the cipher using those weak keys distinguishable for up to 24 rounds. In [9], the multidimensional attack was used to break 25 rounds and also 26 rounds where the latter use the whole code book. Recently, a multiple differential attack was mounted on 18-round of PRESENT [2]. Our focus here is to use the approach described above to find better linear and differential approximations.

Differential Approximations: We use a difference transition submatrix whose input and output differences have Hamming weight less than or equal to four. Now, in order to estimate the differential probability after r rounds, we raise our transition submatrix to r and extract the maximum entry. The time and memory costs of this are negligible. We found that the 2-round iterative characteristic in [5] has probability 2^{-74} for 15-round PRESENT but the differential containing this characteristic has a higher probability equivalent to $2^{-63.50}$ for a 15-round PRESENT. We also found many differentials with probability larger than 2^{-64} for 16 rounds PRESENT where the maximum one occurs with probability $2^{-62.58}$. Moreover, the maximum differential probability we found for 25 rounds is equal to $2^{-97.38}$. This is larger than the 2^{-100} differential characteristic bound for 25 rounds given in [5]. Here we note that the analysis provided in [5] is sound as the authors gave a bound for the differential characteristic and not for the differential which is hard to bound. Nevertheless, this shows that our approach can be useful in bounding the probability of a differential.

Linear Approximations and Statistical Saturation Attacks: All the previous linear attacks on PRESENT used linear trails activating only one Sbox at each round. To find better linear approximations, we considered trails activating at most 4 Sboxes. Thus, we constructed a correlation submatrix using input and output masks of Hamming weight at most 4 bits. By searching for the best approximations among input masks and output masks in one Sbox. We found that there are many approximations whose squared correlation is larger than 2^{-64} when ≤ 24 rounds of PRESENT are used. As noted in [25], these approximations follow the normal distribution with mean zero and variance equal to their squared correlation. Thus, the squared correlation is higher for 32% of keys compared to the whole key space for some approximations where each approximation has a different path. Thus when using multiple linear approximations, each key is more likely to yield a high correlation with respect to some input and output masks [17]. Therefore statistical saturation distinguishers based on linear approximations whose squared correlations are larger than 2^{-64} work exactly as predicted for almost all the keys.

Table 4 described in the Appendix lists the 10 approximations spanned from U_{11} and V_1 and also the 10 approximations spanned from U_{11} and V_3. All these approximations have a squared correlation larger than 2^{-64} and they give us two 24-round statistical saturation distinguishers. Using the input subspace $U_{11} = span\{e_{41}, e_{42}, e_{43}, e_{44}\}$ which corresponds to fixing the 4 bits entering the 11-th Sbox (counting from left to right) and the output subspace $V_1 = span\{e_1, e_{17}, e_{33}, e_{49}\}$ which corresponds to the 4 bits resulted after applying the permutation on the output of the first Sbox, we get a statistical saturation distinguisher on 24 rounds with an average capacity equal to $2^{-60.53}$. Using the same input subspace with another different output subspace $V_3 = span\{e_3, e_{19}, e_{35}, e_{51}\}$, we also get a statistical saturation distinguisher with an average capacity $2^{-60.53}$. Using another input subspace $U_{10} = \{e_{37}, e_{38}, e_{39}, e_{40}\}$

with each of the above two output subspaces we get distinguishers with the same capacities. Now all these 24-round statistical saturation distinguishers can be used to mount a key recovery attack for 16 bits of the last round key on 25 rounds of PRESENT using the whole code book.

Moreover, these 24-round distinguishers could be used to mount a 26-round key recovery attack similar to [9] to recover 16 bits from the 1st round subkey (4 bits from each of the 9th, 10th, 11th and 12th Sbox) and also 16 bits from last round subkey (4 bits from each of the 1st, 5th, 9th and 13th Sbox) but still estimating the success probability and data complexity is difficult. However, the statistical framework developed in [16] in order to estimate the success probability and data complexity of the multidimensional attack could also be used to estimate the success probability and data complexity of this attack, should we assume the independence of the linear approximations used which is not true. This is in fact what has been done in [9] as it has been noted in [17] that the linear approximations used in the 26-round multidimensional attack of PRESENT can not be statistically independent as several approximations share the same input mask.

Therefore, rather than giving the success probability and data complexity, we list in Table 3 the estimated squared Euclidean distance of the statistical saturation distinguisher with input subspace U_{11} and output subspace V_1 for various number of rounds along with the experimental Euclidean distance using 100 random master keys.

Table 3. The table shows the estimated Euclidean distance D together with the experimental Euclidean distance D' averaged over 100 random keys with various amount of plaintexts, namely 2^{10} plaintexts are used for $r = 2, 3$, 2^{12} for $r = 4$, 2^{17} for $r = 5, 6$, and 2^{20} for $r = 7, 8$. $D'_* \equiv$ the Euclidean distance for a wrong key guess.

r	2	3	4	5	6	7	8	23	24
$\log_2(D)$	$-\infty$	-8.00	-9.99	-12.81	-15.23	-17.79	-20.37	-55.52	-64.53
$\log_2(D')$	-10.07	-7.70	-9.67	-12.74	-14.32	-17.09	-19.06	-	-
$\log_2(D'_*)$	-7.69	-9.03	-11.38	-14.34	-16.19	-19.49	-19.92	-	-

Table 3 the Euclidean distance obtained via a wrong key guess which was simulated by encrypting one more round under the right key. The table also shows clearly that the experimental Euclidean distances are close to the estimated capacities and the more plaintext we use the closer our experimental distances get to the expected distances. Thus, using the above mentioned four statistical distinguishers we could find 16 key bits from each of the first and last round keys using the whole code book. We note that these statistical saturation distinguishers are better than the distinguisher reported in the original attack [10] whose input and output subspaces are $U = V = span\{e_{22}, e_{23}, e_{26}, e_{27}, e_{38}, e_{39}, e_{42}, e_{43}\}$. This is because all the linear approximations spanned from U and V do not have a single linear approximation with a squared correlation larger than 2^{-64} even when considering input and output masks with Hamming weight at most 4 bits.

6 Conclusion and Future Work

In this paper, we used sparse difference and correlation submatrices to estimate the probabilities of low-weight differential and linear approximations respectively in PRESENT-like ciphers. This estimation approach can also be used in any cipher allowing low-weight differential and linear characteristics. Using these sparse matrices, we found linear distinguishers for 23-round of SPONGENT-88. While this is far from distinguishing the full 45 rounds of SPONGENT-88, it is the best currently known result against SPONGENT. We also presented four 24-round statistical saturation distinguishers which break 26-round of PRESENT and that is more than the rounds attacked by the original statistical saturation attack [10].

It would be interesting to investigate whether using large difference and correlation submatrices for PRESENT and SPONGENT-88 with entries having Hamming weight at most 5 would make some improvements over this work. Looking at Table 1 we see that the time complexities for constructing these submatrices take around 2^{34} and 2^{36} arithmetic operations for PRESENT and SPONGENT-88 respectively which could be feasible using parallel computing.

Acknowledgements. For His uncountable blessings, unlimited thanks are to ALLAH that are suitable for His majesty and His perfect attributes. Many thanks go to Lars Knudsen, Gregor Leander and the anonymous reviewers for many useful comments.

References

1. Aumasson, J.-P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A lightweight hash. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 1–15. Springer, Heidelberg (2010)
2. Blondeau, C., Gérard, B.: Multiple differential cryptanalysis: Theory and practice. In: Joux (ed.) [19], pp. 35–54
3. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: Spongent: The design space of lightweight cryptographic hashing. IEEE Transactions on Computers PP(99), 1 (2012)
4. Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: spongent: A lightweight hash function. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 312–325. Springer, Heidelberg (2011)
5. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
6. Borghoff, J., Knudsen, L.R., Leander, G., Thomsen, S.S.: Cryptanalysis of present-like ciphers with secret s-boxes. In: Joux (ed.) [19], pp. 270–289
7. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — A family of small and efficient hardware-oriented block ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
8. De Cannière, C., Preneel, B.: Trivium. In: Robshaw, Billet (eds.) [26], pp. 244–266

9. Cho, J.Y.: Linear cryptanalysis of reduced-round present. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 302–317. Springer, Heidelberg (2010)
10. Collard, B., Standaert, F.-X.: A statistical saturation attack against the block cipher PRESENT. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 195–210. Springer, Heidelberg (2009)
11. Daemen, J., Govaerts, R., Vandewalle, J.: Correlation matrices. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 275–285. Springer, Heidelberg (1995)
12. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer (2002)
13. Daemen, J., Rijmen, V.: Probability distributions of correlation and differentials in block ciphers. IACR Cryptology ePrint Archive, 2005:212 (2005)
14. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Heidelberg (2011)
15. Hell, M., Johansson, T., Maximov, A., Meier, W.: The grain family of stream ciphers. In: Robshaw, Billet (eds.) [26], pp. 179–190
16. Hermelin, M., Cho, J.Y., Nyberg, K.: Multidimensional extension of matsui's algorithm 2. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 209–227. Springer, Heidelberg (2009)
17. Hermelin, M., Nyberg, K.: Linear cryptanalysis using multiple linear approximations. Cryptology ePrint Archive, Report 2011/093 (2011)
18. ISO/IEC 29192-2:2012. Information technology Security techniques Lightweight cryptography. Part 2: Block ciphers (2012)
19. Joux, A. (ed.): FSE 2011. LNCS, vol. 6733. Springer, Heidelberg (2011)
20. Lai, X., Massey, J.L.: Markov ciphers and differential cryptanalysis. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 17–38. Springer, Heidelberg (1991)
21. Leander, G.: On linear hulls, statistical saturation attacks, present and a cryptanalysis of puffin. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 303–322. Springer, Heidelberg (2011)
22. Matsui, M.: On correlation between the order of s-boxes and the strength of des. In: Santis (ed.) [28], pp. 366–375
23. Nyberg, K.: Linear approximation of block ciphers. In: Santis (ed.) [28], pp. 439–444
24. O'Connor, L., Golić, J.D.: A unified markov approach to differential and linear cryptanalysis. In: Pieprzyk, J., Safavi-Naini, R. (eds.) ASIACRYPT 1994. LNCS, vol. 917, pp. 387–397. Springer, Heidelberg (1995)
25. Ohkuma, K.: Weak keys of reduced-round PRESENT for linear cryptanalysis. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 249–265. Springer, Heidelberg (2009)
26. Robshaw, M., Billet, O. (eds.): New Stream Cipher Designs. LNCS, vol. 4986. Springer, Heidelberg (2008)
27. Saad, Y.: SPARSKIT: A basic tool kit for sparse matrix computation. Research Institute for Advanced Computer Science, NASA Ames Research Center (1990)
28. De Santis, A. (ed.): EUROCRYPT 1994. LNCS, vol. 950. Springer, Heidelberg (1995)
29. Wang, M.: Differential cryptanalysis of reduced-round PRESENT. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 40–49. Springer, Heidelberg (2008)

A Appendix

Algorithm 1. Finding the best the average squared correlation or absolute correlation

```
Require: Submatrix $M$ of size $l \times l$, $M$ is a submatrix of the average squared correlation matrix (or of the correlation matrix).
Require: Two temporary vectors of length "$l$", tempCorr and tempIndex.
Ensure: Finds the best average squared correlation (absolute correlation) with its corresponding input mask $a$ and output mask $b$.
 1: $counter = 0$.
 2: for $j = 1 \to l$ do
 3:     Extract the $j$th Column $C_j$ from $M$.
 4:     repeat
 5:         $C_j = M \times C_j$
 6:         Increment $counter$
 7:     until $counter$ equals $r - 1$.
 8:     tempCorr($j$)=max($|C_j|$).
 9:     tempIndex($j$) = The index of max($|C_j|$) gives us the corresponding input mask.
10: end for
11: return max(tempCorr) which yields the maximum average squared correlation (absolute correlation) and its index yields the corresponding output mask $b$. Then the corresponding input mask $a$ = max(tempIndex($b$)).
```

Table 4. $(C^{\leq 4}(\alpha,\beta))^2 \equiv$ squared correlation of a 24-round PRESENT linear approximation with input mask α and output mask β computed via a correlation submatrix with Hamming weight at most 4. The first 10 approximations correspond to the output subspace V_1 while the second 10 approximations correspond to the output subspace V_3. $e_i \equiv$ the unit vector with single 1 at position i whose length is 64 (PRESENT's block size). The values between parentheses represent the $\log_2$ of the corresponding number of trails.

α	β	$\log_2((C^{\leq 4}(\alpha,\beta))^2)$
$e_{41} \oplus e_{43}$	$e_1 \oplus e_{17} \oplus e_{33}$	-63.98 (91.67)
$e_{41} \oplus e_{43}$	$e_1 \oplus e_{33} \oplus e_{49}$	-63.77 (90.62)
$e_{41} \oplus e_{43}$	$e_1 \oplus e_{17} \oplus e_{33} + e_{49}$	-63.97 (91.48)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_1 \oplus e_{17} \oplus e_{33}$	-63.80 (91.00)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_1 \oplus e_{17} \oplus e_{49}$	-63.97 (91.12)
$e_{41} \oplus e_{42} \oplus e_{44}$	$e_1 \oplus e_{17} \oplus e_{33}$	-63.97 (91.52)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_1 \oplus e_{33} \oplus e_{49}$	-63.60 (89.95)
$e_{41} \oplus e_{42} \oplus e_{44}$	$e_1 \oplus e_{33} \oplus e_{49}$	-63.77 (90.47)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_1 \oplus e_{17} \oplus e_{33} \oplus e_{49}$	-63.80 (91.48)
$e_{41} \oplus e_{42} \oplus e_{44}$	$e_1 \oplus e_{17} \oplus e_{33} \oplus e_{49}$	-63.96 (91.33)
$e_{41} \oplus e_{43}$	$e_3 \oplus e_{19} \oplus e_{35}$	-63.98 (91.66)
$e_{41} \oplus e_{43}$	$e_3 \oplus e_{35} \oplus e_{51}$	-63.78 (90.62)
$e_{41} \oplus e_{43}$	$e_3 \oplus e_{19} \oplus e_{35} \oplus e_{51}$	-63.97 (91.48)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_3 \oplus e_{19} \oplus e_{35}$	-63.81 (91.00)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_3 \oplus e_{19} \oplus e_{51}$	-63.97 (91.11)
$e_{41} \oplus e_{42} \oplus e_{44}$	$e_3 \oplus e_{19} \oplus e_{35}$	-63.97 (91.51)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_3 \oplus e_{35} \oplus e_{51}$	-63.60 (89.95)
$e_{41} \oplus e_{42} \oplus e_{44}$	$e_3 \oplus e_{35} \oplus e_{51}$	-63.77 (90.47)
$e_{41} \oplus e_{42} \oplus e_{43}$	$e_3 \oplus e_{19} \oplus e_{35} \oplus e_{51}$	-63.80 (90.81)
$e_{41} \oplus e_{42} \oplus e_{44}$	$e_3 \oplus e_{19} \oplus e_{35} \oplus e_{51}$	-63.97 (91.33)

Security Evaluation of Cryptographic Modules against Profiling Attacks

Yongdae Kim[1], Naofumi Homma[2], Takafumi Aoki[2], and Heebong Choi[1]

[1] The Attached Institute of Electronics and Telecommunications Research Institute
{kimyd,gold}@ensec.re.kr
[2] Graduate School of Information Sciences, Tohoku University
homma@aoki.ecei.tohoku.ac.jp, aoki@ecei.tohoku.ac.jp

Abstract. Recently, profiling attacks have been attracting a great deal of attention because of their increasing efficiency. Further investigations are required to determine the potential threats of the profiling attacks. This paper focuses on these attacks. Using hardware and software implementations, we provide a security evaluation of three different types of profiling attacks: template attack, stochastic model attack, and multivariate regression attack. Our experimental results show that multivariate regression attack outperforms other attacks in terms of profiling efficiency and key extraction rates.

Keywords: profiling attack, multivariate regression analysis, template attack, stochastic model attack, power analysis attack.

1 Introduction

Cryptographic algorithms are implemented in various forms: hardware, software, firmware or sometimes in a combination of various forms. These forms are called cryptographic modules. It was believed that cryptographic modules were secure because the underlying cryptographic algorithms are theoretically unbreakable. For this reason, security evaluations were restricted to the algorithm level.

However, a new category of cryptanalysis, power analysis attack has been introduced by P. Kocher, et al. in 1999 [1]. Many cryptographic researchers have begun to investigate not only cryptographic algorithms, but also their concrete implementations.

One of the most efficient power analysis attacks is called the profiling attack, which employs reference modules that have the same characteristics as those of the target module [2], [3]. There are three methods in class of profiling attacks: the template attack [4], the stochastic model attack [5] and multivariate regression attack [6].

Testing methods for cryptographic modules have been developed in many countries under the Cryptographic Module Validation Program (CMVP). However, the methodologies for conducting security evaluation with resistance to side-channel attacks are still under discussion. Federal Information Processing Standard (FIPS) 140-2 is one security requirement for cryptographic modules in

USA and Canada: however, it does not contain concrete metrics for side-channel attacks. The current version of FIPS 140-2 deals with side-channel attacks as mitigations of other attacks. Therefore, the testing methods relative to side-channel attacks will be specified in the new version of the standard, FIPS 140-3. However, it is more difficult to standardize testing methods for profiling attacks than it is for other conventional side-channel attacks, such as correlation power analysis attacks [7], differential power analysis attacks [1], etc.

Because profiling attacks have several issues to be considered: (i) the selection of points that contain data-dependent variations, and (ii) the number of traces for the profiling. These two parameters have significant impacts on profiling attacks. Sometimes they lead to a decrease of performance, especially when there are a limited number of available traces. This can cause unreliable security evaluation of cryptographic modules.

To clear up these two issues, we perform security evaluation on the hardware and software implementations of the Advanced Encryption Standard (AES) to strengthen the security level of the cryptographic modules. In addition to that, we also demonstrate the effect of hiding countermeasures on software implementation. Our experimental results indicate that we can use the multivariate regression attack for an accurate and reliable security evaluation method to test the hardware and software of cryptographic modules.

2 Profiling Attacks

Compared to correlation power analysis attacks [7], profiling attacks require a far lower amount of side-channel information to retrieve the secret key, since such attacks take advantage of prior information from the profiling phase. The basic idea of this technique is to approximate the noise model rather than to reduce or eliminate noise.

Profiling attacks are known to consist of two phases: (i) profiling phase, and (ii) key extraction phase. Each phase uses different modules that have identical physical characteristics. In the profiling phase, an adversary captures physical leakage from a reference module. By analyzing that information, a property of the signal and noise can be characterized. Next, in the key extraction phase, maximum-likelihood estimation is used to determine the correct key using the information built in the profiling phase.

2.1 Template Attack

Profiling Phase. The profiling phase collects a large number of waveforms with different data d_i and key k_j, given as

$$d_i \in \{d_1, d_2, \cdots, d_D\}, \tag{1}$$

$$k_j \in \{k_1, k_2, \cdots, k_K\}, \tag{2}$$

where D and K denote the number of possible pieces of data and keys, respectively. Then, we group together the traces that correspond to the pair of (d_i, k_j), and estimate a mean vector, $\boldsymbol{m}$, and a covariance matrix, $\boldsymbol{C}$, of the multivariate normal distribution.

However, for example in case of 128-bit AES, the number of possible data and keys is $2^8 = 256$. Thus, in total, $256^2 = 65536$ templates are required. It is unrealistic to generate templates corresponding to all possible pairs of keys and data.

Therefore, in practice, templates are generated based on hypothetical power consumption for each pair of (d_i, k_j), and are written as h_{d_i,k_j}. Hence, the number of templates can be reduced. In the case of AES, we need to build only 9 templates corresponding to 9 possible Hamming distance or Hamming weight values, which are dependent on the method of implementation.

Finally, templates $(\boldsymbol{m}, \boldsymbol{C})_{h_{d_i,k_j}}$ that correspond to all possible hypothetical power consumption values are built in this phase.

Thus, the characteristic of W sampling points power consumption trace $\boldsymbol{w} = (w_1, \cdots, w_W)$ can be described as the probability density function of the multivariate normal distribution as follows :

$$\boldsymbol{q} = \boldsymbol{w} - \boldsymbol{m}, \tag{3}$$

$$p(\boldsymbol{w}; (\boldsymbol{m}, \boldsymbol{C})_{h_{d_i,k_j}}) = \frac{exp\left(-\frac{1}{2}\boldsymbol{q}^T\boldsymbol{C}^{-1}\boldsymbol{q}\right)}{\sqrt{(2\pi)^W det(\boldsymbol{C})}}, \tag{4}$$

where $det(\boldsymbol{C})$ and $\boldsymbol{q}^T$ denote the determinant of $\boldsymbol{C}$ and the transpose of vector $\boldsymbol{q}$.

Key Extraction Phase. When a power consumption trace is given, the probability $p(k_j \mid \boldsymbol{w})$ for $j = 1, \cdots, K$ is written as follows using Bayes' theorem.

$$p(k_j \mid \boldsymbol{w}) = \frac{p(\boldsymbol{w} \mid k_j)p(k_j)}{\sum_{l=1}^{K}(p(\boldsymbol{w} \mid k_l)p(k_l))} \tag{5}$$

Note that $p(k_j) = 1/K$, since all possible keys are uniformly distributed. Given a trace, $\boldsymbol{w}$, Eq. (5) indicates a probability when the correct key is equal to k_j.

The original template attack only provides a key extraction strategy based on a single available trace. However, it is difficult in practice to retrieve the correct key using only a trace. Thus, we use the following formula for given D traces: $\boldsymbol{w}_i (i = 1, \cdots, D)$.

$$p(k_j \mid \boldsymbol{w}_{1,\cdots,D}) = \frac{\left(\prod_{i=1}^{D} p(\boldsymbol{w}_i \mid k_j)\right) p(k_j)}{\sum_{l=1}^{K}\left(\left(\prod_{i=1}^{D} p(\boldsymbol{w}_i \mid k_l)\right) p(k_l)\right)} \tag{6}$$

In Eq. (6), $p(\boldsymbol{w}_i \mid k_j)$ is set to $p(\boldsymbol{w}_i; (\boldsymbol{m}, \boldsymbol{C})_{h_{d_i,k_j}})$, which is obtained in the profiling phase in Eq. (4).

Finally, we estimate the correct key k_{ck} using the maximum likelihood estimation with the probability density function, Eq. (6) as follows :

$$k_{ck} = \underset{k_j \in k^*}{\text{argmax}}\, p(k_j \mid \boldsymbol{w}_{1,\cdots,D}), \tag{7}$$

where k^* is the set of all possible key candidates.

2.2 Stochastic Model Attack

In 2005, the stochastic model attack was introduced by W. Schindler, et al. [5]. The fundamental idea of this attack is very similar to that of the template attack. However, the stochastic model attack uses the key-independent noise model instead of the usage noise model associated to all possible key candidates in template attacks.

Profiling Phase. In stochastic model attack, a power trace at time t ($t = 1, \cdots, W$) is represented as,

$$I_t(d_i, k) = h_t(d_i, k) + r_t, \tag{8}$$

with the i-th input d_i and a correct key (which is, however, known to an adversary) k. And $h_t(d_i, k)$ denotes the deterministic part of the trace depending on d_i and k. On the other hand, r_t denotes a random part independent of d_i and k. The profiling phase is divided into two steps in order to approximate the two discrete terms.

In the first step, the deterministic part is profiled using N_1 traces from a reference module. The deterministic part is approximated by a linear combination of u-dimensional vector subspace spanned by the u known function $g_{j,t}$,

$$\hat{h}_t(d_i, k) = \sum_{j=0}^{u-1} \beta_{j,t} \cdot g_{j,t}(d_i, k), \tag{9}$$

where the coefficients $\beta_{0,t}, \cdots, \beta_{u-1,t}$ are estimated value for each instant t. In order to estimate the coefficients $\boldsymbol{\beta}_t = (\beta_{0,t}, \cdots, \beta_{u-1,t})$, the function $g_{j,t}$ is firstly determined in the u-dimensional subspace. For example, in the case of AES, a 9 dimensional subspace is usually chosen as the function $g_{j,t}$, which leads to the best approximation [5]. An adversary generates a matrix, $\boldsymbol{A}$ using N_1 traces corresponding to input d_i and key k as follows:

$$\boldsymbol{A} = \begin{bmatrix} g_{0,t}(d_1, k) & \cdots & g_{u-1,t}(d_1, k) \\ g_{0,t}(d_2, k) & \cdots & g_{u-1,t}(d_2, k) \\ \vdots & & \vdots \\ g_{0,t}(d_{N_1}, k) & \cdots & g_{u-1,t}(d_{N_1}, k) \end{bmatrix} \tag{10}$$

The estimated coefficients

$$\boldsymbol{\beta}_t = (\beta_{0,t}, \beta_{0,1}, \cdots, \beta_{u-1,t}), \tag{11}$$

are then denoted using the least square method,

$$\boldsymbol{\beta}_t = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{w}_t, \tag{12}$$

where the vector $\boldsymbol{w}_t = (w_{1,t}, w_{2,t}, \cdots, w_{N_1,t})$ represents power consumption for each instant t.

After having determined the approximators $\hat{h}_t(d_i, k)$, different set of N_2 traces are used to profile the random part. We first calculate the W-dimensional random vector $\boldsymbol{r} = (r_1, r_2, \cdots, r_W)$ as follows:

$$r_t = I_t(d_i, k) - \hat{h}_t(d_i, k), \tag{13}$$

We assume that the random vector is normally distributed with a covariance matrix $\boldsymbol{C}$. Therefore, $\boldsymbol{C} = (c_{i,j})_{1\le i,j\le W}$ is computed as follows:

$$\begin{aligned} c_{i,j} &= E(r_i r_j) - E(r_i)E(r_j) \quad (14)\\ &= E(r_i r_j) \quad (15)\end{aligned}$$

where $E(X)$ denotes the expected value of the variable X. Finally, we have approximated the deterministic part, $\hat{h}_t(d_i, k)$ and the noise model represented as the covariance, $\boldsymbol{C}$ in this phase.

Key Extraction Phase. In this phase, traces from a target module are analyzed using the model that have been obtained in the profiling phase. We assume that N_3 traces are captured from the target module corresponding to $d_i \in \{d_1, d_2, \cdots, d_{N_3}\}$ and that there is a correct key, k_{ck} (unknown to an adversary). A noise vector $\boldsymbol{z}_i$ is first computed as follows:

$$\boldsymbol{z}_i = I_t(d_i, k_{ck}) - \hat{h}_t(d_i, k_j), \tag{16}$$

where $k_j \in \{k_1, k_2, \cdots, k_K\}$. The noise vector follows a multivariate normal distribution with the profiled covariance matrix $\boldsymbol{C}$ when $j = ck$. So, we can estimate the correct key by computing the following probabilities:

$$p\left(\boldsymbol{z}_i; \hat{h}_t(d_i, k_j)\right) = \frac{exp\left(-\frac{1}{2}\boldsymbol{z}_i^T\boldsymbol{C}^{-1}\boldsymbol{z}_i\right)}{\sqrt{(2\pi)^W det(\boldsymbol{C})}}. \tag{17}$$

The maximum likelihood estimation is applied to determine a correct key k_{ck} using N_3 traces as follows:

$$k_{ck} = \underset{k_j\in k^*}{\operatorname{argmax}}\, \Pi_{i=1}^{N3} p\left(\boldsymbol{z}_i; \hat{h}_t(d_i, k_j)\right). \tag{18}$$

We can simplify the Eq. (17) by applying the logarithm.

$$\begin{aligned} &ln\left(p\left(\boldsymbol{z}_i; \hat{h}_t(d_i, k_j)\right)\right)\\ &= -\frac{1}{2}\boldsymbol{z}_i^T\boldsymbol{C}^{-1}\boldsymbol{z}_i - \frac{1}{2}ln\left((2\pi)^W det(\boldsymbol{C})\right). \quad (19)\end{aligned}$$

The second term $-\frac{1}{2}ln((2\pi)^W det(\boldsymbol{C}))$ in Eq. (19) is constant value. Therefore it can be eliminated. As a result, the estimator can be simplified as a follows:

$$\boldsymbol{z}_i^T \boldsymbol{C}^{-1} \boldsymbol{z}_i. \tag{20}$$

Note that an adversary decides the correct key that minimized the sum of Eq. (20) as follows:

$$k_{ck} = \underset{k_j \in k^*}{\operatorname{argmin}} \sum_{i=1}^{N_3} \boldsymbol{z}_i^T \boldsymbol{C}^{-1} \boldsymbol{z}_i. \tag{21}$$

2.3 Multivariate Regression Attack

This type of profiling attack can improve profiling efficiency using multivariate regression analysis. Even if an adversary can utilize several traces for profiling, the adverse effects for the key extraction can be minimized.

Profiling Phase. In the profiling phase, we need to determine explanatory and response variables to build a multivariate regression model.

Define Response Variable. We consider the response variable as the sum of all hypothetical power consumption of the components (i.e. S-Boxes). Assuming that M components are processed in parallel, we write the l-th component (hypothetical power consumption) as $h^l_{i,ck}$, given by the i-th input ($1 \le i \le N$, N is a number of inputs) and the correct key k_{ck}. We referred the sum of each hypothetical power consumption to s_i, which is defined

$$s_i = \sum_{l=1}^{M} h^l_{i,ck}. \tag{22}$$

We defined the value of s_i as a response variable in the regression model. Note that the value s_i is feasible only if the correct keys are known to the adversary. It is possible that the adversary will use a reference module under full control.

Define Explanatory Variables. First of all, we calculate a squared Pearson correlation coefficient vector, $\boldsymbol{\rho}'_B = (\rho^2_{B,1}, \cdots, \rho^2_{B,W})$ between $\boldsymbol{w}_i$ and s_i considering both negative and positive correlation. If the squared coefficient is high at the t-th time instant, it is usually assumed that the time instant is highly related to the response value, s_i. Thus, the adversary select $P(< W)$ instans with the highest value of the squared correlation coefficient, and referred to as a vector $\boldsymbol{p} = (p_1, p_2, \cdots, p_P)$. Each time instant is sorted in descending order of the value of squared correlation. This means that the squared Pearson correlation have the highest value at time instant, p_1. Now, we select P time instants from $w_{i,t}$ $(1 \le t \le W)$ corresponding to the interesting points. We define the the explanatory variables as followings:

$$w_{i,p_1}, w_{i,p_2}, \cdots, w_{i,p_P} \ (1 \le i \le N_1), \tag{23}$$

where N_1 is the number of inputs for the profiling phase.

Multivariate Regression Model. Next, we can calculate the estimator of regression coefficients. Finally, the following multivariate regression model can be obtained using the regression coefficients in the profiling phase, $(1 \leq i \leq N_1)$

$$\hat{s_i} = \hat{\beta}_0 + \hat{\beta}_1 w_{i,p_1} + \hat{\beta}_2 w_{i,p_2} + \cdots + \hat{\beta}_P w_{i,p_P}, \tag{24}$$

where $\hat{s_i}$ stands for the fitted value using the model. Finally, we obtain the regression coefficients in this phase corresponding to its interesting points.

Key Extraction Phase. In this phase, an adversary measures power traces from a target cryptographic module corresponding to N_2 inputs (known) and secret key k_{ck} (unknown). We utilize the regression model in Eq. (24) to estimate the sum of hypothetical power consumption, $\hat{s_i}$ $(1 \leq i \leq N_2)$ given by the measured traces. h^l_{i,k_j} denotes the hypothetical power consumption of the l-th component (S-Box) associated with each key candidates $k_j \in \{k_1, k_2, \cdots, k_K\}$ and the i-th input. The correct key of the l-th component can be estimated using the Pearson correlation coefficient between $\hat{s_i}$ and h^l_{i,k_j} as follows:

$$k^l_{ck} = \underset{k_j \in \{k_1, k_2, \cdots, k_K\}}{\text{argmax}} \; corr(\hat{s_i}, h^l_{i,k_j}). \tag{25}$$

3 Experimental Analysis

For the hardware implementation, the side-channel attack standard evaluation board (SASEBO) incorporating cryptographic FPGA and ASIC were used for both the target and the reference module. We used an 8-bit AVR microcontroller from the Atmel Corporation for software implementation. In addition, we implemented a hiding countermeasure on the software implementation to investigate of effectiveness of the countermeasure. Ten delays before our target operation (i.e. SubBytes) are inserted. Each delay is composed of several dummy operations. The number of operations is uniformly generated between 0 and 8. Therefore the longest possible delay time is 80 clock cycles.

3.1 Evaluation on Hardware Implementations

We used 15 interesting points and 20,000 traces for profiling for both FPGA and ASIC implementations. In our analysis, the number of available power traces for key extraction is 10,000 and 20,000 for FPGA and ASIC, respectively.

The result of all subkey estimation results are illustrated in Fig. 1. The horizontal axis indicates the number of traces for key extraction: the vertical axis shows the classification rate in percentage computed as follows:

$$R_i = \frac{N^{ck}_i}{16} \times 100, \tag{26}$$

where R_i and N^{ck}_i denote the classification rate and the number of correctly estimated keys using i traces, respectively.

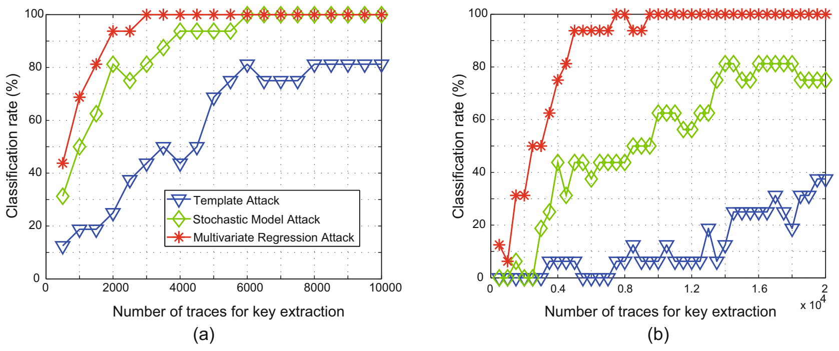


Fig. 1. Classification rate (a) FPGA, and (b) ASIC implementation

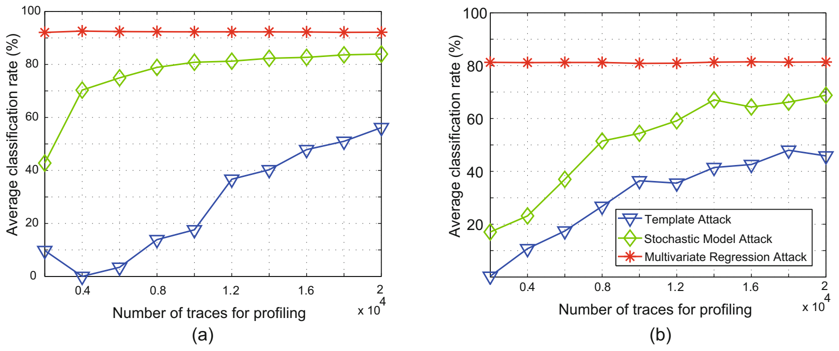


Fig. 2. Average classification rate of FPGA and ASIC implementations associated with the number of traces for profiling (15 interesting points): (a) FPGA implementation (b) ASIC implementation

As can be seen in the Fig. 1, the classification rate is increasing as the number of traces for key extraction increases. However, the multivariate regression attack has the least number of traces to estimate all the subkeys of AES in FPGA and ASIC implementations. In other words, a straightforward AES implementation on FPGA and ASIC is very vulnerable to multivariate regression attack.

In order to examine the effects of the number of traces for profiling and interesting points, we first define an average value of classification rates, $\bar{R}$ as follows:

$$\bar{R} = \frac{1}{D} \sum_{i=1}^{D} R_i, \tag{27}$$

where D represents the number of total available traces for key extraction. In our case, D is 10,000 and 20,000 for FPGA and ASIC, respectively.

Figure 2 shows the average values of classification rates of FPGA and ASIC implementations: the horizontal axis is the number of traces used in the profiling

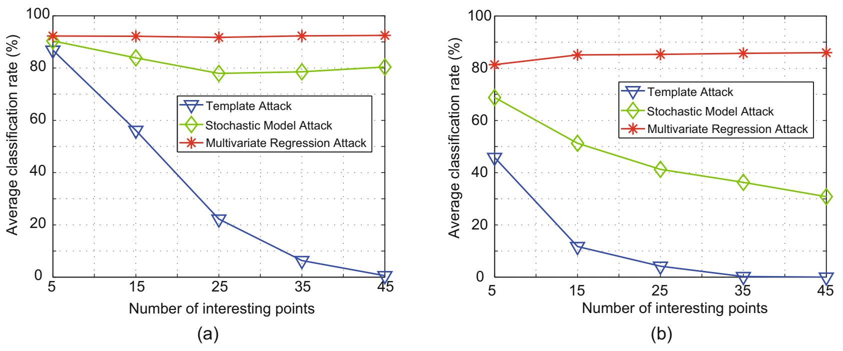


Fig. 3. Average classification rate of FPGA and ASIC implementations associated with the number of interesting points (20,000 traces for profiling): (a) FPGA implementation, and (b) ASIC implementation

phase and the number of interesting points is fixed at 15 points. We do not focus on the selection method of interesting points. However, we do only investigate selection method impact on the classification rate, and thus we assumed that an adversary had already obtained a fixed number of interesting points in the profiling phase.

Figure 3 shows the average classification rates of FPGA and ASIC implementations associated with the number of interesting points: where the number of traces for profiling is 20,000 for both implementations. The results indicate the low average values of the classification rates for the template attack and stochastic model attack as the number of interesting points is increased. The interesting points cover the data dependent time instants. Sometimes, when an adversary selects non-data dependent points as interesting points, the classification rate is negatively affected by these points.

However, it can be clearly observed that the classification rates using multivariate regression attack does not decrease even when the number of interesting points is increased comparing to other profiling attacks. This is because the less data-dependent time instants have a less significant effect on the regression model.

To observe the significance of each regression coefficient, we intentionally selected 20 irrelevant time instants adding them to the 15 selected interesting points, as shown in Fig. 4. The results show squared values of regression coefficients corresponding to the 35 selected points. This clearly confirms that the time instants corresponding to irrelevant points have relatively small values of regression coefficients. This leads to a small impact on the response variable. Thus, a profiling attack using the multivariate regression model takes less effort to determine the time instants as interesting points in order to extract all keys successfully.

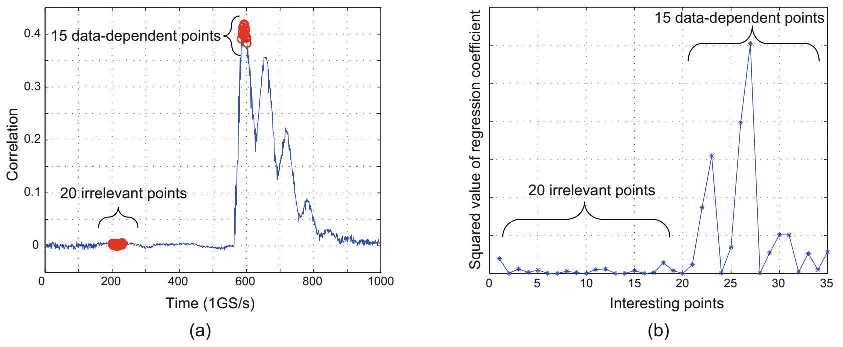


Fig. 4. (a) correlation peaks and interesting points, and (b) squared value of regression coefficient for FPGA implementations

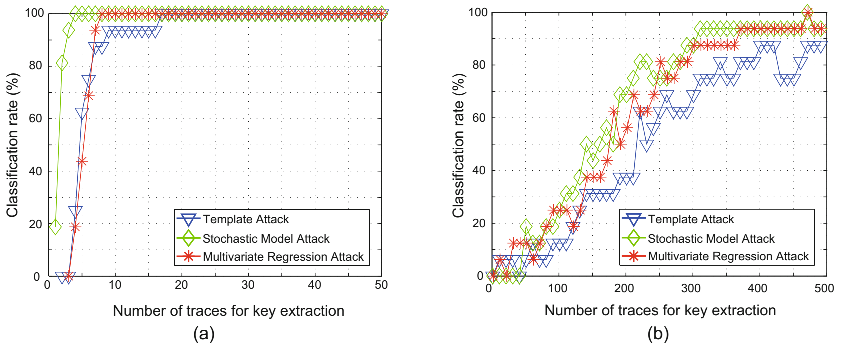


Fig. 5. Classification rate (a) unprotected smart card, and (b) hiding countermeasure implemented smart card

3.2 Evaluation on Software Implementations

Compared to the case of hardware implementations, in software implementations all S-boxes are executed sequentially. Hence we determined different 15 different interesting points for each target S-boxes.

In Fig. 5, we show the classification rates associated with the number of traces for key extraction for an unprotected and hiding countermeasure implemented smart card. For those results in Fig. 5, we used 15 interesting points for both types of implementations: 2,000 and 5,000 traces are used for the profiling phase on the unprotected and hiding countermeasure implemented smart cards, respectively. For the unprotected smart card, it is enough to extract all subkeys using fewer than 20 traces. As can be seen in Fig. 5 (b), even when we used 10 times more power traces than those used with the unprotected smart card, it was impossible to retrieve all the secret keys successfully. This result shows that hiding countermeasure is effective but not perfect method to hinder attacks. Actually, the classification rate increases as the number of traces increases.

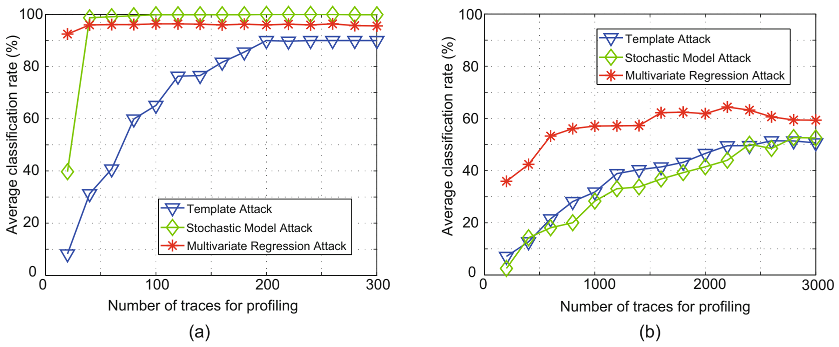


Fig. 6. Average classification rate associated with the number of traces for profiling (15 interesting points): (a) unprotected smart card, and (b) hiding countermeasure implemented smart card

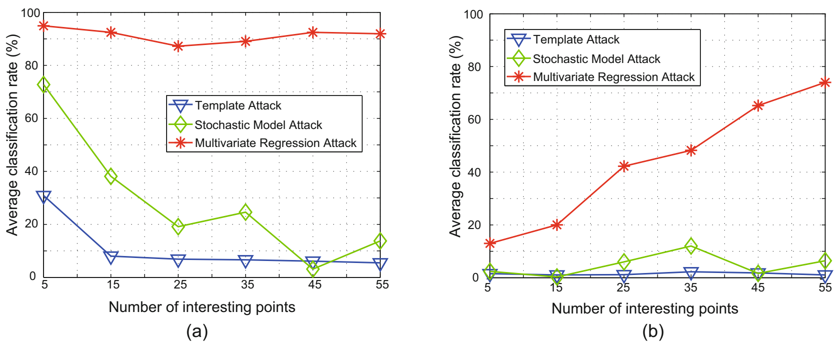


Fig. 7. Average classification rate associated with the number of interesting points: (a) unprotected smart card (20 traces for profiling), and (b) hiding countermeasure implemented smart card (100 traces for profiling)

First, we examine a tendency between the number of traces for profiling and the classification rates. The experimental procedure is the same as that of the hardware implementations experiment. Figure 6 shows the results of this experiment. These results clearly indicate the visible tendencies between the classification rates and the number of traces available for profiling. In addition, the multivariate regression attack has a higher profiling efficiency than that of other profiling attacks for both smart cards. As we already demonstrated experimental results for hardware implementations, we have found that profiling strategies have almost the same performance when a certain number of traces are available in software implementations.

Next, we focus on the efficiency of the profiling phase with the number of interesting points. Figure 7 shows the average classification rate associated with the number of interesting points. Note that we used exactly the same interesting points for each number of traces for profiling. As can be seen in Fig. 7, with respect to hiding countermeasure implementation, the average classification rates show a high improvement as more interesting points are provided. This is because the number of relevant time instants are increased with hiding countermeasures.

4 Conclusion

This paper presented a security evaluation of cryptographic modules against profiling attacks. The profiling attack is one of the side-channel attacks that most effectively expose weaknesses and secret information of cryptographic modules using their physical leakages. However, profiling attacks require a large number of traces to characterize of the power consumption and relevant time instants correctly. The multivariate regression attack is able to compensate for those two issues.

Our evaluation results of hardware and software implementations have shown that multivariate regression attacks, pose a serious threat to the security level of cryptographic modules. The results indicate that we need to consider multivariate regression attacks for a proper security evaluation of profiling attacks, because the attack can perform successfully using a small number of traces in the profiling phase. In addition, such attacks are robust to selection methods of relevant time instants.

Acknowledgements. We would like to gratefully thank Dr. Takeshi Sugawara for his useful support for performing the experiments.

References

1. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
2. Le, T.H., Canovas, C., Clédière, J.: An overview of side channel analysis attacks. In: Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security (ASIACCS), pp. 33–43 (2008)
3. Standaert, F.-X., Gierlichs, B., Verbauwhede, I.: Partition *vs.* Comparison side-channel distinguishers: An empirical evaluation of statistical tests for univariate side-channel attacks against two unprotected CMOS devices. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 253–267. Springer, Heidelberg (2009)
4. Chari, S., Rao, J., Rohatgi, P.: Template attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)
5. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005)
6. Sugawara, T., Homma, N., Aoki, T., Satoh, A.: Profiling attack using multivariate regression analysis. IEICE Electronics Express 7, 1139–1144 (2010)
7. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)

Key-Dependent Weakness of AES-Based Ciphers under Clockwise Collision Distinguisher

Toshiki Nakasone[1,⋆], Yang Li[1], Yu Sasaki[2], Mitsugu Iwamoto[1], Kazuo Ohta[1], and Kazuo Sakiyama[1,⋆]

[1] Dept. of Informatics, The University of Electro-Communications
1-5-1, Chofugaoka, Chofu, Tokyo 182-8585, Japan
{nakasone,liyang,mitsugu,kazuo.ohta,sakiyama}@uec.ac.jp
[2] NTT Secure Platform Laboratories, NTT Corporation
3-9-11 Midori-cho, Musashino-shi, Tokyo 180-8585 Japan
sasaki.yu@lab.ntt.co.jp

Abstract. In 2011, Li *et al.* proposed a series of side-channel attacks that are related to a fundamental side-channel leakage source called clockwise collision. This paper discloses the fact that hardware implementations of AES-based ciphers could have weak keys assuming that the leakage of clockwise collision is distinguishable. In order to explain this, we firstly set up an evaluation method by introducing a threshold-based distinguisher that takes an advantage of the locality of Electro-Magnetic (EM) measurements. Secondly, we discuss that the probability of clockwise collision depends on the key values and the byte positions in the AES states. Thirdly, based on practical EM measurements and mathematical analysis, we quantitatively evaluate the relationship between the probability of clockwise collision and the vulnerability to the side-channel attack. Finally, the discussion is extended to the design methodology of AES-based ciphers, *i.e.*, the parameter selection for S-box and ShiftRows.

Keywords: Side-channel attack, Electromagnetic analysis, Clockwise collision, Weak key, AES-based cipher.

1 Introduction

In recent years, many kinds of Side-Channel Attacks (SCAs) have been introduced [1–7]. The vulnerability to the SCA threatens the security of practical cryptographic implementations. The SCA utilizes physical information, *e.g.*, ElectroMagnetic (EM) radiation and power consumption [6–10].

The SCA with a hypothetical leakage model such as Hamming Distance (HD) model is one of the common approaches to exploit the intermediate value from the physical leakage [1, 5]. As another approach, the template-based SCA was proposed by Chari, Rao, and Rohatgi in 2002 [11]. The template-based SCA

⋆ A part of this research was funded by Ministry of Economy, Trade, and Industry (METI), Japan.

supposes that a template is created with a target device where the parameter such as secret key and input value can be changed by an attacker. By comparing the template with actual measurements, the template-based SCA retrieves the secrets. On the other hand, as one of the SCAs without the template, the collision-based SCA was proposed, which was used against DES firstly in [12], and then applied to AES in [13, 14]. It detects the internal collision and retrieves the intermediate values by exploiting the similarity in the side-channel leakage. As an enhancement of the collision-based analysis, correlation collision analysis was proposed in [15, 16].

In this paper, we explore a new side-channel leakage based on Clockwise Collision (CC) as a further enhancement of collision-based analysis. CC is a fundamental but overlooked internal collision, which was introduced by Li *et al.* [17]. More precisely, when input values for two consecutive cycles collide, we see that most of the calculations of the current cycle are already finished. In fact, Li *et al.* verified that power consumption at the current cycle is quite low when CC happens. Nevertheless, power-based side-channel distinguisher is affected not only by the target byte operation, but also by other ones. Accordingly, it is difficult to distinguish whether or not CC occurs byte by byte from power traces, since we only observe the *total* power consumption of all byte-oriented operations. In this regards, it is a great challenge to ensure that CC happens in a byte operation in the parallelized implementation.

ElectroMagnetic Analysis (EMA) can retrieve a position-specific side-channel leakage, *i.e.*, locality of the EM leakage, by appropriately setting the location of an EM probe [18–20]. Due to such locality, we expect that byte-wise CC can be detected efficiently even for a parallelized hardware implementation. After detecting byte-wise CC, different from previously proposed collision-based analyses, once we can detect byte-wise CC for the EM leakage, the CC leakage can be connected to the secret key directly based on the hypothetical HD model, *i.e.*, HD equals to 0. Hence, it is worth focusing on EMA to exploit the CC leakage from the parallel AES, and discussing how precisely CC can be detected among the data about the EM intensity (the EM traces for short).

In this paper, the AES implementation where 16 byte-oriented operations are executed in parallel is denoted as the parallel AES. Since the probability of CC is dependent on the byte operations in the AES algorithm, we also review the mechanism of how often CC occurs in the parallel AES.

1.1 Our Contribution

Our contribution is fourfold. Firstly, we establish a new evaluation method for CC, which is the collision-based distinguisher that does not require the profiling on a known-key device. The proposed distinguisher utilizes a threshold value of side-channel leakage, *i.e.*, the EM leakage, to distinguish the measured EM traces obtained by an EM probe. Due to the locality of the EM leakage, our evaluation expects that the impact of byte-wise CC can be observed without guessing the states in other bytes. Secondly, we find the key-dependent weakness for AESciphers from the side-channel perspective. We show that, for a specific byte position of the parallel AES, the probability of CC is imbalanced depending on the key value. Thirdly, in order to evaluate the SCA vulnerability due to such imbalance, we quantify the key strength based on statistics from the EM measurements where we approximate the probability density distributions of the EM intensity. We show that we can calculate the relationship between the success probability of detecting CCs and the required amount of measurements. At last, we discuss a better design methodology for ShiftRows and S-box in AES-based ciphers for avoiding the imbalance since such imbalance of key strength is usually considered as weakness for ciphers.

1.2 Paper Organization

The rest of this paper is organized as follows. Section 2 explains the locality of EM leakage and CC. Section 3 shows the mechanism of Clockwise Collision-based EMA (CC-EMA) in detail. In Sect. 4, a new evaluation method is discussed for CC. We demonstrate the experimental result of quantifying the key strength in Sect. 5. Section 6 contains a discussion of the feedback to cipher design, and conclusions are given in Sect. 7.

2 Preliminary

2.1 Clockwise Collision (CC)

Considering the phenomenon of CC, Li *et al.* proposed an attack scenario for an AES implementation with power analysis and fault analysis [17]. When CC occurs, the combinational circuit has no signal transitions because the signal state of the combinational circuit after the previous cycle has already been the state that should be achieved in the current cycle. Therefore, a combinational circuit for HD $= 0$ leads to a much lower power consumption than HD $\neq 0$ for the loop architecture. In contrast, the conventional HD model [5] approximates the power consumption to be proportional to the HD of the register.

2.2 Locality of EM Leakage

The EM leakage varies from the measurement points on a circuit layout, which is inherently different from the power leakage that is measured as total power consumption. The most advantageous point of the EM leakage is that the position-dependent information leakage is available, *i.e., locality of EM leakage* [18–20].

Suppose that an EM probe is able to collect the information leakage only for a certain operation, *e.g.*, a byte operation. Considering that side-channel attacks are based on an input guess of partial operation such as AES S-box, secret information is retrieved byte by byte. In the parallel AES, the locality of EM leakage is especially meaningful for attackers since the leakage derived from a different location significantly reduces the signal to noise ratio [18, 20]. Therefore, EMA can be an analytic tool with the byte-wise locality, and is useful when the suitable measurement position is known.

3 Overview of CC-EMA

The followings are the list of notations used in this paper.

h :	threshold value for separating the EM traces
t :	the collision number of subkey candidates
$T_{\mathrm{HD}=a}$:	event that the EM traces are obtained when a target 8-bit register has HD $= a$ in two consecutive clock cycles, where $a = 0, 1, \ldots, 8$
$T_{x \leq h}$:	event that the EM traces are obtained when a target EM intensity, x, is equal to or less than h
μ_a :	mean of the EM traces of $T_{\mathrm{HD}=a}$
σ_a :	standard deviation of the EM traces of $T_{\mathrm{HD}=a}$
μ :	mean of the EM traces of $T_{\mathrm{HD}\neq 0}$
σ :	standard deviation of the EM traces of $T_{\mathrm{HD}\neq 0}$
N_{cc} :	the number of CCs in the no-shift case when a subkey is fixed

3.1 Target Implementation

This paper focuses on the parallel AES based on the loop architecture. In order to evaluate CC precisely, we use the parallel AES with a composite field S-box that has no countermeasure against SCA. The ciphertext is calculated by a repeated round function, which is usually synchronized with a clock signal. Our work also supposes that AES encrypts unknown plaintexts, while ciphertexts are public. When we provide a random plaintext to AES hardware, the input value for the S-box is considered to be uniformly distributed. In the following discussion, we aim to detect CC in the target byte at the AES last round.

CC and AES S-box. Due to the structure of AES, the mechanism of generating CC is classified into two cases depending on the target-byte positions as illustrated in Fig. 1.

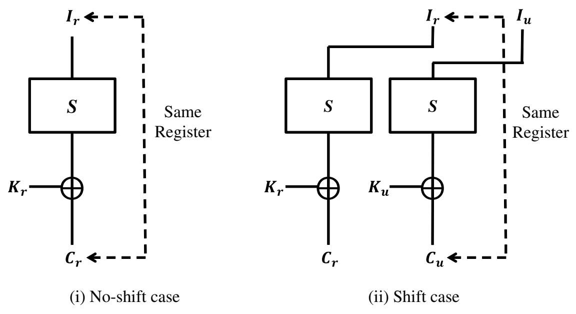


Fig. 1. Two cases of AES last round

Algorithm 1. An example algorithm of CC-EMA against the last round of AES

Input: EM leakages x_i, the target bytes in ciphertexts C_r^i and C_u^i, the threshold h, and the necessary collision number of key candidates t, where i is the number of measurements.

Output: The target subkey, K_r

```
1: Generate a zero array $KeyList[k]$ for counting candidates of $k$;
2: $i = 1$;
3: while $\max(KeyList[k]) < t$ do
4:     if $x_i \leq h$ then
5:         $k \leftarrow S(C_u^i) \oplus C_r^i$;
6:         $KeyList[k] \leftarrow KeyList[k] + 1$;
7:     end if
8:     $i \leftarrow i + 1$;
9: end while
10: Return $K_r \leftarrow \text{argmax}_k\ KeyList[k]$
```

(i) We focus on the byte position where the value in the register is used to compute the r^{th} byte in a round, and after the round transformation, the register stores the results of the r^{th} byte. In this case, the values of the register for two consecutive cycles, I_r and C_r, have the relation $C_r = S(I_r) \oplus K_r$, where S is the AES S-box transformation. In order that CC happens, two values need to be $C_r = I_r$, and hence we know that the probability of CC depends on the property of the S-box and the key value. Hereafter, the case of (i) is called *no-shift case.*
(ii) We focus on the byte position where the value in the register is used to compute the r^{th} byte in a round, and after the round transformation, this register stores the results of the u^{th} byte, where $u \neq r$. In this case, the values of the register for two consecutive cycles, I_r and C_u, do not have any relation. To obtain CC, the value of C_u needs to be the same as I_r. Because C_u and I_r are independently determined, the probability of CC is 1/256. The case of (ii) is called *shift case.*

In the no-shift case, the difference of the input and output, $I_r \oplus C_r$ could be 0 more often than that in the shift case. When an attacker recovers the byte-wise subkey with CC, the key having a high probability of CC is regarded weaker than the shift case. Therefore, the key-strength against CC distinguisher depends on the key value in the no-shift case. The key with $\Pr[I_r = C_r] > 1/256$ is called weak key in this paper, and the detailed information is shown in Appendix A.

3.2 Evaluation Method for CC

The locality of EM leakage aims to detect CC in the target byte at the last round. Therefore, we locate the EM probe so that the side-channel behavior of a single 8-bit AES S-box at the last round can be observed. When CC occurs, the value of K_r, which is constant, can be derived as $K_r = S(C_r) \oplus C_r$ for the no-shift case, and $K_r = S(C_u) \oplus C_r$ for the shift case. That is to say, by using K_r and a ciphertext, we can judge whether or not CC occurs.

Here, we establish an evaluation method for CC in order to construct a simple but efficient distinguisher. Algorithm 1 represents the detailed procedure of CC-EMA in shift case. Note that the same algorithm can be used for the no-shift case assuming that $C_r = C_u$. For steps 4 to 7, when the observed EM intensity is lower than the threshold, we derive the corresponding key candidate k, and count up the frequency. When the $\max(KeyList[k]) \geq t$ where t is the necessary number of key candidates categorized by a derived subkey, the algorithm is stopped and outputs K_r. By taking k that leads to the highest number in $KeyList[k]$, we expect that the corresponding k is likely to be the correct one that was embedded in the hardware. If this expectation is true, it can be said that we could successfully distinguish the EM traces with CC from others. As long as CC has accounted for more than random selection by step 4, theoretically detecting the targeted subkey is feasible.

4 Side-Channel Distinguisher for CC

However, in the no-shift case, the probability of CC is divisible into 5 groups according to the property of the S-box and the key value (See Table. 3 in Appendix A). The key strength for 5 groups are different, which is shown in Sect. 5.1 in detail. For simplicity, we deal with the shift case where S-box uses different registers for storing input and output values in this section. In this case, the probability distribution is determined as

$$P_a = \Pr\big[T_{\mathrm{HD}=a}\big] = \frac{\binom{8}{a}}{256}. \tag{1}$$

Suppose that the EM traces are in accordance with a normal distribution described by the conditional probability density function of

$$f_a(x) = \frac{1}{\sqrt{2\pi\sigma_a^2}} \exp\left(-\frac{(x-\mu_a)^2}{2\sigma_a^2}\right), \tag{2}$$

where μ_a and σ_a are the mean and standard deviation for the case of HD $= a$, respectively. In the conventional HD model, we assume that measured the EM intensity is roughly proportional to the value of HD. This model is very general and well-fit to the various implementations. In fact, a lot of implementations can be broken by using the HD model.

In contrast, we focus on the fact that the EM intensity for HD $= 0$ is much lower than the cases for HD $\neq 0$ considering the nature of CC. Hence, it is expected that the EM measurements corresponding to HD $= 0$ can be efficiently separated among all EM measurements by setting an appropriate threshold. Therefore, we only use the EM traces whose intensity are below the threshold h.

Here, we define the success probability $P_s(t, h)$ when subkey candidates collide t times. For instance, for $t = 1$, as we have only one EM trace for CC and the success probability of $P_s(1, h)$ can be derived as

$$P_s(1,h) = \Pr\big[T_{\mathrm{HD}=0} | T_{x\leq h}\big] = \frac{\Pr\big[T_{\mathrm{HD}=0} \cap T_{x\leq h}\big]}{\Pr\big[T_{x\leq h}\big]} = \frac{S_0}{\sum_{a=0}^{8} S_a}, \tag{3}$$

where

$$S_a = P_a \int_{-\infty}^{h} f_a(x)dx. \tag{4}$$

If we assume that $\mu = \mu_1 = \cdots = \mu_8$, $\sigma = \sigma_1 = \cdots = \sigma_8$, and $\mu_0 < \mu_n$, Eq. (3) becomes

$$\begin{aligned} P_s(1,h) &= \frac{P_0 \cdot \frac{1}{2}\Big(1 + \mathrm{erf}\big(\frac{h-\mu_0}{\sqrt{2\sigma^2}}\big)\Big)}{P_0 \cdot \frac{1}{2}\Big(1 + \mathrm{erf}\big(\frac{h-\mu_0}{\sqrt{2\sigma^2}}\big)\Big) + \big(\sum_{a=1}^{8} P_a\big) \cdot \frac{1}{2}\Big(1 + \mathrm{erf}\big(\frac{h-\mu_n}{\sqrt{2\sigma^2}}\big)\Big)} \\ &= \frac{1 + \mathrm{erf}\big(\frac{h-\mu_0}{\sqrt{2\sigma^2}}\big)}{256 + \mathrm{erf}\big(\frac{h-\mu_0}{\sqrt{2\sigma^2}}\big) + 255 \cdot \mathrm{erf}\big(\frac{h-\mu_n}{\sqrt{2\sigma^2}}\big)}. \end{aligned} \tag{5}$$

Intuitively, we have a higher probability of $P_s(1,h)$ as decreasing the value of h for $\mu_0 < \mu_n$. However, the total number of EM measurements will increase in order to satisfy the condition that the intensity of the EM traces is below h. The amount of EM measurements $M(t,h)$ that we need to perform is

$$M(t,h) = \frac{t}{\Pr\big[T_{\mathrm{HD}=0} \cap T_{x \le h}\big]}. \tag{6}$$

Using the total number of the EM traces $M(t,h)$, we collect the EM leakages satisfying $x \le h$. We derive the key candidate with corresponding ciphertexts, *i.e.*, $k = S(C_u) \oplus C_r$, and count up the key list until a key candidate collides t times.

As for $t \ge 2$, only if the wrong key candidates collide less than t with the number of the EM traces, we can identify the correct candidate. Hence, in order to derive success probability $P_s(t,h)$, we need the probability that the wrong key candidates collide t times, *i.e.*, the multi-collision probability.

To the best of our knowledge, Suzuki *et al.*'s work on the multi-collision probability in [21] is the most strict. However, they use a recursive formula that cannot lead to deriving the probability for arbitrary parameters. Accordingly, we decide to use Monte Carlo simulation. From the simulation results, we determine parameters that minimize the number of side-channel measurements.

5 Experiments with CC-EMA

In this section, CC-EMA is demonstrated based on an actual experiment using Side-channel Attack Standard Evaluation BOard (SASEBO) [22]. We derive the success probability to recover the key value utilizing means and standard deviations of EM measurements. Finally, we quantify the key strength by comparing the success probabilities in shift case and no-shift case.

Table 1 summarizes the setting of our experimental environment. The AES circuit is implemented on the cryptographic FPGA mounted on SASEBO-G. The S-box of AES_Comp [23] is without SCA countermeasures and based on the composite field calculation. In order to verify the concept, we set up the best position for collecting the CC leakage from a target byte. In the following sections, the experimental results will be shown against the AES implementation.

Table 1. Environment for experiments

Target FPGA	Xilinx Virtex-II Pro Series XC2VP7-5FG456C
Clock frequency	8MHz
Oscilloscope	Agilent DSO7032A
EM probe	MORITA-TECH WM7000 Probe Series HC020
Amplifier	MORITA-TECH YCK1000AMP

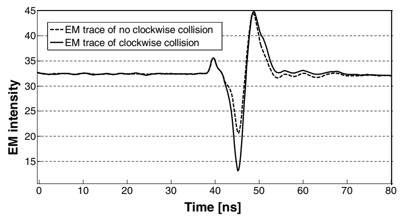


Fig. 2. The EM traces at the last round of AES

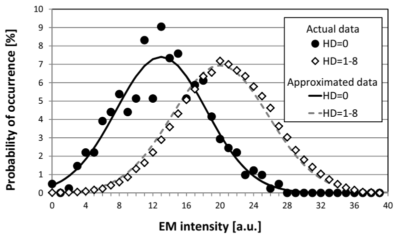


Fig. 3. Conditional probability distributions of the actual EM intensity and its approximation

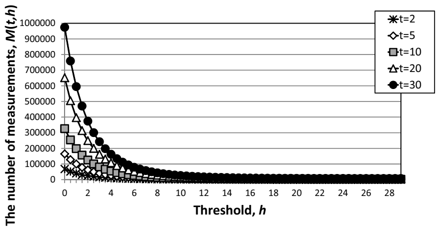


Fig. 4. The total number of measurements $M(t, h)$,, for threshold h when $\mu_0 = 13, \mu = 20, \sigma_0^2 = 29$, and $\sigma^2 = 33$

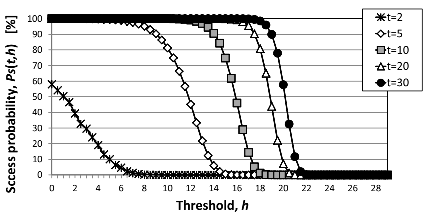


Fig. 5. The success probability $P_s(t, h)$, for threshold h when $\mu_0 = 13, \mu = 20, \sigma_0^2 = 29$, and $\sigma^2 = 33$

5.1 Success Probability Using Experimental Data

Based on an actual experiment, we obtain the EM traces as shown in Fig. 2. We observe the EM intensity x. Table 2 shows that means and standard deviations of the EM intensities in each HD, which are calculated using 100 000 traces. According to the values, we have $\mu_0 = 13, \mu = 20, \sigma_0^2 = 29$, and $\sigma^2 = 33$. Both the conditional probability distribution of HD = 0 and the HD $\neq$ 0 are illustrated in Fig. 3. Using the means and the standard deviations, we approximate conditional probability density functions that are also described in Fig. 3. With the proposed distinguisher and the approximated probability density functions, we represent the probability curve for $P_s(t, h)$ and the number of necessary EM measurements $M(t, h)$, with respect to each threshold in Figs. 4 and 5.

Notice that $M(t, h)$ is calculated with Eq. (6), and the result of $P_s(t, h)$ is based on Monte Carlo simulation, when $t \geq 2$. We can find that the success

Table 2. Means and standard deviations of EM intensity data for HD = a

a	0	1	2	3	4	5	6	7	8
μ_a	12.9	19.4	19.9	20.5	20.7	21.1	20.7	20.7	21.4
σ_a^2	28.8	33.6	34.4	32.6	32.4	32.5	33.1	32.7	32.2

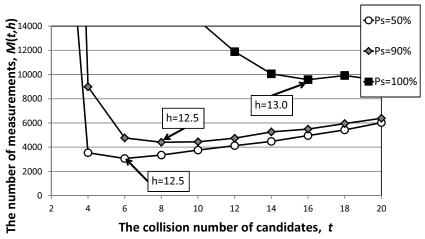


Fig. 6. Relationship between $M(t, h)$ and t in shift case

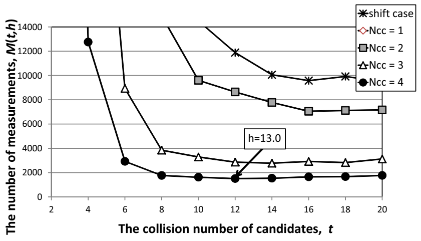


Fig. 7. Relationship $M(t, h)$ and t when the success probability $P_s(t, h) = 100\%$[2]

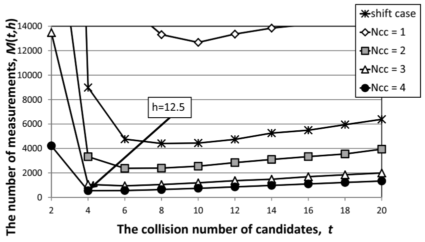


Fig. 8. Relationship $M(t, h)$ and t when the success probability $P_s(t, h) = 90\%$

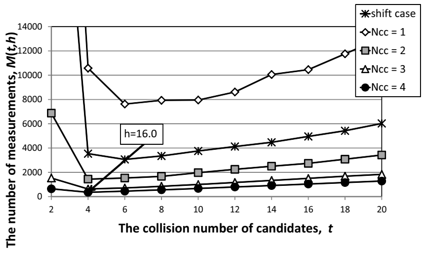


Fig. 9. Relationship $M(t, h)$ and t when the success probability $P_s(t, h) = 50\%$

probability is increased when the value of h is decreased. However, $M(t, h)$ is also increased.

To check more details, we plot the relationship between $M(t, h)$ and t in Fig. 6. We verify the minimum number of measurements $M(t, h)$ for identifying the correct key. When we need the success probability to be more than 90% in our environment, it is possible to identify the key with EM measurements of $M(8, 12.5) = 4431$[1]. From the results, we know that the optimal t exists, because for the small t, *e.g.*, $t = 2$, the correct the EM traces with CC might be mistaken for the wrong ones.

As described in Sect. 3.1, the AES algorithm has two cases *i.e.*, the shift and no-shift cases. In the no-shift case, the probability of CC is imbalanced, and it depends on the key value in the AES last round. Therefore, the no-shift case is divided into 5 different groups depending on the probability of CC[3]. Here, we investigate the success probability in the no-shift case, *i.e.*, N_{cc}=1,

[1] We have $P_s(8, 12.5) = 91.2\%$.
[2] Note that the case of $N_{cc} = 1$ cannot identify the correct key with $M(t, h) \leq 14000$.
[3] Appendix A explains the probability of CC of each group in detail.

2, 3, or 4, where N_{cc} is the number of CCs. Although the N_{cc}=0 is out of scope of this paper, it will be interest to consider the method to identify the correct key when $N_{cc} = 0$. Figures 6, 7, 8 and 9 show the relationship between $M(t, h)$ and t in both the shift and the no-shift cases. When we need the success probability, $P_s(t, h) \geq 90\%$ for $N_{cc} = 4$, it is possible to identify the key with EM measurements of $M(4, 12.5) = 545$[4]. From the result of Figs. 7, 8 and 9, we can retrieve the weak keys with $N_{cc} > 1$ utilizing the less number of measurements than the measurement in shift case.

As shown in Figs. 6, 7, 8 and 9, the success probability largely depends on the probability of CC. Hence, the AES algorithm exists the imbalance of the key strength from side-channel perspective. In addition, the result explains quantitatively analyzed key strength with the success probability to recover its value.

6 Discussion: Feedback to Future Designs of AES-Based Primitives

Designing new block ciphers and other symmetric-key primitives, *e.g.*, hash functions and MACs, is actively discussed even now. This is because the cryptographic community needs not only ciphers for a generic purpose, but also lightweight ciphers or new hash function standards. To design such new primitives, borrowing the idea of the AES algorithm is very natural due to several reasons.

– The security of AES has already been analyzed very well.
– Many implementation techniques are known for AES. Besides, if designers want to exploit the AES-NI [24], the new algorithm must be based on AES.

Therefore, we still have high demand for designing AES-based primitives. The purpose of the discussion in this section is giving a feedback from our results on AES to future AES-based algorithm designs. Such discussion would be useful. For example, the AES-based hash function Grøstl, which is one of the final-round candidates of the SHA-3 competition, explicitly says that the side-channel analysis against MAC usage of Grøstl can be prevented in the same manner as protecting AES.

As discussed in Sect. 3, AES has several weak keys for the no-shift case. In general, the weak keys can be avoided by changing the implementation. In our case, avoiding the parallel execution of 16 S-boxes can be a countermeasure. However, besides the implementation, it is possible to prevent CC by slightly modifying the algorithm without giving any impact to the performance nor the security against theoretical (non-side-channel) analysis. We stress that we never suggest changing the specification of the AES algorithm, but suggest considering the resistance against CC when new AES-based primitives are designed.

An ideally designed cipher should be secure even if the weakest keys are used. In other words, the security is considered by the weakest key rather than the strongest key. Following this philosophy, cipher's designers should avoid the

[4] We have $P_s(4, 12.5) = 90.7\%$.

structure which yields weak keys in terms of CC. There are two possible directions. Note that achieving only one of them is enough.

1. Design an S-box so that all keys have the same strength, even if the no-shift case occurs.
2. Design diffusion so that the no-shift case never occurs.

6.1 Feedback to S-Box Design Principle

As explained in the no-shift case, the probability of CC depends on the S-box transformation, which is written by

$$\begin{aligned} Pr[I_r = C_r] &= \frac{\#C_r|S(C_r) \oplus K_r = C_r}{256}, && (7)\\ &= \frac{\#C_r|S(C_r) \oplus C_r = K_r}{256}. && (8)\end{aligned}$$

This means that if the value of $S(C_r) \oplus C_r$, which is a difference between an input and the corresponding output of the S-box, is biased and takes a particular value more frequently than others, the corresponding K_r is a weak key. Therefore, to avoid weak keys against CC, the S-box needs to satisfy the following property.

$$\max_y \#\{x|S(x) \oplus x = y\} = 1. \tag{9}$$

Equation (9) is satisfied only when the number of solution is exactly 1 for all y. Therefore the S-box must have a so-called fixed point, which means the value x such that $S(x) \oplus x = 0$. In many cases, the S-box is designed so that it does not have a fixed point. However, with respect to CC, having one fixed point and satisfying Eq. (9) is best. As far as we know, this is the first criteria for the S-box design that focuses on how many kinds of differences are generated between the input and output.

6.2 Feedback to ShiftRows Design Principle

Note that the shift case in Sect. 3.1 does not have weak keys. Hence, another design strategy is avoiding the no-shift case. One of the advantages of this strategy is that previously known S-boxes can still be used. For example, the fixed point can be avoided for the S-box and using the same S-box as AES for running the AES-NI is also possible.

There are two methods to avoid the no-shift case. One is avoiding byte-to-byte operation at the last round, *e.g.*, performing the MixColumns operation at the last round of AES. However, this breaks the symmetry in the encryption and decryption and may lose the advantage for the implementation.

The other method is avoiding storing the S-box output in the same register as the input, *i.e.*, avoiding 0 for the parameter of the ShiftRows operation.

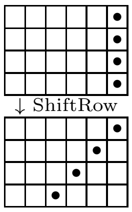


Fig. 10. ShiftRow for Rijndael-192

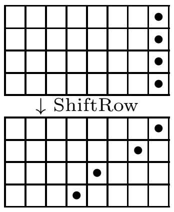


Fig. 11. ShiftRow for Rijndael-256

Let us look the details of the Rijndael cipher as a case study. Rijndael [25] is the cipher proposed by Daemen and Rijmen which is a base of the AES. Different from AES, Rijndael supports larger block sizes; 192-bit and 256-bit blocks. The ShiftRow operations are described in Figs. 10 and 11. According to the specification, the parameters are chosen to resist the (truncated) differential cryptanalysis. It is explicitly mentioned that choosing 0 for the first row is one of the design criteria. However the internal state is rectangle, avoiding parameter 0 is possible. Moreover, avoiding parameter 0 only gives negligible impact to the cipher's performance and cost. As a result, avoiding parameter 0, *e.g.*, parameter (1,2,3,4) for Rijndael-192, is more advantageous to increase the resistance against CC-based analysis. The observation seems useful because most of previously designed AES-based primitives use 0-byte rotation for the ShiftRows parameter. Note that this method cannot be applied to AES-based designs whose row numbers and column numbers are the same. However, also note that the block size of AES-128 is often too small to build hash functions. Hence, making the internal state size rectangle is a natural way to extend the internal state size. For example, Grøstl-512 hash function adopts two AES-based permutations whose internal state size is 8 rows and 16 columns.

7 Conclusions

This paper represented the key-dependent weakness of AES-based ciphers under clockwise collision. Initially, we constructed the new evaluation method for clockwise collision with the threshold-based distinguisher. This paper demonstrated the proposed evaluation method against AES implementation with a 128-bit data path. During the evaluation, we discovered that the probability of clockwise collision was imbalanced, which caused the key-dependent weakness. In order to evaluate the weakness depending on the key value, we quantified the key strength based on practical experimental data and mathematical analysis. Finally, we provided feedback to the design philosophy of S-box and ShiftRows in AES-based ciphers to avoid the imbalance of key strength. As a future work, the clockwise collision distinguisher will be compared with other distinguishers from side-channel perspective.

References

1. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
2. Kocher, P.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
3. Quisquater, J.-J., Samyde, D.: ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In: Attali, S., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001)
4. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)
5. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
6. Messerges, T.S., Dabbish, E.A., Sloan, R.H.: Examining Smart-Card Security under the Threat of Power Analysis Attacks. IEEE Trans. Computers 51(5), 541–552 (2002)
7. Messerges, T.S.: Using Second-Order Power Analysis to Attack DPA Resistant Software. In: Paar, C., Koç, Ç.K. (eds.) CHES 2000. LNCS, vol. 1965, pp. 238–251. Springer, Heidelberg (2000)
8. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM Side-Channel(s). In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 29–45. Springer, Heidelberg (2003)
9. Peeters, E., Standaert, F.-X., Quisquater, J.-J.: Power and Electromagnetic Analysis: Improved Model, Consequences and Comparisons. Integration, the VLSI Journal 40(1), 52–60 (2007)
10. Mateos, E., Gebotys, C.H.: Side Channel Analysis using Giant Magneto-Resistive (GMR) Sensors. In: Proc. of Second International Workshop on Constructive Side-Channel Analysis and Secure Design, COSADE 2011, pp. 42–49 (2011)
11. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)
12. Schramm, K., Wollinger, T., Paar, C.: A New Class of Collision Attacks and Its Application to DES. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 206–222. Springer, Heidelberg (2003)
13. Schramm, K., Leander, G., Felke, P., Paar, C.: A Collision-Attack on AES: Combining Side Channel- and Differential-Attack. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 163–175. Springer, Heidelberg (2004)
14. Bogdanov, A.: Improved Side-Channel Collision Attacks on AES. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 84–95. Springer, Heidelberg (2007)
15. Moradi, A., Mischke, O., Eisenbarth, T.: Correlation-Enhanced Power Analysis Collision Attack. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 125–139. Springer, Heidelberg (2010)
16. Moradi, A.: Statistical Tools Flavor Side-Channel Collision Attacks. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 428–445. Springer, Heidelberg (2012)

17. Li, Y., Nakatsu, D., Li, Q., Ohta, K., Sakiyama, K.: Clockwise Collision Analysis – Overlooked Side-Channel Leakage Inside Your Measurements. Cryptology ePrint Archive, Report 2011/579 (2011), `http://eprint.iacr.org/`
18. Kirschbaum, M., Schmidt, J.-M.: Learning from Electromagnetic Emanations - A Case Study of iMDPL. In: Second International Workshop on Constructive Side-Channel Analysis and Secure Design, COSADE 2011, pp. 50–55 (2011)
19. Dehbaoui, A., Lomne, V., Maurine, P., Torres, L., Robert, M.: Enhancing Electromagnetic Attacks Using Spectral Coherence Based Cartography. In: Becker, J., Johann, M., Reis, R. (eds.) VLSI-SoC 2009. IFIP AICT, vol. 360, pp. 135–155. Springer, Heidelberg (2011)
20. Meynard, O., Réal, D., Flament, F., Guilley, S., Homma, N., Danger, J.-L.: Enhancement of Simple Electro-Magnetic Attacks by Pre-characterization in Frequency Domain and Demodulation Techniques. In: Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2011, pp. 1004–1009. IEEE (2011)
21. Suzuki, K., Tonien, D., Kurosawa, K., Toyota, K.: Birthday paradox for multi-collisions. In: Rhee, M.S., Lee, B. (eds.) ICISC 2006. LNCS, vol. 4296, pp. 29–40. Springer, Heidelberg (2006)
22. National Institute of Advanced Industrial Science and Technology (AIST), Side-channel Attack Standard Evaluation Board (SASEBO), `http://www.rcis.aist.go.jp/special/SASEBO/index-en.html`
23. Tohoku University, Cryptographic Hardware Project: Project Webpage, `http://www.aoki.ecei.tohoku.ac.jp/crypto/`
24. Gueron, S.: Intel's New AES Instructions for Enhanced Performance and Security. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 51–66. Springer, Heidelberg (2009)
25. Daemen, J., Rijmen, V.: AES Proposal: Rijndael (1998)

A Classification of Last-Round Subkey

In Sect. 3, we explain that the probability of CC is different depending on the key value in the no-shift case. Based on our evaluation using the CC distinguisher, the subkeys that have $N_{cc} > 1$ are recovered with the less number of measurements than the subkeys in shift case. Here, Table 3 shows that the AES last-round subkey value of the first row are classified according to the probability of CC. The detailed relationship between the key values and the number of CCs, N_{cc}, are represented in Table 4. Note that, the N_{cc} for each subkey is calculated by $\#\{C_r | S(C_r) \oplus K_r = C_r\}$.

Table 3. Classification of subkey of the first row in AES last round

N_{cc}	#key values	Probability
0	93	36.3
1	91	35.5
2	54	21.1
3	15	5.9
4	3	1.2

Table 4. Relationship between the key value and the number of CCs, N_{cc}

Key value	N_{cc}	Key value	N_{cc}	Key value	N_{cc}	Key value	N_{cc}
00	0	40	1	80	2	c0	0
01	2	41	1	81	0	c1	0
02	1	42	1	82	3	c2	3
03	1	43	0	83	0	c3	2
04	0	44	1	84	0	c4	1
05	1	45	3	85	1	c5	0
06	3	46	0	86	0	c6	1
07	0	47	0	87	1	c7	2
08	1	48	0	88	0	c8	1
09	0	49	1	89	0	c9	1
0a	0	4a	2	8a	0	ca	0
0b	0	4b	1	8b	0	cb	1
0c	1	4c	3	8c	2	cc	0
0d	0	4d	1	8d	4	cd	1
0e	0	4e	2	8e	1	ce	0
0f	0	4f	1	8f	1	cf	2
10	2	50	1	90	3	d0	1
11	2	51	3	91	1	d1	1
12	2	52	1	92	1	d2	1
13	1	53	0	93	2	d3	0
14	2	54	0	94	0	d4	2
15	0	55	1	95	1	d5	0
16	1	56	0	96	0	d6	2
17	0	57	1	97	2	d7	2
18	0	58	0	98	2	d8	1
19	1	59	1	99	0	d9	3
1a	3	5a	2	9a	0	da	3
1b	0	5b	0	9b	0	db	2
1c	3	5c	0	9c	2	dc	1
1d	0	5d	1	9d	0	dd	1
1e	1	5e	0	9e	0	de	2
1f	2	5f	1	9f	1	df	1
20	3	60	2	a0	2	e0	0
21	1	61	1	a1	0	e1	0
22	2	62	2	a2	0	e2	1
23	0	63	2	a3	2	e3	0
24	0	64	0	a4	0	e4	1
25	0	65	1	a5	1	e5	0
26	0	66	0	a6	0	e6	1
27	1	67	1	a7	0	e7	4
28	1	68	2	a8	0	e8	2
29	0	69	1	a9	3	e9	1
2a	0	6a	1	aa	0	ea	0
2b	2	6b	0	ab	0	eb	1
2c	1	6c	2	ac	0	ec	1
2d	1	6d	2	ad	2	ed	2
2e	1	6e	3	ae	1	ee	1
2f	0	6f	1	af	0	ef	2
30	1	70	1	b0	1	f0	2
31	0	71	0	b1	1	f1	1
32	2	72	2	b2	0	f2	2
33	1	73	2	b3	0	f3	0
34	1	74	1	b4	2	f4	2
35	0	75	1	b5	2	f5	1
36	0	76	0	b6	2	f6	2
37	0	77	1	b7	1	f7	1
38	2	78	1	b8	2	f8	2
39	1	79	2	b9	4	f9	0
3a	1	7a	3	ba	1	fa	0
3b	0	7b	1	bb	0	fb	1
3c	2	7c	1	bc	0	fc	2
3d	1	7d	1	bd	0	fd	0
3e	0	7e	0	be	2	fe	1
3f	1	7f	0	bf	1	ff	0

Efficient Group Signatures in the Standard Model*

Laila El Aimani and Olivier Sanders**

Technicolor, Security & Content Protection Labs
1 avenue de Belle Fontaine, 35576 Cesson-Sévigné Cedex, France

Abstract. In a group signature scheme, group members are able to sign on behalf of the group. Since the introduction of this cryptographic authentication mechanism, several schemes have been proposed but only few of them enjoy a security in the standard model. Moreover, those provided in the standard model suffer the recourse to non standard-assumptions, or the expensive cost and bandwidth of the resulting signature.

We provide three practical group signature schemes that are provably secure in the standard model under standard assumptions. The three schemes permit dynamic enrollment of new members while keeping a constant size for both keys and group signatures, and they improve the state-of-the art by several orders of magnitude.

Keywords: Group signature, bilinear groups, standard model, non-interactive zero-knowledge.

1 Introduction

Group signatures, introduced in 1991 by Chaum and Van Heyst [12] allow members of a group to anonymously sign messages on behalf of the whole group. However, to prevent abuses, the group is controlled by a group manager that has the ability to *open* the group signature, *i.e.* to identify the signer of a message. Group signatures have proved to be extremely useful in various applications, for example keycard access to restricted areas, where it is necessary to secure areas to only employees of the group without tracking individual employee's movements.

Related Works. Since their introduction, a great number of security properties that group signatures should meet have been introduced until Bellare, Micciancio and Warinschi [4] provided appropriate definitions and formalized the intuitive informal requirements of previous works. In fact, they proposed two properties

* This is an extended abstract. The full version [14] is available at the Cryptology ePrint Archive.

** Laila El Aimani is now an independent researcher and Olivier Sanders is with Orange Labs.

for static groups, namely *full anonymity* and *full traceability*, that captured all previous requirements. Full anonymity requires that group signatures reveal no information about the signer, even in the presence of a powerful adversary who has access to an opening oracle and to all users secret keys. Full traceability requires that the group manager is always able to identify the signer of a valid group signature or a member of the coalition that issued it. Bellare, Shi and Zhang [5] extended these notions to dynamic groups and added the notion of *non-frameability*, which requires that even a dishonest group manager and a coalition of group members cannot falsely accuse an honest user of having issued some group signature.

Boneh and Shacham [7] proposed a weaker notion of anonymity, called *selfless anonymity*, where signers can trace their own signatures. Further schemes ([8], [9]) introduced another weak version of anonymity, namely the CPA-*anonymity* where the adversary does not have access to an opening oracle. As mentioned in [6], this is a much more serious limitation because the opening functionality becomes virtually useless.

Most practical group signatures schemes ([6], [13]) that have been proposed are proven secure in the random oracle model (ROM). Indeed, despite its inherent flaws ([11], [3]), the ROM compares much better than the standard model with respect to efficiency. Ateniese *et al.* [2] gave an efficient group signature scheme in the standard model but proved its security under non-standard assumptions. Moreover, the security proof of anonymity (lemma A.2 of their paper) does not mention any access to opening oracles by the adversary and thus differs from the standard CCA-anonymity experiment. Actually, this CCA-anonymity seems hard to reach since their group signatures are partially re-randomizable, so any adversary against the selfless anonymity is able to re-randomize the first part of the challenge group signature and make an opening request on the result. In 2007, Groth [15] gave the first efficient realization, achieving full anonymity in the standard model, where the size of group signatures is about 50 elements.

Our Contributions. We propose three groups signatures schemes, proven secure in the standard model under standard assumptions, that improve the size of group signatures and the number of pairings required for the signature verification.

In our constructions, each user joining the group receives a certificate on a key while the group manager receives some information allowing him to open the group signatures produced by this user. To produce a group signature on a message, the user will use his key to sign, with a digital signature scheme, the message, a randomized part of the certificate and the verification key of a strong one-time signature. Then he will produce a non-interactive zero-knowledge proof (NIZK) to prove that the key used to sign the message is certified. Finally, to prevent anyone else to randomize his group signature, the user will sign some of its elements with the strong one-time signature scheme.

Our first group signature scheme, whose group signatures comprise only 22 group elements, achieves selfless anonymity without the need for encryption.

Indeed, the use of the Camenish-Lysyanskaya signature scheme [10] makes it possible to disclose the digital signature on the message in the group signature, and thus to decrease the cost and size of the non-interactive proof of knowledge. The scheme resorts however to a trusted party to set up the system parameters. Moreover the cost of the opening algorithm is linear in the number of members.

Our second group signature scheme overcomes the drawbacks of the first scheme but at the expense of a slight increase in the size of the group signature (28 group elements). In fact, users certify now their verification keys (instead of their signing keys), allowing the group manager to set up the system parameters and to extract the identity of the group signature issuer using his extraction key. Since the verification keys are group elements, we now require that our certificate scheme is structure-preserving [1], which explains the extra-cost of the scheme.

Finally we propose a generic group signature scheme that generalizes Groth's scheme, and which achieves full anonymity. The construction makes use of an encryption scheme in order to be able to use any EUF-CMA digital signature scheme, which increases the size/cost of the group signature. An instantiation of this construction results in a group signature with 30 group elements.

We summarize in Figure the signature sizes (number of group elements), the verification costs (number of pairings) and the opening cost (as a function of the number of participants, denoted r) of our contributions (GS1, GS2, GS3) and compare them with the state of the art.

Scheme	Signature size	Verification cost	Opening cost	Standard model?
BCNSW [6]	5	2	O(r)	No
DP [13]	9	1	O(1)	No
Groth [15]	50	246	O(1)	Yes
GS1 (Section 4)	22	73	O(r)	Yes
GS2 (Section 5)	28	124	O(1)	Yes
GS3 (Section 6)	30	92	O(r)	Yes

Fig. 1. Comparison between group signatures performances

Outline of the Paper. The rest of this paper is organized as follows. In section 2 we recall definitions of the notions that we use in our schemes. Section 3 defines the syntax and the security model of a secure group signature scheme. Following sections present our group signature schemes. We defer the security proofs to the full version [14].

2 Preliminaries

In this section we recall some necessary bricks that will be used throughout the document, namely bilinear maps, Camenisch-Lysyanskaya's signatures, and signatures on committed values. Note that we defer in full version [14] the recall of further primitives such as (one-time, structure-preserving) signatures, tag-based encryption, and non-interactive proofs of knowledge.

2.1 Bilinear Groups

Our constructions use cyclic groups of prime order that have a non-degenerate efficiently computable bilinear map e. We use the following notations:

- $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are cyclic groups of prime order q.
- We write the group operations multiplicatively in each group.
- A bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ has the following properties:
  1. For all $x \in \mathbb{G}_1, y \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_q$ we have $e(x^a, y^b) = e(x, y)^{ab}$.
  2. For $x \neq 1_{\mathbb{G}_1}$ or $y \neq 1_{\mathbb{G}_2}$, $e(x, y) \neq 1_{\mathbb{G}_T}$.
  3. e is efficiently computable.

The Symmetric External Diffie-Hellman (SXDH) Assumption: We say $SXDH$ holds in the bilinear groups $\mathbb{G}_1$ and $\mathbb{G}_2$ if DDH is hard in both groups, where DDH (Decisional Diffie-Hellman) is the following problem:

Given (g, g^a, g^b, g^c) it is hard to decide whether $c = ab \bmod q$ or random.

2.2 Camenisch-Lysyanskaya Signatures

Our group signature schemes make use of the pairing-based Camenish-Lysyankaya signature schemes [10] (scheme A and scheme C in their paper), which are provably secure under the LSRW assumption [16]. The two schemes operate in three cyclic groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ of prime order q, equipped with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, and two generators $g \in \mathbb{G}_1$ and $\widetilde{g} \in \mathbb{G}_2$.

Lysyanskaya-Sahai-Rivest-Wolf (LSRW) Assumption:
Let $\widetilde{g} \in \mathbb{G}_2$, $\widetilde{x} = \widetilde{g}^{\alpha}$, $\widetilde{y} = \widetilde{g}^{\beta}$ and $O_{\widetilde{x},\widetilde{y}}(.)$ be an oracle that, on input a value $u \in \mathbb{Z}_q$, outputs $A = (a, a^{\beta}, a^{\alpha+u\alpha\beta}) \in \mathbb{G}_1^3$ for a randomly chosen $a \in \mathbb{G}_1$. Then for all probabilistic polynomial time adversaries $\mathcal{A}$, the quantity ϵ is negligible:

$$\epsilon = \texttt{Pr}[\alpha, \beta \leftarrow \mathbb{Z}_q; \widetilde{x} \leftarrow \widetilde{g}^{\alpha}, \widetilde{y} \leftarrow \widetilde{g}^{\beta}; (u, a, b, c) \leftarrow \mathcal{A}^{O_{\widetilde{x},\widetilde{y}}}(\widetilde{x}, \widetilde{y}) :$$

$$u \notin Q \wedge a \in \mathbb{G}_1 \wedge b = a^{\beta} \wedge c^{\alpha+u\alpha\beta}]$$

where Q is the set of queries asked by $\mathcal{A}$ to his oracle $O_{\widetilde{x},\widetilde{y}}(.)$.

2.3 Re-randomizable Signatures on Committed Values

Our constructions use *i.e.* digital signature schemes Σ with the following properties:

[Key generation]	Choose $\alpha, \beta \xleftarrow{R} \mathbb{Z}_d$ then compute $X \leftarrow \widetilde{g}^{\alpha}$ and $Y \leftarrow \widetilde{g}^{\beta}$
	set $pk \leftarrow \{X, Y\}$ and $sk \leftarrow \{\alpha, \beta\}$.
[Signature on m**]**	Choose a random $a \in \mathbb{G}_1$
	and outputs : $\sigma = (a, a^{\beta}, a^{\alpha + m\alpha\beta})$
[Verification]	Given pk, m and $\sigma = (a, b, c)$
	check if the following equations holds:
	$e(a, Y) = e(b, \widetilde{g})$ and $e(a, X).e(b, X)^m = e(c, \widetilde{g})$

Fig. 2. Camenisch-Lysyanskaya's signature A

[Key generation]	Choose $\alpha, \beta, z_i \xleftarrow{R} \mathbb{Z}_d$ then compute $X \leftarrow \widetilde{g}^{\alpha}$, $Y \leftarrow \widetilde{g}^{\beta}$
	and $Z_i \leftarrow \widetilde{g}^{z_i}$ for $1 \leq i \leq l$
	set $pk \leftarrow \{X, Y, Z_i\}_{1 \leq i \leq l}$ and $sk \leftarrow \{\alpha, \beta, z_i\}_{1 \leq i \leq l}$.
[Signature on $(m_0, ..., m_l)$**]**	Choose a random $a \in \mathbb{G}_1$
	$A_i = a^{z_i}$ for $1 \leq i \leq l$
	$b = a^{\beta}$, $B_i = (A_i)^{\beta}$
	$c = a^{\alpha + \alpha\beta m_0} \prod_{i=1}^{l} A^{\alpha\beta m_i}$
	and outputs the signature $\sigma = (a, \{A_i\}, b, \{B_i\}, c)$
[Verification]	Given pk, m and σ
	check if the following verification equations holds:
	$e(a, Z_i) = e(A_i, \widetilde{g})$
	$e(a, Y) = e(b, \widetilde{g})$ and $e(A_i, Y) = e(B_i, \widetilde{g})$
	$e(a, X).e(b, X)^{m_0} \prod_{i=1}^{l} e(B_i, X)^{m_i} = e(c, \widetilde{g})$

Fig. 3. Camenisch-Lysyanskaya's signature C

- The signature scheme is *re-randomizable* i.e. it admits a function `SigRand` such that for each signature σ on a message m, $\texttt{SigRand}(\sigma) = \sigma'$, where σ' is a valid signature on m and such that no probabilistic polynomial time adversary $\mathcal{A}$, with access to a signing oracle, is able, given a signature σ, to distinguish a randomized version of σ from a signature on a random element of the message space.
- Σ admits an algorithm `comSign` to obtain signatures on committed values which, on input c (where $(c, r) \leftarrow \texttt{Commit}(m)$) and $\pi \leftarrow \text{POK}\{(m, r) : (c, r) = \texttt{Commit}(m)\}(c)$, outputs σ such that $\Sigma.\texttt{verify}(\sigma, (m, r)) = 1$.

We define EUF-CMA security the same way as the standard digital signature schemes (a re-randomizable signature scheme cannot obviously reach the SEUF-CMA security). As shown in [10] the signature scheme described in figure 3 is EUF-CMA secure and admits an efficient protocol (described in Fig. 4) for obtaining signature on committed value. Moreover, using a proof similar to that of [6], this scheme is re-randomizable if the DDH-assumption holds in $\mathbb{G}_1$.

Signer (input: sk, pk)		**User** (input: $m_1, m_2, ..., m_k$ and pk)
$U_i \leftarrow g^{z_i}$ for $1 \leq i \leq k$		
	$\xrightarrow{\quad U_i \quad}$	
		$l \overset{R}{\longleftarrow} \mathbb{Z}_q$
		$C \leftarrow g^l \prod_{i=1}^{k} U_i^{m_i}$
		$\pi \leftarrow \text{POK}\{(l, m_1, ..., m_k) : C = g^l \prod_{i=1}^{k} U_i^{m_i}\}(C)$
	$\xleftarrow{\quad C, \pi \quad}$	
$x \overset{R}{\longleftarrow} \mathbb{Z}_q$		
$a \leftarrow g^x$		
$A_i = a^{z_i}$ for $1 \leq i \leq k$		
$b = a^\beta$, $B_i = (A_i)^\beta$		
$c = a^\alpha C^{x\alpha\beta}$		
	$\xrightarrow{\sigma \leftarrow (a, \{A_i\}, b, \{B_i\}, c)}$	

Fig. 4. Protocol for obtaining a CL signature C on committed values

3 Defining Group Signatures

3.1 Syntax

A group signature scheme consists of a set of users, identified by a unique index i, that can produce signatures on behalf of the group. Users must interact with a certification authority and the group manager to join the group. At the end of this interaction, each user obtains a signature key pair and the group manager obtains some information that will allow him to identify the group signatures issuers. The syntax that we require is as follows.

– $\texttt{Keygen}(\lambda)$: This algorithm inputs a security parameter λ and outputs $(gpk, sk_{cert}, sk_M,$**Sreg**,**reg**$)$, where gpk is the group public key, sk_{cert} is the issuer's secret key, sk_M is the group manager's secret key, **Sreg** is the group manager's secret register and **reg** is a public register.

– $\texttt{Join}(gpk, upk_i)$: The `Join` protocol is used to add a new user to the group. It consists of one interactive protocol between the user and the issuer and another one between the user and the group manager. The common inputs of this protocol is gpk and the user's public key upk_i. The private input of the user is usk_i whereas the private inputs of the issuer and the group manager are sk_{cert} and sk_M respectively. As a result of the first interaction, the user obtains his group signing key sk_i, the verification key vk_i and a certificate, $\sigma_{cert}(i)$, proving membership of the group. Then he uses usk_i to compute σ_i, a digital signature on vk_i. At the end of the second interaction, the group manager obtains and stores vk_i and σ_i in **Sreg**[i] and publishes upk_i in **reg**[i].

– $\texttt{Sign}(gpk, sk_i, \sigma_{cert}(i), m)$: This algorithm inputs a message m, the user's secret key and certificate and outputs a group signature μ on m.

- $\texttt{Verify}(gpk, \mu, m)$: This algorithm inputs a message m and a group signature μ and outputs 1 if μ is a valid group signature on m, and 0 otherwise.

- $\texttt{Open}(gpk, m, \mu, sk_M)$: This algorithm inputs a message m, a group signature μ and the group manager's secret data sk_M and **Sreg**. If μ is a valid group signature on m from user i it returns a proof τ of this statement and the identity i, else it returns $\perp$.

- $\texttt{Judge}(gpk, m, \mu, i, \tau)$: This algorithm inputs a message m, a group signature μ, an identity i and a proof of knowledge τ. It returns 1 if τ is a valid proof that μ is a group signature on m, issued by user i. Otherwise it returns 0.

3.2 Security Model

In this section we give the security definitions that we require for group signature schemes. We adhere to the model defined by [5], except that we use the *selfless anonymity* notion from [7] instead of the *full anonymity* for our first two group signature schemes. Informally, selfless anonymity, contrarily to full anonymity, does not protect dishonest users, i.e. the adversary does not know the private keys of the identities he wishes to be challenged on. As in [6], we consider a setting with n users divided statically into sets $\mathcal{HU}$ and $\mathcal{DU}$ of honest and dishonest users respectively (*i.e* an honest user cannot be corrupted during the experiment). This static division implies, by guessing the indices of "target" users in the anonymity and non-frameability experiment, security in the dynamic corruption case.
The security notions make use of the following oracles:

- $\mathcal{O}\texttt{Join}_{UD}(gpk, upk_i, sk_M)$ is an oracle that executes the user's side of the join protocol for the input user $i \in \mathcal{HU}$. This oracle will be used by an adversary playing the role of the corrupted group manager.

- $\mathcal{O}\texttt{Join}_{DM}(gpk, upk_i, usk_i)$ is an oracle that executes the join protocol with the honest group manager. This oracle will be used by an adversary to register a corrupted user.

- $\mathcal{O}\texttt{Sign}(i, m)$ is an oracle that accepts as input an identity i and a message m and returns a group signature μ if user i is honest and registered.

- $\mathcal{O}\texttt{Open}(m, \mu)$ inputs a message-signature pair (m, μ) and returns the result of the function call
$\texttt{Open}(\mathbf{Sreg}, sk_M, \mu, m)$. We use the notation $\mathcal{O}\texttt{Open}\neg(m, \mu)$ when the query (m, μ) is not allowed.

Correctness: We define the correctness of a group signature scheme through a game in which an adversary is allowed to request a signature on some message by any of the honest group members. The adversary wins if either the resulting signature does not pass the verification test, the signature is opened as if it were produced by a different user, or the judging algorithm returns 0. The group signature scheme is correct if for any adversary $\mathcal{A}$ and any security parameter λ (we keep this notation in the following experiments), $\texttt{Pr}[\mathbf{Exp}_{\mathcal{A}}^{corr}(\lambda) = 1]$ is negligible in λ, where $\mathbf{Exp}_{\mathcal{A}}^{corr}(\lambda)$ is defined as follows:

1. $\mathcal{HU} \leftarrow \{1..n\}$.
2. $(gpk, sk_{cert}, sk_M) \leftarrow \texttt{Keygen}(\lambda)$.
3. $(sk_i, vk_i, \sigma_{cert}(i)) \leftarrow \texttt{Join}(gpk, sk_{cert}, sk_M)$ for each user $i \in \mathcal{HU}$.
4. $(i, m) \leftarrow \mathcal{A}^{\mathcal{O}\texttt{Sign},\mathcal{O}\texttt{Open}}(gpk)$.
5. If $i \notin \mathcal{HU}$ then return 0.
6. $\mu \leftarrow \texttt{Sign}(sk_i, m)$.
7. If $\texttt{Verify}(gpk, m, \mu) = 0$ then return 1.
8. If $\texttt{Open}(\mathbf{Sreg}, sk_M, \mu, m) = (j, \tau)$ and $j \neq i$ then return 1.
9. If $\texttt{Judge}(m, \mu, \tau, i) = 0$ then return 1.

Anonymity: Informally, anonymity requires that group signatures do not reveal the signer's identity. In the selfless-anonymity game the adversary's goal is to determine which of the two users generated the challenge signature. The difference with the full anonymity of [5] consists in not giving the adversary $\mathcal{A}$ access to either private key. As in [7] and [6] we define the selfless-anonymity experiment $\mathbf{Exp}_{\mathcal{A}}^{anon-b}(\lambda)$ as follows:

1. $\mathcal{DU} \leftarrow \mathcal{A}(\lambda)$.
2. $\mathcal{HU} \leftarrow \{1..n\} \setminus \mathcal{DU}$.
3. $(gpk, sk_{cert}, sk_M) \leftarrow \texttt{Keygen}(\lambda)$.
4. $(sk_i, vk_i, \sigma_{cert}(i)) \leftarrow \texttt{Join}(gpk, sk_{cert}, sk_M)$ for each user $i \in \mathcal{HU}$.
5. $(m, i_0, i_1) \leftarrow \mathcal{A}^{\mathcal{O}\texttt{Sign},\mathcal{O}\texttt{Open},\mathcal{O}\texttt{Join}_{DM}}(gpk)$ with $(i_0, i_1 \in \mathcal{HU})$.
6. $\mu \leftarrow \texttt{Sign}(sk_{i_b}, m)$ for $b \xleftarrow{R} \{0, 1\}$.
7. $b^* \leftarrow \mathcal{A}^{\mathcal{O}\texttt{Sign},\mathcal{O}\texttt{Open}\neg(m,\mu),\mathcal{O}\texttt{Join}_{DM}}(gpk)$.
8. Return b^*.

We define $\mathbf{Adv}_{\mathcal{A}}^{anon-b}(\lambda) = |\texttt{Pr}[b = b^*] - \frac{1}{2}|$. The group signature scheme is anonymous if for any probabilistic polynomial time adversary, this advantage is a negligible function of λ. In the full-anonymity experiment, we no longer require that $i_0, i_1 \in \mathcal{HU}$, we then say that the group signature scheme is fully-anonymous.

Traceability: Traceability requires that no adversary is able to create a valid signature that cannot be traced to some user already registered. We define the traceability experiment as follows:

1. $\mathcal{DU} \leftarrow \{1..n\}$.
2. $(gpk, sk_{cert}, sk_M) \leftarrow \texttt{Keygen}(\lambda)$.
3. $(m, \mu) \leftarrow \mathcal{A}^{\mathcal{O}\texttt{Open},\mathcal{O}\texttt{Join}_{DM}}(gpk)$.

4. If $\texttt{Verify}(gpk, m, \mu) = 1$ and $\texttt{Open}(\mathbf{Sreg}, sk_M, \mu, m) = \perp$ then return 1.
5. Return 0.

We define $\mathbf{Adv}_{\mathcal{A}}^{trace}(\lambda) = \texttt{Pr}[\mathbf{Exp}_{\mathcal{A}}^{trace}(\lambda) = 1]$. The group signature scheme is traceable if for any probabilistic polynomial time adversary, this advantage is a negligible function of λ.

Non-frameability: Informally, non-frameability requires that a cheating group manager cannot falsely accuse an honest user of having signed a given message m. We define the non-frameability experiment as follows:

1. $\mathcal{DU} \leftarrow \mathcal{A}(\lambda)$.
2. $\mathcal{HU} \leftarrow \{1..n\} \setminus \mathcal{DU}$.
3. $(gpk, sk_{cert}, sk_M) \leftarrow \texttt{Keygen}(\lambda)$.
4. $(sk_i, vk_i, \sigma_{cert}(i)) \leftarrow \texttt{Join}(gpk, sk_{cert}, sk_M)$ for each user $i \in \mathcal{HU}$.
5. $(i, m, \mu) \leftarrow \mathcal{A}^{\mathcal{O}\texttt{Sign}, \mathcal{O}\texttt{Join}_{UD}}(sk_M, gpk)$.
6. If $i \notin \mathcal{HU}$ or $\texttt{Verify}(gpk, m, \mu) = 0$ then return 0.
7. If m was queried to $\mathcal{O}\texttt{Sign}$ then return 0.
8. If $\texttt{Judge}(m, \mu, \tau, i) = 0$ then return 0.
9. Return 1.

We define $\mathbf{Adv}_{\mathcal{A}}^{nf}(\lambda) = \texttt{Pr}[\mathbf{Exp}_{\mathcal{A}}^{nf}(\lambda) = 1]$. The group signature scheme is non-frameable if for any probabilistic polynomial time adversary, this advantage is a negligible function of λ.

Remark: We can also define the strong non-frameability experiment in which the adversary's goal is to accuse an honest user of having created a given signature μ (we replace the line 7 by: if μ was produced by $\mathcal{O}\texttt{Sign}$ then returns 0).

4 A Group Signature without Encryption

4.1 Description

The core of this group signature scheme is the combination of a re-randomizable signature scheme Σ_{cert} and the signature scheme Σ_1 described in Figure 2. We assume that the system parameters are set up by a trusted party and that each user i has a key pair (usk_i, upk_i) for a signature scheme Σ_0 where upk_i is public and associated to i. At the end of the $\texttt{Keygen}$ algorithm, the issuer receives sk_{cert}. When a new member joins the group, he creates a key pair (sk_i, vk_i) for Σ_1, then gets a certificate $\sigma_{cert}(i)$ on sk_i (using the protocol for obtaining signatures on committed values) and finally sends vk_i and a signature on it (using usk_i) to the group manager who records it in his secret register **Sreg**. When making a group signature on a message m, the member will first re-randomize his certificate and generate a key pair (sk_{ots}, vk_{ots}) for a strong one-time signature. Then he will use his secret key sk_i to sign the concatenation of m, the certificate and vk_{ots} and give a non-interactive zero-knowledge proof of knowledge that the certificate is a valid signature on sk_i. To prevent an adversary from randomizing the signature or the non-interactive proof, the user will sign them with the strong one-time signature. We use the following notations to describe our group signature scheme:

Notations:

- λ is the security parameter.
- $\mathcal{G}$ is a probabilistic polynomial time algorithm that, on input λ, generates $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \widetilde{g})$ where $g \in \mathbb{G}_1$ and $\widetilde{g} \in \mathbb{G}_2$.
- K_{NI} is a probabilistic polynomial time algorithm that, on input $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \widetilde{g})$, generates a common reference string crs for the Groth-Sahai proof system.
- **reg** is a public register.
- **Sreg** is the secret register of the group manager.
- gpk is the group public key.

The algorithms defining our group signature scheme are described in figure 5.

```
Keygen($\lambda$)
$gk \leftarrow \mathcal{G}(\lambda)$
$crs \leftarrow K_{NI}(gk)$
$(sk_{cert}, pk_{cert}) \leftarrow \Sigma_{cert}.\mathtt{keygen}(gk)$
$gpk \leftarrow (gk, pk_{cert}, crs, \mathbf{reg})$

Join(user: $usk_i$, issuer: $sk_{cert}$, group manager: Sreg)
$(sk_i, vk_i) \leftarrow \Sigma_1.\mathtt{Keygen}(gk)$
$\sigma_{cert}(i) \leftarrow \Sigma_{cert}.\mathtt{comSign}(sk_{cert}, \mathtt{Commit}(sk_i), \pi)$
$\sigma_i \leftarrow \Sigma_0.\mathtt{sign}(usk_i, vk_i)$
$\mathbf{Sreg}[i] \leftarrow (vk_i, \sigma_i)$
$\mathbf{reg}[i] \leftarrow (i, upk_i)$

Sign($sk_i, \sigma_{cert}(i), m$)
$\sigma_{cert}(i) \leftarrow \Sigma_{cert}.\mathtt{sigRand}(\sigma_{cert}(i))$
$(sk_{ots}, vk_{ots}) \leftarrow \Sigma_{ots}.\mathtt{keygen}$
$(a, b, c) \leftarrow \Sigma_1.\mathtt{sign}(sk_i, m \| vk_{ots} \| \sigma_{cert}(i))$
$\pi \leftarrow \mathrm{POK}\{(sk_i):$
    $(a, b, c) = \Sigma_1.\mathtt{sign}(sk_i, m \| vk_{ots} \| \sigma_{cert}(i)) \wedge$
    $\sigma_{cert}(i) = \Sigma_{cert}.\mathtt{sign}(sk_{cert}, sk_i)$
    $\}(m, (a, b, c), \sigma_{cert}(i))$
$\sigma_{ots} \leftarrow \Sigma_{ots}.\mathtt{sign}(sk_{ots}, a \| \pi)$
$\mu \leftarrow (m, vk_{ots}, \sigma_{ots}, (a, b, c), \sigma_{cert}(i), \pi)$
```

```
Verify($m, \mu$)
If $V_{NI}(crs, \mu, m, \pi) = 1 \wedge$
    $\Sigma_{ots}.\mathtt{verify}(vk_{ots}, \sigma_{ots}, a \| \pi) = 1$
    then return 1
Else return 0.

Open($\mathbf{Sreg}, m, \mu$)
If $\mathtt{Verify}(m, \mu) = 0$
    then return 0.
Parse $\mu$ as $(m, vk_{ots}, \sigma_{ots}, \sigma_1, \sigma_{cert}, \pi)$
for $vk_i$ in $\mathbf{Sreg}[i]$:
    if $\Sigma_1.\mathtt{Verify}(vk_i, m \| vk_{ots} \| \sigma_{cert}, \sigma_1) = 1$
        return $i$ and a proof $\tau$.
with $\tau \leftarrow \mathrm{POK}\{(vk_i, \sigma_i):$
    $\Sigma_1.\mathtt{Verify}(vk_i, m \| vk_{ots} \| \sigma_{cert}, \sigma_1) = 1 \wedge$
    $\Sigma_0.\mathtt{Verify}(upk_i, vk_i, \sigma_i) = 1$
    $\}(m, \mu, \mathbf{reg})$

Judge($m, \mu, \tau, i$)
If $V_{NIZK}(crs, m, \mu, \tau, i) = 1$
    return 1.
Else return 0.
```

Fig. 5. Group signature scheme 1

Theorem 1. *If Σ_{cert} is a EUF-CMA secure signature scheme and (P, V) is a sound zero-knowledge proof system, then the group signature is fully traceable.*

Theorem 2. *If Σ_0 and Σ_1 are EUF-CMA secure signature schemes, Σ_{ots} is a strong one-time signature scheme and (P, V) is a sound, zero-knowledge proof system, then the group signature scheme is strongly non-frameable.*

Theorem 3. *If Σ_0 and Σ_1 are EUF-CMA secure signature schemes, Σ_{ots} is a strong one-time signature and (P, V) is a sound, zero-knowledge proof system, then the group signature scheme is anonymous.*

The proofs are provided in the full version [14].

4.2 A Concrete Realization

We instantiate this system with the Λ_{sxdh} setting which refers to the case of asymmetric pairings for which the DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$. We use the CL signature scheme described in Fig. 3 for Σ_{cert}, the weakly secure Boneh-Boyen signature scheme described in Fig. 8 for Σ_{ots} and the structure-preserving signature described in Fig. 9 for Σ_0.
We use the following notations to describe the proofs of knowledge:

Notations:

- $(sk_i, pk_i) \leftarrow ((\alpha_i, \beta_i), (\widetilde{g}^{\alpha_i}, \widetilde{g}^{\beta_i}))$
- $(sk_{cert}, pk_{cert}) \leftarrow ((u, v, z_1, z_2), (U, V, Z_1, Z_2))$
- $(sk_M, pk_M) \leftarrow ((x_1, \widetilde{x}_1, x_2, \widetilde{x}_2), (X_1, \widetilde{X}_1, X_2, \widetilde{X}_2))$
- $\sigma_1 \leftarrow (a_1, b_1, c_1)$; $\sigma_{cert} \leftarrow (a, A_1, A_2, b, B_1, B_2, c)$
- $\eta \leftarrow m \| vk_{ots} \| \sigma_{cert}$

A group signature, $(vk_{ots}, \sigma_{ots}, (a, b, c), \sigma_{cert}(i))$ involves 13 group elements, and a proof π of the following equations (the variables are underlined):

$$e(a_1, \widetilde{g})^{\underline{\beta_i}} = e(b_1, \widetilde{g}) \tag{1}$$

$$(e(a_1, \widetilde{g}).e(b_1, \widetilde{g})^{\eta})^{\underline{\alpha_i}} = e(c_1, \widetilde{g}) \tag{2}$$

$$e(c, \widetilde{g}) = e(a, U).e(b, U)^{\underline{l}} .e(B_1, U)^{\underline{\alpha_i}} .e(B_2, U)^{\underline{\beta_i}} \tag{3}$$

where η is $m \| vk_{ots} \| \sigma_{cert}$ and l is the randomness used by user i in the join stage to obtain signature on the committed values α_i and β_i.
These equations involve 3 variables (α_i, β_i, l) and thus increase the size of the proof by 6 group elements. Equations (1) and (2) proves that (a_1, b_1, c_1) is a valid Camenisch-Lysyanskaya signature from user i on η. These two equations cost 1 group element each. Equation (3) proves that the certificate is a valid signature on α_i and β_i and costs 1 group element. The total size of the proof is then 9 group elements.

The total size of the group signature is thus 22 group elements, which amounts to 700 Bytes if we consider an implementation using 256-bit groups sizes. Using naive computations, equation (1) needs for the verification 18 pairings, equation (2), 19 pairings and equation (3), 25 pairings. Checking the consistency of the certificate and of the one-time signature adds 11 pairings. The total cost of the `Verify` algorithm is then 73 pairings.

Remark: We can further reduce the size of the group signature if the group manager and the certificate issuer are the same entity. Indeed, in the `Join` protocol we no longer need to use the hiding randomness l which decreases both the size of the certificate and the size of the proof by two elements. The size of the group signature is then 18 group elements.

5 A Group Signature without Encryption and without Trusted Parties

5.1 Description

The main drawbacks of the previous scheme lie in requiring a trusted party to set up the system parameters, and having the cost of the algorithm `Open` linear in the number of users. To solve these two problems, users will now certify their verification keys (instead of their signing keys), allowing thus the group manager to set up the system parameters and to use the extraction keys ek of the Groth-Sahai proofs to extract the identity behind a group signature.

This construction is similar to the previous scheme except that we require a different signature scheme for Σ_{cert}. We now assume that any digital signature σ, generated using Σ_{cert} on an arbitrary message m can be efficiently transformed in a reversible way to a pair (r, s) where r (r may be the empty string) is information theoretically independent from m, *i.e.* there exists an algorithm `Simulate` that inputs $\Sigma_{cert}.sk$ and a message from the message space and outputs a string indistinguishable from r. We require further the partial randomizability property for Σ_{cert}, *i.e.* Σ_{cert} admits two algorithms $\texttt{SigPrand}_0$ and $\texttt{SigPrand}_1$, defined as follows:

For each signature $\sigma \to (r, s)$ on a message m:

– $(r', state) \leftarrow \texttt{SigPrand}_0(r)$
– $s' \leftarrow \texttt{SigPrand}_1(s, state)$

where r' is unlinkable to r and $\sigma' \leftarrow (r', s')$ is a valid signature on m.

We use the same notations as in the previous scheme to define in figure 6 the algorithms of our second group signature scheme. X_{NI} will be the extraction algorithm for the non-interactive Groth-Sahai proofs.

Theorem 4. *If Σ_{cert} is a EUF-CMA secure signature scheme and (P, V) is a sound zero-knowledge proof system, then the group signature is fully traceable.*

Theorem 5. *If Σ_0 and Σ_1 are EUF-CMA secure signature schemes, Σ_{ots} is a strong one-time signature scheme and (P, V) is a sound, zero-knowledge proof system, then the group signature scheme is strongly non-frameable.*

Theorem 6. *If Σ_0 and Σ_1 are EUF-CMA secure signature schemes, Σ_{ots} is a strong one time signature and (P, V) is a sound, zero-knowledge proof system, then the group signature scheme is anonymous.*

The proofs are provided in the full version [14].

```
**Keygen**$(\lambda)$
$gk \leftarrow \mathcal{G}(\lambda)$
$(crs, ek) \leftarrow K_{NI}(gk)$
$(sk_{cert}, pk_{cert}) \leftarrow \Sigma_{cert}.\texttt{keygen}(gk)$
$gpk \leftarrow (gk, pk_{cert}, crs, \mathbf{reg})$

**Join**(user: $usk_i$, issuer: $sk_{cert}$, group manager: **Sreg**)
$(sk_i, vk_i) \leftarrow \Sigma_1.\texttt{Keygen}(gk)$
$\sigma_{cert}(i) \leftarrow \Sigma_{cert}.\texttt{sign}(sk_{cert}, vk_i)$
$\sigma_i \leftarrow \Sigma_0.\texttt{sign}(usk_i, vk_i)$
$\mathbf{Sreg}[i] \leftarrow (vk_i, \sigma_i)$
$\mathbf{reg}[i] \leftarrow (i, upk_i)$

**Sign**$(sk_i, \sigma_{cert}(i), m)$
$(r, s) \leftarrow \sigma_{cert}(i)$
$(r', state) \leftarrow \texttt{SigPrand}_0(r)$
$s' \leftarrow \texttt{SigPrand}_1(s, state)$
$(sk_{ots}, vk_{ots}) \leftarrow \Sigma_{ots}.\texttt{keygen}(\lambda)$
$(a, b, c) \leftarrow \Sigma_1.\texttt{sign}(sk_i, m \| vk_{ots} \| r')$
$\pi \leftarrow \mathrm{POK}\{(vk_i, s') :$
    $(a, b, c) = \Sigma_1.\texttt{sign}(sk_i, m \| vk_{ots} \| r') \wedge$
    $(r', s') = \Sigma_{cert}.\texttt{sign}(sk_{cert}, vk_i)$
    $\}(m, (a, b, c), r')$
$\sigma_{ots} \leftarrow \Sigma_{ots}.\texttt{sign}(sk_{ots}, a \| \pi)$
$\mu \leftarrow (m, vk_{ots}, \sigma_{ots}, (a, b, c), r', \pi)$
```

```
**Verify**$(m, \mu)$
If $V_{NI}(crs, \mu, m, \pi) = 1 \wedge$
    $\Sigma_{ots}.\texttt{verify}(vk_{ots}, \sigma_{ots}, a \| \pi) = 1$
    then return 1
Else return 0.

**Open**$(\mathbf{Sreg}, ek, m, \mu)$
$vk_i \leftarrow X_{NI}(crs, \mu)$
If $vk_i$ is registered in **Sreg**
    return $i$ and $\tau$
Else return $\perp$
with $\tau \leftarrow \mathrm{POK}\{(vk_i, \sigma_i) :$
    $\Sigma_1.\texttt{Verify}(vk_i, m \| vk_{ots} \| r', \sigma_1) = 1 \wedge$
    $\Sigma_0.\texttt{Verify}(upk_i, vk_i, \sigma_i) = 1$
    $\}(m, \mu, \mathbf{reg})$

**Judge**$(m, \mu, \tau, i)$
If $V_{NIZK}(crs, m, \mu, \tau, i) = 1$
    return 1.
Else return 0.
```

Fig. 6. Group signature scheme 2

5.2 A Concrete Realization

We use the Λ_{sxdh} setting and the structure-preserving signature scheme described in Fig. 9 for Σ_{cert}. We further use the weakly secure Boneh-Boyen signature scheme described in Fig 8 for Σ_{ots}.
Using the same notations as in section 4 (with $(\widetilde{X}, \widetilde{Y}) \leftarrow (\widetilde{g}^{\alpha_i}, \widetilde{g}^{\beta_i})$) and [1], π is a proof of the following equations:

$$e(a_1, \underline{Y}) = e(b_1, \widetilde{g}) \tag{1}$$

$$e(a_1, \underline{X}).e(b_1, \underline{X})^{\eta} = e(c_1, \widetilde{g}) \tag{2}$$

$$e(g_z, \underline{z'})e(g_r, \underline{r'})e(s', t')e(g_1, \underline{X})e(g_2, \underline{Y}) = A \tag{3}$$

$$e(h_z, \underline{z'})e(h_u, \underline{u'})e(v', w')e(h_1, \underline{X})e(h_2, \underline{Y}) = B \tag{4}$$

where $\sigma_{cert} \leftarrow (z', r', s', t', u', v', w')$ and η is $m \| vk_{ots} \| (s', t', v', w')$. The authors of [1] proved that (s', t', v', w') are information theoretically independent of the signature element z' and (X, Y).

These equations involve then 5 variables (X, Y, z', r', u') and thus increase the size of the proof by 10 group elements. Equations (1) and (2) prove that (a_1, b_1, c_1) is a valid Camenisch-Lysyanskaya signature from user i on η. These two equations costs 2 group elements each. Equation (3) and (4) prove that the certificate is a valid signature on X and Y and cost 2 group elements each. The total size of the proof is then 18 group elements.

A group signature, $(vk_{ots}, \sigma_{ots}, (a, b, c), (s', t', v', w'))$ involves 10 group elements, the total size is thus 28 group elements, which amounts to 900 Bytes if we consider an implementation using 256-bit groups size. Using naive computations, equation (1) involves 22 pairings, equation (2), 27 pairings and equation (3) and (4), 37 pairings each. Verifying the one-time signature adds 1 pairing. The total cost of the `Verify` algorithm is then 124 pairings.

In the full version, we present a third construction which can be seen as a generalization of Groth's [15] group signature. Appropriate instantiation of this construction results in a group signature consisting of 30 group elements, which is more compact that the realization in [15] where the total size of the group signature is 50 elements. Moreover, the verification cost of such a signature is also much smaller than that of [15].

References

1. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 209–236. Springer, Heidelberg (2010)
2. Ateniese, G., Camenisch, J., Hohenberger, S., de Medeiros, B.: Practical group signatures without random oracles. IACR Cryptology ePrint Archive, 2005:385 (2005)
3. Bellare, M., Boldyreva, A., Palacio, A.: An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 171–188. Springer, Heidelberg (2004)
4. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 614–629. Springer, Heidelberg (2003)
5. Bellare, M., Shi, H., Zhang, C.: Foundations of group signatures: The case of dynamic groups. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 136–153. Springer, Heidelberg (2005)
6. Bichsel, P., Camenisch, J., Neven, G., Smart, N.P., Warinschi, B.: Get shorty via group signatures without encryption. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 381–398. Springer, Heidelberg (2010)
7. Boneh, D., Shacham, H.: Group signatures with verifier-local revocation. In: Atluri, V., Pfitzmann, B., McDaniel, P.D. (eds.) ACM Conference on Computer and Communications Security, pp. 168–177. ACM (2004)
8. Boyen, X., Waters, B.: Compact group signatures without random oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 427–444. Springer, Heidelberg (2006)
9. Boyen, X., Waters, B.: Full-domain subgroup hiding and constant-size group signatures. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 1–15. Springer, Heidelberg (2007)
10. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Heidelberg (2004)
11. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited (preliminary version). In: Vitter, J.S. (ed.) STOC, pp. 209–218. ACM (1998)

12. Chaum, D.: Some weaknesses of "Weaknesses of undeniable signatures". In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 554–556. Springer, Heidelberg (1991)
13. Delerablée, C., Pointcheval, D.: Dynamic fully anonymous short group signatures. In: Nguyen, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 193–210. Springer, Heidelberg (2006)
14. El Aimani, L., Sanders, O.: Efficient Group Signatures in the Standard Model. Cryptology ePrint Archive (2012)
15. Groth, J.: Fully anonymous group signatures without random oracles. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 164–180. Springer, Heidelberg (2007)
16. Lysyanskaya, A., Rivest, R.L., Sahai, A., Wolf, S.: Pseudonym Systems (Extended Abstract). In: Heys, H.M., Adams, C.M. (eds.) SAC 1999. LNCS, vol. 1758, pp. 184–199. Springer, Heidelberg (2000)

Batch Verification Suitable for Efficiently Verifying a Limited Number of Signatures

Keisuke Hakuta[1,2], Yosuke Katoh[2], Hisayoshi Sato[1], and Tsuyoshi Takagi[3]

[1] Hitachi, Ltd., Yokohama Research Laboratory,
292, Yoshida-cho, Totsuka-ku, Yokohama, Kanagawa, 244-0817, Japan
{keisuke.hakuta.cw,hisayoshi.sato.th}@hitachi.com
[2] Graduate School of Mathematics, Kyushu University,
744, Motooka, Nishi-ku, Fukuoka, Fukuoka, 819-0395, Japan
k-hakuta@math.kyushu-u.ac.jp
[3] Institute of Mathematics for Industry, Kyushu University,
744, Motooka, Nishi-ku, Fukuoka, Fukuoka, 819-0395, Japan
takagi@imi.kyushu-u.ac.jp

Abstract. Batch verification is a method for verifying digital signatures at once. Batch verification can reduce the computational cost compared to that of verifying each signature one by one, and in particular, batch verification is especially appropriate for systems which are required to verify a large amount of signatures. However, in addition to the above requirement, several types of systems might also require verifying a limited number of digital signatures more and more efficiently in real-time. For this purpose, to improve the efficiency of verifying a limited number of signatures is presumably an important matter. This paper deals with the second requirement and proposes an efficient batch verification technique suitable for verifying a limited number of signatures in real-time. Our method can only be applied to elliptic curve based signatures, and uses one of the two special families of elliptic curves.

Keywords: digital signature, batch verification, elliptic curve, Frobenius expansion, endomorphism.

1 Introduction

A batch verification for digital signature scheme is a method to verify multiple signatures simultaneously in order to significantly speed up signature verification. If signatures are generated by multiple signers, we call it *multi-signer* batch verification; otherwise, we call it *single-signer* batch verification in accordance with [9]. Batch verification was firstly proposed by Naccache, M'Raïhi, Raphaeli, and Vaudenay [22] and Yen and Laih [31], respectively. Although several batch verification methods have been proposed (cf. [11,12,21,23]), almost all methods were broken (cf. [4,7,15,29]).

In 1998, Bellare, Garay, and Rabin [3] introduced the notion of single-signer batch verification (and also introduced *screening* which is a weaker notion of batch verification) and proposed three general tests for discrete logarithm based signature schemes. The small exponents test was improved by Cheon and Lee [6] to speed up batch verification for elliptic curve based signature schemes, and proposed two batch verification tests. One is called the Sparse Exponent Test and the other is called "*the Complex Exponent Test*" (CE test for short). So as to achieve further improvement, Cheon and Yi [8] generalized the idea of CE test in [6].

In 2007, Camenisch, Hohenberger, and Pedersen [5] extended the notion of multi-signer batch verification. Moreover, Camenisch *et al.* applied the small exponents test to some pairing-based signature schemes. An excellent overview of batch verification can be found in [5].

1.1 Motivation

In this paper, we focus on single-signer batch verification for discrete logarithm based signature schemes. Batch verification is especially appropriate for systems which are required to verify **a large amount of signatures**. Namely, for many applications of batch verification, it is an important requirement to verify a large amount of signatures efficiently. However, in addition to the above requirement, several types of systems might also require verifying **a limited number of digital signatures** more and more efficiently. This means that it might be a requirement to verify a limited number of digital signatures generated by a signer, as fast as possible.

– *IP camera surveillance system.*

One such example which might need the above requirement is IP camera surveillance system. In IP camera surveillance systems, each IP camera captures live images and sends them directly to image storage servers over an IP network. Digital signature provide an effective solution to prevent manipulation of live images. In order to facilitate key management, all IP cameras which are monitoring the same area often use the same signing key.

A typical IP camera supports real time frame rate of $5 \sim 30$ frames per second (fps). If the frame rate is 15 fps and an IP camera captures live images throughout the day, more than one million images are stored in the storage server(s). Normally the storage server verify the signatures which have already stored in the storage server. Thus it is an important requirement to verify a large amount of stored signatures efficiently.

On the other hand, the Internet is inherently insecure and is facing various security threads. So one might verify the signatures immediately after the storage server received the live images. Therefore, it might be an important requirement to verify a limited number of live images immediately after the storage server received the live images.

– *Wireless sensor network.*

Another example is wireless sensor network (WSN). Wireless sensor networks are widely used in various kinds of fields. Wireless sensor network is typically composed of a large number of sensor nodes and a base station [30]. In WSN,

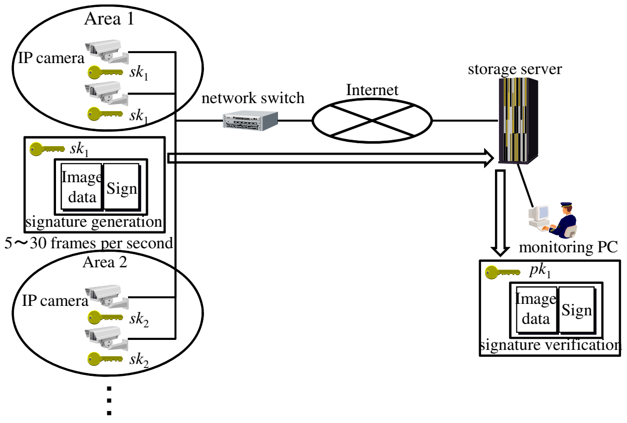


Fig. 1. A typical IP camera surveillance system

the sensed data is transmit to the base station through a multi-hop network consisting of several sensor nodes. Because multi-hop networks are potentially vulnerable to manipulation, digital signature is used to validate the data integrity of the sensed data. As is the case with the above example, several sensor nodes are used the same signing key. Each sensed data is verify by the base station. In this situation, the base station might verify a limited number of sensed data for some reasons such as efficient verification.

1.2 Contribution of this Paper

The contribution of this paper is to propose a new batch verification for discrete logarithm based signature schemes. Whereas the CE test is suitable for systems which need to verify a large amount of signatures, the proposed test is suitable for verifying a limited number of signatures in real-time. The performance analysis shows that the proposed test is faster than previous methods when a limited number of signatures is verified (See Table 5). For example, our test with a 286 bit curve (Table 7) is faster than previous methods when the number of sigantures is less than or equal to 18 (resp. 6) for security parameter 80 (resp. 140).

The rest of this paper is organized as follows. Section 2 prepares some notations. Section 3 reviews some batch verification tests. In Section 4, we propose two new batch verification tests. Section 5 compares the computational costs of several previous tests and our proposed test. Section 6 concludes the paper.

2 Notation

Throughout this paper, we use the following notation. For any field k, we denote by $p = \mathrm{char}(k)$ the characteristic of the field k. We use the symbols $\mathbb{Z}$, $\mathbb{C}$, and $\mathbb{F}_q$ to represent the integers, complex numbers, and a finite field with q elements respectively, where $q = p^r$ $(r \geq 1)$, $p = \mathrm{char}(\mathbb{F}_q)$. For a positive integer n, we denote the residue class ring modulo n by $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$. For $x \in \mathbb{C}$, we denote by $|x|$ the absolute value of x. For a finite set S, we denote the cardinality of S by $\#S$.

For any non-zero complex number $\psi \in \mathbb{C} \setminus \{0\}$, we denote ψ-adic expansion $\sum_{i=0}^{\ell-1} c_i \psi^i$ with $c_i \in \mathbb{Z}$ by $(c_{\ell-1}, \ldots, c_0)_\psi$, and denote the number of non-zero c_i's by $wt((c_{\ell-1}, \cdots, c_0)_\psi)$. We call ℓ the length of the ψ-adic expansion. Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$, $\phi : E \to E, \quad (x, y) \mapsto (x^q, y^q)$ be the q^{th}-power Frobenius map on E. We put $t_m := q^m + 1 - \#E(\mathbb{F}_{q^m})$, $t := t_1$, and call t_m the trace of $E(\mathbb{F}_{q^m})$. We can regard ϕ as a complex number which satisfies the characteristic equation $\phi^2 - t\phi + q = 0$. We denote the cardinality of the set of the $\mathbb{F}_{q^m}$-rational points of E by $\#E(\mathbb{F}_{q^m}) = hn$ (h: cofactor, n: prime). We assume that the order of a point $P \in E(\mathbb{F}_{q^m})$ is n.

$\mathcal{A}$, $\mathcal{F}$ stand for the computational cost of point addition on $E(\mathbb{F}_{q^m})$, q^{th}-power Frobenius map on $E(\mathbb{F}_{q^m})$, respectively. An elementary multiplication in $\mathbb{F}_{q^m}$ will be abbreviated by M.

3 Preliminaries

In this section we briefly review batch verification. For detail, refer to [3], [5], [6] and [8].

3.1 Batch Verification

Let $\mathbb{G}$ be a cyclic group of prime order n, and g a generator of $\mathbb{G}$. For $a \in \mathbb{G}$, $\mathrm{ord}_{\mathbb{G}}(a)$ is the order of a, namely, $\mathrm{ord}_{\mathbb{G}}(a)$ is the smallest non-negative integer x that holds $a^x = 1_G$, where 1_G is the identity element in the group $\mathbb{G}$. $X = \{(x_1, y_1), \cdots, (x_N, y_N)\}$ $(x_i \in \mathbb{Z}_n, y_i \in \mathbb{G})$ is a batch instance which we would like to check whether or not $y_i = g^{x_i}$ for all $i = 1, \cdots, N$. We define a Boolean relation $R(\cdot)$ on the set $\{(x, y) | x \in \mathbb{Z}_n, y \in \mathbb{G}\}$ by the following. We define $R(x, y) = 1$ if $g^x = y$, and $R(x, y) = 0$ otherwise. We say that the batch instance $X = \{(x_1, y_1), \cdots, (x_N, y_N)\}$ $(x_i \in \mathbb{Z}_n, y_i \in \mathbb{G})$ is correct if $R(x_i, y_i) = 1$ for all $i = 1, \ldots, N$, and incorrect if there exists an index $i \in \{1, \ldots, N\}$ such that $R(x_i, y_i) = 0$. The following definition is the notion of single-signer batch verification introduced by Bellare, Garay, and Rabin [3].

Definition 1. [Batch Verifier (Batch Verification Test) [3]] *A batch verifier* (*also called a batch verification test*) *for relation R is a probabilistic algorithm $\mathcal{V}$ that takes as input a batch instance $X = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ for R and k_{bv} a security parameter for batch verification, and*
(1) *If X is correct then $\mathcal{V}$ outputs* 1.
(2) *If X is incorrect then the probability that $\mathcal{V}$ outputs* 1 *is at most $2^{-k_{\mathrm{bv}}}$. This probability is called the error probability.*

3.2 Small Exponents Test

Bellare, Garay, and Rabin [3] proposed three general batch verification tests, the (*Atomic*) *Random Subset Test*, the *Small Exponents Test*, and the *Bucket Test.* The atomic random subset test checks whether $g^{\sum_{i=1}^{N} s_i x_i} = \prod_{i=1}^{N} {y_i}^{s_i}$, where each s_i is picked at random bit from $\{0,1\}$. Note that the upper bound of the error probability of the atomic random subset test is at most $1/2$. The random subset test is repetition of the atomic random subset test independently k_{bv} times in order to lower the error probability of the random subset test to $2^{-k_{\mathrm{bv}}}$. The small exponents test is an extension of the atomic random subset test. The small exponents test checks whether $g^{\sum_{i=1}^{N} s_i x_i} = \prod_{i=1}^{N} {y_i}^{s_i}$, where each s_i is picked at random bit from $\{0,1\}^{k_{\mathrm{bv}}}$.

3.3 Complex Exponent Test

Cheon and Lee [6] proposed two improved tests of the small exponents test, one is the *Sparse Exponent Test*, and the other is the *Complex Exponent Test.* The complex exponent test can only be applied to a special family of elliptic curves whereas the sparse exponent test can be applied to elliptic curves over prime/extension fields. In the complex exponent test, subfield elliptic curves (cf. [17], [26], [28]) are used.

Digital signature schemes based on an elliptic curve over a finite field (e.g., ECDSA* [1] which is a modified version of ECDSA [20], EC-Schnorr signature [27]), for $x \in \mathbb{Z}_n$, $Q \in E$, it is checked whether $xP = Q$ or not. We consider how to check $x_i P = Q_i$ $(i = 1, \ldots, N)$ for a batch instance $\{(x_1, Q_1), \ldots, (x_N, Q_N)\}$. We define a Boolean relation $\mathrm{ScMul}_{E,P}(x, Q)$ as follows: $\mathrm{ScMul}_{E,P}(x, Q) := 1$ if $xP = Q$, otherwise $\mathrm{ScMul}_{E,P}(x, Q) := 0$.

We prepare some notations to explain the complex exponent test. We define

$$S_1(\ell_1, k_1, q) \underset{\mathrm{def}}{=} \left\{ d = \sum_{i=0}^{\ell_1 - 1} a_i \phi^i \,\middle|\, a_i \in \mathbb{Z}, |a_i| \leq q-1, a_{i+1}a_i = 0,\ wt(d) \leq k_1 \right\},$$

$$S_2(\ell_2, k_2, q) \underset{\mathrm{def}}{=} \left\{ d = \sum_{i=0}^{\ell_2 - 1} a_i \phi^i \,\middle|\, a_i \in \mathbb{Z}, |a_i| < q^2/2, q \nmid a_i,\ a_{i+1}a_i = 0,\ wt(d) \leq k_2 \right\},$$

for an elliptic curve E defined over $\mathbb{F}_q$. For all $P \in E(\mathbb{F}_q^m)$, we always have $(\phi^m - 1)P = \mathcal{O}$, the point at infinity. So we sometimes identify ϕ^m and 1 as elements in $\mathbb{Z}[\phi]$, or we consider the endomorphisms on E in $\mathbb{Z}[\phi]/(\phi^m - 1)$ instead of $\mathbb{Z}[\phi]$. Thus we have to choose an appropriate value of ℓ_j in order to fulfill that for any two different elements in $S_j(\ell_j, k_j, q)$, these two elements are different from each other as endomorphisms on $G = \langle P \rangle$ (e.g., $\sum_{i=0}^{m-1} \phi^i = \sum_{i=0}^{m-1} 0\phi^i = 0$).

The following theorem guarantees the above situation if the range of ℓ_j is chosen appropriately.

Theorem 1. [6, Theorem 3] *For $j = 1, 2$ and $6 \leq \ell_j \leq \frac{\log_2 n - 2\log_2 M_j - \log_2 30}{\log_2 q}$, each endomorphism on G as an element in $S_j(\ell_j, k_j, q)$ is different from each other, where $M_1 = 2(q-1)$, $M_2 = 2\lfloor (q^2-1)/2 \rfloor$, and $k_j \leq \ell_j$.*

Under the condition of Theorem 1, the cardinalities of $S_j(\ell_j, k_j, q)$ are given by the following theorem.

Theorem 2. [6, Theorem 4] *$\#S_1(\ell_1, k_1, q) = \sum_{i=0}^{k_1} \binom{\ell_1+1-i}{i}(2q-2)^i$ and $\#S_2(\ell_2, k_2, q) = \sum_{i=0}^{k_2} \binom{\ell_2+1-i}{i}(q^2-q)^i$.*

We fix an extension degree m, and we choose the maximal ℓ_j such that the condition of Theorem 1 holds. Moreover, for each $k_{\mathrm{bv}} \in \mathbb{Z}$, we choose the maximal k_j so that $2^{k_{\mathrm{bv}}} \geq \#S_j(\ell_j, k_j, q)$. The complex exponents test check whether

$$\left(\sum_{i=1}^{N} s_i x_i \bmod n\right) P = \sum_{i=1}^{N} s_i Q_i \tag{1}$$

or not for randomly chosen $s_i \in S_j(\ell_j, k_j, q) (j = 1, 2)$. The naive test verifies whether $x_i P = Q_i$ or not for each i ($i \in \{1, \ldots, N\}$), i.e., N scalar multiplications with n **bits long scalars**. On the other hand, in order to reduce the computational cost of the CE test, Cheon and Lee apply BGMW method ([2]) to their CE test (See [6, Fig. 3] for detail). If we rewrite step 4 of [6, Fig. 3] (right-to-left BGMW method [2]) as the left-to-right version of BGMW method, the computational costs are

$$\big(k_1 N + 2(q-2)\big)\mathcal{A} + (q-1)\ell_1\mathcal{F} + (1 \text{ scalar multiplication}), \tag{2}$$

$$\begin{aligned} &\big(k_2 N + 2(\lfloor (q^2-1)/2 \rfloor + \lfloor (q-1)/2 \rfloor)\big)\mathcal{A} \\ &\quad + (\lfloor (q^2-1)/2 \rfloor + \lfloor (q-1)/2 \rfloor)\ell_2\mathcal{F} + (1 \text{ scalar multiplication}), \end{aligned} \tag{3}$$

respectively. Remark that the dominant operation in Equation (1) is multi-scalar multiplication (right hand side of Equation (1)). The computational cost of the multi-scalar multiplication depends heavily on the number of signatures N and q. If $\{(x_1, Q_1), \ldots, (x_N, Q_N)\}$ is a correct batch instance, Equation (1) is always true. The probability that the complex exponent test accepts an incorrect batch instance $\{(x_1, Q_1), \ldots, (x_N, Q_N)\}$ is at most $2^{-k_{\mathrm{bv}}}$ $(j = 1, 2)$ ([6, Theorem 1])D

3.4 Improved Complex Exponent Test

Cheon and Yi in [8] generalized the idea of the CE test. In order to improve the efficiency of the CE test, they use *the width-w τ-adic non-adjacent form* (cf. [13], [18], [28] for the explanation of τ-NAF and w-τ-NAF). Although the improved CE test is quite fast for verifying a large amount of signatures, the improved CE test is *not* suitable for verifying a limited number of signatures, because the precomputation cost is relatively large compared to that of the CE test. Therefore, we omit the details of the improved CE test, and we do not deal with the improved CE test any more. See [8] for details.

4 Proposed Methods

In this section, we propose two new methods. Due to space limitations, we only describe one of the new batch verification tests and omit the description of the other test. One can easily be obtained the other test in a similar way. For details, we refer to the *preproceedings version* of our paper ([16]). In Section 4.1, we show some properties of two types of elliptic curves ([24]) which are used in our proposed methods. In Section 4.2, we propose one of the new batch verification tests. Moreover we analyze the security of our batch verification test, and estimates the computational costs of the proposed method.

4.1 Properties of Two Types of Elliptic Curves

Proposition 1. [Endomorphism of elliptic curve $E_{p,b}/\mathbb{F}_p : y^2 = x^3 - b$ [10,19,24,25]] *Let p be a prime such that $p \equiv 1 \bmod 3$, and $E_{p,b} : y^2 = x^3 - b$ be the elliptic curve defined over $\mathbb{F}_p$. Suppose that $\beta \in \mathbb{F}_p$ is an element of order 3. Then the curve $E_{p,b}$ has the endomorphism $\omega : E_{p,b} \to E_{p,b}$, $(x, y) \mapsto (\beta x, y)$.*

Proposition 2. [Endomorphism of elliptic curve $E_{p,a}/\mathbb{F}_p : y^2 = x^3 + ax$ [10,19,24,25]] *Let p be a prime such that $p \equiv 1 \bmod 4$, and $E_{p,a} : y^2 = x^3 + ax$ be the elliptic curve defined over $\mathbb{F}_p$. Suppose that $\alpha \in \mathbb{F}_p$ is an element of order 4. Then the curve $E_{p,a}$ has the endomorphism $\lambda : E_{p,a} \to E_{p,a}$, $(x, y) \mapsto (-x, \alpha y)$.*

Park, Lee, and Park proved the existence of a Frobenius expansion on $E_{p,b}$ (resp. $E_{p,a}$). In the following, for an elliptic curve E over $\mathbb{F}_q$, we regard an endomorphism ψ as a complex number (See [25] for detail).

Proposition 3. [$D_{\mathbf{E}}$-Frobenius expansion [24, Theorem 2]] *Let $E_{7,b} : y^2 = x^3 - b$ be the elliptic curve defined over $\mathbb{F}_7$, $\beta \in \mathbb{F}_7$ be an element of order 3, and ϕ be the 7th-power Frobenius map on $E_{7,b}$. For any $d \in \mathbb{Z}[\omega]$, we can write $d = \sum_{i=0}^{\ell-1} c_i \phi^i$, where $c_i \in D_E := \{0, \pm 1, \pm\omega, \pm\omega^2\}$, $\ell \leq \lceil 2 \log_7 \sqrt{N_{\mathbb{Z}[\omega]/\mathbb{Z}}(d)} \rceil + 1$.*

Proposition 4. [$D_{\mathbf{G}}$-Frobenius expansion [24, Theorem 1]] *Let $E_{5,a} : y^2 = x^3 + ax$ be the elliptic curve defined over $\mathbb{F}_5$, $\alpha \in \mathbb{F}_5$ be an element of order 4, and ϕ be the 5th-power Frobenius map on $E_{5,a}$. For any $d \in \mathbb{Z}[\sqrt{-1}]$, we can write $d = \sum_{i=0}^{\ell-1} c_i \phi^i$, where $c_i \in D_G := \{0, \pm 1, \pm\lambda\}$, $\ell \leq \lceil 2 \log_5 \sqrt{N_{\mathbb{Z}[\lambda]/\mathbb{Z}}(d)} \rceil + 1$.*

We define

$$S_{\mathrm{E}}(\ell_\omega, k_\omega, q) := \left\{ d = \sum_{i=0}^{\ell_\omega - 1} a_i \phi^i \,\middle|\, a_i \in D_{\mathrm{E}}, wt(d) \leq k_\omega \right\}, \tag{4}$$

for an elliptic curve $E_{7,b}$, and

$$S_{\mathrm{G}}(\ell_\lambda, k_\lambda, q) := \left\{ d = \sum_{i=0}^{\ell_\lambda - 1} a_i \phi^i \,\middle|\, a_i \in D_{\mathrm{G}}, wt(d) \leq k_\lambda \right\}, \tag{5}$$

for an elliptic curve $E_{5,a}$. The proposed methods are modifications of the complex exponent test.

Roughly speaking, our main observation is as follows. Since each number s_i is randomly chosen from the set $S_1(\ell_1, k_1, q)$ (resp. $S_2(\ell_2, k_2, q)$) in the complex exponent test, the number of point additions in the right hand side of Equation (1) (dominant part) depends heavily on the number of signatures N and q. However, each number s_i is randomly chosen from the set $S_{\mathrm{E}}(\ell_\omega, k_\omega, q)$ (resp. $S_{\mathrm{G}}(\ell_\lambda, k_\lambda, q)$) in our proposed method, the number of point additions in the right hand side of Equation (1) depends only on the number of signatures N.

In order to evaluate the security of our method, we have to evaluate an upper bound of ℓ_ω (resp. ℓ_λ) which satisfies that each element in the set S_{E} (resp. S_{D}) is different from each other as an element in $G = \langle P \rangle$. We also have to calculate the cardinality of the set S_{E} (resp. S_{D}). Namely, we need analogous results of [6, Theorem 3] and [6, Theorem 4] in Section 3.3. In Section 4.2, we evaluate $\#S_{\mathrm{E}}(\ell_\omega, k_\omega, q)$.

4.2 Proposed Test Using the Curve $E_{7,b}$

We show the classification of the elliptic curves of the form $E_{7,b}/\mathbb{F}_7 : y^2 = x^3 - b$ in Table 1 (See [19] and [24] for details).

Table 1. Classification of elliptic curves of the form $E_{7,b}$ (cf. [19], [24])

b	t	ϕ	relation between ϕ and ω
1	4	$2 \pm \sqrt{-3}$	$\phi - 2\omega = 3$ or $\phi + 2\omega = 1$
-1	-4	$-2 \pm \sqrt{-3}$	$\phi - 2\omega = -1$ or $\phi + 2\omega = -3$
2	1	$\frac{1 \pm 3\sqrt{-3}}{2}$	$\phi - 3\omega = 2$ or $\phi + 3\omega = -1$
-2	-1	$\frac{-1 \pm 3\sqrt{-3}}{2}$	$\phi - 3\omega = 1$ or $\phi + 3\omega = -2$
3	5	$\frac{5 \pm \sqrt{-3}}{2}$	$\phi - \omega = 3$ or $\phi + \omega = 2$
-3	-5	$\frac{-5 \pm \sqrt{-3}}{2}$	$\phi - \omega = -2$ or $\phi + \omega = -3$

Indeed, it can be shown that the Frobenius expansion with coefficients in D_{E} in $\mathbb{Z}[\omega]$ have unique representation. For the proof of the uniqueness, we prepare the following lemma.

Lemma 1. *Let $\alpha = x + y\omega \in \mathbb{Z}[\omega]$ $(x, y \in \mathbb{Z})$. Then, the following equivalences hold:*

- *If $b = 1$, then $\phi | \alpha \iff 7 | (x + 2y)$.*
- *If $b = -1$, then $\phi | \alpha \iff 7 | (2x + y)$.*
- *If $b = 2$, then $\phi | \alpha \iff 7 | (x - 3y)$.*
- *If $b = -2$, then $\phi | \alpha \iff 7 | (3x - y)$.*
- *If $b = 3$, then $\phi | \alpha \iff 7 | (x - 3y)$.*

– *If $b = -3$, then $\phi|\alpha \Leftrightarrow 7|(x + 2y)$.*

In particular, for a rational integer $\alpha \in \mathbb{Z}$, we have $\phi|\alpha \Leftrightarrow 7|x$.

Proof. The proof can be found in [16]. □

Proposition 5. [Uniqueness of the Frobenius expansion with coefficients $D_{\mathbf{E}}$] *Every $d \in \mathbb{Z}[\omega]$ has a unique Frobenius expansion with coefficients in D_E.*

Proof. The proof can be found in [16]. □

Note that if $b = \pm 3$ then any D_{E}-coefficients Frobenius expansion can be rewritten as $\mathbb{Z}$-coefficients Frobenius expansion, and if $b = \pm 1, \pm 2$ then any D_{E}-coefficients Frobenius expansion can be rewritten as $\mathbb{Q}$-coefficients Frobenius expansion. Moreover, if $b = \pm 1$, denominator of each coefficient of $\mathbb{Q}$-coefficients Frobenius expansion is 1 or 2. In a similar way, if $b = \pm 2$, denominator of each coefficient of $\mathbb{Q}$-coefficients Frobenius expansion is 1 or 3.

For any D_{E}-coefficients Frobenius expansion with appropriate length, it can be proven that each Frobenius expansion as above is different from each other in $\mathbb{Z}[\omega]/(\phi^m - 1)$.

Proposition 6. *Let $M \in \mathbb{Z}_{>0}$, $\ell_\omega \geq 4$. Let $C_b \in \mathbb{Z}$ be a rational integer such that if $b = \pm 1$ then $C_b = 2$, if $b = \pm 2$ then $C_b = 3$, and if $b = \pm 3$ then $C_b = 1$. We put $f(X) = \sum_{i=0}^{\ell_\omega - 1} a_i X^i \in \mathbb{C}[X]$, $a_i \in D_E$ and $M = 8$. We assume that $C_b \times 68 \times 7^{\ell_\omega} \times M^2 \leq n$. Then $f(\phi)P = \mathcal{O}$ implies that $C_b \times f(X)$ is divided by $X^2 - tX + 7$ in $\mathbb{Z}[X]$.*

Proof. The proof can be found in [16]. □

If ℓ_ω satisfies the condition of Proposition 6, each element in $S_{\mathrm{E}}(\ell_\omega, k_\omega, 7)$ is different from each other.

Theorem 3. [Upper bound of the length of the $D_{\mathbf{E}}$-coefficients Frobenius expansion] *Let $4 \leq \ell_\omega \leq \frac{\log_2 n - 2\log_2 M - \log_2 68 - \log_2 C_b}{\log_2 7}$, $M = 8$. Then each element in $S_E(\ell_\omega, k_\omega, 7)$ is different endomorphism on G.*

Proof. The proof can be found in [16]. □

By Theorem 3, if $C_b \times 68 \times 7^{\ell_\omega} \times 8^2 \leq n$, the cardinality of the set S_{E} is given by the following theorem.

Theorem 4. [Cardinality of the set $S_{\mathbf{E}}$]

$$\#S_E(\ell_\omega, k_\omega, 7) = \sum_{i=0}^{k_\omega} \binom{\ell_\omega}{i} 6^i.$$

Proof. By elementary combinatorics, we obtain the above equation. □

Our proposed test using the curve $E_{7,b}$ is described in Algorithm 1.

Algorithm 1. Propsed Batch Verification Test using the curve $E_{7,b}$

```
Input: batch instance $\{(x_1, Q_1), \dots, (x_N, Q_N)\}$
Output: Accept or Reject
 1: Choose $N$ random elements $s_1, \dots, s_N$ from $S_{\mathrm{E}}(\ell_\omega, k_\omega, 7)$.
    Write $s_i = \sum_{j=0}^{\ell_\omega - 1} c_{i,j}\phi^j$ and $\epsilon_{i,j} = 0$ (if $c_{i,j} = 0$), $\epsilon_{i,j} = 1$ (if $c_{i,j} \in \{1, \omega, \omega^2\}$),
    $\epsilon_{i,j} = -1$ (if $c_{i,j} \in \{-1, -\omega, -\omega^2\}$) for nonzero $c_{i,j}$ for each $i$.
 2: $s \leftarrow \sum_{i=1}^{N} s_i x_i \bmod (\phi^{m-1} + \phi^{m-2} + \cdots + \phi + 1)$
 3: $R[i] \leftarrow \mathcal{O}$ for $i \in \{1, \omega, \omega^2\}$
 4: for $j$ from $\ell_\omega - 1$ downto 0 do
 5:    $R[1] \leftarrow \phi(R[1])$, $R[\omega] \leftarrow \phi(R[\omega])$, $R[\omega^2] \leftarrow \phi(R[\omega^2])$
 6:    for $i$ from 1 to $N$ do
 7:       if $c_{i,j} \neq 0$ then
 8:          if $c_{i,j} = \pm 1$ then
 9:             $R[1] \leftarrow R[1] + \epsilon_{i,j}(Q_i)$
10:          else if $c_{i,j} = \pm\omega$ then
11:             $R[\omega] \leftarrow R[\omega] + \epsilon_{i,j}(Q_i)$
12:          else
13:             $R[\omega^2] \leftarrow R[\omega^2] + \epsilon_{i,j}(Q_i)$
14:          end if
15:       end if
16:    end for
17: end for
18: $Q \leftarrow \omega(R[\omega^2])$
19: $Q \leftarrow \omega(R[\omega] + Q)$
20: $Q \leftarrow Q + R[1]$
21: if $Q = sP$ then
22:    return Accept
23: else
24:    return Reject
25: end if
```

By the same argument as Section 3.3, the computational cost of the proposed test using the Curve $E_{7,b}$ is given by

$$\big(k_\omega N + 2\big)\mathcal{A} + 3\ell_\omega\mathcal{F} + 2M + (1 \text{ scalar multiplication}). \tag{6}$$

5 Comparison

5.1 Timings

In order to evaluate the computational costs of the CE test and the proposed test, we implement arithmetic operations a finite field $\mathbb{F}_{7^{103}}$ (See Table 7 in Appendix A). We use $f(x) = x^{103} + x^{54} + 6$ and $E_{7,-3}$-103 (Table 7) for the irreducible trinomial of degree 103 and the estimation of the computational costs, respectively. The bit length of the prime order n of this curve is 286. For the finite field implementation, we use polynomial basis.

The platform was a Intel® [1] Core$^{\mathrm{TM}}$2 Duo Processor E8400 (2.99GHz) with 2GB RAM computer, Windows® [2] XP. Programs were all written in ANSI C language with gcc 3.4.4 compiler using the flags "-O3".

In the case of $p = 3$, the three elements in $\mathbb{F}_3$ are represened by two bits, and it is known that the method to construct the addition in $\mathbb{F}_3$ by using bitwise operations such as "AND" operation or "XOR" operation (cf. [14])D In our implementation, the seven elements in $\mathbb{F}_7$ are represened by three bits, and we extend the method in [14] to implement the arithmetic operations in $\mathbb{F}_{7^{103}}$. The timings are listed in Table 2.

Table 2. Timings of the Finite Field Operations

Finite Field Operations	timing (in μsec)
Addition	0.063
Subtraction	0.063
7-th powering	1.400
Multiplication	15.230

5.2 Estimate

We estimate batch verification costs of the CE test and our proposed test in the case of $E_{7,-3}$-103 (See Table 7 in Appendix A). The computational cost of the CE tests are given by (2)C(3), respectively. The computational cost of the proposed test using $E_{7,b}$ is given by (6). From Theorem 1 and Theorem 3, for n for $E_{7,-3}$-103 in Table 7, the maximal value of ℓ_1, ℓ_2, ℓ_ω are $\ell_1 = 97$, $\ell_2 = 96$, $\ell_\omega = 98$, respectively. We use the Jacobian coordinate for point addition (12 times finite field multiplications and four times squarings), and the Jacobian coordinate for Frobenius map (three times 7th-powerings) for the estimation. Moreover, it is assumed that

- the costs of finite field multiplication and finite field squaring are equal to each other.
- the cost of a 7th powering and 0.092 times finite field multiplication are equal (from the result in Table 2).

We show the numbers of finite field multiplications that need for the CE test and the proposed test in Table 3 by using (2), (3), and (6) [3] .

[1] Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

[2] Intel, Core are trademarks or a registered trademarks of Intel Corporation in the United States and other countries.

[3] In order to simplify the computational costs of the algorithm in [6, Fig. 3] and our proposed test (Algorithm 1), we use left-to-right version of BGMW method. However, even if we compare the computational costs of the algorithm in [6, Fig. 3] and the right-to-left version of Algorithm 1, our algorithm is faster than the algorithm [6, Fig. 3] for verifying a limited number of signatures because the number of the Frobenius map in the right-to-left version of Algorithm 1 is larger than that of [6, Fig. 3].

Table 3. Estimate the computational costs of CE test and our test

Method	# of M
CE test (S_1)	$16k_1N + 284.21$
CE test (S_2)	$16k_2N + 1222.72$
Our test ($S_{\mathbf{E}}$)	$16k_\omega N + 59.04$

For $\ell_1 = 97$, $\ell_2 = 96$, $\ell_\omega = 98$ and for a given $k_{\rm bv}$, we describe the values k_1, k_2, k_ω so that $\#S_j \approx 2^{k_{\rm bv}}$, $\#S_{\rm E} \approx 2^{k_{\rm bv}}$ in Table 4.

Table 4. Values k_j and k_ω such that $\#S_j \approx 2^{k_{\rm bv}}$, $\#S_{\rm E} \approx 2^{k_{\rm bv}}$

Method \ Security Param. $k_{\rm bv}$	40	60	80	100	120	140
CE test (S_1)	5	8	11	14	18	22
CE test (S_2)	4	6	9	11	14	16
Our test ($S_{\rm E}$)	6	9	12	16	21	25

For a given method in Table 3, a given number of signatures N, and a given security parameter $k_{\rm bv}$, we can estimate the number of multiplications in $\mathbb{F}_{7^{103}}$ by substituting N and k_j or k_ω in Table 4 into the method in Table 3. Our method is faster than the complex exponent test when the number of signatures N satisfy Table 5:

Table 5. the number of signatures N for which our method is faster than CE test

Security Param. $k_{\rm bv}$	40	60	80	100	120	140
# of signatures	$N \leq 18$	$N \leq 18$	$N \leq 18$	$N \leq 9$	$N \leq 6$	$N \leq 6$

From Table 5, we can see that our test is faster than the CE test when the number of signatures is relatively small, Thus, our proposed test is suitable for verifying a limited number of signatures in real-time such as IP camera surveillance systems.

6 Conclusion

In this paper, we described a new batch verification test based on elliptic curves defined over fields with characteristics 5 and 7. Our proposed method is a modification of the complex exponent test (CE test) proposed by Cheon and Lee. The difference between the new test and the CE test is that our test use another digit set of Frobenius expansion in order to accelerate batch verification. We evaluated the security of our new test, and the proof of the security of our new test can be treated similar to the one of CE test. We also evaluated the computational cost of the new test. Our estimation indicates that the proposed test is suitable for verifying a limited number of signatures.

References

1. Antipa, A., Brown, D., Gallant, R., Lambert, R., Struik, R., Vanstone, S.: Accelerated Verification of ECDSA Signatures. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 307–318. Springer, Heidelberg (2006)
2. Brickell, E.F., Gordon, D.M., McCurley, K.S., Wilson, D.B.: Fast Exponentiation with Precomputation. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 200–207. Springer, Heidelberg (1993)
3. Bellare, M., Garay, J.A., Rabin, T.: Fast Batch Verification for Modular Exponentiation and Digital Signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 236–250. Springer, Heidelberg (1998), `http://www-cse.ucsd.edu/users/mihir`
4. Boyd, C., Pavlovski, C.: Attacking and Repairing Batch Verification Schemes. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 58–71. Springer, Heidelberg (2000)
5. Camenisch, J., Hohenberger, S., Pedersen, M.Ø.: Batch Verification of Short Signatures. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 246–263. Springer, Heidelberg (2007)
6. Cheon, J.H., Lee, D.H.: Use of Sparse and/or Complex Exponents in Batch Verification of Exponentiations. IEEE Transactions on Computers 55(12), 1536–1542 (2006)
7. Coron, J.-S., Naccache, D.: On the Security of RSA Screening. In: Imai, H., Zheng, Y. (eds.) PKC 1999. LNCS, vol. 1560, pp. 197–203. Springer, Heidelberg (1999)
8. Cheon, J.H., Yi, J.H.: Fast Batch Verification of Multiple Signatures. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 442–457. Springer, Heidelberg (2007)
9. Guo, F., Mu, Y., Chen, Z.: Efficient Batch Verification of Short Signatures for a Single-Signer Setting without Random Oracles. In: Matsuura, K., Fujisaki, E. (eds.) IWSEC 2008. LNCS, vol. 5312, pp. 49–63. Springer, Heidelberg (2008)
10. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 190–200. Springer, Heidelberg (2001)
11. Harn, L.: Batch Verifying Multiple DSA Digital Signatures. Electronics Letters 34(9), 870–871 (1998)
12. Harn, L.: Batch Verifying Multiple RSA Digital Signatures. Electronics Letters 34(12), 1219–1220 (1998)
13. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, New York (2004)
14. Harrison, K., Page, D., Smart, N.P.: Software Implementation of Finite Fields of Characteristic Three, for Use in Pairing-based Cryptosystems. LMS Journal of Computation and Mathematics 5(1), 181–193 (2002)
15. Hwang, M.S., Lee, C.C., Lu, E.J.: Cryptanalysis of the Batch Verifying Multiple DSA-Type Digital Signatures. Pakistan Journal of Applied Sciences 1(3), 287–288 (2001)
16. Hakuta, K., Katoh, Y., Sato, H., Takagi, T.: Batch Verification Suitable for Efficiently Verifying A Limited Number of Signatures. In: Preproceedings of the 15th Annual International Conference on Information Security and Cryptology, ICISC 2012 (2012)
17. Hakuta, K., Sato, H., Takagi, T.: Efficient arithmetic on subfield elliptic curves over small finite fields of odd characteristic. J. Math. Cryptol. 4(3), 199–238 (2010)

18. Hakuta, K., Sato, H., Takagi, T.: Explicit lower bound for the length of minimal weight τ-adic expansions on Koblitz curves. J. Math-for-Ind. 2A, 75–83 (2010)
19. Koblitz, N.: An Elliptic Curve Implementation of the Finite Field Digital Signature Algorithm. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 327–337. Springer, Heidelberg (1998)
20. National Institute for Standards and Technology, Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186-3 (June 2009), `http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf`
21. Lee, S., Cho, S., Choi, J., Cho, J.: Efficient Identification of Bad Signatures in RSA-Type Batch Signatures. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E89-A(1), 74–80 (2006)
22. Naccache, D., M'Raïhi, D., Vaudenay, S., Raphaeli, D.: Can D.S.A be Improved? Complexity trade-offs with the Digital Signature Standard. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 77–85. Springer, Heidelberg (1995)
23. Pastuszak, J., Michałek, D., Pieprzyk, J., Seberry, J.: Identification of Bad Signatures in Batches. In: Imai, H., Zheng, Y. (eds.) PKC 2000. LNCS, vol. 1751, pp. 28–45. Springer, Heidelberg (2000)
24. Park, T.-J., Lee, M.-K., Park, K.: New Frobenius Expansions for Elliptic Curves with Efficient Endomorphisms. In: Lee, P.J., Lim, C.H. (eds.) ICISC 2002. LNCS, vol. 2587, pp. 264–282. Springer, Heidelberg (2003)
25. Silverman, J.H.: The Arithmetic of Elliptic Curves. GTM, vol. 106. Springer (1986)
26. Smart, N.P.: Elliptic Curve Cryptosystems over Small Fields of Odd Characteristic. Journal of Cryptology 12(2), 141–151 (1999)
27. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology 4(3), 161–174 (1991)
28. Solinas, J.A.: Efficient Arithmetic on Koblitz Curves. Des. Codes Cryptogr. 19(2-3), 195–249 (2000)
29. Stanek, M.: Attacking LCCC Batch Verification of RSA Signatures. International Journal of Network Security 6(2), 238–240 (2008)
30. Kumar, V., Madria, S.: Secure Data Aggregation in Wireless Sensor Networks. In: Hara, T., Zadorozhny, V.I., Buchmann, E. (eds.) Wireless Sensor Network Technologies for the Information Explosion Era. SCI, vol. 278, pp. 77–107. Springer, Heidelberg (2010)
31. Yen, S.M., Laih, C.S.: Improved Digital Signature Suitable for Batch Verification. IEEE Transactions on Computers 44(7), 957–959 (1995)

A Sample Parameters

Table 6 and Table 7 list sample parameters for an elliptic curve $E_{5,a}/\mathbb{F}_5$ and an elliptic curve $E_{7,a}/\mathbb{F}_7$, respectively. The extension degree m's are prime and are selected so that there exists the elliptic curve $E_{5,b}/\mathbb{F}_5$ or an elliptic curve $E_{7,a}/\mathbb{F}_7$ having the cofactor $\#E_{5,a}(\mathbb{F}_5)$ or $\#E_{7,b}(\mathbb{F}_7)$, respectively. The prime order n is presented in hexadecimal form. A backslash at the end of a line indicates that the number (hexadecimal) is continued in the next line.

m	The extension degree of the finite field $\mathbb{F}_{5^m}$ (resp. $\mathbb{F}_{7^m}$).
$f(x)$	The irreducible trinomial of degree m in $\mathbb{F}_5[x]$ (resp. $\mathbb{F}_7[x]$).
h	The cofactor. $h = \#E(\mathbb{F}_5)$ (resp. $\#E(\mathbb{F}_7)$).
n	$n = \#E(\mathbb{F}_{5^m})/h$ (resp. $\#E(\mathbb{F}_{7^m})/h$).

Table 6. Sample Parameters for the elliptic curves $E_{5,a}/\mathbb{F}_5$

$E_{5,2}$-97: $m = 97$, $f(x) = x^{97} + x^{58} + 2$, $a = 2$, $h = 2$, $\lvert h\rvert = 2$, $\lvert n\rvert = 225$						
$n =$ 0x	00000001	2BA095DC	7701D9CB	7743E3A2	B0E3BDBC	E284B04F\
	367F1914	ED5FBA01				
$E_{5,2}$-107: $m = 107$, $f(x) = x^{107} + x^{9} + 1$, $a = 2$, $h = 2$, $\lvert h\rvert = 2$, $\lvert n\rvert = 248$						
$n =$ 0x	00AE67F1	E9AEC071	87ECD859	0680A3AA	02A9DFE5	BD41A836\
	BA41FAF8	6F4E279D				
$E_{5,-2}$-151: $m = 151$, $f(x) = x^{151} + x^{61} + 1$, $a = -2$, $h = 10$, $\lvert h\rvert = 4$, $\lvert n\rvert = 348$						
$n =$ 0x	09C69A97	284B578D	7FF2A760	414536EF	BCA758CB	F4FBCB0A\
	32CC8363	8555BB9A	84122DF7	C08209C8	5F656D4D	
$E_{5,-2}$-167: $m = 167$, $f(x) = x^{167} + x^{66} + 1$, $a = -2$, $h = 10$, $\lvert h\rvert = 4$, $\lvert n\rvert = 385$						
$n =$ 0x	00000001	5B4E5998	400A95D3	5EAC354F	34215CD4	6E417018\
	FB1DC739	C5E736BD	C153819F	71B64393	465FC46B	F4AB38FF\
	A407344D					
$E_{5,1}$-227: $m = 227$, $f(x) = x^{227} + x^{53} + 1$, $a = 1$, $h = 4$, $\lvert h\rvert = 3$, $\lvert n\rvert = 526$						
$n =$ 0x	000021C5	29DD78FA	571E196B	3EBB0D20	429C476A	1848CAB5\
	E0E8A121	378DE187	888F99D2	99F404EE	4F9BC974	D5035A62\
	AC9F5E1E	0DA29A51	0B4012E2	3ECD1590	9A4B1065	
$E_{5,-2}$-317: $m = 317$, $f(x) = x^{317} + x^{24} + 4$, $a = -2$, $h = 10$, $\lvert h\rvert = 4$, $\lvert n\rvert = 733$						
$n =$ 0x	1A8662F3	B3919708	2BF4C0D2	548C2A3D	779053F3	CF932023\
	7E5EC14C	922CB561	85BD758D	2B99F28C	ADBC9799	509367D4\
	E379DA56	D0582105	28C0C11C	7B00363E	9042A70F	4F3A60D4\
	8CE325D4	4171E2E9	68316505	D41F503A	36E361D1	
$E_{5,-2}$-439: $m = 439$, $f(x) = x^{439} + 4x^{35} + 1$, $a = -2$, $h = 10$, $\lvert h\rvert = 4$, $\lvert n\rvert = 1017$						
$n =$ 0x	0100CCFD	6D32E361	2A7F4535	9B775336	206E4053	5E362EC5\
	7D5C0F1E	B832C2D9	C2B8B468	3B54F5EC	018C1A10	A8DECBFE\
	C96B211E	E977B7C4	6DCEF7E2	C81A8F0E	7DDFC28A	051C3168\
	8DA47D5C	7E6B890F	F1F5A336	8CAFE22B	9A5A9357	7377B7A8\
	342935DB	09E38487	A25E300C	5F681788	B3A4E79A	331AE55C\
	5091770A	DF9A5B4D				

Table 7. Sample Parameters for the elliptic curves $E_{7,b}/\mathbb{F}_7$

$E_{7,3}$-71: $m = 71,\ f(x) = x^{71} + x^{10} + 4,\ b = 3,\ h = 3,\ \lvert h\rvert = 2,\ \lvert n\rvert = 198$						
$n =$ 0x	00000035	57E6DA5B	FB1EC95B	4614A377	DBAF4B91	89919638\
	64505C87					
$E_{7,-3}$-103: $m = 103,\ f(x) = x^{103} + x^{54} + 6,\ b = -3,\ h = 13,\ \lvert h\rvert = 4,\ \lvert n\rvert = 286$						
$n =$ 0x	2BEDF1E0	41D10B7B	55B514D7	9FE59CB4	0B53BF6B	9DB9FE62\
	3B89B385	159E5565	1BA268D5			
$E_{7,1}$-127: $m = 127,\ f(x) = x^{127} + x^{2} + 6,\ b = 1,\ h = 4,\ \lvert h\rvert = 3,\ \lvert n\rvert = 355$						
$n =$ 0x	00000005	CAC4104D	859A6DF5	82D57312	11D9947A	4AE9CFD1\
	F4E36489	97D050DC	E03624B8	91381F19	AA1824CF	98DE5637
$E_{7,3}$-167: $m = 167,\ f(x) = x^{167} + 3x^{39} + 1,\ b = 3,\ h = 3,\ \lvert h\rvert = 2,\ \lvert n\rvert = 468$						
$n =$ 0x	0009783A	48EDC313	BA408497	7B4E4849	883BF3D0	8EC233E6\
	2CB37954	33A61505	385CAF38	732C2337	EDA85316	2CA773D5\
	1053D953	7248F3EA	7C290A47			
$E_{7,2}$-181: $m = 181,\ f(x) = x^{181} + x^{5} + 3,\ b = 2,\ h = 7,\ \lvert h\rvert = 3,\ \lvert n\rvert = 506$						
$n =$ 0x	0280DE7F	5E056B02	C82C2CFE	C4038DDB	6CA57BF4	2FBB56FB\
	F74DAADB	65B85853	0E19FF02	B0B2748C	C32514A1	4F5975AB\
	53254133	EC718BF1	50C86198	19A6FF59		
$E_{7,-3}$-191: $m = 191,\ f(x) = x^{191} + 3x^{31} + 1,\ b = -3,\ h = 13,\ \lvert h\rvert = 4,\ \lvert n\rvert = 533$						
$n =$ 0x	0016B21C	CFE2FB4E	0B541889	60D8B31F	5711D7D5	A8AE5602\
	AEEDEA73	33DAAC30	8706656A	62F57B75	1A9FFBD9	D470E482\
	0D7CE237	021A828C	B8BD9503	4AE2D557	ECD12537	
$E_{7,1}$-223: $m = 223,\ f(x) = x^{223} + 4x^{16} + 6,\ b = 1,\ h = 4,\ \lvert h\rvert = 3,\ \lvert n\rvert = 625$						
$n =$ 0x	00010739	5CC42964	0E1C6468	886BEC6F	6478026C	3BB31334\
	297DC5EC	9BB482E1	748405E2	807FB6D7	1CB4BE06	F6CD1379\
	5ABC7679	0E24ABC6	0B4CF9BB	65600766	528640F7	EE46D8EF\
	61ECB724	6B049C97				
$E_{7,1}$-383: $m = 383,\ f(x) = x^{383} + x^{24} + 6,\ b = 1,\ h = 4,\ \lvert h\rvert = 3,\ \lvert n\rvert = 1074$						
$n =$ 0x	00025313	FC8FCB0A	1214B25F	E5235E4E	98F78982	632A525B\
	57FC01BB	C71B6491	C54FFF95	E9822C33	9B81C0A3	57472671\
	0F6B4A58	9FB61465	A3450E13	FF323D32	ECD4F0F5	EC177CD6\
	6EA567B4	EC0BDD2F	6DF59C4C	2CC17741	373453DD	06EB3E66\
	63D4D191	86294806	59E26EA1	20C39651	43F3B8AC	66FC3262\
	7061D995	046ADEA9	198BB1B1	A1E92737		

Linear Recurring Sequences for the UOV Key Generation Revisited

Albrecht Petzoldt[1] and Stanislav Bulygin[2]

[1] Technische Universität Darmstadt, Department of Computer Science
Hochschulstraße 10, 64289 Darmstadt, Germany
apetzoldt@cdc.informatik.tu-darmstadt.de
[2] Center for Advanced Security Research Darmstadt - CASED
Mornewegstraße 32, 64293 Darmstadt, Germany
Stanislav.Bulygin@cased.de

Abstract. Multivariate cryptography is one of the main candidates to guarantee the security of communication in a post quantum era. While multivariate signature schemes are very fast and require only modest computational resources, the key sizes of such schemes are quite large. In [17] Petzoldt et al. proposed a way to use Linear Recurring Sequences (LRS's) for the key generation of the Unbalanced Oil and Vinegar (UOV) signature scheme by which they were able to reduce the public key size of this scheme by a factor of 7. In this paper we describe a modification of their scheme, which enables us not only to reduce the public key size, but also to speed up the verification process of the UOV scheme by a factor of 5.

Keywords: Multivariate Cryptography, UOV Signature Scheme, Key Size Reduction, Fast Verification.

1 Introduction

When quantum computers arrive, classical public-key cryptosystems like RSA and ECC will be broken [1]. The reason for this is Shor's algorithm [18] which solves number theoretic problems like integer factorization and discrete logarithms in polynomial time on a quantum computer. So, to guarantee the security of communication in a post quantum era, we need alternatives to those classical schemes. Besides lattice-, code-, and hash-based cryptosystems, multivariate cryptography seems to be a candidate for this.

Additionally to its (believed) resistance against quantum computer attacks, multivariate cryptosystems are very fast, especially for signatures [3, 5]. Furthermore they require only modest computational resources, which makes them attractive for the use on low-cost devices like smartcards and RFID chips. However, multivariate schemes are not widely used yet, mainly because of the large size of their public and private keys.

In [17] Petzoldt et al. proposed to use Linear Recurring Sequences (LRS) for the key generation of the Unbalanced Oil and Vinegar (UOV) Signature Scheme.

They did this by inserting a matrix B generated by an LRS into the coefficient matrix of the public key. Therefore, the Macauley matrix of the public key has the form $M_P = (B|C)$, where C is a matrix without visible structure. By doing so, they were able to decrease the public key size of UOV by a factor of 7, namely from 100 kB to about 14 kB.

In this paper we propose a variation of their scheme, which not only decreases the size of the public key, but also enables us to speed up the verification process. We show how to use the large structure of the matrix B to reduce the number of field multiplications needed during the verification process by a factor of 5. We derive our results both theoretically and show them using a C implementation of the scheme.

The structure of this paper is as follows: Section 2 gives a very short introduction on Linear Recurring Sequences (LRS). In Section 3 we give an overview on multivariate signature schemes and describe the UOV signature scheme. Section 4 reviews the approach of [16] to create UOV schemes with structured public keys . In Section 5 we describe our new approach in detail. Furthermore we look at the security of our scheme and consider the question how to choose the parameters of it. Section 6 demonstrates, how we can use the special structure of our polynomials to speed up the verification process. Finally, Section 7 presents the results of our computer experiments and Section 8 concludes the paper.

2 Linear Recurring Sequences (LRS)

In this section we repeat briefly results from the theory of linear recurring sequences (LRS's) needed in the following sections. For a more detailed introduction we refer to [13].

Definition 1. *Let L be a positive integer and $\gamma_1, \dots, \gamma_L$ be elements of a finite field $\mathbb{F}$. A Linear Recurring Sequence (LRS) of length L is a sequence $\{s_1, s_2, \dots\}$ of $\mathbb{F}$-elements satisfying the relation*

$$s_j = \gamma_1 \cdot s_{j-1} + \gamma_2 \cdot s_{j-2} + \cdots + \gamma_L \cdot s_{j-L} = \sum_{i=1}^{L} \gamma_i \cdot s_{j-i} \quad (\forall j > L). \qquad (1)$$

The values $s_1, \dots, s_L$ are called the initial values of the LRS.

Definition 2. *The connection polynomial of an LRS is defined as*

$$C(X) = \gamma_L \cdot X^L + \gamma_{L-1} \cdot X^{L-1} + \cdots + \gamma_1 \cdot X + 1 = \sum_{i=1}^{L} \gamma_i X^i + 1.$$

The LRS S is uniquely determined by its initial values $s_1, \dots, s_L$ and the connection polynomial C (due to equation (1)). Therefore we denote the LRS by $S = LRS(s_1, \dots, s_L, C)$.

3 Multivariate Public Key Cryptography

The basic idea behind multivariate cryptography is to choose a system $\mathcal{F}$ of m quadratic polynomials in n variables over a finite field $\mathbb{F}$ which can be easily inverted (central map). After that one chooses two affine invertible maps $\mathcal{S}$ and $\mathcal{T}$ to hide the structure of the central map. The public key of the cryptosystem is the composed quadratic map $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$ which is (hopefully) difficult to invert. The private key consists of $\mathcal{S}$, $\mathcal{F}$ and $\mathcal{T}$ and therefore allows to invert $\mathcal{P}$.

Due to this construction, the security of multivariate cryptography is based on two mathematical problems:

Problem MQ: Solve the system $p_1 = \ldots = p_m = 0$, where each p_i is a quadratic polynomial in the n variables $x_1, \ldots, x_n$ with coefficients and variables in $\mathbb{F}$.

The MQ-problem is proven to be NP-hard even for quadratic polynomials over $GF(2)$ [10].

Problem EIP (Extended Isomorphism of Polynomials): Given a class of central maps $\mathcal{C}$ and a map $\mathcal{P}$ expressible as $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$, where $\mathcal{S}$ and $\mathcal{T}$ are affine maps and $\mathcal{F} \in \mathcal{C}$, find a decomposition of $\mathcal{P}$ of the form $\mathcal{P} = \mathcal{S}' \circ \mathcal{F}' \circ \mathcal{T}'$, with affine maps $\mathcal{S}'$ and $\mathcal{T}'$ and $\mathcal{F}' \in \mathcal{C}$.

In this paper we concentrate on the case of multivariate signature schemes. The standard process for signature generation and verification works as follows:

$$\begin{tikzcd}
d \arrow[r, "\mathcal{H}"] & \mathbf{h} \in \mathbb{F}^m \arrow[r, "\mathcal{S}^{-1}"] & \mathbf{x} \in \mathbb{F}^m \arrow[r, "\mathcal{F}^{-1}"] & \mathbf{y} \in \mathbb{F}^n \arrow[r, "\mathcal{T}^{-1}"] & \mathbf{z} \in \mathbb{F}^n \arrow[lll, bend left=60, "\mathcal{P}"]
\end{tikzcd}$$

Fig. 1. Signature generation and verification

Signature Generation. To sign a document d, we use a hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{F}^m$ to compute the value $\mathbf{h} = \mathcal{H}(d) \in \mathbb{F}^m$. Then we compute $\mathbf{x} = \mathcal{S}^{-1}(\mathbf{h})$, $\mathbf{y} = \mathcal{F}^{-1}(\mathbf{x})$ and $\mathbf{z} = \mathcal{T}^{-1}(\mathbf{y})$. The signature of the document is $\mathbf{z} \in \mathbb{F}^n$. Here, $\mathcal{F}^{-1}(\mathbf{x})$ means finding one (of the possibly many) pre-images of $\mathbf{x}$ under the central map $\mathcal{F}$.

Verification. To verify the authenticity of a document, one simply computes $\mathbf{h}' = \mathcal{P}(\mathbf{z})$ and the hash value $\mathbf{h} = \mathcal{H}(d)$ of the document. If $\mathbf{h}' = \mathbf{h}$ holds, the signature is accepted, otherwise rejected.

There are several ways how to build the central map $\mathcal{F}$ of multivariate schemes. In this paper we concentrate on the so called SingleField constructions. In contrast to BigField schemes like Matsumoto-Imai [14] and MiddleField schemes like ℓiC [8], here all the computations are done in one (relatively small) field. In the following subsection we describe one well known example for such a scheme in detail.

3.1 The UOV Signature Scheme

One way to create an easily invertible multivariate quadratic system is the principle of Oil and Vinegar, which was first proposed by J. Patarin in [15].

Let $\mathbb{F}$ be a finite field. Let o and v be two integers and set $n = o + v$. We set $V = \{1, \ldots, v\}$ and $O = \{v + 1, \ldots, n\}$. We call $x_1, \ldots, x_v$ the Vinegar variables and $x_{v+1}, \ldots, x_n$ Oil variables and define o quadratic polynomials $f^{(k)}(\mathbf{x}) = f^{(k)}(x_1, \ldots, x_n)$ by

$$f^{(k)}(\mathbf{x}) = \sum_{i \in V,\ j \in O} \alpha_{ij}^{(k)} x_i x_j + \sum_{i,j \in V,\ i \leq j} \beta_{ij}^{(k)} x_i x_j + \sum_{i \in V \cup O} \gamma_i^{(k)} x_i + \eta^{(k)} \ (1 \leq k \leq o). \tag{2}$$

Note that Oil and Vinegar variables are not fully mixed, just like oil and vinegar in a salad dressing.

The map $\mathcal{F} = (f^{(1)}(\mathbf{x}), \ldots, f^{(o)}(\mathbf{x}))$ can be easily inverted. First, we choose the values of the v Vinegar variables $x_1, \ldots, x_v$ at random. Thus we get a system of o linear equations in the o variables $x_{v+1}, \ldots, x_n$ which can be solved e.g. by Gaussian Elimination. If the system does not have a solution, one has to choose other values of $x_1, \ldots, x_v$ and try again.

To hide the structure of $\mathcal{F}$ in the public key, one composes it with an affine map $\mathcal{T} : \mathbb{F}^n \to \mathbb{F}^n$. Therefore, the public key has the form $\mathcal{P} = \mathcal{F} \circ \mathcal{T}$. The private key consists of $\mathcal{F}$ and $\mathcal{T}$ and therefore allows to invert the public key.

Remark: In opposite to other multivariate schemes the second affine map $\mathcal{S}$ is not needed for the security of UOV. So it can be dropped.

In his original paper [15] Patarin suggested to choose $o = v$ (Balanced Oil and Vinegar (OV)). After this scheme was broken by Kipnis and Shamir in [12], it was recommended in [11] to choose $v > o$ (Unbalanced Oil and Vinegar (UOV)).

The UOV signature scheme over $GF(2^8)$ is commonly believed to be secure for $o \geq 28$ equations [19] and $v = 2 \cdot o$ Vinegar variables. For UOV schemes over $GF(2^4)$ we need at least $o = 40$ equations and $v = 2 \cdot o$ Vinegar variables.

4 Improved versions of UOV

In this section we review the approach of [16] to create UOV-based schemes with a structured public key.

Recall that, in the case of the Unbalanced Oil and Vinegar signature scheme [11], the public key $\mathcal{P}$ is given as the composition of the central UOV-map $\mathcal{F}$ and an affine invertible map $\mathcal{T}$ (given by a matrix M_T and a vector c_T), i.e.

$$\mathcal{P} = \mathcal{F} \circ \mathcal{T}. \tag{3}$$

In [16] it is observed, that this equation (after fixing the affine map $\mathcal{T}$), leads to a linear relation between the coefficients of the quadratic monomials of $\mathcal{P}$ and $\mathcal{F}$ of the form

$$p_{ij}^{(k)} = \sum_{r=1}^{v} \sum_{s=r}^{n} \alpha_{ij}^{rs} \cdot f_{rs}^{(k)}, \tag{4}$$

where $p_{ij}^{(k)}$ and $f_{ij}^{(k)}$ are the coefficients of $x_i x_j$ in the k-th component of $\mathcal{P}$ and $\mathcal{F}$ respectively and the α_{ij}^{rs} are given as

$$\alpha_{ij}^{rs} = \begin{cases} t_{ri} \cdot t_{si} & (i = j) \\ t_{ri} \cdot t_{sj} + t_{rj} \cdot t_{si} & \text{otherwise} \end{cases}. \tag{5}$$

Here $t_{ij} \in \mathbb{F}$ denote the elements of the matrix M_T. Let $D := \frac{v \cdot (v+1)}{2} + ov$ be the number of non-zero quadratic terms in any component of $\mathcal{F}$ and $D' := \frac{n \cdot (n+1)}{2}$ be the number of quadratic terms in the public polynomials. Let M_P and M_F be the coefficient matrices of $\mathcal{P}$ and $\mathcal{F}$ respectively (w.r.t. the graded lexicographic ordering of monomials). The matrices M_P and M_F are divided into submatrices as shown in Figure 2. Note that, due to the absence of oil $\times$ oil terms in the central polynomials, we have a block of zeros in the middle of M_F.

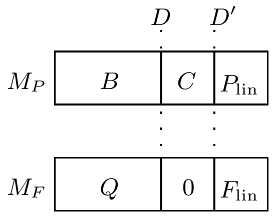


Fig. 2. Layout of the matrices M_P and M_F

Furthermore, the authors of [16] defined the so called transformation matrix $A_{UOV} \in \mathbb{F}^{D \times D}$ containing the coefficients α_{ij}^{rs} of equation (4)

$A_{UOV} = \left(\alpha_{ij}^{rs}\right)$ $(1 \leq r \leq v, r \leq s \leq n$ for the rows, $1 \leq i \leq v, i \leq j \leq n$ for the columns), i.e.

$$A_{UOV} = \begin{pmatrix} \alpha_{11}^{11} & \alpha_{12}^{11} & \dots & \alpha_{vn}^{11} \\ \alpha_{11}^{12} & \alpha_{12}^{12} & \dots & \alpha_{vn}^{12} \\ \vdots & & & \vdots \\ \alpha_{11}^{vn} & \alpha_{12}^{vn} & \dots & \alpha_{vn}^{vn} \end{pmatrix}. \tag{6}$$

With this notation, equation (4) yields

$$B = Q \cdot A_{UOV}. \tag{7}$$

If the matrix A_{UOV} is invertible, this equation has a solution for Q. Experiments indicate that this condition is fulfilled with overwhelming probability. We can then use Algorithm 1 to generate a key pair for UOV.

Algorithm 1. Alternative Key Generation for UOV schemes

```
Input: parameters $(\mathbb{F}, o, v)$
Output: UOV keypair $(\mathcal{F}, \mathcal{T}), \mathcal{P}$
1: $D \leftarrow \frac{v \cdot (v+1)}{2} + o \cdot v$
2: Choose an $o \times D$ matrix $B$ (e.g. generated by an LRS).
3: Choose randomly an affine map $\mathcal{T}$ (represented by an $n \times n$-matrix $M_T$ and an
   $n$-vector $c_T$). If $M_T$ is not invertible, choose again.
4: Compute for $\mathcal{T}$ the corresponding transformation matrix $A_{\mathrm{UOV}}$ (using equations
   (5) and (6)). If $A_{\mathrm{UOV}}$ is not invertible, go back to step 2.
5: Solve the linear system given by equation (7) to get the matrix $Q$ and therewith
   the quadratic coefficients of the central polynomials.
6: Choose the linear and constant terms of the central map $\mathcal{F}$ at random.
7: Compute the public key as $\mathcal{P} = \mathcal{F} \circ \mathcal{T}$.
8: return $(\mathcal{F}, \mathcal{T}), \mathcal{P}$
```

5 Our Choice of *B*

The authors of [17] used a matrix B, whose elements were given by a single Linear Recurring Sequence, i.e. for a given LRS $S = (s_1, s_2, \dots)$ the matrix B was of the form

$$B^{(PB11)} = \begin{pmatrix} s_1 & s_2 & \dots & s_D \\ s_{D+1} & s_{D+2} & \dots & s_{2 \cdot D} \\ \vdots & & & \vdots \\ s_{(o-1) \cdot D+1} & s_{(o-1) \cdot D+2} & \dots & s_{o \cdot D} \end{pmatrix} \tag{8}$$

To guarantee the security of the scheme, they had to choose a Linear Recurring Sequence of length $L \geq o$.

In our new scheme, we use not only one, but o different Linear Recurring Sequences. The goal of this strategy is to reduce the lengths of the single LRS's, which will later help us to speed up the verification process of the scheme (see Section 6). In fact, we will use Linear Recurring Sequences of length 1.

We choose randomly two vectors $\alpha,\ \gamma \in \mathbb{F}^o$ and define for each $i = 1, \ldots, o$ a univariate polynomial C_i by $C_i(X) = \gamma_i \cdot X + 1$. For $i = 1, \ldots, o$ we compute the first D elements of the Linear Recurring Sequence $S^{(i)} = (s_1^{(i)}, s_2^{(i)}, \ldots) = LRS(\alpha_i, C_i)$ and put this sequence into the i-th row of the matrix B. Therefore, the matrix B will have the following structure:

$$B = \begin{pmatrix} s_1^{(1)} & s_2^{(1)} & \ldots & s_D^{(1)} \\ s_1^{(2)} & s_2^{(2)} & \ldots & s_D^{(2)} \\ \vdots & & & \vdots \\ s_1^{(o)} & s_2^{(o)} & \ldots & s_D^{(o)} \end{pmatrix}. \tag{9}$$

We denote the scheme obtained by using this matrix B and Algorithm 1 by UOVLRS2.

5.1 Choice of α and γ

First, we look at the question what happens if two elements of the vector γ, say γ_i and γ_j $(i \neq j)$ are equal.

Theorem 1. *If $\gamma_i = \gamma_j$ for $i \neq j \in \{1, \ldots, o\}$, the homogeneous quadratic parts of the polynomials $p^{(i)}$ and $p^{(j)}$ are linearly dependent.*

Proof. If $\gamma_i = \gamma_j$ for $i \neq j \in \{1, \ldots, o\}$, the two rows $B[i]$ and $B[j]$ are linearly dependent. Since we have $Q = B \cdot A^{-1}$ (c.f. equation (7)), the same holds for $Q[i]$ and $Q[j]$ (see Figure 2). Note that this matrix contains all the private coefficients of quadratic terms, which means that the homogeneous quadratic parts of the i-th and j-th central polynomials are linearly dependent. Since during the key generation of UOV the rows of the central map $\mathcal{F}$ are not mixed, the same holds for the homogeneous quadratic part of the i-th and j-th public polynomial. □

Theorem 1 states that by computing $p^{(i)} - \frac{\alpha_i}{\alpha_j} \cdot p^{(j)}$ the attacker will get a linear equation in the system variables, which means that he can reduce the number of variables in the quadratic system by 1. We can conclude

Corollary 1. *Attacking an instance of UOVLRS2 with m equations and $t < m$ different values in the vector γ is only as hard as solving a (UOVLRS2) system of t equations.*

To check this theoretical result, we created instances of UOVLRS2 for different values of o and v and different types of vectors γ and solved the resulting public systems with MAGMA v.2-13.10 (with fixing of v variables to create determined systems).

Table 1. Running time of direct attacks with MAGMA

t [1]	(o,v)	(9,18)	(10,20)	(11,22)	(12,24)	(13,26)	(14,28)
9	time (s)	5.4	5.7	5.7	5.8	5.8	5.9
10	time (s)	———	38.9	40.8	41.8	43.0	44.6
11	time (s)	———	———	287.3	301.4	309.8	315.2

[1] number of different values in γ

To achieve the optimal security level, the elements of the vector γ must be pairwise distinct. Furthermore, all the elements have to be $\neq 0$.

Remark: The above condition gives a lower bound to the cardinality of the underlying field. In particular, we can not define our scheme over $GF(2^4)$.

On the contrary, there seem to be no major conditions for the choice of the vector α. We have to ensure only that $\alpha_i \in \mathbb{F} \setminus \{0\} \;\; \forall i = 1, \ldots, o$. For simplicity we choose $\alpha = (1, \ldots, 1)$.[2] Therefore, we get a matrix B of the Vandermonde-type:

$$B = \begin{pmatrix} 1 & \gamma_1 & \gamma_1^2 & \cdots & \gamma_1^{D-1} \\ 1 & \gamma_2 & \gamma_2^2 & \cdots & \gamma_2^{D-1} \\ \vdots & & & & \vdots \\ 1 & \gamma_o & \gamma_o^2 & \cdots & \gamma_o^{D-1} \end{pmatrix} \tag{10}$$

which can be used in Algortihm 1 to generate a key pair of UOVLRS2.

5.2 Security

As mentioned above, the matrix B of our scheme is of the Vandermonde type. If the elements of the vector γ are pairwise distinct, there is not any relationship between the rows of B at all. This is in contrast to the schemes of [16] and [17] and prevents therefore possible attacks against schemes of this type which use such relationships. Furthermore this is very similar to the case of standard UOV, which seems to show that direct attacks against our scheme are as difficult as direct attacks against standard UOV. Further evidence for this result was given by experiments with MAGMA [2].

Furthermore we checked experimentally the security of our scheme against other attacks affecting UOV-like schemes, including

– UOV attack of Kipnis and Shamir [11]
– UOV Reconciliation attack [7]

and found that these attacks cannot use the structure in our systems. Details on these experiments can be found in the appendix of this paper.

[2] In fact, the attacker is allowed to multiply each public polynomial $p^{(i)}$ $(i = 1, \ldots, o)$ by a number $a_i \in \mathbb{F} \setminus \{0\}$ of his choice. By doing so, he can produce a vector α' of this form.

6 The Verification Process

The central part of the verification process for multivariate signature schemes is the evaluation of the public polynomials. Normally this is done as follows: For a given (valid or invalid) signature $\mathbf{z} = (z_1, \ldots, z_n) \in \mathbb{F}^n$ one first computes an $\frac{(n+1)\cdot(n+2)}{2}$ vector mon, which contains the values of all monomials of degree ≤ 2, i.e.

$$\text{mon} = (z_1^2, z_1 z_2, \ldots, z_n^2, z_1, \ldots, z_n, 1). \tag{11}$$

Then we have

$$\mathcal{P}(\mathbf{z}) = \begin{pmatrix} M_P[1] \cdot \text{mon}^T \\ \vdots \\ M_P[o] \cdot \text{mon}^T \end{pmatrix}, \tag{12}$$

with $M_P[i]$ being the i-th row of the Macauley matrix M_P.

For our new scheme, the following strategy seems to be more promising:

6.1 Notations

Let $\mathbf{h} = (h_1, \ldots, h_o)$ be the hash value of the signed message.
The public polynomials can be written as

$$p^{(k)}(x_1, \ldots, x_n) = \sum_{i=1}^{n} \sum_{j=i}^{n} p_{ij}^{(k)} \cdot x_i x_j + \sum_{i=1}^{n} p_i^{(k)} \cdot x_i + p_0^{(k)} \quad (k = 1, \ldots, o). \tag{13}$$

For $k = 1, \ldots, o$ we define upper triangular matrices $MP^{(k)}$ by

$$MP^{(k)} = \begin{pmatrix} p_{11}^{(k)} & p_{12}^{(k)} & p_{13}^{(k)} & \cdots & p_{1n}^{(k)} & p_1^{(k)} \\ 0 & p_{22}^{(k)} & p_{23}^{(k)} & \cdots & p_{2n}^{(k)} & p_2^{(k)} \\ 0 & 0 & p_{33}^{(k)} & & p_{3n}^{(k)} & p_3^{(k)} \\ \vdots & & \ddots & & \vdots & \vdots \\ 0 & 0 & \ldots & 0 & p_{nn}^{(k)} & p_n^{(k)} \\ 0 & 0 & \ldots & 0 & 0 & p_0^{(k)} \end{pmatrix}. \tag{14}$$

For a (valid or invalid) signature $\mathbf{z} = (z_1, \ldots, z_n)$ of the message we define the extended signature vector

$$\text{sign} = (z_1, \ldots, z_n, 1). \tag{15}$$

With this notation we can write the verification process in the following form

$$\text{accept the signature } \mathbf{z} \iff \text{sign} \cdot MP^{(k)} \cdot \text{sign}^T = h_k \ \forall k \in \{1, \ldots, o\}. \tag{16}$$

In the following subsection we consider the question how we can evaluate this equation more efficiently for our scheme.

6.2 Verification of UOVLRS2

In the case of UOVLRS2, the matrices $MP^{(k)}$ are of the form shown in Figure 3.

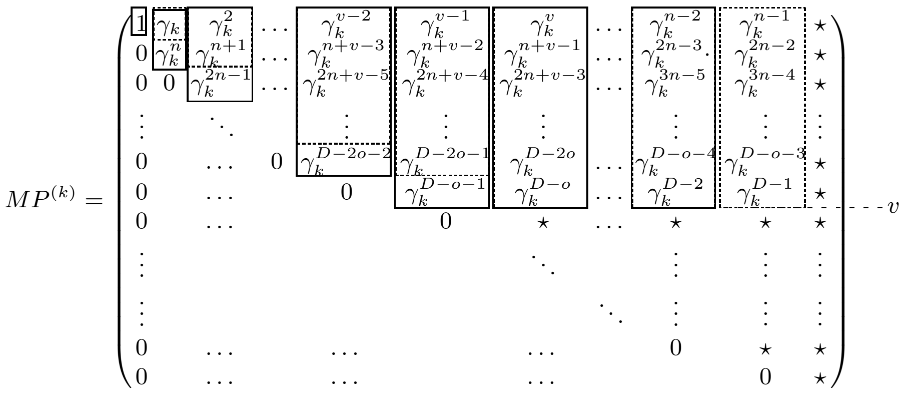


Fig. 3. Matrices $MP^{(k)}$ for UOVLRS2

We have

$$MP_{ij}^{(k)} = \gamma_k \cdot MP_{i,j-1}^{(k)} \ \forall i \in \{1, \ldots, v\},\ j \in \{i+1, \ldots, n\},\ k \in \{1, \ldots, o\}. \quad (17)$$

Therefore we get

$$(\mathrm{sign}_1, \ldots, \mathrm{sign}_i) \cdot \begin{pmatrix} MP_{1,j}^{(k)} \\ MP_{2,j}^{(k)} \\ \vdots \\ MP_{i,j}^{(k)} \end{pmatrix} = \gamma_k \cdot (\mathrm{sign}_1, \ldots, \mathrm{sign}_i) \cdot \begin{pmatrix} MP_{1,j-1}^{(k)} \\ MP_{2,j-1}^{(k)} \\ \vdots \\ MP_{i,j-1}^{(k)} \end{pmatrix} \quad (18)$$
$$\forall i \in \{1, \ldots v\}, \quad j \in \{i+1, \ldots, n\},\ k \in \{1, \ldots, o\}.$$

The boxes in Figure 3 illustrate this equation: Boxes with continuous lines show the vector $(MP_{1,j-1}^{(k)}, \ldots, MP_{i,j-1}^{k})^T$ on the right hand side of equation (18), while the boxes with dotted lines show the vector $(MP_{1,j}^{(k)}, \ldots, MP_{i,j}^{(k)})^T$ on the left hand side. Any box with dotted lines can be computed by multiplying the corresponding box with continuous lines by γ_k.

We can use this fact to speed up the verification process of UOVLRS2 by a large factor (see Algorithm 2).

Algorithm 2 works as follows:

From line 2 to 14 the public polynomials are evaluated. From line 3 to 12 we hereby compute the matrix vector product sign $\cdot\, MP^{(k)}\cdot$ whose result is stored

Algorithm 2. Verification process for UOVLRS2

```
Input: signature $\mathbf{z} \in \mathbb{F}^n$, hash value $\mathbf{h} \in \mathbb{F}^m$
Output: Boolean value TRUE or FALSE
 1: sign $\leftarrow (z_1, \ldots, z_n, 1)$
 2: for $k = 1$ to $o$ do
 3:     $\text{temp}_1 \leftarrow \text{sign}_1$
 4:     for $j = 2$ to $v$ do
 5:         $\text{temp}_j \leftarrow \gamma_k \cdot \text{temp}_{j-1} + MP^{(k)}_{jj} \cdot \text{sign}_j$
 6:     end for
 7:     $a \leftarrow \text{temp}_v$
 8:     for $j = v + 1$ to $n$ do
 9:         $a \leftarrow \gamma_k \cdot a$
10:         $\text{temp}_j \leftarrow a + \sum_{i=v+1}^{j} MP^{(k)}_{ij} \cdot \text{sign}_i$
11:     end for
12:     $\text{temp}_{n+1} \leftarrow \sum_{i=1}^{n+1} MP^{(k)}_{i,n+1} \cdot \text{sign}_\text{i}$
13:     $h'_k \leftarrow \sum_{i=1}^{n+1} \text{temp}_i \cdot \text{sign}_i$
14: end for
15: if $h_k = h'_k \ \forall k \in \{1, \ldots, o\}$ then
16:     return TRUE
17: else
18:     return FALSE
19: end if
```

in the vector temp. In line 5 and line 9-10 we hereby use the special structure of our public key, which allows us to compute each temp_i $(i = 2, \ldots, n)$ using only two multiplications. Finally, in line 13 of the algorithm, we compute the scalar product of temp and sign.

In line 15 to 19 we test, if the result is equal to the hash value of the message.

Computational Effort. To evaluate $\mathcal{P}$ in the standard way (i.e. by using equations (11) and (12)), one needs

$$\frac{n+1}{2} \cdot (n + o \cdot (n+2)) \text{ field multiplications.} \tag{19}$$

Algorithm 2 needs (for each iteration of the main loop)

- in the first loop (step 4 to 6) $2 \cdot (v - 1)$ field multiplications,
- in the second loop (step 8 to 11) $o + \frac{o \cdot (o+1)}{2}$ field multiplications,
- in step 12 $n + 1$ field multiplications,
- and in step 13 again $n + 1$ field multiplications.

Therefore, to evaluate equation (16) (o iterations of the main loop), Algorithm 2 needs

$$o \cdot \left(3 \cdot n + v + \frac{o \cdot (o+1)}{2}\right) \text{ field multiplications.} \tag{20}$$

For $\mathbb{F} = GF(2^8)$, $(o, v) = (28, 56)$ this means a reduction of the number of field multiplications needed during the verification process by a factor of 5.3.

7 Parameters and Experiments

Based on our security analysis (see Subsection 5.2 and Appendix A), we propose for our scheme the same parameters as for the standard UOV scheme, namely

$$\mathbb{F} = GF(256),\ (o, v) = (28, 56).$$

The elements of the vector $\gamma \in \mathbb{F}^o$ are chosen pairwise distinct and we set $\alpha = (1, \ldots, 1) \in \mathbb{F}^o$.

To check our theoretical results regarding the verification process, we created a straightforward C implementation of our scheme and the standard UOV. Table 2 shows the results:

Table 2. Comparison of our scheme with standard UOV

Scheme	private key size (kB)	hash length (bit)	signature length (bit)	public key size (kB)	public key red. factor	verification time ms	verification time red. factor
$\mathrm{UOV}(2^8, 28, 56)$	96.6	224	672	99.9	-	0.99	-
$\mathrm{UOVLRS2}(2^8, 28, 56)$	96.6	224	672	13.5	7.4	0.18	5.5
$\mathrm{UOV}(2^8, 30, 60)$	117.0	240	720	122.6	-	1.21	-
$\mathrm{UOVLRS2}(2^8, 30, 60)$	117.0	240	720	16.4	7.5	0.21	5.7

8 Conclusion and Future Work

In this paper we proposed a variation of the UOVLRS scheme of [17], which not only achieves a similar reduction of the public key size but also speeds up the verification process of UOV by a large factor. In particular, we achieved a reduction of the public key size of UOV by a factor of 7.5 and a speed up factor of 5.5 for the verification process. We showed the latter both theoretically and by a C implementation of the schemes. Furthermore, experiments seem to show that the security of UOV is not weakened by our modifications.

Future work includes the extension of our ideas to the Rainbow Signature scheme [6] and the implementation of the scheme on hardware. Furthermore we want to apply our techniques to the QUAD stream cipher [4] to speed up its key stream generation process.

Acknowledgements. We thank the anonymous reviewers for their comments which helped to improve the paper. The first author thanks the Horst-Görtz Foundation for financial support, while the second author is funded by DFG grant BU630/22-1.

References

[1] Bernstein, D.J., Buchmann, J., Dahmen, E. (eds.): Post-Quantum Cryptography. Springer, Heidelberg (2009)

[2] Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. J. Symbolic Comput. 24(3-4), 235–265 (1997)

[3] Bogdanov, A., Eisenbarth, T., Rupp, A., Wolf, C.: Time-area optimized public-key engines: -cryptosystems as replacement for elliptic curves? In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 45–61. Springer, Heidelberg (2008)

[4] Berbain, C., Gilbert, H., Patarin, J.: QUAD: A Practical Stream Cipher with Provable Security. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 109–128. Springer, Heidelberg (2006)

[5] Chen, A.I.-T., Chen, M.-S., Chen, T.-R., Cheng, C.-M., Ding, J., Kuo, E.L.-H., Lee, F.Y.-S., Yang, B.-Y.: SSE implementation of multivariate pkcs on modern x86 cpus. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 33–48. Springer, Heidelberg (2009)

[6] Ding, J., Schmidt, D.: Rainbow, a new multivariable polynomial signature scheme. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 164–175. Springer, Heidelberg (2005)

[7] Ding, J., Yang, B.-Y., Chen, C.-H.O., Chen, M.-S., Cheng, C.-M.: New Differential-Algebraic Attacks and Reparametrization of Rainbow. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 242–257. Springer, Heidelberg (2008)

[8] Ding, J., Wolf, C., Yang, B.-Y.: ℓ-invertible Cycles for Multivariate Quadratic Public Key Cryptography. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 266–281. Springer, Heidelberg (2007)

[9] Faugère, J.C.: A new efficient algorithm for computing Groebner bases (F4). Journal of Pure and Applied Algebra 139, 61–88 (1999)

[10] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979)

[11] Kipnis, A., Patarin, J., Goubin, L.: Unbalanced Oil and Vinegar Signature Schemes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 206–222. Springer, Heidelberg (1999)

[12] Kipnis, A., Shamir, A.: Cryptanalysis of the Oil and Vinegar Signature scheme. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 257–266. Springer, Heidelberg (1998)

[13] Lidl, R., Niederreiter, H.: Introduction to Finite Fields and their Applications. Cambridge University Press (1986)

[14] Matsumoto, T., Imai, H.: Public Quadratic Polynomial-Tuples for efficient Signature-Verification and Message-Encryption. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 419–453. Springer, Heidelberg (1988)

[15] Patarin, J.: The oil and vinegar signature scheme. Presented at the Dagstuhl Workshop on Cryptography (September 1997)

[16] Petzoldt, A., Bulygin, S., Buchmann, J.: A Multivariate Signature Scheme with a partially cyclic public key. In: Proceedings of SCC 2010, pp. 229–235 (2010)

[17] Petzoldt, A., Bulygin, S., Buchmann, J.: Linear Recurring Sequences for the UOV Key Generation. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 335–350. Springer, Heidelberg (2011)

[18] Shor, P.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM J. Comput. 26(5), 1484–1509
[19] Thomae, E., Wolf, C.: Solving underdetermined Systems of Multivariate Quadratic Equations Revisited. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 156–171. Springer, Heidelberg (2012)

A Details of the Experiments

In this section we present the results of our experiments with known attacks against the UOV scheme. In particular, we test the security of our scheme against

– Direct attacks
– UOV-Reconciliation attack
– UOV-attack

A.1 Direct Attacks

In a direct attack an attacker tries to solve the public system $\mathcal{P}(x) = h$ by a system solver like XL or a Gröbner Basis method. Direct attacks can be used against each multivariate scheme as a message recovery attack (encryption schemes) or a signature forgery attack (signature schemes). To check the security of the UOVLRS2 scheme under direct attacks, we carried out a number of experiments with MAGMA [2] v.2-13.10, which contains an efficient implementation of Faugère's F_4 algorithm [9] to compute Gröbner Bases. For each of the parameter sets listed in Table 3 we created 100 instances of UOV and UOVLRS2 and solved the public systems using the MAGMA command `Variety`.

Table 3. Running time of the direct attack against UOV and UOVLRS2

parameters $(2^8, o, v)$	(9,18)	(10,20)	(11,22)	(12,24)	(13,26)	(14,28)
UOV	5.5 s	40.0 s	289.2 s	2,383 s	18,928 s	196,638 s
UOVLRS2	5.4 s	39.9 s	288.6 s	2,378 s	18,917 s	195,963 s

A.2 UOV-Reconciliation Attack

In the UOV-Reconciliation attack [7] the attacker tries to find an affine transformation which brings the public key in the form of a UOV central map (i.e. no Oil $\times$ Oil terms). To do this, the attacker has to solve a number of multivariate quadratic systems. The complexity of the attack is mainly given by the complexity of solving the first of these systems, which contains o equations in v variables. Table 4 shows the time, MAGMA needs for solving this first system for standard UOV and UOVLRS2.

Table 4. Running time of the UOV-Reconciliation attack against UOV and UOVLRS2

parameters $(2^8, o, v)$	(9,18)	(10,20)	(11,22)	(12,24)	(13,26)	(14,28)
UOV	5.5 s	40.1 s	289.3 s	2,381 s	18,924 s	196,712 s
UOVLRS2	5.4 s	39.9 s	286.8 s	2,379 s	18,923 s	196,683 s

A.3 UOV Attack

In the UOV attack of Kipnis and Shamir [12] the attacker tries to reconstruct the essential parts of the affine transformation $\mathcal{T}$ (i.e. the parts which mix Oil and Vinegar variables). To do this, he tries to find the space $\mathcal{T}^{-1}(\mathcal{O})$, where $\mathcal{O}$ is the so called Oil space

$$\mathcal{O} = \{x \in \mathbb{F}^n : x_1 = \ldots = x_v = 0\}.$$

This can be done by looking at the invariant subspaces of the linear maps $W = P_i^{-1} \cdot \sum_{j=1}^{o} P_j$, where P_j is the symmetric matrix associated with the homogeneous part of the j-th public polynomial. Table 5 shows the base 2-logarithm of the number of matrices W we had to test until finding a basis of $\mathcal{T}^{-1}(\mathcal{O})$.

Table 5. Results of the experiments with the UOV attack

parameters $(2^8, o, v)$	(2,4)	(3,6)	(4,8)	(5,10)
UOV	16.1	24.3	32.2	40.0
UOVLRS2	16.0	24.1	32.0	39.9

As the Tables 3 - 5 show, known attacks against the UOV signature scheme cannot use the special structure of our public keys. Of course, this is no proof that no dedicated attacks against our scheme exist. However, as long as no such attack is known, we believe our scheme to be secure and propose for it the same parameters as for the standard UOV scheme (see Section 7).

Galindo-Garcia Identity-Based Signature Revisited

Sanjit Chatterjee, Chethan Kamath, and Vikas Kumar

Dept. of Computer Science and Automation,
Indian Institute of Science,
Bangalore
{sanjit,chethan0510,vikaskumar}@csa.iisc.ernet.in

Abstract. In Africacrypt 2009, Galindo-Garcia [12] proposed a lightweight identity-based signature (IBS) scheme based on the Schnorr signature. The construction is simple and claimed to be the most efficient IBS till date. The security is based on the discrete-log assumption and the security argument consists of two reductions: $\mathcal{B}_1$ and $\mathcal{B}_2$, both of which use the multiple-forking lemma [4] to solve the discrete-log problem (DLP).

In this work, we revisit the security argument given in [12]. Our contributions are two fold: (i) we identify several problems in the original argument and (ii) we provide a detailed new security argument which allows significantly tighter reductions. In particular, we show that the reduction $\mathcal{B}_1$ in [12] fails in the standard security model for IBS [1], while the reduction $\mathcal{B}_2$ is incomplete. To remedy these problems, we adopt a two-pronged approach. First, we sketch ways to fill the gaps by making minimal changes to the structure of the original security argument; then, we provide a new security argument. The new argument consists of three reductions: $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ and in each of them, solving the DLP is reduced to breaking the IBS. $\mathcal{R}_1$ uses the general forking lemma [2] together with the programming of the random oracles and Coron's technique [8]. Reductions $\mathcal{R}_2$ and $\mathcal{R}_3$, on the other hand, use the multiple-forking lemma along with the programming of the random oracles. We show that the reductions $\mathcal{R}_1$ and $\mathcal{R}_2$ are significantly tighter than their original counterparts.

Keywords: Identity-based signatures, Galindo-Garcia identity-based signature, Schnorr signatures, Forking lemma, Discrete-log assumption.

1 Introduction

The notion of identity-based signatures (IBS) is an extension of the idea of digital signatures to the identity-based setting. As in traditional signature schemes, the signer uses her secret key to sign a message. However, the signature can be verified by anyone using the signer's identity and public parameters of the private-key generator (PKG). IBS or more generally, identity-based cryptosystems [18] do not require any certificates to be exchanged and hence can be advantageous over the traditional PKI based systems in certain scenarios.

Several RSA based IBS [10,13] have been proposed in the literature after the notion of IBS was introduced by Shamir in 1984 [18]. In recent times, a few pairing based constructions were also proposed [7,14,15,9]. Galindo-Garcia [12], on the other hand, used the technique of concatenated Schnorr's signature to propose an identity-based signature that works in the discrete-log setting but does not require pairing. The authors came up with a security proof of the proposed IBS scheme in the so-called `EU-ID-CMA` model using the random oracle methodology [3] and a variant of the forking lemma [2,4,16]. The security is based on the discrete-log problem in any prime order group. The authors suggest to implement their scheme in a suitable elliptic curve group and after a comparative study concluded that the proposed construction has an overall better performance than the existing RSA-based and pairing-based schemes. The Galindo-Garcia IBS, due to its efficiency and simplicity, has been used as a building block for a couple of other cryptosystems [17,20].

Our Contribution. Critical examination of the security argument of a cryptographic construction to see whether the claimed security is indeed achieved or not is an important topic in cryptographic research. Two such well-known examples are Shoup's work on OAEP [19] and Galindo's work on Boneh-Franklin IBE [11]. Another important question in the area of provable security is to obtain tighter security reduction for existing construction. One such classical example is Coron's analysis of FDH [8]. In this work we revisit the security argument of Galindo-Garcia IBS [12] with the above two questions in mind.

The security argument of Galindo-Garcia IBS consists of two reductions, $\mathcal{B}_1$ and $\mathcal{B}_2$, the choice of which is determined by an event E. The authors construct $\mathcal{B}_1$ to solve the DLP when the event E occurs. Similarly, $\mathcal{B}_2$ is used to solve the DLP in case the complement of E occurs. Both the reductions use the multiple-forking lemma [4] to show that the DLP is reduced to breaking the IBS scheme.

In this work, we make several observations about the security argument in [12]. In particular, we show that the reduction $\mathcal{B}_1$ fails to provide a proper simulation of the unforgeability game in the standard security model for IBS [1], while $\mathcal{B}_2$ is incomplete. We adopt a two-pronged approach to address these problems. First, we sketch ways to fill the gaps by making minimal changes to the structure of original security argument; then, we provide a new security argument. The new argument consists of three reductions: $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$. At a high level, our first reduction, $\mathcal{R}_1$, addresses the problems identified in the original $\mathcal{B}_1$ in [12], while $\mathcal{R}_2$ and $\mathcal{R}_3$ together address the incompleteness of the original $\mathcal{B}_2$. The reduction $\mathcal{R}_1$ uses the general forking lemma [2] and the technique first introduced by Coron [8] to prove the security of FDH. We show that this results in a significantly tighter security reduction. On the other hand, both $\mathcal{R}_2$ and $\mathcal{R}_3$ are structurally similar to $\mathcal{B}_2$ but uses two different versions of the multiple forking lemma [4], together with an algebraic technique similar to one adopted by Boneh-Boyen in [5]. The security reduction $\mathcal{R}_2$ is also significantly tighter than the original $\mathcal{B}_2$ (see *Table* 1 for a comparison). All the three reductions use the programmability of the random oracles in a crucial way.

Notations. We adopt the notations commonly used in the literature. In addition, for an oracle H, $\#\mathrm{H}(x)$ indicates the index on which the oracle query for input x was made. Also, in a group $\mathbb{G}$, the discrete-log to a base g is denoted by $\log_g^{\mathbb{G}}$.

2 Revisiting the Galindo-Garcia Security Argument

We now describe some of the problems that we observed with the original security argument. We reproduce the original reductions from §4 of [12] in the full version of this paper [6]. In the following, $\mathcal{B}_i.j$ refers to the j^{th} step in the construction of $\mathcal{B}_i$, $i \in \{1, 2\}$.

2.1 Observations on $\mathcal{B}_1$

We now note the following points about the reduction $\mathcal{B}_1$. We also mention ways to fix the problems.

(i) Correctness of signatures on $\hat{\mathsf{id}}$*:* In $\mathcal{B}_1.4$, when $\mathcal{A}$ makes a signature query on $\hat{\mathsf{id}}$, $\mathcal{B}_1$ returns $(A, B, R) \in \mathbb{G}^3$ as the signature. However, in the protocol definition, the signatures are elements of $\mathbb{G} \times \mathbb{Z}_p \times \mathbb{G}$. Therefore, the signatures on $\hat{\mathsf{id}}$ will fail the verification in the general group setup – i.e., $\mathbb{G}$ is any cyclic group of prime order p, and in particular, in the elliptic curve setting – as the operation g^B is not defined in $\mathbb{G}$. What the authors *could* have intended in $\mathcal{B}_1.4$ is

- When $\mathcal{A}$ makes a call to the signature oracle $\mathcal{O}_s$ for (id, m) with $\mathsf{id} = \hat{\mathsf{id}}$, $\mathcal{B}_1$ chooses $t, b \in_R \mathbb{Z}_p$, *sets* $B := g^b, R := g^{-zc}(g^\alpha)^t, c := \mathrm{H}(\mathsf{id}, R)$ and $A := B(g^\alpha g^{zc})^{-d}$. Then it returns the signature (A, b, R) to $\mathcal{A}$.

Even after the above correction is applied, the signatures on $\hat{\mathsf{id}}$ fail the verification algorithm. For the signatures to verify, the following equality should hold.

$$\begin{aligned} g^b &= A(R \cdot (g^\alpha)^c)^d \\ &= g^b(g^\alpha g^{zc})^{-d}(g^{-zc}(g^\alpha)^t g^{zc})^d \\ 1 &= g^{(\alpha+zc)(-d)} g^{\alpha t d} \end{aligned}$$

However, it holds only if $(\alpha t - zc - \alpha)\, d \equiv 0 \bmod p$. It is easy to check that the LHS is a random element of $\mathbb{Z}_p$. Hence, the signatures on $\hat{\mathsf{id}}$ given by $\mathcal{B}_1$ will fail to verify with an overwhelming probability of $1 - \frac{1}{p}$. The equality holds if we set $t := 1 + \frac{zc}{\alpha}$, instead of selecting t uniformly at random from $\mathbb{Z}_p$. However, setting $t := 1 + \frac{zc}{\alpha}$ results in R being set to the problem instance g^α, removing t from the picture altogether. Thus, $\mathcal{B}_1.4$ would finally look like:

- When $\mathcal{A}$ makes a call to the signature oracle $\mathcal{O}_s$ for (id, m) with $\mathsf{id} = \hat{\mathsf{id}}$, $\mathcal{B}_1$ chooses $b, d \in_R \mathbb{Z}_p$, sets $B := g^b, R := g^\alpha, c := \mathrm{H}(\mathsf{id}, R), A := B(g^\alpha g^{zc})^{-d}$ and programs the random oracle in such a way that $d := \mathrm{G}(\mathsf{id}, A, m)$. Then it returns the signature (A, b, R) to $\mathcal{A}$.

Although it may appear that the reduction $\mathcal{B}_1$ can be rescued with the modification mentioned above, the line of argument in $\mathcal{B}_1$ has another inherent – much more serious – problem, which we describe next.

(ii) Ambiguity due to the choice of $\hat{\mathsf{id}}$*:* $\mathcal{B}_1$ sets the identity involved in the $\hat{\imath}^{\text{th}}$ G-oracle query as the target identity $\hat{\mathsf{id}}$ (see $\mathcal{B}_1$.1). Hence, the target identity can be fixed only *after* the $\hat{\imath}^{\text{th}}$ query to the G-oracle has been made. However, whenever a signature query is made on any identity, $\mathcal{B}_1$ has to decide whether the identity is the target identity or not. Therefore, when $\mathcal{A}$ makes a signature query before the $\hat{\imath}^{\text{th}}$ G-oracle call, $\mathcal{B}_1$ has no way to decide whether to proceed to $\mathcal{B}_1$.3 or $\mathcal{B}_1$.4 (as it depends on whether $\mathsf{id} = \hat{\mathsf{id}}$ or not). $\mathcal{B}_1$ can provide a proper simulation of the protocol environment only if *no* signature query is made on the target identity $\hat{\mathsf{id}}$ *before* the $\hat{\imath}^{\text{th}}$ G-oracle call. However, $\mathcal{B}_1$ cannot really restrict the adversarial strategy this way. In fact, $\mathcal{B}_1$ will fail to give a proper simulation of the protocol environment if $\mathcal{A}$ makes one signature query on $\hat{\mathsf{id}}$ before the $\hat{\imath}^{\text{th}}$ G-oracle query and one more signature query on $\hat{\mathsf{id}}$ after the $\hat{\imath}^{\text{th}}$ G-oracle query.

One way to fix the problem noted above is to guess the "index" of the target identity *instead* of guessing the index of the G-oracle query in which the target identity is involved. Suppose n distinct identities are involved in the queries to the G-oracle, where $1 \leq n \leq q_G$.[1] The strategy would be to guess the index $\hat{\imath}$ of the target identity $\hat{\mathsf{id}}$ among all the identities, i.e. if $\{\mathsf{id}_1, \ldots, \mathsf{id}_n\}$ were the *distinct* identities involved in the queries to the G-oracle (in that order), we set $\mathsf{id}_{\hat{\imath}}$ with $1 \leq \hat{\imath} \leq n$ as the target identity. Now, by assumption no identity queried to the G-oracle prior to $\mathsf{id}_{\hat{\imath}}$ can be the target identity. Hence, the ambiguity noted before can be avoided. Although this strategy works well with the "mended" reduction which we ended up in *Observation* (i), it will still incur a tightness loss of the order $\mathcal{O}\left(q_{\mathrm{G}}^3\right)$.

In our alternative security argument given in §3, we show how to get around the problem in $\mathcal{B}_1$ by using Coron's technique, together with some algebraic manipulation and non-trivial random oracle programming. In addition to correcting the errors in $\mathcal{B}_1$, we end up with a much tighter reduction as a result.

2.2 Observations on $\mathcal{B}_2$.

We now note the following points about the reduction $\mathcal{B}_2$ and, as in $\mathcal{B}_1$, we discuss the possible fixes.

(i) Incorrect solution of the DLP instance : In Step $\mathcal{B}_2$.4 (see [6] for details), the reduction obtains the solution of the DLP instance by solving the four equations given in (2) of [6]. However, on substituting the values of b_ks from those equations in (3) of [6], we get

$$\frac{b_2 + b_1 - b_0 - b_3}{c_1(d_2 - d_3) - c_0(d_0 - d_1)} = \alpha + \log_g^{\mathbb{G}} R \cdot \frac{d_2 + d_1 - d_0 - d_3}{c_1(d_2 - d_3) - c_0(d_0 - d_1)}, \quad (1)$$

[1] $\mathcal{B}_1$ will maintain a counter and increment it by 1 each time a new identity is queried to the G-oracle.

which is *not* the correct solution to the DLP instance. Note that the simulator does not know the value of $\log_g^{\mathbb{G}} R$ and hence cannot extract α from the above expression. However, it is not difficult to get the correct solution. The more fundamental problem is that $\mathcal{B}_2$ fails to capture all possible adversarial strategies as we show next.

(ii) Incompleteness of $\mathcal{B}_2$: In Step $\mathcal{B}_2$.4, $\mathcal{B}_2$ runs $\mathcal{M}_{\mathcal{Y},3}$[2] to get four forged signatures as shown in (2) of [6]. The b_k component of the forged signatures, though, need not always have this particular structure. The structure depends on the precise order in which $\mathcal{A}$ makes the oracle calls: $\mathrm{G}(\mathsf{id}, A, m)$ and $\mathrm{H}(R, \mathsf{id})$, during the simulation. (Here, (id, m) corresponds to the target identity and the message pair in the forgery while (A, R) are part of the forged signature.) Thus, only one of the two possible adversarial behaviors is covered in $\mathcal{B}_2$. We call this Case 1:– $\mathcal{A}$ calling the H-oracle before the G-oracle. But one cannot rule out the second case, i.e. Case 2:– $\mathcal{A}$ calling the G-oracle before the H-oracle.

Let's look into the structure of the forged signatures in Case 2. As a result of the ordering of the oracle calls, $\mathcal{Y}$ returns J_0 as the index of the G-oracle call on (id, A, m) and I_0 as the index of the H-oracle call on (R_0, id), at the end of Run 0. As G-oracle is forked before the H-oracle, we get $d_1 = d_0$, $d_3 = d_2$ and $R_1 = R_0$, $R_3 = R_2$ in the subsequent forkings, while all the c_is, $0 \leq i \leq 3$ will be different. On the other hand, the value A returned as part of the forged signature remains the same in all the four runs. Hence, the signatures returned by $\mathcal{M}_{\mathcal{Y},3}$ will contain b_ks of the form

$$\begin{aligned} b_0 &= \log A + (\log R_0 + c_0\alpha)d_0\ , & b_1 &= \log A + (\log R_0 + c_1\alpha)d_0, \\ b_2 &= \log A + (\log R_2 + c_2\alpha)d_2 \ \text{ and } & b_3 &= \log A + (\log R_2 + c_3\alpha)d_2. \end{aligned} \tag{2}$$

When the signatures have the structure as in (2), we cannot use the original equation to get a solution of the DLP. This is because $d_1 = d_0$ and $d_3 = d_2$ makes the denominator part in the corresponding expression zero. As we cannot rule out this particular adversary, the reduction does not address all the cases, rendering it incomplete.

To summarize, the same strategy to solve the DLP will *not* work for the two aforementioned complementary cases. Still it is possible to distinguish between the two cases, Case 1 and Case 2, simply by looking at the structure of the forged signatures. In Case 1, all the Rs will be equal, i.e. $R_3 = R_2 = R_1 = R_0$; on the other hand, in Case 2, all the As will be equal, i.e. $A_3 = A_2 = A_1 = A_0$. We could then use the appropriate relations to solve for the DLP instance. However, this results in an *unnecessary* forking being executed in Case 2. We address this in §3 by *splitting* $\mathcal{B}_2$ into two reductions $\mathcal{R}_2$ and $\mathcal{R}_3$, with $\mathcal{R}_2$ involving only a single forking. The single forking, in turn, leads to a significantly tighter reduction (see *Table* 1).

[2] See the full version of the paper [6] for explanation of the general forking algorithm $\mathcal{F}_{\mathcal{Y}}$ and the multiple-forking algorithm $\mathcal{M}_{\mathcal{Y},n}$.

3 New Security Argument

On the basis of the observations made in the previous section, we now proceed to provide a detailed security argument for Galindo-Garcia IBS. In a nutshell, we have effectively *modularised* the security argument into three mutually exclusive parts so that each of the three situations mentioned in the previous section can be studied in more detail. We also show that it is possible to obtain significantly tighter reductions in two of the three cases.

In order to address the problem in $\mathcal{B}_1$ we redefine the event E and to address the incompleteness of $\mathcal{B}_2$ we introduce another event F. The security argument involves constructing three algorithms: $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ and in each of them solving the DLP is reduced to breaking the IBS. $\mathcal{R}_1$, unlike its counterpart $\mathcal{B}_1$, uses the general forking algorithm, whereas $\mathcal{R}_2$ and $\mathcal{R}_3$, the counterparts of $\mathcal{B}_2$, still use the multiple-forking algorithm. The new reductions $\mathcal{R}_1$ and $\mathcal{R}_2$ are also tighter than their counterparts in [12]. We refer the reader to the full version of this paper [6] for a comprehensive explanation on the working of the general forking algorithm $\mathcal{F}_\mathcal{Y}$ [2] and the multiple-forking algorithm $\mathcal{M}_{\mathcal{Y},n}$ [4].

Theorem 1. *Let $\mathcal{A}$ be an $(\epsilon, t, q_\varepsilon, q_s, q_H, q_G)$-adversary against the IBS in the* `EU-ID-CMA` *model. If H and G are modelled as random oracles, we can construct either*

(i) Algorithm $\mathcal{R}_1$ which (ϵ_1, t_1)-breaks the DLP, where

$$\epsilon_1 \geq \frac{\epsilon^2}{\exp(1) q_G q_\varepsilon} \quad \text{and} \quad t_1 \leq t + 2(q_\varepsilon + 3q_s)\tau, \tag{3}$$

(ii) Algorithm $\mathcal{R}_2$ which (ϵ_2, t_2)-breaks the DLP, where

$$\epsilon_2 \geq \epsilon \left(\frac{\epsilon}{(q_H + q_G)^2} - \frac{1}{p} \right) \quad \text{and} \quad t_2 \leq t + 2(2q_\varepsilon + 3q_s)\tau, \tag{4}$$

(iii) Algorithm $\mathcal{R}_3$ which (ϵ_3, t_3)-breaks the DLP, where

$$\epsilon_3 \geq \epsilon \left(\frac{\epsilon^3}{(q_H + q_G)^6} - \frac{3}{p} \right) \quad \text{and} \quad t_3 \leq t + 4(2q_\varepsilon + 3q_s)\tau. \tag{5}$$

Here q_ε and q_s denote the upper bound on the number of extract and signature queries, respectively, that $\mathcal{A}$ can make; q_H and q_G denote the upper bound on the number of queries to the H-oracle and G-oracle respectively. τ is the time taken for an exponentiation in the group $\mathbb{G}$ and exp *is the base of natural logarithm.*

Proof. $\mathcal{A}$ is successful if it produces a valid forgery $\hat{\sigma} = (\hat{A}, \hat{b}, \hat{R})$ on $(\hat{\mathsf{id}}, \hat{m})$. Consider the following event in the case that $\mathcal{A}$ is successful.

E:– $\mathcal{A}$ makes at least one signature query on $\hat{\mathsf{id}}$ and $\hat{R}$ was returned by the simulator as part of the output to a signature query on $\hat{\mathsf{id}}$.

The complement of this event is

$\bar{\mathrm{E}}$:– Either $\mathcal{A}$ does not make any signature queries on $\hat{\mathsf{id}}$ or $\hat{R}$ was never returned by the simulator as part of the output to a signature query on $\hat{\mathsf{id}}$.

Note that the definition of the new event E (and $\bar{\mathrm{E}}$) is slightly different from the one given in the security argument of [12], i.e. event E (and NE).

In order to come up with the forgery $\hat{\sigma}$ with a non-negligible probability, the adversary, at some point during its execution, has to make the two random oracle calls: $\mathrm{H}(\hat{R}, \hat{\mathsf{id}})$ and $\mathrm{G}(\hat{\mathsf{id}}, \hat{A}, \hat{m})$. Depending on the order in which $\mathcal{A}$ makes these calls, we further subdivide the event $\bar{\mathrm{E}}$ into an event F and its complementary event $\bar{\mathrm{F}}$, where

F:– The event that $\mathcal{A}$ makes the call $\mathrm{G}(\hat{\mathsf{id}}, \hat{A}, \hat{m})$ before the call $\mathrm{H}(\hat{R}, \hat{\mathsf{id}})$.
$\bar{\mathrm{F}}$:– The event that $\mathcal{A}$ makes the call $\mathrm{H}(\hat{R}, \hat{\mathsf{id}})$ before the call $\mathrm{G}(\hat{\mathsf{id}}, \hat{A}, \hat{m})$.

In the case of the events E, $\bar{\mathrm{E}} \wedge \mathrm{F}$ and $\bar{\mathrm{E}} \wedge \bar{\mathrm{F}}$, we give the reductions $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ respectively. They are described in the subsequent sections.

Simulating the Random Oracles. A random oracle query is defined to be *fresh* if it is the first query involving that particular input. If a query is not fresh for an input, in order to maintain consistency, the random oracle has to respond with the same output as in the previous query on that input. We say that a fresh query does not require *programming* if the simulator can simply return a random value as the response. The crux of most security arguments involving random oracles, including ours, is the way the simulator answers the queries that require programming. In our case, random oracle programming is used to resolve the circularity involved while dealing with the *implicit* random oracle queries. A random oracle query is said to be implicit if it is not an explicit query from the adversary or the simulator. As usual, to simplify the book-keeping, all implicit random oracle queries involved in answering the extract and signature queries are put into the account of $\mathcal{A}$.

3.1 Reduction $\mathcal{R}_1$

$\mathcal{R}_1$ uses the so-called "partitioning strategy", first used by Coron in the security argument of FDH [8]. The basic idea is to divide the identity-space $\mathbb{I}$ into two disjoint sets, $\mathbb{I}_\varepsilon$ and $\mathbb{I}_s$, depending upon the outcome of a biased coin. The simulator is equipped to respond to both extract and signature queries on identities from $\mathbb{I}_\varepsilon$. But it fails if the adversary does an extract query on any identity from $\mathbb{I}_s$; it can answer only to signature queries on identities from $\mathbb{I}_s$. Finally, the simulator hopes that the adversary produces a forgery on an identity from $\mathbb{I}_s$. The optimal size of the sets is determined on analysis.

In $\mathcal{R}_1$ the problem instance is embedded in the randomiser R, depending on the outcome of a biased coin. As $\mathcal{R}_1$ maintains a unique R for each identity, the structure of R decides whether an identity belongs to $\mathbb{I}_\varepsilon$ or $\mathbb{I}_s$. The details follow.

Let $\Delta := (\mathbb{G}, p, g, g^\alpha)$ be the given DLP instance. $\mathcal{R}_1$ sets $z \in_R \mathbb{Z}_p$ as the *master* secret key. The public parameters $\mathsf{mpk} := (\mathbb{G}, p, g, g^z, \mathrm{H}, \mathrm{G})$ are released to the adversary. The hash functions H and G are modelled as random oracles. This is done with the aid of two tables, $\mathfrak{L}_{\mathrm{H}}$ and $\mathfrak{L}_{\mathrm{G}}$.

Handling the Queries.

H-oracle Query. $\mathfrak{L}_{\mathrm{H}}$ contains tuples of the form

$$\langle R, r, \mathsf{id}, c, \beta \rangle \in \mathbb{G} \times \mathbb{Z}_p \cup \{\bot\} \times \{0,1\}^* \times \mathbb{Z}_p \times \{0, 1, \phi\}.$$

Here, (R, id) is the query to the H-oracle and c is the corresponding output. Therefore, a query $\mathrm{H}(R, \mathsf{id})$ is fresh if there exists no tuple $\langle R_i, r_i, \mathsf{id}_i, c_i, \beta_i \rangle$ in $\mathfrak{L}_{\mathrm{H}}$ such that $(\mathsf{id}_i = \mathsf{id}) \wedge (R_i = R)$. If such a tuple exists, then the oracle has to return c_i as the output.

The r-field is used to store additional information related to the R-field. The tuples corresponding to the explicit H-oracle queries, made by $\mathcal{A}$, are tracked by storing '$\bot$' in the r-field. This indicates that $\mathcal{R}_1$ does not have any additional information regarding R. In these tuples, the β-field is irrelevant and this is indicated by storing 'ϕ'. In tuples with $r \neq \bot$, the field β indicates whether the DLP instance is embedded in R or not. If $\beta = 0$ then $R = (g^\alpha)^r$ for some known $r \in \mathbb{Z}_p$, which is stored in the r-field. On the other hand, $\beta = 1$ implies $R = g^r$ for some known $r \in \mathbb{Z}_p$, which is, again, stored in the r-field. We now explain how the fresh H-oracle queries are handled.

$\mathrm{H}(R, \mathsf{id})$:– The query may be

(i) H_1, Explicit query made by $\mathcal{A}$:– In this case $\mathcal{R}_1$ returns $c \in_R \mathbb{Z}_p$ as the output. $\langle R, \bot, \mathsf{id}, c, \phi \rangle$ is added to $\mathfrak{L}_{\mathrm{H}}$.

(ii) H_2, Explicit query made by $\mathcal{R}_1$:– As in the previous case, $\mathcal{R}_1$ returns $c \in_R \mathbb{Z}_p$ as the output. As $\mathcal{R}_1$ knows $r = \log_g^{\mathbb{G}} R$, $\langle R, r, \mathsf{id}, c, 1 \rangle$ is added to $\mathfrak{L}_{\mathrm{H}}$.

(iii) H_3, Implicit query by $\mathcal{R}_1$ in order to answer a signature query made by $\mathcal{A}$:– See *Sign* (iii) on how to program the random oracle in this situation.

G-oracle Query. $\mathfrak{L}_{\mathrm{G}}$ contains tuples of the form

$$\langle \mathsf{id}, A, m, d \rangle \in \{0,1\}^* \times \mathbb{G} \times \{0,1\}^* \times \mathbb{Z}_p.$$

Here, (id, A, m) is the query to the G-oracle and d is the corresponding output. Therefore, a random oracle query $\mathrm{G}(\mathsf{id}, A, m)$ is fresh if there exists no tuple $\langle \mathsf{id}_i, A_i, m_i, d_i \rangle$, in $\mathfrak{L}_{\mathrm{G}}$ such that $(\mathsf{id}_i = \mathsf{id}) \wedge (A_i = A) \wedge (m_i = m)$. If such a tuple exists, then the oracle has to return d_i as the output. We now explain how the fresh G-oracle queries are handled.

$\mathrm{G}(\mathsf{id}, A, m)$:– The query may be

(i) G_1, Explicit query made by the either $\mathcal{A}$ or $\mathcal{R}_1$:– In this case $\mathcal{R}_1$ returns $d \in_R \mathbb{Z}_p$ as the output. $\langle \mathsf{id}, A, m, d \rangle$ is added to $\mathfrak{L}_{\mathrm{G}}$.

(ii) G_2, Implicit query by $\mathcal{R}_1$ in order to answer a signature query made by $\mathcal{A}$:– See *Sign* (i), (iii) on how to program the random oracle in this situation.

Now that $\mathcal{R}_1$ can handle the random oracle queries, the extract and signature queries are answered as follows.

Extract Query. $\mathcal{O}_\varepsilon(\mathsf{id})$:–

<u>If</u> there exists a tuple $\langle R_i, r_i, \mathsf{id}_i, c_i, \beta_i\rangle$ in $\mathfrak{L}_\text{H}$ such that $(\mathsf{id}_i = \mathsf{id}) \wedge (r_i \neq \perp)$
(i) If $\beta_i = 0$, $\mathcal{R}_1$ *aborts* ($\mathsf{abort}_{1,1}$).
(ii) Otherwise, $\beta_i = 1$ and $\mathcal{R}_1$ returns $\mathsf{usk} := (r_i + zc_i, R_i)$ as the secret key.
<u>Otherwise</u>
(iii) $\mathcal{R}_1$ chooses $r \in_R \mathbb{Z}_p$, sets $R := g^r$ and asks the H-oracle for $c := \text{H}(R, \mathsf{id})$. It returns $\mathsf{usk} := (r + zc, R)$ as the secret key.

Signature Query. $\mathcal{O}_s(\mathsf{id}, m)$:–

<u>If</u> there exists a tuple $\langle R_i, r_i, \mathsf{id}_i, c_i, \beta_i\rangle$ in $\mathfrak{L}_\text{H}$ such that $(\mathsf{id}_i = \mathsf{id}) \wedge (r_i \neq \perp)$
(i) If $\beta_i = 0$, $\mathcal{R}_1$ selects $s, d \in_R \mathbb{Z}_p$ and sets $A := g^s(g^\alpha)^{-r_i d}$. Then (id, A, m, d) is added to $\mathfrak{L}_\text{G}$ (*Deferred case* G_2)[3]. The signature returned is $\sigma := (A, s + zcd, R_i)$.
(ii) Otherwise, $\beta_i = 1$ and the secret key for id is $\mathsf{usk} = (y, R_i)$, where $y = r_i + zc_i$ and $R_i = g^{r_i}$. $\mathcal{R}_1$ selects $a \in_R \mathbb{Z}_p$, sets $A := g^a$ and asks G-oracle for $d := \text{G}(\mathsf{id}, A, m)$. The signature returned is $\sigma := (A, a{+}yd, R_i)$.
<u>Otherwise</u>, $\mathcal{R}_1$ tosses a coin β with a bias δ (i.e, $\Pr[\beta = 0]{=}\delta$). The value of δ will be quantified on analysis.
(iii) If $\beta = 0$, $\mathcal{R}_1$ selects $c, d, s, r \in_R \mathbb{Z}_p$ and sets $R := (g^\alpha)^r$, $A := g^s(g^\alpha)^{-rd}$. Next, it adds $\langle (g^\alpha)^r, r, \mathsf{id}, c, 0\rangle$ to $\mathfrak{L}_\text{H}$ (*Deferred case* H_3) and $\langle \mathsf{id}, A, m, d\rangle$ to $\mathfrak{L}_\text{G}$ (*Deferred case* G_2).[4] The signature returned is $\sigma := (A, s{+}zc_i d, R)$.
(iv) Otherwise, $\beta = 1$ and $\mathcal{R}_1$ selects $a, r \in_R \mathbb{Z}_p$ and sets $A := g^a, R := g^r$. It then asks the respective oracles for $c := \text{H}(R, \mathsf{id})$ and $d := \text{G}(\mathsf{id}, A, m)$. The signature returned is $\sigma := (A, a + (r + zc)d, R)$.

Correctness. For $\beta = 0$, the signature given by $\mathcal{R}_1$ is of the form (A, b, R), where $A = g^s(g^\alpha)^{-rd}$, $b = s + zcd$ and $R = (g^\alpha)^r$. $\mathcal{R}_1$ also sets $c := \text{H}(R, \mathsf{id})$ and $d := \text{G}(\mathsf{id}, A, m)$. The signature verifies as shown below.

$$\begin{aligned} g^b &= g^{s+zcd} = g^{s-\alpha rd+\alpha rd+zcd} \\ &= g^s(g^\alpha)^{-rd}((g^\alpha)^r(g^z)^c)^d = A(R(g^z)^c)^d. \end{aligned}$$

For $\beta = 1$, the signatures are generated as in the protocol. Therefore they fundamentally verify.

To conclude the queries section, we calculate the probability of the event $\neg\mathsf{abort}_{1,1}$. $\mathcal{R}_1$ aborts only when $\mathcal{A}$ does an extract query on an identity from $\mathbb{I}_s$, i.e. an identity with $\beta = 0$. Therefore, $\mathcal{R}_1$ does not abort if all the extract queries are from $\mathbb{I}_\varepsilon$ and we have

$$\Pr[\neg\mathsf{abort}_{1,1}] = (1 - \delta)^{q_\varepsilon}. \tag{6}$$

[3] If there exists a tuple $\langle \mathsf{id}_i, A_i, m_i, d_i\rangle$ in $\mathfrak{L}_\text{G}$ with $\mathsf{id}_i = \mathsf{id} \wedge A_i = A \wedge m_i = m$ but $d_i \neq d$ then $\text{G}(\mathsf{id}, A, m)$ cannot be set to d. In that case $\mathcal{R}_1$ can simply choose a fresh set of randomisers s, d and repeat the process.

[4] $\mathcal{R}_1$ chooses different randomisers if there is a collision as explained in *Footnote* 3.

Solving the DLP. $\mathcal{R}_1$ now uses the general forking algorithm $\mathcal{F}_\mathcal{Y}$ to solve the DLP challenge. It runs $\mathcal{F}_\mathcal{Y}$ on the given DLP instance Δ, with the G-oracle involved in the replay attack. If $\mathcal{F}_\mathcal{Y}$ fails, $\mathcal{R}_1$ *aborts* ($\mathsf{abort}_{1,2}$). On the other hand, if $\mathcal{F}_\mathcal{Y}$ is successful, it gets two valid forgeries

$$\hat{\sigma}_0 = (\hat{A}, \hat{b}_0, \hat{R}_0) \text{ and } \hat{\sigma}_1 = (\hat{A}, \hat{b}_1, \hat{R}_1)$$

on $(\hat{\mathsf{id}}, \hat{m})$. $\mathcal{R}_1$ now retrieves two tuples

$$\mathsf{t}_i := \langle R_i, r_i, \mathsf{id}_i, c_i, \beta_i \rangle \mid (\mathsf{id}_i = \hat{\mathsf{id}}) \wedge (R_i = \hat{R}_0) \text{ and}$$

$$\mathsf{t}_j := \langle R_j, r_j, \mathsf{id}_j, c_j, \beta_j \rangle \mid (\mathsf{id}_j = \hat{\mathsf{id}}) \wedge (R_j = \hat{R}_1)$$

from $\mathfrak{L}_\mathrm{H}$. $\mathcal{R}_1$ *aborts* ($\mathsf{abort}_{1,3}$) if both β_i and β_j are equal to 1. Otherwise it solves for α as shown below. Note that d_0 and d_1 represent the value of $\mathrm{G}(\hat{\mathsf{id}}, \hat{A}, \hat{m})$ in the two runs. Let $\hat{a} := \log_g^{\mathbb{G}} \hat{A}$.

(i) $(\beta_i = 1) \wedge (\beta_j = 0)$:– In this case, $\hat{R}_0 = g^{r_i}$ and $\hat{R}_1 = g^{r_j \alpha}$. Thus we have $\hat{b}_0 = \hat{a} + (r_i + zc_i)d_0$ and $\hat{b}_1 = \hat{a} + (r_j\alpha + zc_j)d_1$.

$$\alpha = \frac{z(c_i d_0 - c_j d_1) + r_i d_0 - (\hat{b}_0 - \hat{b}_1)}{r_j d_1}. \tag{7}$$

(ii) $(\beta_i = 0) \wedge (\beta_j = 1)$:– In this case, $\hat{R}_0 = g^{r_i \alpha}$ and $\hat{R}_1 = g^{r_j}$. Thus we have $\hat{b}_0 = \hat{a} + (r_i\alpha + zc_i)d_0$ and $\hat{b}_1 = \hat{a} + (r_j + zc_j)d_1$.

$$\alpha = \frac{z(c_j d_1 - c_i d_0) + r_j d_1 - (\hat{b}_1 - \hat{b}_0)}{r_i d_0}. \tag{8}$$

(iii) $(\beta_i = 0) \wedge (\beta_j = 0)$:– In this case, $\hat{R}_0 = g^{r_i \alpha}$ and $\hat{R}_1 = g^{r_j \alpha}$. Thus we have $\hat{b}_0 = \hat{a} + (r_i\alpha + zc_i)d_0$ and $\hat{b}_1 = \hat{a} + (r_j\alpha + zc_j)d_1$.

$$\alpha = \frac{(\hat{b}_0 - \hat{b}_1) - z(c_i d_0 - c_j d_1)}{(r_i d_0 - r_j d_1)}. \tag{9}$$

Remark 1. The equations (7), (8) and (9) hold even if $\hat{R}_1 = \hat{R}_0$ (and consequently $r_j = r_i$ and $c_j = c_i$). Note that this can happen if the adversary makes the random oracle query $\mathrm{H}(\hat{R}_0, \hat{\mathsf{id}})$ before the query $\mathrm{G}(\hat{\mathsf{id}}, \hat{A}, \hat{m})$ in $\mathsf{Run\ 0}$. Hence, the order in which $\mathcal{A}$ makes the aforementioned random oracle calls is not relevant.

We conclude by calculating the probability of $\mathsf{abort}_{1,3}$ provided $\mathsf{abort}_{1,2}$ has not occurred. It is same as the probability with which $(\beta_i = 1) \wedge (\beta_j = 1)$, i.e.

$$\Pr[\mathsf{abort}_{1,3} \mid \neg\mathsf{abort}_{1,2}] = (1-\delta)^2. \tag{10}$$

Let gfrk be the probability with which $\mathcal{F}_\mathcal{Y}$ is successful. Since $\mathsf{abort}_{1,2}$ occurs if $\mathcal{F}_\mathcal{Y}$ fails, we have

$$\Pr[\neg\mathsf{abort}_{1,2}] = \mathsf{gfrk}. \tag{11}$$

Analysis. The probability analysis is done in terms of the aborts $\mathsf{abort}_{1,1}$, $\mathsf{abort}_{1,2}$ and $\mathsf{abort}_{1,3}$ from (6), (11) and (10). $\mathcal{F}_{\mathcal{Y}}$ is successful during $\mathsf{Run\ 0}$ if there is no abort during the query phase ($\neg\mathsf{abort}_{1,1}$) and $\mathcal{A}$ produces a valid forgery. We denote this probability by acc_1. Thus

$$\mathsf{acc}_1 \geq \Pr\left[\neg\mathsf{abort}_1\right] \cdot \epsilon \geq (1-\delta)^{q_\varepsilon} \cdot \epsilon.$$

Applying the general forking lemma with $\mid \mathbb{S} \mid = p$ and $\gamma = q_{\mathrm{G}}$, we get

$$\mathsf{gfrk} \geq \mathsf{acc}_1 \cdot \left(\frac{\mathsf{acc}_1}{q_{\mathrm{G}}} - \frac{1}{p}\right) \geq (1-\delta)^{q_\varepsilon}\epsilon \cdot \left(\frac{(1-\delta)^{q_\varepsilon}\epsilon}{q_{\mathrm{G}}} - \frac{1}{p}\right).$$

If we assume $p \gg 1$, the above expression approximates to

$$\mathsf{gfrk} \geq \frac{(1-\delta)^{2q_\varepsilon}\epsilon^2}{q_{\mathrm{G}}}.$$

Now, $\mathcal{R}_1$ is successful in solving DLP if neither of the aborts, $\mathsf{abort}_{1,2}$ and $\mathsf{abort}_{1,3}$, occur. Thus the advantage it has is

$$\begin{aligned}\epsilon_1 &= \Pr\left[\neg\mathsf{abort}_{1,3} \wedge \neg\mathsf{abort}_{1,2}\right] = \Pr\left[\neg\mathsf{abort}_{1,3} \mid \neg\mathsf{abort}_{1,2}\right] \cdot \Pr\left[\neg\mathsf{abort}_{1,2}\right] \\ &\geq \left(1-(1-\delta)^2\right) \cdot \mathsf{gfrk} \geq (2\delta - \delta^2)\frac{(1-\delta)^{2q_\varepsilon}\epsilon^2}{q_{\mathrm{G}}}.\end{aligned}$$

The above expression is maximised when $\delta = \left(1 - \sqrt{\frac{q_\varepsilon - 2}{q_\varepsilon - 1}}\right)$, at which we get

$$\epsilon_1 \geq \frac{1}{\exp(1)(q_\varepsilon - 1)}\left(1 - \frac{1}{q_\varepsilon - 1}\right)\frac{\epsilon^2}{q_{\mathrm{G}}}.$$

Here, exp is the base of natural logarithm. Assuming $q_\varepsilon \gg 1$, we get the approximation

$$\epsilon_1 \geq \frac{\epsilon^2}{\exp(1) q_{\mathrm{G}} q_\varepsilon}.$$

Remark 2. The above reduction is tighter than the original reduction $\mathcal{B}_1$ given in [12]. This can be attributed to two reasons: (i) $\mathcal{R}_1$ using the general forking algorithm $\mathcal{F}_{\mathcal{Y}}$ instead of the multiple-forking algorithm $\mathcal{M}_{\mathcal{Y},1}$ and (ii) $\mathcal{B}_1$ in [12] randomly chooses one of the identities involved in the G-oracle call as the target identity which contributes a factor of q_{G} to the degradation in $\mathcal{B}_1$. In contrast, we apply Coron's technique in $\mathcal{R}_1$ to partition the identity space in some optimal way.

Time Analysis. If τ is the time taken for an exponentiation in $\mathbb{G}$ then the time taken by $\mathcal{R}_1$ is $t_1 \leq t + 2(q_\varepsilon + 3q_s)\tau$. It takes at most one exponentiation for answering the extract query and three exponentiations for answering the signature query. This contributes the $(q_\varepsilon + 3q_s)\tau$ factor in the running time. The factor of two comes from the forking algorithm, since it involves running the adversary twice.

3.2 Reductions $\mathcal{R}_2$ and $\mathcal{R}_3$

The reduction $\mathcal{R}_2$ is similar in some aspects to the (incomplete) reduction $\mathcal{B}_2$ in [12]. However, a *major* difference is that $\mathcal{R}_2$ uses the multiple-forking algorithm $\mathcal{M}_{\mathcal{Y},1}$ instead of $\mathcal{M}_{\mathcal{Y},3}$ to solve the DLP challenge. Therefore, only one forking is involved leading to a much tighter reduction than $\mathcal{B}_2$. This is described in *Appendix* A. As for the reduction $\mathcal{R}_3$, the approach used is the same as in the reduction $\mathcal{B}_2$ in [12]. Therefore, we refer the reader to the full version of the paper [6] for the description.

3.3 A Comparison with the Original Reduction.

Recall that we replaced the reduction $\mathcal{B}_1$ in the original security argument with the new reduction $\mathcal{R}_1$. Similarly, $\mathcal{B}_2$ was replaced with the two reductions $\mathcal{R}_2$ and $\mathcal{R}_3$. The resulting effect on tightness is tabulated below. The security degradation involved in original $\mathcal{B}_1$ is of the order $\mathcal{O}\left(q_{\mathrm{G}}^3\right)$. In comparison, $\mathcal{R}_1$ incurs a degradation of order $\mathcal{O}\left(q_{\mathrm{G}}q_{\varepsilon}\right)$ which is much lower than that of $\mathcal{B}_1$. Note that $q_{\mathrm{G}} \gg q_{\varepsilon}$, i.e. the bound on the number of random oracle queries is much greater than the bound on the number of extract queries. For example, for 80-bit security one usually assumes $q_{\mathrm{G}} \approx 2^{60}$ while $q_{\varepsilon} \approx 2^{30}$. The degradation involved in the original $\mathcal{B}_2$ would be of the order of $\mathcal{O}\left((q_{\mathrm{G}}q_{\mathrm{H}})^6\right)$ (as pointed out in *Footnote* 5). In comparison, the security degradation involved in $\mathcal{R}_2$ and $\mathcal{R}_3$ is of order $\mathcal{O}\left((q_{\mathrm{G}}+q_{\mathrm{H}})^2\right)$ and $\mathcal{O}\left((q_{\mathrm{G}}+q_{\mathrm{H}})^6\right)$ respectively.

Table 1. A comparison of degradation in the original [12] and the new argument

Original reductions [12]	$\mathcal{B}_1$	$\mathcal{B}_2$	
Degradation	$\mathcal{O}\left(q_{\mathrm{G}}^3\right)$	$\mathcal{O}\left((q_{\mathrm{G}}q_{\mathrm{H}})^6\right)$	

Our new reductions	$\mathcal{R}_1$	$\mathcal{R}_2$	$\mathcal{R}_3$
Degradation	$\mathcal{O}\left(q_{\mathrm{G}}q_{\varepsilon}\right)$	$\mathcal{O}\left((q_{\mathrm{G}}+q_{\mathrm{H}})^2\right)$	$\mathcal{O}\left((q_{\mathrm{G}}+q_{\mathrm{H}})^6\right)$

4 Conclusion

In this work we have identified certain shortcomings in the original security argument of the Galindo-Garcia IBS. Based on our observations we provide a new elaborate security argument for the same scheme. Two of the reductions are significantly tighter than their counterparts in the original security argument in [12]. However, all the reductions are still non-tight. We would like to pose the question of constructing an identity-based signature scheme in discrete-log setting (without pairing) with a tighter security reduction as an interesting open research problem.

Acknowledgements. We thank the anonymous reviewers for their helpful suggestions.

References

1. Bellare, M., Namprempre, C., Neven, G.: Security proofs for identity-based identification and signature schemes. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 268–286. Springer, Heidelberg (2004)
2. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, pp. 390–399. ACM, New York (2006)
3. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security, CCS 1993, pp. 62–73. ACM, New York (1993)
4. Boldyreva, A., Palacio, A., Warinschi, B.: Secure proxy signature schemes for delegation of signing rights. Journal of Cryptology 25, 57–115 (2012)
5. Boneh, D., Boyen, X.: Efficient selective-id secure identity-based encryption without random oracles. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004)
6. Chatterjee, S., Kamath, C., Kumar, V.: Galindo-Garcia identity-based signature revisited. Cryptology ePrint Archive, Report 2012/646 (2012)
7. Choon, J., Cheon, J.H.: An identity-based signature from gap Diffie-Hellman groups. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 18–30. Springer, Heidelberg (2002)
8. Coron, J.-S.: On the exact security of full domain hash. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 229–235. Springer, Heidelberg (2000)
9. Sharmila Deva Selvi, S., Sree Vivek, S., Pandu Rangan, C.: Identity-based deterministic signature scheme without forking-lemma. In: Iwata, T., Nishigaki, M. (eds.) IWSEC 2011. LNCS, vol. 7038, pp. 79–95. Springer, Heidelberg (2011)
10. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
11. Galindo, D.: Boneh-franklin identity based encryption revisited. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 791–802. Springer, Heidelberg (2005)
12. Galindo, D., Garcia, F.D.: A Schnorr-like lightweight identity-based signature scheme. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 135–148. Springer, Heidelberg (2009)
13. Guillou, L.C., Quisquater, J.-J.: A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 216–231. Springer, Heidelberg (1990)
14. Herranz, J.: Deterministic identity-based signatures for partial aggregation. The Computer Journal 49(3), 322–330 (2005)
15. Hess, F.: Efficient identity based signature schemes based on pairings. In: Nyberg, K., Heys, H. (eds.) SAC 2002. LNCS, vol. 2595, pp. 310–324. Springer, Heidelberg (2003)
16. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology 13, 361–396 (2000)
17. Radhakishan, V., Selvakumar, S.: Prevention of man-in-the-middle attacks using id-based signatures. In: Second International Conference on Networking and Distributed Computing - ICNDC 2011, pp. 165–169 (2011)
18. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985)

19. Shoup, V.: OAEP reconsidered. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 239–259. Springer, Heidelberg (2001)
20. Xie, M., Wang, L.: One-round identity-based key exchange with perfect forward security. Information Processing Letters 112(14-15), 587–591 (2012)

A Reduction $\mathcal{R}_2$

Let $\varDelta := (\mathbb{G}, p, g, g^{\alpha})$ be the given DLP instance. $\mathcal{R}_2$ sets $(\mathbb{G}, p, g, g^{\alpha}, \mathrm{H}, \mathrm{G})$ as public parameters mpk and releases it to $\mathcal{A}$. Note that $\mathcal{R}_2$ does not know the *master* secret key msk, which is α, the solution to the DLP challenge. The hash functions H and G are modelled as random oracles. This is done with the aid of two tables, $\mathfrak{L}_{\mathrm{H}}$ and $\mathfrak{L}_{\mathrm{G}}$.

A.1 Handling the Queries

H-oracle Query. $\mathfrak{L}_{\mathrm{H}}$ contains tuples of the form

$$\langle R, \mathsf{id}, c, y\rangle \in \mathbb{G} \times \{0,1\}^* \times \mathbb{Z}_p \times \mathbb{Z}_p \cup \{\perp\}.$$

Here, (R, id) is the query to the H-oracle and c the corresponding output. The y-field stores either the corresponding component of the secret key for id or '$\perp$' if the field is invalid. A random oracle query $\mathrm{H}(R, \mathsf{id})$ is fresh if there exists no tuple $\langle R_i, \mathsf{id}_i, c_i, y_i\rangle$ in $\mathfrak{L}_{\mathrm{H}}$ such that $(\mathsf{id}_i = \mathsf{id}) \wedge (R_i = R)$. If such a tuple exists, then the oracle has to return c_i as the output. We now explain how the H-oracle queries are answered.

$\mathrm{H}(R, \mathsf{id})$:– The query may be

(i) H_1, Explicit query made by $\mathcal{A}$:– In this case $\mathcal{R}_2$ returns $c \in_R \mathbb{Z}_p$ as the output. $\langle R, \mathsf{id}, c, \perp\rangle$ is added to $\mathfrak{L}_{\mathrm{H}}$.

(ii) H_2, Implicit query by $\mathcal{R}_2$ in order to answer an extract query made by $\mathcal{A}$:– See *Extract* (ii) on how to program the random oracle in this situation.

G-oracle Query. $\mathfrak{L}_{\mathrm{G}}$ has the same structure as in $\mathcal{R}_1$ (See §3.1). The queries to G-oracle are handled as shown below.

$\mathrm{G}(\mathsf{id}, A, m)$:– $\mathcal{R}_2$ returns $d \in_R \mathbb{Z}_p$ as the output. $\langle \mathsf{id}, A, m, d\rangle$ is added to $\mathfrak{L}_{\mathrm{G}}$.

Signature and Extract Queries. Since $\mathcal{R}_2$ does not know the *master* secret key α, it has to use the algebraic technique used in $\mathcal{R}_1$ to come up with the secret key corresponding to an identity. The choice of R and c enables it to give the secret key. The circularity involved in this choice is resolved by programming the H-oracle appropriately. Signature queries are answered by generating the usk as in the extract query, followed by calling $\mathcal{S}$.

Extract query. $\mathcal{O}_\varepsilon(\mathsf{id})$:–

(i) If there exists a tuple $\langle R_i, \mathsf{id}_i, c_i, y_i\rangle$ in $\mathfrak{L}_{\mathrm{H}}$ such that $(\mathsf{id}_i = \mathsf{id}) \wedge (y_i \neq \perp)$, $\mathcal{R}_2$ returns $\mathsf{usk} := (y_i, R_i)$ as the secret key.

(ii) Otherwise, it chooses $c, y \in_R \mathbb{Z}_p$, sets $R := (g^\alpha)^{-c} g^y$ and adds $\langle R, \mathsf{id}, c, y\rangle$ to $\mathfrak{L}_{\mathrm{H}}$(*Deferred case* H_2). It returns $\mathsf{usk} := (y, R)$ as the secret key.

Signature query. $\mathcal{O}_s(\mathsf{id}, m)$:–

(i) If there exists a tuple $\langle R_i, \mathsf{id}_i, c_i, y_i\rangle$ in $\mathfrak{L}_{\mathrm{H}}$ such that $(\mathsf{id}_i = \mathsf{id}) \wedge (y_i \neq \perp)$, then $\mathsf{usk} = (y_i, R_i)$. $\mathcal{R}_2$ now uses the knowledge of usk to run $\mathcal{S}$ and returns the signature.

(ii) Otherwise, $\mathcal{R}_2$ generates the usk as in *Extract*(ii) and runs $\mathcal{S}$ to return the signature.

We conclude the queries section with the remark that $\mathcal{R}_2$ never aborts during the query stage.

A.2 Solving the DLP

$\mathcal{R}_2$ now uses the multiple-forking algorithm $\mathcal{M}_{\mathcal{Y},1}$ to solve the DLP challenge. It runs $\mathcal{M}_{\mathcal{Y},1}$ on mpk, with both H and G-oracle involved in the replay attack. If $\mathcal{M}_{\mathcal{Y},1}$ fails, $\mathcal{R}_2$ *aborts* ($\mathsf{abort}_{2,1}$). On the other hand, if $\mathcal{M}_{\mathcal{Y},1}$ is successful, $\mathcal{R}_2$ gets two valid forgeries

$$\hat{\sigma}_0 = (\hat{A}, \hat{b}_0, \hat{R}) \text{ and } \hat{\sigma}_1 = (\hat{A}, \hat{b}_1, \hat{R}) \tag{12}$$

on $(\hat{\mathsf{id}}, \hat{m})$ with

$$\hat{b}_0 = \hat{a} + (\hat{r} + \alpha c_0)\hat{d} \text{ and } \hat{b}_1 = \hat{a} + (\hat{r} + \alpha c_1)\hat{d}, \tag{13}$$

where $\hat{a} := \log_g^{\mathbb{G}} \hat{A}$ and $\hat{r} := \log_g^{\mathbb{G}} \hat{R}$. Note that c_0 and c_1 represent the value of $\mathrm{H}(\hat{R}, \hat{\mathsf{id}})$ in the two runs. Also, the event F guarantees that $\mathcal{A}$ makes the G-oracle call $\mathrm{G}(\hat{\mathsf{id}}, \hat{A}, \hat{m})$, before the H-oracle call $\mathrm{H}(\hat{R}, \hat{\mathsf{id}})$. Finally it outputs the solution to the DLP instance,

$$\alpha = \frac{\hat{b}_0 - \hat{b}_1}{\hat{d}(c_0 - c_1)}. \tag{14}$$

Structure of the forgeries. Now we justify the structure of the component b of the forgeries given in (13). Recall that the signature queries are answered by doing an extract query on the identity followed by calling $\mathcal{S}$. Therefore, the resultant secret keys are of the form $\mathsf{usk} = (y, R)$, where $R = (g^\alpha)^{-c} g^y$ and we have

$$r = -\alpha c + y.$$

If a forgery is produced using the same R as given by $\mathcal{R}_2$ as part of the signature query on id, then b will be of the form $b = a + (-\alpha c + y + \alpha c)d = a + yd$. Therefore, it will not contain the solution to the DLP challenge α, and such forgeries are of no use to $\mathcal{R}_2$. But the event $\bar{\mathrm{E}}$ guarantees that $\mathcal{A}$ does not forge using an

R which was given as part of the signature query on id and hence, for the forgery to be valid b will necessarily be of the form:

$$b = a + (r + \alpha c)d. \tag{15}$$

We conclude with the remark that the event $\mathsf{abort}_{2,1}$ does not occur if the multiple-forking algorithm is successful (let this probability be mfrk). Therefore

$$\Pr\left[\neg\mathsf{abort}_{2,1}\right] = \mathsf{mfrk}. \tag{16}$$

A.3 Analysis

The only abort involved in $\mathcal{R}_2$ is $\mathsf{abort}_{2,1}$, which occurs when $\mathcal{M}_{\mathcal{Y},1}$ fails. Therefore $\mathcal{R}_2$ is successful if $\mathcal{M}_{\mathcal{Y},1}$ is and from (16) we have

$$\epsilon_2 = \Pr\left[\neg\mathsf{abort}_{2,1}\right] = \mathsf{mfrk}.$$

We denote the probability with which $\mathcal{M}_{\mathcal{Y},1}$ is successful during $\mathsf{Run\ 0}$ as acc_2. Since there is no abort involved during query phase, $\mathcal{M}_{\mathcal{Y},1}$ is successful during $\mathsf{Run\ 0}$ if $\mathcal{A}$ produces a valid forgery, i.e. $\mathsf{acc}_2 = \epsilon$. Applying the multiple-forking lemma with $n := 1$, $\gamma := q_{\mathrm{H}} + q_{\mathrm{G}}$,[5] and $|\ \mathbb{S}\ | = p$, we have

$$\begin{aligned}\epsilon_2 = \mathsf{mfrk} &\geq \mathsf{acc}_2 \cdot \left(\frac{\mathsf{acc}_2}{(q_{\mathrm{H}} + q_{\mathrm{G}})^2} - \frac{1}{p}\right)\\ &\geq \epsilon\left(\frac{\epsilon}{(q_{\mathrm{H}} + q_{\mathrm{G}})^2} - \frac{1}{p}\right).\end{aligned}$$

Time Analysis. Drawing analogy from the time analysis of $\mathcal{R}_1$, the time taken by $\mathcal{R}_2$ is easily seen to be bounded by $t_2 \leq t + 2(2q_\varepsilon + 3q_s)\tau$.

[5] In the analysis of $\mathcal{B}_2$ in [12], γ was assumed to be $q_{\mathrm{H}} \cdot q_{\mathrm{G}}$. However, γ actually denotes the size of the set of responses to the random oracle queries involved in the replay attack. As both H and G-oracle is involved in the replay attack in $\mathcal{B}_2$, the size of the set is $q_{\mathrm{H}} + q_{\mathrm{G}}$ rather than $q_{\mathrm{H}} \cdot q_{\mathrm{G}}$.

Private Over-Threshold Aggregation Protocols

Myungsun Kim[1,⋆], Abedelaziz Mohaisen[2], Jung Hee Cheon[3,⋆], and Yongdae Kim[4]

[1] University of Suwon, Suwon, South Korea
msunkim@suwon.ac.kr
[2] VeriSign Labs, VA, USA
amohaisen@verisign.com
[3] Seoul National University, Seoul, South Korea
jhcheon@snu.ac.kr
[4] Korea Advanced Institute of Science and Technology, Daejeon, South Korea
yongdaek@ee.kaist.ac.kr

Abstract. In this paper, we revisit the private κ^+ data aggregation problem, and formally define the problem's security requirements as both data and user privacy goals. To achieve both goals, and to strike a balance between efficiency and functionality, we devise a novel cryptographic construction that comes in two schemes; a fully decentralized construction and its practical but semi-decentralized variant. Both schemes are provably secure in the semi-honest model. We analyze the computational and communication complexities of our construction, and show that it is much more efficient than the existing protocols in the literature.

Keywords: Privacy-preservation, over-threshold, data privacy, user privacy.

1 Introduction

Of particular interest in many applications is the problem of *computing the over-threshold elements*, elements whose count is greater than a given value, in a *private* manner. A typical application that involves such primitive is network traffic distribution, where n network sensors need to jointly analyze the security alert broadcasted by different sources in order to find suspect sites. In such an application, and without losing generality, each of such sensors has a set of suspects and would like to collaboratively compute the most frequent on each of these sets (e.g., the count greater than κ, referred to as κ^+) without revealing the set of suspects to other sensors with whom she collaborates.

Problem Definition: Let there be n users denoted by $u_i, 1 \le i \le n$, and each of them has a private (multi-)set $\mathtt{X}_i$ of cardinality k. For simplicity, assume that the cardinality of each multiset is equal to each other. (We can efficiently handle the case where the cardinality of all multisets are different from each other by adding random elements.) There may exist one or more elements such that $\alpha_{i,j} = \alpha_{i,j'}$ for some $j \neq j'$.

PRIVATE κ^+ AGGREGATION PROBLEM: By the multiplicity of an element of a multiset we mean the number of times it appears in the multiset. Let $\kappa \in \mathbb{N}$, and assume κ

⋆ This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2012-0001243).

has been implicitly predefined among all users. Then the problem at hand is defined as follows: Given n multisets of cardinality k, find a set $Z = \{\alpha_1, \dots, \alpha_\kappa\} \subset \mathtt{U} = \bigcup_{i=1}^{n} \mathtt{X}_i$ such that (*i*) for all elements $\alpha \in \mathtt{U}$, if α has a multiplicity greater than or equal to κ, then $\alpha \in Z$, (*ii*) no polynomial-time algorithm can learn any element other than the output of a κ^+ protocol, and (*iii*) no polynomial-time algorithm should know which output of the execution belongs to which user [15].

One straightforward technique to solve the problem is to use a trusted third party (TTP), where each user sends his private set to such TTP which performs the κ^+ aggregation task and reports the result back to each user. However, finding such TTP is not always possible in many applications. Moreover, compromising the TTP could lead to a complete privacy loss for all participating users.

Another approach is to use secure multiparty computation (SMC), which allows to securely compute a function over private data where users only learn the result of the function and nothing else. However, despite recent advances in their efficiency, SMC methods still require substantial computation and communication costs, making them impractical for real-world applications that mainly operate on large datasets.

A final approach is to use existing private set-operation protocols such as [14,21,13], especially multiset union protocols. These protocols securely compute all elements appearing in the union of input multisets at least τ times. Here all of them demand a priori-knowledge of the threshold value τ.

Remark 1. There have been many results [22,23,25,26] in the literature with titles containing the term "top-κ". We stress that these protocols produce the greatest κ elements in the union of the given sets in a private manner, and thus are different in their end results from our protocol. For example, there is a secure method for finding the κ-th ranked element in multiple multisets by Aggarwal et al. [1]. Repeatedly applying this protocol we can efficiently find the biggest κ elements in a distributed list.

Our Approach—Informal Descriptions: The most non-trivial part of κ^+ protocols is that we should satisfy two privacy requirements, namely the data privacy and user-privacy, at the same time. First let us take a closer look at e-voting protocols. In e-voting protocols, each ballot is mixed with a shuffle scheme which plays a crucial role in removing linkability between voters and ballots, which would hint user privacy. In fact, in e-voting literature this notion is called unlinkability. In order to emphasize the difference with e-voting protocols, we use the term user privacy. Now assuming that each element in multisets is encrypted and shuffled as in e-voting protocols, all encrypted elements can be decrypted, especially in a threshold manner, while satisfying user privacy. However, all non-κ^+ elements also are revealed, which violates data privacy in our application. Thus we need a way to keep data privacy even when all encrypted elements were decrypted. For this purpose we introduce an efficient function E that commutes with an underlying public-key encryption. More specifically, let Enc be a public-key encryption algorithm and Dec be the corresponding decryption algorithm. We demand that: (i) for all s and for all pk, $E_s \circ \mathsf{Enc}_{pk} = \mathsf{Enc}_{pk} \circ E_s$, and (ii) for all elements α, given $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(E_s(\alpha)))$ no algorithm can efficiently find α without s. We call this notion *double encryption*. In conclusion, our main technique is to shuffle doubly encrypted elements by each user. We should notice that all shuffle algorithms used in e-voting protocols rely on the re-randomizable property of underlying homomorphic encryption (e.g. see [8,17,18,12,11]), but its re-randomization algorithm does not change the plaintexts of input ciphertexts. However, in our protocol a double encryption scheme will change the plaintexts of input ciphertexts, which is the main difference from existing shuffle algorithms.

Summary of Our Results: In this paper, we present a formal definition of private κ^+ protocol and its security. Our operation setting is *fully decentralized* among n users over non-partitioned data. For the efficiency of our protocol, we refrain from using secure multiparty computation and construct an efficient private κ^+ protocol which is both data-private and user-private. Our construction strikes a real balance in the consumed resources and achieved security, and satisfies both privacy requirements. In particular, in its efficiency, our construction is comparable to the work in [3], which achieves its efficiency by giving up decentralized communication model, and in its security guarantees it is comparable to the work [4], which is secure but resources exhaustive. Our protocol on the other hand overcome the limitations and shortcomings of those protocols.

More specifically, our scheme does not requires a trusted party to set up the keys. Note that using Paillier cryptosystem [20] in a threshold manner demands some trusted setup. Moreover, our proposed protocol has several desirable features as follows: (1) It has $\mathcal{O}(n^2k)$ computational complexity where n is the number of users and k is the cardinality of each user's set, assuming $\kappa \leq k$, (2) It has $\mathcal{O}(n^2k)$ communication complexity, and (3) It has a linear round complexity in the number of users.

In general, real-world applications has n much smaller than k, which further justifies the efficiency of our protocol. This is, our protocol is beneficial in such real-world applications where the number of participating users is small but the size of their multisets is large. It remains an important open problem to devise a protocol whose round complexity does not depend on the number of users. Then we could make our protocol have $\mathcal{O}(nk)$ computation and communication complexity.

Organization. The rest of this paper is organized as follows. In Section 2, we discuss the related work with extensive analysis and comparison to our work. In Section 3, we outline the preliminaries required for understanding the rest of the paper, including double encryption, our formalism of κ^+ protocol and its security, and cryptographic primitives used in the context of computing the κ^+. In Section 4, we introduce our construction that comes into two forms with different requirements and guarantees and meets data privacy and user privacy. In Sections 5 and 6 we analyze the security and complexity of our work, by proving its security and showing its resources consumption requirements. Finally, we draw concluding remarks and point future work in Section 7.

2 Related Work

There has been plenty of work in the literature to solve the problem of private data aggregation. Such schemes can be classified under three schools of thoughts: fully centralized, fully decentralized, and semi-decentralized. While the centralized schemes assume the existence of a trusted third party (TTP), which makes them of less interest from the cryptographic and practical point of views, the fully decentralized schemes utilize cryptographic primitives and protocols to replace the centralized TTP. Finally, semi-decentralized schemes try to bridge the functional and security gap between other directions. As they are of particular relevance to our work, we limited our discussion to the decentralized and semi-decentralized protocols.

Decentralized solutions to the problem try to replace the centralized TTP with cryptographic constructions, which comes in different forms leading to several directions of research. One direction is based on SMC, as it is the case in [4]. However, at the core of that protocol's drawbacks is its inefficiency: since it uses Yao's garbled circuits [24], the computational resources required for ordinary data sizes is overwhelmingly high. Furthermore, as the datasets become disjoint, the efficiency of such construction decreases sharply. In [4], Burkhart and Dimitropoulos—in what we consider to be the most relevant work to ours—have devised a construction in which *the round complexity is linear to the number of bits* in the data elements. However, due to using sketches to improve the efficiency of subprotocols, the aggregate results are *probabilistic*. Furthermore, while the work in [4] is efficient in terms of its computational complexity, this efficiency comes at the expense of high round complexity. Kissner and Song [14] devised an over-threshold set union protocol—where *a threshold value τ should be given in advance*—to find all elements appearing in the union of input multisets at least τ times. The protocol requires a priori knowledge of the threshold, although operates in a decentralized manner. We compare it to our work in Section 6.

Finally, *semi-decentralized* constructions, represented by the work of Applebaum et al. in [3], aim to enhance the efficiency of fully-decentralized instantiations by adding new entities: proxy and database (DB). However the proxy and the DB are assumed to be semi-honest restricting the possibility of coalition between proxy and DB. This allows to obtain a constant round protocol. While the authors claim that both proxy and DB are expected to act as semi-honest, it might be a strong assumption both theoretically and practically. Furthermore, their scheme extensively relies on oblivious transfer (OT) [16], which is a very expensive public-key primitive since it may require two modular exponentiations per invocation, and run for each bit of the user's data element.

To sum up, Table 1 summarizes properties and efficiency of existing solutions compared with our proposed protocol. Computational complexity is expressed in terms of the number of multiplications over modulo p. For simplicity, we assume that multisets have values less than the prime p. Note that Applebaum et al.'s protocol requires both ElGamal encryption [7] and Goldwasser-Micali encryption [10], but we assume that both encryption systems use the same size modulus.

Table 1. Summary and Comparison

	Comm. Model	Round Cpx	Comp. Cpx	Comm. Cpx
Ours	Fully decentralized	$\mathcal{O}(n)$	$\mathcal{O}(n^2k\log p)$	$\mathcal{O}(n^2k\log p)$
[4]	Fully decentralized	$\mathcal{O}(n(n+k\log k)\log p)$	$\mathcal{O}(n^2k)$	$\mathcal{O}(n^2k\log p)$
[3]	Semi-decentralized	$\mathcal{O}(1)$	$\mathcal{O}(nk\log^2 p)$	$\mathcal{O}(nk\log p)$

3 Preliminaries

Notation. Let us denote $F(\alpha)$ for the multiplicity (also known as frequency) of an element α in a multiset $\mathtt{X}$ and $F(\mathtt{X})$ for the collection of multiplicities for all elements in the multiset $\mathtt{X}$—here the multiplicity $F(\alpha)$ of an element α refers to how many times the element appears in $\mathtt{X}$. For $n \in \mathbb{N}$, $[1, n]$ denotes the set $\{1, \ldots, n\}$. If A is a probabilistic polynomial-time (PPT) machine, we use $a \leftarrow A$ to denote A which produces output according to its internal randomness. In particular, if U is a set, then $r \overset{\$}{\leftarrow} U$ is used to denote sampling from the uniform distribution on U. A function $g : \mathbb{N} \to \mathbb{R}$ is negligible if for every positive polynomial $\mu(\cdot)$ there exists an integer L such that $g(\eta) < 1/\mu(\eta)$ for all $\eta > L$.

3.1 Definitions

Informally, a double encryption is a pair of encryption schemes $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ and $\mathsf{E} = (G, E, D)$ such that $\mathsf{Enc}(E(a)) = E(\mathsf{Enc}(a))$. We demand that an encryption scheme E be *deterministic*. The reason is that we need to know a complete distribution of multisets while hiding their elements. See Appendix A for a formal definition of public-key cryptosystem and its standard security definition.

Definition 1 (Double Encryption). *Let $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme defined as in Definition 4 with a pair of keys $(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$ and a message space (resp., ciphertext space) $\mathbf{M}_{pk}$ (resp., $\mathbf{C}_{pk}$). A pair $(\mathcal{E}, \mathsf{E})$ is called* double encryption *if there exists a triple of polynomial-time computable functions, $\mathsf{E} = (G, E, D)$, that satisfies the following properties:*

- *A probabilistic function $G(1^\lambda)$ takes as input a parameter λ, and outputs (s, s') s.t. $\forall s, s'$ and for any $m \in \mathbf{M}_{pk}$, $m = D_{s'}(E_s(m))$, E and D are deterministic.*
- *Commutativity: For all pk, s and for all $m \in \mathbf{M}_{pk}$, $\mathsf{Enc}_{pk}(E_s(m)) = E_s(\mathsf{Enc}_{pk}(m))$ up to the randomness of $\mathsf{Enc}_{pk}(\cdot)$.*
- *For all $c \leftarrow \mathsf{Enc}(m)$, $E_s(m) = \mathsf{Dec}_{sk}(E_s(c))$.*

We give an instantiation of a double encryption scheme in the following example.

Example 1. Let p be a large prime of the form $p = 2q + 1$, where q is also prime. Let $\mathbb{G}_q$ be a subgroup of $\mathbb{Z}_p^\times$ of order q with a generator g. Then a standard CPA-secure ElGamal encryption $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ is defined as follows: Selecting $x \overset{\$}{\leftarrow} \mathbb{Z}_q$, $\mathsf{KG}(1^\lambda)$ outputs (pk, sk) where $pk := (p, q, g, y = g^x \pmod p)$ and $sk := (p, q, g, x)$. Given a message $m \in \mathbb{G}_q$, the encryption algorithm Enc outputs $c = (g^r, m \cdot y^r)$ for a randomness $r \overset{\$}{\leftarrow} \mathbb{Z}_q$. Given an ElGamal ciphertext $c = (u, v)$, the decryption algorithm Dec computes $v \cdot u^{-x}$ using the secret key x. Now $\mathsf{E} = (G, E, D)$ is defined as follows:

- A probabilistic function $G(1^\lambda)$ outputs $(s, s') \in (\mathbb{Z}_q)^2$ such that $ss' = 1 \pmod q$.
- Given $\alpha \in \mathbb{G}_q$, $E : \mathbb{G}_q \to \mathbb{G}_q$ is given by $\alpha \mapsto \alpha^s \pmod p$.
- A deterministic function $D_{s'}(\beta)$ computes $\beta^{s'} \pmod p$.

Then, $(\mathcal{E}, \mathsf{E})$ is a *double encryption*:

- For all values $m \in \mathbb{G}_q$, $m = (m^s)^{s'} \pmod p$.
- For any message $m \in \mathbb{G}_q$, there exists $r' = rs$ s.t. $\left(g^{r'}, (m^s) \cdot y^{r'}\right) = ((g^r)^s, (my^r)^s)$.
- For any ElGamal ciphertext $c = (u, v) \in (\mathbb{G}_q)^2$, where $u = g^r$ and $v = my^r$, $m^s = v^s \cdot (u^s)^{-x} \pmod p$.

We use a standard definition of shuffle given by Nguyen et al. [18]; see details of the definition in [18, Def. 3 & Def. 4] and its extend version. As we mentioned before, our shuffle algorithm takes as input a list of ciphertexts, and outputs a list of *permuted* and *doubly encrypted ciphertexts*. Since a double encryption scheme leads to change the plaintexts of input ciphertexts, we need to devise proving the correctness of the shuffle.

Now we define κ^+ protocol and give its algorithmic description. Throughout the paper, we use Σ_n to denote the set of all permutations on $[1, n]$. A private κ^+ protocol consists of five computable (in polynomial time) algorithms, (Setup, DEncrypt, Shuffle, Aggregate, Reveal), over a double encryption $(\mathcal{E}, \mathsf{E})$, which are explained as follows:

$(pk, sk, s, s') \leftarrow \mathsf{Setup}(1^\lambda)$**:** The setup algorithm takes as an input the security parameter λ, and outputs the public and secret parameters for doubly encrypting input ciphertexts, by invoking $(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$ and $(s, s') \leftarrow G(1^\lambda)$.

$(E_s(c_1), \ldots, E_s(c_n)) \leftarrow \mathsf{DEncrypt}(pk, s, c_1, \ldots, c_n)$**:** The algorithm DEncrypt takes as input system parameters (pk, s) and a list of ciphertexts $(c_1, \ldots, c_n)$, and produces a list of doubly encrypted ciphertexts $(E_s(c_1), \ldots, E_s(c_n))$.

$\left(E_s(c_{\pi(1)}), \ldots, E_s(c_{\pi(n)})\right) \leftarrow \mathsf{Shuffle}(\pi, E_s(c_1), \ldots, E_s(c_n))$**:** The algorithm Shuffle chooses a random permutation $\pi \in \Sigma_n$ and shuffles the doubly encrypted ciphertexts $(E_s(c_1), \ldots, E_s(c_n))$, and then outputs the mixed list.

$(E_s(\alpha_1), \ldots, E_s(\alpha_\kappa)) \leftarrow \mathsf{Aggregate}\left(pk, sk, E_s\left(c_{\pi(1)}\right), \ldots, E_s\left(c_{\pi(n)}\right)\right)$**:** The algorithm Aggregate takes as input a pair of keys (pk, sk) and a list of permuted, doubly encrypted ciphertexts, and for all $i \in [1, n]$ computes $\mathsf{Dec}_{sk}\left(E_s\left(c_{\pi(i)}\right)\right) = E_s\left(\alpha_{\pi(i)}\right)$. Finally it computes $F\left(E_s\left(\alpha_{\pi(i)}\right)\right), i \in [1, n]$ and outputs only the elements whose multiplicity is greater than or equal than κ. Here $\{E_s(\alpha_1), \ldots, E_s(\alpha_\kappa)\} = \left\{E_s\left(\alpha_{\pi(i)}\right) \middle| F\left(E_s\left(\alpha_{\pi(i)}\right)\right) \geq \kappa\right\}$.

$(\alpha_1, \ldots, \alpha_\kappa) \leftarrow \mathsf{Reveal}\left(pk, s', E_s(\alpha_1), \ldots, E_s(\alpha_\kappa)\right)$**:** The algorithm Reveal outputs the most frequent κ^+ elements by computing $D_{s'}\left(E_s(\alpha_j)\right)$ for all $j \in [1, \kappa]$.

In the next section, we will define the meaning of *secure* κ^+ protocol. Then we describe cryptographic building blocks for constructing a secure κ^+ protocol under proper cryptographic assumptions.

3.2 Security Definition

Ideal Functionality. we define the ideal functionality $\mathcal{F}_{\mathsf{topk}}$ for the κ^+ protocol

Definition 2. *There are a set of n users, $U = \{u_i\}_{i=1}^n$, a trusted party $\mathcal{T}$, and an ideal adversary $\mathcal{S}$ controlling a set of corrupted users $\Upsilon_t = \{u_{i_j}\}_{j=1}^t$ for some $t \in [0, n-1]$. Let $\mathtt{X}_i = \{\alpha_{i,j}\}_{j=1}^{k_i}$ be a multiset of user $u_{i \in [1,n]}$.*

1. *Each user u_i sends $\mathtt{X}_i$ to $\mathcal{T}$.*
2. *$\mathcal{T}$ computes the following functionality, and returns the output Z_l to each $u_{l \in [1,n]}$:*

$$Z_l = \left\{ \alpha_{i,j} \in \bigcup\nolimits_{i \in [1,n]} \mathtt{X}_i \middle| F(\alpha_{i,j}) \geq \kappa \right\}.$$

Data Privacy. Informally, *data privacy* requires that no user, or coalition of users, should learn anything about honest users' inputs except what can be trivially derived from the output itself. We can easily derive the formal definition for data privacy in κ^+ protocols following the standard privacy definition of existing protocols in the literature; an excellent reference on that is Goldreich's textbook in [9]. More specifically, we use Definition 7.5.1 (resp., Definition 7.5.3) in [9] for the semi-honest model (resp. the malicious model). Roughly speaking, this is formalized by considering an ideal world where $\mathcal{T}$ receives the inputs of the users and outputs the result of the defined functionality. We demand that in the real world application of the protocol–that is, one without the $\mathcal{T}$–no user should learn more information than in the ideal world.
User Privacy. The remaining part to conclude our definitions is user privacy. Let $Z = \{\alpha_1, \ldots, \alpha_\kappa\}$ be an output of a κ^+ protocol. Roughly speaking, no user or coalition of users should gain a non-negligible advantage in distinguishing, for all $\alpha \in Z$, an honest user u_i such that $\alpha \in \mathtt{X}_i$.

Definition 3 (User Privacy). *Let $\Pi_{\kappa,\mathcal{E},\mathsf{E}}$ be a κ^+ protocol defined as in Section 3.1 over a double encryption scheme $(\mathcal{E}, \mathsf{E})$ and $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary.*

```
Experiment $\mathsf{Exp}^{\kappa^+}_{\mathcal{A}}(\Pi_{\kappa,\mathcal{E},\mathsf{E}}, \lambda)$
    $(pk, sk, s, s') \leftarrow \mathsf{Setup}(\lambda)$;
    $(\mathsf{state}, \Upsilon_t, m_1, \ldots, m_{n-t}) \leftarrow \mathcal{A}_1(pk, n, t)$ s.t. $\Upsilon_t$ is a set of corrupted $t$ users;
    $\sigma \overset{\$}{\leftarrow} \Sigma_n$ and assign $m_{\sigma(i)}$ to each honest $i$-th user $u_i \notin \Upsilon_t$;
    $(\alpha_1, \ldots, \alpha_\kappa) \leftarrow \Pi_{\kappa,\mathcal{E},\mathsf{E}}$, where $\mathcal{A}_1$ interacts with the $n-t$ honest users;
    $(i, j) \leftarrow \mathcal{A}_2(pk, m_1, \ldots, m_{n-t}, \mathsf{state})$;
```

We define the advantage of an adversary $\mathcal{A}$, running in probabilistic polynomial time:

$$\mathsf{Adv}^{\kappa^+}_{\mathcal{A}}(\Pi_{\kappa,\mathcal{E},\mathsf{E}}, \lambda) = \left| \Pr[\sigma(i) = j] - \frac{1}{n-t} \right| .$$

A κ^+ protocol is user-private *if the advantage $\mathsf{Adv}^{\kappa^+}_{\mathcal{A}}(\Pi_{\kappa,\mathcal{E},\mathsf{E}}, \lambda)$ is negligible in the security parameter λ.*

3.3 Cryptographic Assumptions and Tools

Next we outline the cryptographic tools and assumptions we use in our protocol. Let $\mathcal{G}$ be a finite cyclic group of prime order q, and let $g \in \mathcal{G}$ be a generator. Given $h \in \mathcal{G}$, the discrete logarithm problem requires us to compute $x \in \mathbb{Z}_q$ such that $g^x = h$. We denote this (unique) x by $\log_g h$. In particular groups $\mathcal{G}$ and for q large, it is assumed hard to compute x, which is said to be the Discrete Logarithm (DL) assumption.

A stronger assumption is the Decisional Diffie-Hellman (DDH) assumption. Here, given $\mathcal{G}$, a generator g of $\mathcal{G}$, and three elements $a, b, c \in \mathcal{G}$, we are asked (informally) to decide whether there exist x, y such that $a = g^x, b = g^y$, and $c = g^{xy}$. More formally, the DDH assumption states that the following two distributions are computationally indistinguishable: $\{\mathcal{G}, g, g^x, g^y, g^{xy}\}$ and $\{\mathcal{G}, g, g^x, g^y, g^z\}$ where $x, y, z \overset{\$}{\leftarrow} \mathbb{Z}_q$.

We extensively use ElGamal encryption defined in Example 1. This scheme secure against CPA attack in a DDH group $\mathbb{G}$; see Appendix A for the CPA security. In addition, we need an efficient scheme which works as follows: When each user holds a shared secret key s_i such that $s = \prod_{i=1}^{n} s_i$, the scheme allows each user to have a share s'_i satisfying $s' = \prod_{i=1}^{n} s'_i$ and $s' = s^{-1} \pmod q$ for a public modulus q. Indeed, we may realize the scheme by techniques studied by Algesheimer et al. [2, §5].

4 Our Construction

In this section, we describe our construction for computing the κ^+ elements privately. We begin by considering a basic setting of n users, denoted by $u_1, \dots, u_n$. Let $\mathtt{X}_i = \{\alpha_{i,1}, \dots, \alpha_{i,k}\}$ for all $i \in [1, n]$. Each user u_i has its private multiset $\mathtt{X}_i$, and the users wish to jointly compute $\{\alpha \in \bigcup_{i=1}^{n} \mathtt{X}_i | F(\alpha) \geq \kappa\}$. For simplicity, assume that all elements are in the proper message domain $\mathbf{M}_{pk}$ of an ElGamal encryption scheme, e.g., a finite cyclic subgroup $\mathbb{G}_q$ of $\mathbb{Z}_p$ in which the DDH assumption holds. For a multiset $\mathtt{X} = \{\alpha_1, \dots, \alpha_k\}$, we denote $\mathtt{X}^s$ as $\{\alpha_1^s, \dots, \alpha_k^s\}$ for some $s \in \mathbb{Z}_q$. With such notation in mind, we proceed to describe our construction.

4.1 Description

Let λ be a security parameter, p be a λ-bit prime such that for some prime q, $p = 2q+1$, and $\mathbb{G}_q$ be a finite cyclic subgroup of $\mathbb{Z}_p^\times$ whose order is q, and g be a generator of $\mathbb{G}_q$.

Setup(1^λ). Each user agrees to a threshold ElGamal encryption $\mathcal{E}$ with a public/private key pair (pk, sk), which are computed as follows. Define $\mathsf{params} := (p, q, g, \mathbb{G}_q)$. Each user selects a value $x_i \xleftarrow{\$} \mathbb{Z}_q$, computes $y_i = g^{x_i}$, and sets $sk = (\mathsf{params}, x_i)$; the public key is then given by $pk := \left(\mathsf{params}, y = \prod_{i=1}^{n} y_i = g^{\sum_{i \in [1,n]} x_i} \pmod p\right)$. In addition, all users are distributed a share (s_i, s'_i) such that $s = \prod_{i=1}^{n} s_i$, $s' = \prod_{i=1}^{n} s'_i$, and $s \cdot s' = 1 \pmod q$. Notice that in threshold decryption schemes, users generally produce shares of the decrypted element, and during the operation of the schemes if one user sends a uniformly generated share instead of a valid one the decrypted element is uniform. Also, if the decrypted element is uniform, the resulting decryption reveals no information to the users.

DEncrypt. Let $I = \{1, \dots, n\}$ be a set of indices, and let the power function $E_{s_i}(\alpha) = \alpha^{s_i} \pmod p$ which is deterministic.

1. Every user u_i encrypts his multiset $\mathtt{X}_i$ as follows:

$$\mathsf{Enc}_{pk}(\mathtt{X}_i) = \{\mathsf{Enc}_{pk}(\alpha_{i,1}), \dots, \mathsf{Enc}_{pk}(\alpha_{i,k})\}$$

   where $\mathsf{Enc}_{pk}(\alpha_{i,j}) = (g^{r_{i,j}}, \alpha_{i,j} \cdot y^{r_{i,j}})$ for some randomizer $r_{i,j} \in \mathbb{Z}_q$, and sends $\mathsf{Enc}_{pk}(\mathtt{X}_i)$ to u_1.
2. User u_1 computes $\{E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_1)), \dots, E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_n))\}$, which is denoted by Y_0.

Shuffle & DEncrypt. For $i \in [1, n]$, u_i receives vector Y_{i-1} and computes a permuted, doubly encrypted version Y_i as follows:

1. $u_{i\neq 1}$ computes

$$\begin{aligned} E_{s_i}(Y_{i-1}) &= \{c_1, \ldots, c_{nk}\} \\ &= \left\{E_{s_i}\left(E_{s_{i-1}}\left(\cdots E_{s_1}\left(\alpha_{\pi_{i-1}(1)}\right)\cdots\right)\right), \ldots, \right. \\ &\qquad \left. E_{s_i}\left(E_{s_{i-1}}\left(\cdots E_{s_1}\left(\alpha_{\pi_{i-1}(nk)}\right)\cdots\right)\right)\right\}. \end{aligned}$$

   More precisely, here $\alpha_{\pi_{i-1}(\ell)} = \alpha_{\pi_{i-1}\circ\cdots\circ\pi_1(\ell)}$ for all $\ell \in [1, nk]$.
2. u_i chooses a random permutation $\pi_i \in \Sigma_{nk}$, and applies π_i to the list of $c_{\ell\in[1,nk]}$ computed above; denote the result by Y_i.
3. u_i sends Y_i to u_{i+1}; the last user u_n sends Y_n to all users.

Aggregate. Let $\mathtt{U} = \bigcup_{i=1}^{n} \mathtt{X}_i$. Every user has $E_s(\mathsf{Enc}_{pk}(\mathtt{U}))$.

1. Every user participates in a group decryption and obtains

$$E_s(\mathtt{U}) = \left\{E_s\left(\alpha_{\pi(1)}\right), \ldots, E_s\left(\alpha_{\pi(nk)}\right)\right\}$$

   where $\pi = \pi_n \circ \cdots \circ \pi_1$.
2. Every user computes $Z = \{E_s(\alpha) \in E_s(\mathtt{U}) \big| F(E_s(\alpha)) \geq \kappa\}$.

Reveal.

1. For every $E_s(\alpha) \in Z$, user u_i sends its share of $D_{s'_i}(E_s(\alpha))$ to $u_{i'}$.
2. After receiving all the shares, every user u_i computes $\alpha = D_{s'}(E_s(\alpha))$, thereby recovering the κ^+, $\{\alpha \in \mathtt{U} | F(\alpha) \geq \kappa\}$.

Efficiency. The advantage of the above protocol is multifold. First, compared to Kissner and Song's protocol [14], our protocol provides the functionality of finding a threshold value and computing the "over threshold" at the same computation and communication cost—whereas they incur different and higher costs in [14]. Second, compared to the κ^+ protocol described in [4], our protocol has a much better computational complexity. See details in Section 5. In order to present a fair comparison between our proposed protocol and Applebaum et al.'s protocol [3] we devise our protocol for a semi-decentralized model in the next section. The other purpose of our modification is to reduce the round complexity to a constant.

4.2 Semi-decentralized Construction

The most crucial drawback of the previous protocol is its $\mathcal{O}(n)$ round complexity. To avoid this problem, Applebaum et al. introduced two semi-honest users: a *proxy* which shuffles a list of input ciphertexts, and a *database* which aggregates κ^+ elements. Applying the same technique to our protocol, we obtain a constant-round κ^+ protocol.

– Assume that there are n_1 proxies and n_2 databases described as in [3].
– Each database engages in setting up a threshold ElGamal encryption and publishes a public key. Instead of users, all proxies are distributed secret shares $(s_l, s'_l)_{l\in[1,n_1]}$.
– Each user computes a list of ElGamal ciphertexts and sends it to a proxy.
– Each proxy runs DEncrypt and Shuffle, and returns the result to all databases.
– Databases perform group decryption, and get the list of encrypted κ^+ elements
– Finally, all proxies decrypt the encrypted κ^+ list and return the κ^+ to all users.

Compared to [3], our protocol does not require OT operations, nor an extra encryption scheme. Recall that Applebaum et al.'s protocol requires ElGamal encryption and Goldwasser-Micali (GM) encryption: ElGamal encryption is used to encrypt elements in multisets and GM encryption is used to encrypt their multiplicity.

5 Security Analysis

Theorem 1 (Correctness). *In the private top-κ protocol in sec. 4.1, every honest user learns the joint set distribution of all users' private inputs, i.e., each element $E_s(\alpha)$ such that $\alpha \in \bigcup_{i=1}^{n} \mathtt{X}_i$ and the number of times it appears, with overwhelming probability.*

Proof. Each player learns a randomly permuted joint multiset $E_s(\mathtt{U}) = \left\{E_s\left(\alpha_{\pi(1)}\right), \ldots, E_s\left(\alpha_{\pi(nk)}\right)\right\}$. We know that $|\mathtt{U}^s| = nk$. Since π is a permutation, for each $E_s\left(\alpha_{\pi(\ell)}\right)$ and for all $\ell \in [1, nk]$, there exist a pair of the unique index ℓ^* such that

$$\begin{aligned} \ell^* &= \pi^{-1}(\ell) \\ &= \pi_n^{-1}(\ell) \circ \cdots \circ \pi_1^{-1}(\ell). \end{aligned}$$

Namely, $E_s\left(\alpha_{\pi(\ell)}\right)$ is a unique blinded version of $\alpha_{\ell^*} \in \bigcup_{i=1}^{n} \mathtt{X}_i$. Moreover, $\forall \ell, \ell^* \in [1, nk]$, $\alpha_\ell = \alpha_{\ell^*}$ if and only if $E_s(\alpha_\ell) = E_s(\alpha_{\ell^*})$ with overwhelming probability. □

Corollary 1. *In the private top-κ protocol in Section 4.1, every honest user learns the κ^+ in the union of private multisets with overwhelming probability.*

Now we show that our protocol satisfies the privacy requirements in the semi-honest model. Let $\mathcal{T}$ be a trusted party in the ideal world which receives the private input multiset $\mathtt{X}_i$ of size k from user u_i for $i \in [1, n]$, and then returns to every user the joint multiset distribution $\{F(\alpha)\}$ for all $\alpha \in \bigcup_{i=1}^{n} \mathtt{X}_i$.

Theorem 2 (Data Privacy). *Assume that the threshold ElGamal encryption $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ is secure against CPA. In the private top-κ protocol in Section 4.1, any coalition of less than n semi-honest users learn no more information than would be given by using the same private inputs in the ideal-world model with $\mathcal{T}$.*

Proof. We assume that the ElGamal encryption scheme is CPA-secure, and so each user learns only

$$\begin{aligned} &\mathsf{Enc}_{pk}(\mathtt{X}_1), \ldots, \mathsf{Enc}_{pk}(\mathtt{X}_n); \\ &E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_1)), \ldots, E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_1)), \ldots, E_{s_{i-1}}(\mathsf{Enc}_{pk}(\mathtt{X}_1)), \ldots, E_{s_{i-1}}(\mathsf{Enc}_{pk}(\mathtt{X}_n)); \\ &\qquad\vdots \\ &E_{s_{i-1}}(\cdots E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_1)) \cdots), \ldots, E_{s_{i-1}}(\cdots E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_n)) \cdots) \end{aligned}$$

during an execution. At the end of the protocol all users further know $E_s(\mathsf{Enc}_{pk}(\mathtt{U}))$ where $\mathtt{U} = \bigcup_{i=1}^{n} \mathtt{X}_i$, and for some $\gamma_{\ell \in [1,nk]} \in \mathbb{Z}_q$

$$\begin{aligned} E_s(\mathsf{Enc}_{pk}(\mathtt{U})) &= \{E_s(\mathsf{Enc}_{pk}(\mathtt{X}_1)), \ldots, E_s(\mathsf{Enc}_{pk}(\mathtt{X}_n))\} \\ &= \left(g^{\gamma_1}, \left(\alpha_{\pi(1)}\right)^s \cdot y^{\gamma_1}\right), \ldots, \left(g^{\gamma_{nk}}, \left(\alpha_{\pi(nk)}\right)^s \cdot y^{\gamma_{nk}}\right). \end{aligned}$$

Note that π is a composition of random permutations and is unknown to all users, as the maximum coalition size is smaller than n. That is, if there exists at least an honest user, then a composition of random permutations $\pi = \pi_n \circ \cdots \circ \pi_1$ is a random permutation because at least a permutation $\pi_{i \in [1,n]}$ is secure. What is more, note that s is uniformly distributed and unknown to all users for the same reason. As s is uniformly distributed for any user inputs and π is random, no user or coalition can learn more than a set of re-randomized ElGamal encryptions. As s is uniformly distributed, a group decryption of ElGamal encryptions reveals no more than

$$\{E_s(\alpha_\ell)\}_{\ell \in [1,nk]} = E_s\left(\bigcup_{i=1}^{n} \mathtt{X}_i\right) = E_s(\mathtt{U}).$$

We know the fact that $F(\alpha) = F(\alpha^s)$ for two multisets $\mathtt{X}$ and $E_s(\mathtt{X}) \in (\mathbb{G}_q)^k$, for all $s \in \mathbb{Z}_q$ and for all $\alpha \in \mathtt{X}$. Hence we see that

$$F(E_s(\mathtt{U})) = F\left(E_s\left(\bigcup_{i=1}^{n} \mathtt{X}_i\right)\right) = F\left(\bigcup_{i=1}^{n} \mathtt{X}_i\right) = F(\mathtt{U}),$$

which can be derived from the output returned by $\mathcal{T}$ in the ideal-world model. □

Theorem 3 (User Privacy). *Assume that the threshold ElGamal encryption* Enc *is CPA-secure. The private top-κ protocol in Section 4.1 is user-private against any coalition of less than n semi-honest users.*

Proof. Assume that there is at least an honest user in the system, and that the threshold ElGamal encryption $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ is CPA-secure. After performing $\mathsf{DEncrypt}$ and $\mathsf{Shuffle}$ algorithms, every user obtains a collection of ElGamal encryptions $\{c_1, \ldots, c_{nk}\}$. By the second assumption, the adversary cannot learn any further information except that which encryptions have been sent from which users. Running these algorithms, each user should raise the power of the received encryptions with his shared secret s_i. Namely, each user holds the modified list of the encryptions,

$$\{E_{s_i}(c_1), E_{s_i}(c_2), \ldots, E_{s_i}(c_{nk})\}.$$

Next the user should apply his private permutation π_i to the list to transform it to

$$\left\{E_{s_i}\left(c_{\pi(1)}\right), E_{s_i}\left(c_{\pi(2)}\right), \ldots, E_{s_i}\left(c_{\pi(nk)}\right)\right\}.$$

At the end of running the algorithms, all users get a permuted and doubly encrypted list

$$\left\{E_s\left(c_{\pi(1)}\right), E_s\left(c_{\pi(2)}\right), \ldots, E_s\left(c_{\pi(nk)}\right)\right\}$$

where the permutation $\pi = \pi_n \circ \cdots \circ \pi_1$ and $s = \prod_{i=1}^{n} s_i$. As there exists at least an honest user, even when $n-1$ users collude, s is uniformly distributed and unknown to all users and π is a random permutation. This completes the proof of the claim. □

Theorem 4. *Assuming that the threshold ElGamal encryption is CPA-secure and the DL assumption holds, the proposed top-κ protocol is secure in the semi-honest model.*

Proof. We complete the proof of security by Theorem 2 and Theorem 3. □

6 Efficiency Analysis

The private κ^+ protocol has not yet been implemented, but we give a detailed analysis of the running time and space requirements as follows. We base our protocol on ElGamal encryption and the power function with primes $|p| = 1024, |q| = 160$. To measure users' overhead, we count the number of exponentiations using a 1024-bit modulus.

Table 2. Complexity Analysis

	Comp. Cpx (expo.)	Comm. Cpx (bits)	Rounds Cpx
Setup	n	$n \log p$	1
DEncrypt & Shuffle	$4nk + 2n^2k$	$2(n-1)k \log p + 2n^2k \log p$	n
Aggregate	n^2k	$2n^2k \log p$	1
Reveal	$n\kappa$	$n\kappa \log p$	$n-1$

In Table 2 we show a summary of the complexity result for our proposed protocol. The total computational complexity is dominated by DEncrypt and Shuffle algorithms. Putting the computational complexities together shows that total computation complexity is $\mathcal{O}(n^2k)$ in $\mathcal{O}(n)$ rounds. The proposed protocol has $\mathcal{O}(n^2k \log p)$ bits in total. It is impossible to directly compare our protocol with Applebaum et al.'s protocol, since it runs in the semi-decentralized model, we just present the computational complexity.

Comparison. We consider three protocols: Kissner and Song (KS) protocol [14], Burkhart and Dimitropoulos (BD) protocol [4], and Applebaum et al. protocol.

Based on KS Protocol. We first compare our work with a KS-based κ^+ protocol. As mentioned earlier, since it does not provide a way for finding τ, we do not know computational and communication complexity in computing τ. Assuming τ is given, their protocol has $\mathcal{O}(n^2k)$ computation complexity in $\mathcal{O}(n)$ rounds.

BD Protocol. In turn, we give a comparison with BD protocol. To our knowledge, it is the only fully decentralized κ^+ protocol that does not use Yao's garbled circuit evaluation. Their protocol utilizes two special-purpose sub-protocols–`equality` and `lessthan` (see [6,19]), but in [5] as the authors pointed out, comparison of two shared secrets is very expensive and computational intensive. Thus, they use a computationally efficient version of the basic sub-protocols as follows: `equality` requires $\log p$ rounds and `lessthan` requires $(2 \log p + 10)$ rounds. Their protocol consists of two key ingredients as follows:

- Finding the correct τ: takes $(\log k(2 \log p + 10) + \log p + 2 \log p + 10)\, nk$ rounds.
- Resolving collisions: Requires $\frac{n(n-1)}{2} \log p + 2(n-1) \log p + 10(n-1)$ rounds.

Note that BD protocol also should know τ as in KS protocol. Hence, the total round complexity is $\mathcal{O}(n(n + k \log k) \log p)$ for hash tables of size $\log k$ and U of nk. We find their protocol takes $4\left(\frac{n(n-1)}{2}k + k(n-1)\right)$ multiplications in $\mathbb{Z}_p^{\times}$.

Applebaum et al. Protocol. Let us use $\mathcal{O}_p(\cdot)$ to denote complexity using modulus prime p and $\mathcal{O}_N(\cdot)$ complexity using modulus composite N. Assume all elements are integers less than p and the maximum multiplicity is less than $\log\log p$.
Their major computation-intensive parts are as follows:

- Interactive computation between Users and Proxy: First, users should run a protocol for oblivious evaluation of pseudorandom function by communicating with proxies, then encrypt the result with ElGamal encryption. This requires $n(k(2\log p + 2) + 2k)$ exponentiations over $\mathbb{Z}_p^\times$. Also, users should encrypt the multiplicity of each element with GM encryption, requiring $nk \log\log p$ multiplications over $\mathbb{Z}_N^\times$. Finally each user doubly encrypts their elements using ElGamal encryption. This requires $2nk$ exponentiations over $\mathbb{Z}_p^\times$.
- Aggregation by Database: The most computationally-intensive part is ElGamal and GM decryption. Since database receives two types of ElGamal ciphertexts, it performs $2nk$ exponentiations over $\mathbb{Z}_p^\times$. GM decryption requires $2nk \log\log p$ exponentiations over $\mathbb{Z}_N^\times$.

Thus, the complexity is $\mathcal{O}_p(nk\log p) + \mathcal{O}_N(nk\log\log p)$ exponentiations.

7 Conclusion

In this paper we have looked at the problem of finding the κ^+ element securely, and formally defined what it means for a protocol to be a secure κ^+ protocol. We developed two protocols, with varying operation overhead, analyzed their security, and demonstrated their practicality. In the near future, we will investigate two directions. First, since our constructions' security is proven in the semi-honest model—which is rationalized by the application domain, we will investigate constructions that are provably secure in the malicious model, and their potential applications. Second, as the shuffling algorithm in our current construction requires sending messages among players in a relay manner, we will consider the practical and security aspects of a construction that relies on sending such messages in a broadcast manner.

Acknowledgement. We thank Burt Kaliski for his feedback on an earlier version.

References

1. Aggarwal, G., Mishra, N., Pinkas, B.: Secure computation of the k^{th}-ranked element. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 40–55. Springer, Heidelberg (2004) 473
2. Algesheimer, J., Camenisch, J., Shoup, V.: Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 417–432. Springer, Heidelberg (2002) 479
3. Applebaum, B., Ringberg, H., Freedman, M.J., Caesar, M., Rexford, J.: Collaborative, privacy-preserving data aggregation at scale. In: Atallah, M.J., Hopper, N.J. (eds.) PETS 2010. LNCS, vol. 6205, pp. 56–74. Springer, Heidelberg (2010) 474, 475, 480, 481
4. Burkhart, M., Dimitropoulos, X.: Fast privacy-preserving top-k queries using secret sharing. In: IEEE ICCCN (2010) 474, 475, 480, 483

5. Burkhart, M., Strasser, M., Many, D., Dimitropoulos, X.: SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In: USENIX Security (2010) 483
6. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 285–304. Springer, Heidelberg (2006) 483
7. El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985) 475
8. Furukawa, J., Sako, K.: An efficient scheme for proving a shuffle. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 368–387. Springer, Heidelberg (2001) 474
9. Goldreich, O.: The foundations of cryptography. Cambridge University Press (2004) 478
10. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. (1984) 475, 486
11. Groth, J.: A verifiable secret shuffle of homomorphic encryptions. J. of Cryptology (2010) 474
12. Groth, J., Lu, S.: Verifiable shuffle of large size ciphertexts. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 377–392. Springer, Heidelberg (2007) 474
13. Hong, J., Kim, J.W., Kim, J., Park, K., Cheon, J.H.: Constant-round privacy preserving multiset union. In: Cryptology ePrint Archive, 2011/138 (2011) 473
14. Kissner, L., Song, D.: Privacy-preserving set operations. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 241–257. Springer, Heidelberg (2005) 473, 475, 480, 483
15. Mohaisen, A., Hong, D., Nyang, D.: Privacy in location based services: Primitives toward the solution. In: NCM (2008) 473
16. Naor, M., Pinkas, B.: Oblivious transfer with adaptive queries. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 573–590. Springer, Heidelberg (1999) 475
17. Neff, C.: A verifiable secret shuffle and its application to e-voting. In: ACM Conference on Computer and Communications Security, pp. 116–125 (2001) 474
18. Nguyen, L., Safavi-Naini, R., Kurosawa, K.: Verifiable shuffles: A formal model and a paillier-based efficient construction with provable security. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 61–75. Springer, Heidelberg (2004) 474, 477
19. Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 343–360. Springer, Heidelberg (2007) 483
20. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999) 474
21. Sang, Y., Shen, H.: Efficient and secure protocols for privacy-preserving set operations. ACM Transactions on Information and System Security (TISSEC) 13(1), 9:1–9:35 (2009) 473
22. Vaidya, J., Clifton, C.: Privacy-preserving top-k queries. In: ICDE (2005) 473
23. Xiong, L., Chitti, S., Liu, L.: Top$k$ queries across multiple private databases. In: International Conference on Distributed Computing Systems (ICDCS), pp. 145–154 (2005) 473
24. Yao, A.: Protocols for secure computations. In: FOCS, pp. 160–164 (1982) 475
25. Zhang, R., Shi, J., Liu, Y., Zhang, Y.: Verifiable fine-grained top-k queries in tiered sensor networks. In: INFOCOM, pp. 2633–2641 (2010) 473
26. Zhang, R., Zhang, Y., Zhang, C.: Secure top-k query processing via untrusted location-based service providers. In: INFOCOM, pp. 1170–1178 (2012) 473

A Basic Definitions

We first give a formal definition of a public key cryptosystem and then its standard security definition. We shall write

$$\Pr[x_1 \overset{\$}{\leftarrow} X_1, x_2 \overset{\$}{\leftarrow} X_2(x_1), \ldots, x_n \overset{\$}{\leftarrow} X_n(x_1, \ldots, x_{n-1}) : \varphi(x_1, \ldots, x_n)]$$

to denote the probability that when x_1 is drawn from a certain distribution X_1, and x_2 is drawn from a certain distribution $X_2(x_1)$, possibly depending on the particular choice of x_1, and so on, all the way to x_n, the predicate $\varphi(x_1, \ldots, x_n)$ is true.

Definition 4. *A* public-key cryptosystem $\mathcal{E}$ *is a 3-tuple of PPT algorithms* $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ *such that*

1. *The key generation algorithm* KG *takes as input the security parameter* λ *and outputs a pair of keys* (pk, sk)*. For given* pk*, the message space* $\mathbf{M}_{pk}$ *and the randomness space* $\mathbf{R}_{pk}$ *are uniquely determined.*
2. *The encryption algorithm* Enc *takes as input a public key* pk *and a message* $m \in \mathbf{M}_{pk}$*, and outputs a ciphertext* $c \in \mathbf{C}_{pk}$ *where* $\mathbf{C}_{pk}$ *is a finite set of ciphertexts. We write this as* $c \leftarrow \mathsf{Enc}_{pk}(m)$*. We sometimes write* $\mathsf{Enc}_{pk}(m)$ *as* $\mathsf{Enc}_{pk}(m, r)$ *when the randomness* $r \in \mathbf{R}_{pk}$ *used by* Enc *needs to be emphasized..*
3. *The decryption algorithm* Dec *takes as input a private key* sk *and a ciphertext* c*, and outputs a message* m *or a special symbol* $\perp$ *which means failure.*

We say that a public-key cryptosystem $\mathcal{E}$ is *correct* if, for any key-pair $(pk, sk) \leftarrow \mathsf{KG}(\lambda)$ and any $m \in \mathbf{M}_{pk}$, it is the case that: $m \leftarrow \mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m))$.

Definition 5 ([10]). *A public-key cryptosystem* $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ *with a security parameter* λ *is called to be* semantically secure (IND-CPA secure) *if after the standard CPA game being played with any PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the advantage* $\mathsf{Adv}^{\mathsf{cpa}}_{\mathcal{E},\mathcal{A}}(\lambda)$*, formally defined as*

$$\left| \Pr_{b,r}\left[\begin{array}{l} (pk, sk) \leftarrow \mathsf{KG}(\lambda), (\mathsf{state}, m_0, m_1) \leftarrow \mathcal{A}_1(pk), \\ c = \mathsf{Enc}_{pk}(m_b; r) : b \leftarrow \mathcal{A}_2(\mathsf{state}, m_0, m_1, c) \end{array} \right] - \frac{1}{2} \right|,$$

is negligible in λ *for all sufficiently large* λ*.*

In the experiment above, when we allow $\mathcal{A}_1$ to query the decryption oracle, if the advantage $\mathsf{Adv}^{\mathsf{cca2}}_{\mathcal{E},\mathcal{A}}(\lambda)$ is negligible, we say $\mathcal{E}$ is IND-CCA1 secure, in short, CCA1 secure.

Retracted: An Enhanced Anonymous Authentication and Key Exchange Scheme Using Smartcard

Kyung-kug Kim and Myung-Hwan Kim

ISaC and Department of Mathematical Sciences,
Seoul National University, Seoul 151-747, Korea
gggim1231000@gmail.com, mhkim@math.snu.ac.kr

Abstract. Nowadays, anonymity property of user authentication scheme becomes important. In 2003, Park et al. proposed an authentication and key exchange scheme using smart card. However, Juang et al. pointed out that Park et al.'s scheme did not provide the user anonymity. Then, they presented an anonymous authenticated key exchange protocol using smart cards in 2008. They argued that their scheme provided identity privacy, mutual authentication, and half-forward secrecy. In 2009, however, Lee et al. showed that Juang et al.'s scheme is not secure against stolen-verifier attack. Moreover, Juang's scheme does not satisfy the user anonymity. To solve this problem, we proposed an improved anonymous authentication and key exchange scheme. Then, we show that the proposed scheme is secure against various well-known attacks.

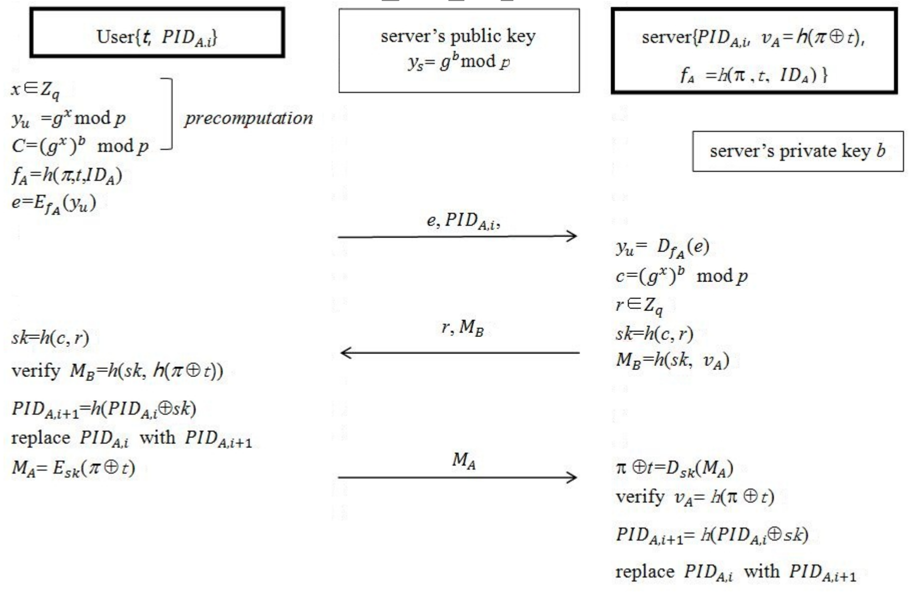
Keywords: anonymous, user authentication, password key exchange protocol.

1 Introduction

In 1981, Lamport presented the password based user authentication scheme [1]. After Lamport's scheme, many researchers have been proposed various security schemes [2-12]. As Juang et al. and Lee et al. mentioned, since then, all of those protocols were not suitable for the public wireless LAN services. Unlike general services, the public wireless LAN services should satisfy its unique characteristics such as billing, roaming and security. Especially, security in public wireless LAN service is an essential issue. The security requirements of the public wireless LAN services are as follows. First, it should ensure the user anonymity. If the user anonymity is not ensured, the location information of the respective user is totally exposed to an attacker. Once a user's sensitive information such as location information is collected by an attacker, it leads to a violation of privacy. Here, the user anonymity means that not only a user's identity should be protected from being exposed to an attacker but also to the server. To protect a user's privacy, user information must not be exposed even to servers. That is because there are chances that a server may abuse the user information. It is called anonymous communication to prevent exposing user's identity only during communication [13]. Second, a mutual authentication between a user and a server should be possible. If it is not guaranteed, an attacker may impersonate as an authenticated user or server. Third, a public wireless LAN service should satisfy the forward secrecy. If it does not satisfy the forward secrecy, an attacker is able to compute a session key based on the intercepted information. In 2003, Park et al. proposed an authentication and key exchange protocol satisfying the above requirements [14]. However, in 2008, Juang et al. proved that the protocol proposed by Park et al. did not ensure the user anonymity and proposed a new protocol ensuring the user anonymity [15]. The protocol proposed by Juang et al. had a merit to require less computational overhead than the existing protocol while ensuring the anonymity, but we discover that their protocol does not satisfy the user anonymity and is vulnerable to the stolen-verifier attack. Moreover, it will be showed a shortcoming that the server has high computational overhead. In this paper, we analyze the vulnerability of the protocol proposed by Juang et al. and propose an improved protocol to ensure not only the anonymous communication but also the user anonymity. The structure of this paper is organized as follows. In Section 2, we demonstrate out proposed protocol. In Section 3, we discuss the efficiency and security of our protocol. Finally, we make conclusions in Section 4.

2 Proposed Protocol

In this section, we propose an improved anonymous authentication and key exchange protocol. While maintaining the merits of the existing protocol, our protocol provides the user anonymity, is secure against the stolen-verifier attack, and has low computational overhead than Juang et al.'s protocol.



2.1 Notations

- p: a large prime number
- q: a prime divisor of $(p-1)$
- g: an element of order q in $Z^*{}_p$
- b: a private key of the server
- y_s: a public key of the server (= g^{b}mod p)
- A: the user
- B: the server
- π: the password of the user
- t: the shared symmetric key between the user and the server for symmetric key encryption
- i: the index of the session
- ID_A: the user's identification
- $h()$: a secure one-way hash function
- $E_K()$: a symmetric encryption function with the symmetric key K
- $D_K()$: a symmetric decryption function with the symmetric key K
- sk: a session key generated by the user and the server
- $PID_{A,i}$: a temporary ID of the user (=$PID_{A,i}$= $h(PID_{A,i-1} \oplus sk)$)

2.2 Registration Stage

A and B share a password π and a symmetric key t. A remembers π and stores t and $PID_{A,i}$ in a smart card. B stores $PID_{A,i}$, $v = h(\pi \oplus t)$, $f = h(\pi, t, ID_A)$ in a storage. Moreover, B selects a random number b as a private key and computes $y_s = g^b$mod p and sets it as a public key. b must be securely stored in B's database.

2.3 Precomputation Stage

A selects a random value x in Z_q and computes $y_u = g^x$mod p. Then, to reduce the computational overhead in the authentication and key exchange stage, A calculates $c = g^{bx}$ mod p and stores it.

2.4 Authentication and Key Exchange Stage

In this stage, a mutual authentication between a user and the server is performed and a session key is established.

(1) A computes $f_A = h(\pi, t, ID_A)$, $e = E_{f_A}(y_u)$ and sends e and $PID_{A,i}$ to B.

(2) B acquires v and f by verifying $PID_{A,i}$. Based on f, B computes $y_u = D_{f_A}(e)$. After decryption, B computes $c = (g^x)^b \bmod p$ and selects a random number r. B computes $sk = h(c, r)$ and $M_B = h(sk, v)$. Now, B sends r and M_B to A.

(3) A computes a session key $sk = h(c, r)$. Now, A and B share the session key sk. Then A calculates $M_B = h(sk, h(\pi \oplus t))$ and compares it with the M_B sent by B. If so, A authenticates B as a legitimate server. To ensure the user anonymity, A computes $PID_{A,i+1} = h(PID_{A,i} \oplus sk)$ and replaces $PID_{A,i}$ stored in the smart card with $PID_{A,i+1}$. Finally A computes $M_A = E_{sk}(\pi \oplus t)$ and sends it to B.

(4) B decrypts M_A using sk and gets $\pi \oplus t$. With this value, B computes $v_A = h(\pi \oplus t)$ and verifies whether right v_A comes out, then B authenticates A as a legitimate user. Then B computes $PID_{A,i+1} = h(PID_{A,i} \oplus sk)$ and replaces $PID_{A,i}$ stored in the server's database with $PID_{A,i+1}$.

3 Efficiency and Security Analysis

3.1 Efficiency Analysis

To analyze the efficiency, we assumed the following environment.

- Hash function: SHA-1 (Hash size: 160 bits)
- Symmetric key algorithm: AES (Key length: 128 bits)
- r(nonce) : 128 bits
- b(long-term secret key) : 128 bits
- *ID* length: 32 bits
- Number of users registered in the server: n

In the above environment, the following is the result of the efficiency analysis of each scheme.

In case of user's computation cost, the proposed protocol showed the least number of hash functions with 4 compared to other protocols but it has one more encryption process added. All protocols show same number of exponentiation computation. In case of server's computation cost, especially number of hash function computation, the protocol proposed by Juang et al. must try until right *SID* comes out by substituting

Table 1. Efficiency analysis proposed protocol

Schemes	User's computation cost	Server's Computation cost	Communication Cost
Park et al.'s scheme [14]	5 Hash+ 1 Encryption+ 2 Exponentiation	O(n) Hash+ 1 Decryption+ 1 Exponentiation	1664 bits
Juang et al.'s scheme [15]	5 Hash+ 1 Encryption + 2 Exponentiation	O(n) Hash+ 1 Decryption+ 1 Exponentiation	1684 bits
Proposed scheme	4 Hash+ 2 Encryption + 2 Exponentiation	4 Hash+ 2 Decryption+ 1 Exponentiation	1632 bits

every π and t that the server has to $h(\pi, t, i)$ to verify the SID. In the worst case, for a user to verify itself in a single session, it must perform as many hash functions as the number of users registered in the server. This results in too high computational overhead. On the other hand, the proposed protocol does not have a step to substitute a specific value into a hash function and it only has to perform 4 hash function computations. However, it performs one more decryption process than other protocols. In case of communication cost, the proposed protocol has the smallest with 1632 bits. As a conclusion, although all protocols have similar numbers of encryption/decryption, exponentiation computations, and communication cost, the proposed protocol is more efficient than the others since the computational cost of the server's hash function is very low.

3.2 Security Analysis

The followings are the security analysis results of each scheme.

(1) User Anonymity: In this protocol, users' ID is not exposed during communication and it is not stored in the server but stored in temporary ID, PID forms. Therefore, it satisfies the user anonymity.

(2) Mutual Authentication: Since an attacker cannot know sk and $\pi \oplus t$, it cannot compute M_A and M_B. That is, only legitimate users and the server can compute M_A and M_B. Therefore, users and the server can mutually authenticate each other through M_A and M_B.

(3) Half-forward Secrecy: Half Forward Secrecy means the loss of one side's long-lived key should not be damaging to the previous sessions [16]. Let's assume that an attacker figured out a user's secret information (ID, π, t). The attacker may

Table 2. Efficiency analysis of proposed protocol

	Park et al. [14]	Juang et al. [15]	Proposed protocol
Providing the user anonymity	X	X	X
Providing the mutual authentication	X	X	X
Providing the half-forward secrecy	X	X	X
Secure against the user impersonation attack	X	X	X
Secure against the server impersonation attack	X	X	X
Secure against the replay attack	X	X	X
Secure against the stolen-verifier attack	X	X	X
Secure against the guessing attack	X	X	X

find out $g^x \bmod p$ by decrypting $E_f(g^x \bmod p)$. However, since the attacker does not know b, it cannot compute c, and cannot compute the session key. Therefore, the user side satisfies the forward secrecy. Again, let's assume the attacker find out the server's secret information (ID, π, t, b). The attacker may find out $g^x \bmod p$ by decrypting $E_f(g^x \bmod p)$, and since it knows b, it can compute c. Finally, the attacker can compute the session key. Therefore, since this protocol satisfies the forward secrecy only from the user side, it satisfies the half-forward secrecy.

(4) User Impersonation Attack: For an attacker impersonates as the legitimate user, it must transmit correct M_A to the server. However, since the attacker cannot compute the session key sk without knowing c, it cannot figure out the correct M_A. Therefore, this protocol is secure against the user impersonation attack.

(5) Server Impersonation Attack: For an attacker impersonates as the legitimate server, it must send correct M_B. However, since the attacker cannot compute the session key sk without knowing c, it cannot figure out the correct M_B. Therefore, this protocol is secure against the server impersonation attack.

(6) Replay Attack: A user randomly creates x on each session. Therefore, a different value is created every time for the e. Moreover, since a different value is created each time for M_A and M_B due to a random value r, this protocol is secure against the replay attack.

(7) Stolen-verifier Attack: An attacker is able to get PID, $v = h(\pi \oplus t)$ and $f = h(\pi, t, ID_A)$ by attacking the server's database. However, it cannot find out users' ID, password and symmetric key only with those information. An attacker may impersonate as the legitimate user based on the acquired information using the stolen-verifier

attack. In this case, an attacker should find out M_A, but since it cannot compute $\pi \oplus t$, it is unable to compute M_A. Therefore, an attacker cannot impersonate as the legitimate user. An attacker may also impersonate as a legitimate server based on the acquired information through the stolen-verifier attack, but it must know the secret key b to succeed an attack. However, b is secret information, which cannot be acquired through the stolen verifier attack, so an attacker cannot impersonate as the legitimate server. Therefore, this protocol is secure against the stolen-verifier attack.

(8) Guessing Attack: The secret information for an attacker to try guessing attack is v, M_A, M_B and f. To analogize the password π based on those information, it must know t, ID and c. However, there is no way for an attacker to find out those values. Moreover, the attacker may perform the guessing attack by trying XOR with the acquired values. If there is secret information separately with low entropy such as password or *ID* in the combined values by the attacker, the attacker may find out the password by performing the guessing attack. However, as a result of analyzing the combinations of every information that an attacker can get, we confirmed that there is no information combination separately including password or *ID*. We also confirmed that some other information do not include secret information at all and they are nonce or meaningless information. Therefore, this protocol is secure against the guessing attack. The table 1 in the Appendix describes possible combinations of the values to break our protocol using a guessing attack.

4 Conclusion

In this paper, we have pointed out that the Juang's protocol was vulnerable to the stolen-verifier attack and did not satisfy the user anonymity, and proposed an improved protocol. The proposed protocol ensures the user anonymity, is secure against the stolen-verifier attack and satisfies the half-forward secrecy.

References

1. Lamport, L.: Password authentication with insecure communication. Communications of the ACM 24(11), 770–772 (1981)
2. Awasthi, A., Lal, S.: A remote user authentication scheme using smart cards with forward secrecy. IEEE Trans. Consumer Electronic 49(4), 1246–1248 (2003)
3. Awasthi, A., Lal, S.: An enhanced remote user authentication scheme using smart cards. IEEE Trans. Consumer Electronic 50(2), 583–586 (2004)
4. Juang, W.: Efficient password authenticated key agreement using smart card. Computers & Security 23, 167–173 (2004)
5. Ku, W., Chen, S.: Weaknesses and improvements of an efficient password based remote user authentication scheme using smart cards. IEEE Trans. Consumer Electronic 50(1), 204–207 (2004)
6. Kumar, M.: New remote user authentication scheme using smart cards. IEEE Trans. Consumer Electronic 50(2), 597–600 (2004)

7. Kwon, T., Park, Y., Lee, H.: Security analysis and improvement of the efficient password-based authentication protocol. IEEE Communications Letters 9(1), 93–95 (2005)
8. Park, Y., Park, S.: Two factor authenticated key exchange (TAKE) protocol in public wireless LANs. IEICE Trans. Communications E87-B(5), 1382–1385 (2004)
9. Sun, H.: An efficient use authentication scheme using smart cards. IEEE Trans. Consumer Electronic 46(4), 958–961 (2000)
10. Wang, X., Zhang, W., Zhang, J., Khan, M.: Cryptanalysis and improvement on two efficient remote user authentication scheme using smart cards. Computer Standards & Interfaces 29(5), 507–512 (2007)
11. Yang, C., Hwang, M.: Cryptanalysis of simple authenticated key agreement protocols. IEICE Trans. Communications E87-A(8), 2174–2176 (2004)
12. Yang, C., Wang, R.: Cryptanalysis of a user friendly remote authentication scheme with smart cards. Computer Security 23, 425–427 (2004)
13. Chai, Z., Cao, Z., Lu, R.: Efficient Password-Based Authentication and Key Exchange Scheme Preserving User Privacy. In: Cheng, X., Li, W., Znati, T. (eds.) WASA 2006. LNCS, vol. 4138, pp. 467–477. Springer, Heidelberg (2006)
14. Park, Y.M., Park, S.K.: Two factor authenticated key exchange(TAKE) protocol in public wireless LANs. IEICE Trans. Communications E87-B(5), 1382–1385 (2004)
15. Juang, W.-S., Wu, J.-L.: Two efficient two-factor authenticated key exchange protocols in public wireless LANs. Computers and Electrical Engineering 10, 1–8 (2008)
16. Yeh, T.-C., Shen, H.-Y., Hwang, J.-J.: A Secure One-Time Password Authentication Scheme Using Smart Cards E85-B(11), 2515–2518 (2002)
17. Lee, H., Choi, D., Lee, Y., Won, D., Kim, S.: Cryptanalysis of Two-Factor Authenticated Key Exchange Protocol in Wireless LANs. World Academy of Science, Engineering and Technology 59, 390–393 (2009)

Efficient Proofs for CNF Formulas on Attributes in Pairing-Based Anonymous Credential System

Nasima Begum, Toru Nakanishi, and Nobuo Funabiki

Department of Communication Network Engineering, Okayama University, Japan
nasima@sec.cne.okayama-u.ac.jp, {nakanisi,funabiki}@cne.okayama-u.ac.jp

Abstract. To enhance user privacy, anonymous credential systems allow the user to convince a verifier of the possession of a certificate issued by the issuing authority anonymously. In the systems, the user can prove relations on his/her attributes embedded into the certificate. Previously, a pairing-based anonymous credential system with constant-size proofs in the number of attributes of the user was proposed. This system supports the proofs of the inner product relations on attributes, and thus can handle the complex logical relations on attributes as the CNF and DNF formulas. However this system suffers from the computational cost: The proof generation needs exponentiations depending on the number of the literals in OR relations. In this paper, we propose a pairing-based anonymous credential system with the constant-size proofs for CNF formulas and the more efficient proof generation. In the proposed system, the proof generation needs only multiplications depending on the number of literals, and thus it is more efficient than the previously proposed system. The key of our construction is to use an extended accumulator, by which we can verify that multiple attributes are included in multiple sets, all at once. This leads to the verification of CNF formulas on attributes. Since the accumulator is mainly calculated by multiplications, we achieve the better computational costs.

1 Introduction

1.1 Backgrounds

Electronic identification has been widely applied to access authorization to buildings, use of facilities, Web services, etc. Currently, electronic identity (eID) such as eID card is often used. The eID is issued by a trusted organization such as the government, company, or university, and is used for its service. Trusted ID is attractive for secondary use in commercial services. The eID includes attributes of the user such as the gender, the occupation, and the date of birth. In commercial cases, the attribute-based authentication is desired. For example, a service provider can deny access to kids, by checking the age in the eID. One of serious issues in the existing eID systems is user's privacy. In the systems, the eID may reveal the user's identity. The service provider can collect the use history of each user. Anonymous credential systems [8,6,7,13,10] are one of the solutions.

Anonymous credential systems allow an issuer to issue a certificate to a user. Each certificate is a proof of membership, qualification, or privilege, and contains user's attributes. The user can anonymously convince a verifier of the possession of the certificate. In the proof protocol of the anonymous credential system, only selected attributes can be disclosed without revealing any other information about the user's privacy. Proofs of more complex relation on attributes are also available. The AND relation is used when proving the possession of all of the multiple attributes. For example, the user can prove that he belongs to the department and that he is a professor, when entering the room of examination papers. The OR relation represents the proof for possession of one of multiple attributes. For example, he can prove that he is a technical staff, an assistant, or a professor when using a copy machine in a laboratory. An implementation on a standard Java card is shown in [3].

1.2 Previous Works

In [6], an RSA-based anonymous credential system with proofs for the AND and OR relation was proposed, where the proofs have the constant size in the number of attributes. Thus, this system provides the practicality for the ID card application. In [13], the pairing-based system with the constant-size proofs was proposed to achieve the short data size by excluding the RSA-related assumptions. However, both the RSA-based system and pairing-based one have a drawback: They allow us to prove only simple AND or OR relations on attributes. Namely, we cannot prove any combination of AND and OR relations simultaneously. In lots of applications, the proofs of such a combination are needed.

In [10], a pairing-based system with the constant-size proofs was proposed, where inner-product relations on attributes can be proved. This means that we can handle CNF or DNF formulas on attributes via the polynomial-based encoding shown in [11]. However, this system has a problem of the computational cost: The proof generation needs exponentiations depending on the number of the literals in OR relations. In usual cases that the formulas include OR relations for lots of literals, the user devices with limited computational power such as electronic ID cards need lots of time.

1.3 Our Contributions

We propose a pairing-based anonymous credential system with the constant size of proofs, where the combinations of AND and OR relations on attributes can be proved as CNF formulas. In our system, the proof generation cost is more efficient than the system [10], since only multiplications depending on the number of literals are needed. We extend the efficient accumulators in [7,13] to handle the proof of the CNF formula for the construction. Using the accumulator, lots of attributes are accumulated to one value, and we can verify that a value (or multiple values) is included in the accumulator. In our extended accumulator, we can verify that multiple attributes are included in multiple sets, all at once. This leads to the verification of CNF formulas on attributes. As the underlying anonymous credential system, our system is derived from the group signature scheme [12], which adopts structure-preserving signature in [2] as the certificate and Groth-Sahai proofs [9] as non-interactive witness-indistinguishable proofs. As a result, our system is secure in the standard model, as in [10]. In addition, due to the non-interactive proofs, our system is non-interactive where a user can generate the proof on certified attributes by himself and the verifier can verify the proof by himself, as in [10].

A demerit of our system is the increase of public parameters. Let V_ℓ be the set of attributes in the ℓ-th OR clause in the proved CNF formula, and let U be the set of user's certified attributes. This increase happens when the maximum of $|V_\ell \cap U|$ is large for multiple ℓ. In Section 6, we can demonstrate that the increase of the public parameters is not so huge in a likely example of CNF formula in eID applications.

2 Preliminaries

2.1 Bilinear Groups

Our scheme utilizes the following bilinear groups:

1. $\mathcal{G}$ and $\mathcal{T}$ are multiplicative cyclic groups of prime order p,
2. g is a randomly chosen generator of $\mathcal{G}$,
3. e is an efficiently computable bilinear map: $\mathcal{G} \times \mathcal{G} \to \mathcal{T}$, i.e., (1) for all $u, v \in \mathcal{G}$ and $a, b \in Z$, $e(u^a, v^b) = e(u, v)^{ab}$, and (2) $e(g, g) \neq 1_{\mathcal{T}}$.

2.2 Assumptions

As in the underlying system [12], the security of our system is based on the DLIN (Decision LINear) assumption [4], and the q-SFP (Simultaneous Flexible Pairing) assumption [2]. We also adopt n-DHE (DH Exponent) assumption [7] for the accumulator.

Definition 1 (DLIN assumption). *For all PPT algorithm $\mathcal{A}$, the probability*

$$|\Pr[\mathcal{A}(g, g^a, g^b, g^{ac}, g^{bd}, g^{c+d}) = 1] - \Pr[\mathcal{A}(g, g^a, g^b, g^{ac}, g^{bd}, g^z) = 1]|$$

is negligible, where $g \in_R \mathcal{G}$ and $a, b, c, d, z \in_R Z_p$.

Definition 2 (q-SFP assumption). *For all PPT algorithm $\mathcal{A}$, the probability*

$$\begin{aligned}
&\Pr[\mathcal{A}(g_z, h_z, g_r, h_r, a, \tilde{a}, b, \tilde{b}, \{(z_j, r_j, s_j, t_j, u_j, v_j, w_j)\}_{j=1}^q) \\
&= (z^*, r^*, s^*, t^*, u^*, v^*, w^*) \in \mathcal{G}^7 \\
&\wedge e(a, \tilde{a}) = e(g_z, z^*)e(g_r, r^*)e(s^*, t^*) \wedge e(b, \tilde{b}) = e(h_z, z^*)e(h_r, u^*)e(v^*, w^*) \\
&\wedge z^* \neq 1_{\mathcal{G}} \wedge z^* \neq z_j \text{ for all } 1 \leq j \leq q]
\end{aligned}$$

is negligible, where $(g_z, h_z, g_r, h_r, a, \tilde{a}, b, \tilde{b}) \in \mathcal{G}^8$ and all tuples $\{(z_j, r_j, s_j, t_j, u_j, v_j, w_j)\}_{j=1}^q)$ satisfy the above relations.

Definition 3 (n-DHE assumption). *For all PPT algorithm $\mathcal{A}$, the probability*

$$\Pr[\mathcal{A}(g, g^a, \ldots, g^{a^n}, g^{a^{n+2}}, \ldots, g^{a^{2n}}) = g^{a^{n+1}}]$$

is negligible, where $g \in_R \mathcal{G}$ and $a \in_R Z_p$.

2.3 Structure-Preserving Signatures (AHO Signatures)

We utilize the structure-preserving signatures, since the knowledge of the signature can be proved by Groth-Sahai proofs. As in [12], we adopt the AHO signature scheme in [2,1]. Using the AHO scheme, we can sign multiple group elements to obtain a constant-size signature. In our construction, a single group element is signed, and thus we describe the case of single message to be signed.

AHOKeyGen: Select bilinear groups $\mathcal{G}, \mathcal{T}$ with a prime order p and a bilinear map e. Select $g, G_r, H_r \in_R \mathcal{G}$, and $\mu_z, \nu_z, \mu, \nu, \alpha_a, \alpha_b \in_R Z_p$. Compute $G_z = G_r^{\mu_z}, H_z = H_r^{\nu_z}, G = G_r^{\mu}, H = H_r^{\nu}$, $A = e(G_r, g^{\alpha_a}), B = e(H_r, g^{\alpha_b})$. Output the public key as $pk = (\mathcal{G}, \mathcal{T}, p, e, g,\ G_r, H_r, G_z, H_z, G, H, A, B)$, and the secret key as $sk = (\alpha_a, \alpha_b, \mu_z, \nu_z, \mu, \nu)$.

AHOSign: Given message M together with sk, choose $\beta, \epsilon, \eta, \iota, \kappa \in_R Z_p$, and compute $\theta_1 = g^{\beta}$, and

$$\theta_2 = g^{\epsilon - \mu_z \beta} M^{-\mu}, \quad \theta_3 = G_r^{\eta}, \quad \theta_4 = g^{(\alpha_a - \epsilon)/\eta},$$
$$\theta_5 = g^{\iota - \nu_z \beta} M^{-\nu}, \quad \theta_6 = H_r^{\kappa}, \quad \theta_7 = g^{(\alpha_b - \iota)/\kappa}.$$

Output the signature $\sigma = (\theta_1, \ldots, \theta_7)$.

AHOVerify: Given the message M and the signature $\sigma = (\theta_1, \ldots, \theta_7)$, accept these if the following equations hold:

$$A = e(G_z, \theta_1) \cdot e(G_r, \theta_2) \cdot e(\theta_3, \theta_4) \cdot e(G, M),$$
$$B = e(H_z, \theta_1) \cdot e(H_r, \theta_5) \cdot e(\theta_6, \theta_7) \cdot e(H, M).$$

This signature is existentially unforgeable against chosen-message attacks under the q-SFP assumption [2].

Using the re-randomization algorithm in [2], this signature can be publicly randomized to obtain another signature $(\theta'_1, \ldots, \theta'_7)$ on the same message. As a result, in the following Groth-Sahai proof, $(\theta'_i)_{i=3,4,6,7}$ can be safely revealed, while $(\theta'_i)_{i=1,2,5}$ have to be committed, as mentioned in [12].

2.4 Groth-Sahai (GS) Proofs

To prove the secret knowledge in relations of the bilinear maps, we utilize Groth-Sahai (GS) proofs [9]. As in [12], we adopt the instantiation based on DLIN assumption. For the bilinear groups, the proof system needs a common reference string $(\boldsymbol{f}_1, \boldsymbol{f}_2, \boldsymbol{f}_3) \in \mathcal{G}^3$ for $\boldsymbol{f}_1 = (f_1, 1, g), \boldsymbol{f}_2 = (1, f_2, g)$ for some $f_1, f_2 \in \mathcal{G}$.

The commitment to an element X is computed as $\boldsymbol{C} = (1, 1, X) \cdot \boldsymbol{f}_1^r \cdot \boldsymbol{f}_2^s \cdot \boldsymbol{f}_3^t$ for $r, s, t \in_R Z_p^*$. In case of the CRS setting for perfectly sound proofs, $\boldsymbol{f}_3 = \boldsymbol{f}_1^{\xi_1} \cdot \boldsymbol{f}_2^{\xi_2}$ for $\xi_1, \xi_2 \in_R Z_p^*$. Then, the commitment $\boldsymbol{C} = (f_1^{r+\xi_1 t}, f_2^{s+\xi_2 t}, X g^{r+s+t(\xi_1+\xi_2)})$ is the linear encryption in [4]. On the other hand, in the setting of the witness indistinguishability, $\boldsymbol{f}_1, \boldsymbol{f}_2, \boldsymbol{f}_3$ are linearly independent, and thus $\boldsymbol{C}$ is perfectly hiding. The DLIN assumption implies the indistinguishability of the CRS. To prove that the committed variables satisfy the pairing relations, the prover prepares the commitments, and replaces the variables in the pairing relations by the commitments. The GS proof allows us to prove the set of pairing product equations:

$$\prod_{i=1}^{n} e(A_i, X_i) \cdot \prod_{i=1}^{n} \prod_{j=1}^{n} e(X_i, X_j)^{a_{ij}} = t$$

for variables $X_1, \ldots, X_n \in \mathcal{G}$ and constants $A_1, \ldots, A_n \in \mathcal{G}, a_{ij} \in Z_p, t \in \mathcal{T}$.

3 Extended Accumulator for Inclusions in Multiple Sets

In [7], an efficient pairing-based accumulator is proposed. The accumulator is generated from a set of values, and we can verify that a single value is included in the set. In [13], the extended version is proposed, where we can verify that multiple values are included in the specified set, all at once. This paper furthermore extends the accumulator, where we can verify that, for a set U, for all multiple sets $V_1, \ldots, V_L$, a value from U is included in each V_ℓ, i.e., $U \cap V_\ell \neq \emptyset$, all at once. The verification of this type is applied to our construction of anonymous credential system with proofs for CNF formulas on attributes.

3.1 Proposed Construction

Let $V_1, \ldots, V_L$ be L subsets of $\{1, \ldots, n\}$. Define $\mathcal{V} = (V_1, \ldots, V_L)$. Let U be a subset of $\{1, \ldots, n\}$ of size L' satisfying $U \cap V_\ell \neq \emptyset$ for all $1 \leq \ell \leq L$. For any instance of $\mathcal{V}$, we need to fix L. For any instance, we set the maximum of $|V_\ell|$ as η for all $1 \leq \ell \leq L$. For any instance, we set the maximum of $|U \cap V_\ell|$ as ζ_ℓ for each $1 \leq \ell \leq L$. These conditions are needed to apply this accumulator to our anonymous credential system.

Then, the following accumulator allows us to confirm $U \cap V_\ell \neq \emptyset$ for all $1 \leq \ell \leq L$, all at once.

AccSetup: This is the algorithm to output the public parameters. For all $1 \leq \ell \leq L$, compute $c_\ell = (\eta + 1)^{\ell - 1}$ and set $\mathcal{C} = (c_1, \ldots, c_L)$. We assume that $(\eta + 1)c_L < p$. Select bilinear groups $\mathcal{G}, \mathcal{T}$ with a prime order p and a bilinear map e. Select $g \in_R \mathcal{G}$. Select $\gamma \in_R Z_p$, and compute and publish

$\{\zeta_\ell\}_{1\le\ell\le L}, \mathcal{C}, p, \mathcal{G}, \mathcal{T}, e, g, g_1 = g^{\gamma^1}, \ldots, g_n = g^{\gamma^n}, g_{n+2} = g^{\gamma^{n+2}}, \ldots, g_{2n} = g^{\gamma^{2n}}$ and $z = e(g,g)^{\gamma^{n+1}}$ as the public parameters.

AccGen: This is the algorithm to compute the accumulator using the public parameters. The accumulator $acc_{\mathcal{V}}$ of $\mathcal{V}$ is computed as

$$acc_{\mathcal{V}} = \prod_{1\le\ell\le L} (\prod_{j\in V_\ell} g_{n+1-j})^{c_\ell}.$$

AccWitGen: This is the algorithm to compute the witness that $U \cap V_\ell \neq \emptyset$ for all $1 \le \ell \le L$, using the public parameters. Given U, $\mathcal{V}$, and the accumulator $acc_{\mathcal{V}}$, the witness is computed as $W = \prod_{i\in U}\prod_{1\le\ell\le L}(\prod_{j\in V_\ell}^{j\neq i} g_{n+1-j+i})^{c_\ell}$. Furthermore, the auxiliary parameters are computed as $\delta_\ell = |U \cap V_\ell|$ for all $1 \le \ell \le L$.

AccVerify: This is the algorithm to verify that $U \cap V_\ell \neq \emptyset$ for all $1 \le \ell \le L$, using the witness, the auxiliary parameters, and the public parameters. Given $acc_{\mathcal{V}}, U$, W and $\{\delta_\ell\}_{1\le\ell\le L}$, set $u = \delta_1 c_1 + \ldots + \delta_L c_L$. Then, accept if

$$\frac{e(\prod_{i\in U} g_i, acc_{\mathcal{V}})}{e(g,W)} = z^u, \qquad 1 \le \delta_\ell \le \zeta_\ell,$$

for all $1 \le \ell \le L$.

3.2 Security

We can show the correctness and the security.

Theorem 1. *Assume that* **AccSetup**, **AccGen**, **AccWitGen** *correctly compute all parameters. Then,* **AccVerify** *accepts* $U, acc_{\mathcal{V}}, W, \{\delta_\ell\}_{1\le\ell\le L}$ *that they outputs.*

Theorem 2. *Under the n-DHE assumption, any adversary cannot output* $(U, \mathcal{V} = \{V_\ell\}_{1\le\ell\le L}, W, \{\delta_\ell\}_{1\le\ell\le L})$ *where* $U, V_1, \ldots, V_L$ *are subsets of* $\{1,\ldots,n\}$ *and* $\delta_\ell \in Z_p$, *on inputs* $\{\zeta_\ell\}_{1\le\ell\le L}, \mathcal{C}, p, \mathcal{G}, \mathcal{T}, e, g, g_1, \ldots, g_n, g_{n+2}, \ldots, g_{2n}$ *and* z *s.t.* **AccVerify** *accepts* $U, acc_{\mathcal{V}}, W, \{\delta_\ell\}_{1\le\ell\le L}$ *but there exists some* V_ℓ *satisfying* $U \cap V_\ell = \emptyset$.

The proof of Theorem 1 is in Appendix A. The proof of Theorem 2 will be shown in the full paper.

Remark 1. Note that, in Theorem 2, the adversary is allowed to output $\delta_\ell \neq |U \cap V_\ell|$, since the condition in **AccVerify** is only $1 \le \delta_\ell \le \zeta_\ell$. This implies that, under the n-DHE assumption, for any $u' = \delta_1 c_1 + \cdots + \delta_L c_L$ s.t. $u' \neq u = |U \cap V_1|c_1 + \cdots + |U \cap V_L|c_L$ and $1 \le \delta_\ell \le \zeta_\ell$, the adversary cannot output $(U, \mathcal{V}, W, \{\delta_\ell\}_{1\le\ell\le L})$ s.t. $\frac{e(\prod_{i\in U} g_i, acc_{\mathcal{V}})}{e(g,W)} = z^{u'}$, when there exists some V_ℓ satisfying $U \cap V_\ell = \emptyset$.

4 Syntax and Security Model of Anonymous Credential System

We consider *non-interactive* anonymous credential system, where a user can generate the proof on certified attributes by himself and the verifier can verify the proof by himself. This is similar concept to the group signature scheme, and thus our security model is derived from that of the group signature scheme.

The security model of the group signature scheme consists of traceability, non-frameability, and anonymity. The traceability means that once a group signature is opened, it identifies a group member who joined the group. The non-frameability means that no one except a group member can issue a valid group signature that can be identified to the member. In this paper, since we concentrate on the function of the anonymous attribute proof, we do not care about the tracing. Thus, in the following model, we omit the functions on the tracing. This is why the non-frameability is omitted, and the traceability is replaced by the similar requirement *misauthentication resistance*. The misauthentication resistance means the soundness of the attribute proof. Note that the combination of our construction and the construction of the group signature scheme [12] can achieve the tracing.

4.1 Syntax

The attribute value is indexed by an integer from $\{1, \ldots, n\}$, where n is the total number of attribute values. As in [13], all attribute values in all attribute types are indexed by using the universal set $\{1, \ldots, n\}$. We describe CNF formula Ψ on attributes using the indexes as follows: $(a_{11} \vee a_{12} \vee \cdots) \wedge (a_{21} \vee a_{22} \vee \cdots) \wedge \cdots$ with $a_{11}, a_{12}, \ldots, a_{21}, a_{22}, \ldots \in \{1, \ldots, n\}$. Each literal $a_{11}, a_{21}, \cdots$ means that the proving user owns the attribute of the index. Set $V_1 = (a_{11}, a_{12}, \ldots)$ and $V_2 = (a_{21}, a_{22}, \ldots) \ldots$. Set U be the set of attributes (indexes) of the proving user. We assume that $|V_\ell|$ has the upper bound, η, for all $1 \leq \ell \leq L$, and assume that the size of $|U \cap V_\ell|$ has the upper bound, ζ_ℓ, for each $1 \leq \ell \leq L$. Also, we assume the maximum number of clauses in any CNF formula, L.

The anonymous credential system consists of the following algorithms:

IssuerKeyGen: The inputs of this algorithm are n, L, η, ζ_ℓ for all $1 \leq \ell \leq L$. The outputs are issuer's public key *ipk* and issuer's secret key *isk*.

CertObtain: This is an interactive protocol between a probabilistic algorithm **CertObtain-**$\mathcal{U}_k$ for the k-th user and a probabilistic algorithm **CertObtain-**$\mathcal{I}$ for an issuer, where the issuer issues the certificate including the attributes to the user. **CertObtain-**$\mathcal{U}_k$, on input *ipk* and $U_k \subset \{1, \ldots, N\}$ that is indexes corresponding to the attribute values of the user, outputs the certificate $cert_k$ ensuring the attributes of the user. On the other hand, **CertObtain-**$\mathcal{I}$ is given ipk, isk as inputs.

ProofGen: This probabilistic algorithm, on inputs *ipk*, U_k, $cert_k$, Ψ that is the predicate on attributes to be proved, outputs the proof σ.

Verify: This is a deterministic algorithm for verification. The input is ipk, a proof σ, and the predicate Ψ. Then the output is 'valid' if the attributes in U_k satisfy Ψ, or 'invalid' otherwise.

4.2 Security Model

The security model consists of misauthentication resistance and anonymity. The misauthentication resistance requirement captures the soundness of the attribute proof. This means that an adversary $\mathcal{A}$ cannot try to forge a proof for a predicate, where the attributes of any user corrupted by $\mathcal{A}$ do not satisfy the predicate. The anonymity requirement captures the anonymity and unlinkability of proofs, as in the group signatures. The formal definitions are in Appendix B.

5 Proposed Anonymous Credential System

5.1 Construction Idea

For the verification of CNF formulas, we use our extended accumulator. Consider the following CNF formula: $(a_{11} \vee a_{12} \vee \ldots) \wedge (a_{21} \vee a_{22} \ldots) \wedge \ldots$, where $a_{11}, a_{12} \ldots a_{21}, a_{22} \ldots \in \{1, \ldots, n\}$ means that the proving user owns the corresponding attribute. Set $V_1 = (a_{11}, a_{12}, \ldots)$ and $V_2 = (a_{21}, a_{22}, \ldots) \ldots$. Let U be the set of attributes (indexes) of the proving user. Then, using the accumulator, we can confirm that $U \cap V_\ell \neq \emptyset$ for any V_ℓ. This means that the attributes in U satisfies this CNF formula, since some attribute in U is one of attributes in every OR clause expressed by V_ℓ.

Our construction is based on the anonymous credential system using the AHO signatures and GS proofs. This is derived from the construction of a group signature scheme in [12], which is secure in the standard model. In the underlying system, the certificate is an AHO signature, where attributes of the user are unified to one element and embedded as $\prod_{i \in U} g_i$, to apply it to the verification of accumulator.

In our system, a part of accumulator verification, $\frac{e(\prod_{i \in U} g_i, acc_{\mathcal{V}})}{e(g, W)} = z^u$, can be proved without revealing secret information by directly using the GS proof for the pairing relation. However, the other part of the verification, $1 \leq \delta_\ell \leq \zeta$ where $u = \delta_1 c_1 + \ldots + \delta_L c_L$, needs another technique. We utilize the set membership proof technique [5] to prove this relation, while hiding parameters δ_ℓ. Since δ_ℓ may indicate some secret information (i.e., $|U \cap V_\ell|$) of the user, we need the zero-knowledge type of proof. As the preparation, the issuer publishes signatures on possible values as g_1^u, denoted by set Φ. In the attribute proof, the user proves the knowledge of the signature on a committed value to convince the verifier that the committed value is g_1^u such that u satisfies the conditions.

To use the accumulator, L, which is the number of V_ℓ, has to be fixed. On the other hand, in the CNF formula of the input, the number of clauses, L', is less than or equal to L. Then, we introduce a special attribute a_{SP} that every user always owns. To the CNF formula with L' clauses, a_{SP} is added such as the number of clauses becomes L. Namely, given formula $\Psi = (a_{11} \vee a_{12} \vee \cdots) \wedge (a_{21} \vee a_{22} \vee \cdots) \wedge \cdots (a_{L'1} \vee a_{L'2} \vee \cdots)$ is extended to $\Psi' = (a_{11} \vee a_{12} \vee \cdots) \wedge (a_{21} \vee a_{22} \vee \cdots) \wedge \cdots (a_{L'1} \vee a_{L'2} \vee \cdots) \wedge a_{\mathrm{SP}} \wedge \cdots a_{\mathrm{SP}}$. The literal a_{SP} is always true, this extended formula is the same as the original.

5.2 Proposed Construction

IssuerKeyGen. It is given n, L, η, and ζ_ℓ for all $1 \le \ell \le L$.

1. Select bilinear groups $\mathcal{G}$, $\mathcal{T}$ with the same order p and the bilinear map e, and $g \in_R \mathcal{G}$.
2. Generate public parameters of the extended accumulator: For all $1 \le \ell \le L$, compute $c_\ell = (\eta + 1)^{\ell-1}$ and set $\mathcal{C} = (c_1, \dots, c_L)$. Select $\gamma \in_R Z_p$, and compute
$$pk_{\mathrm{acc}} = (\mathcal{C}, g_1 = g^{\gamma^1}, \dots, g_n = g^{\gamma^n}, g_{n+2} = g^{\gamma^{n+2}}, \dots, g_{2n} = g^{\gamma^{2n}},\\ z = (g,g)^{\gamma^{n+1}}).$$
3. Generate two key pairs for the AHO signature:
$$pk^{(d)}_{\mathrm{AHO}} = (G^{(d)}_r, H^{(d)}_r, G^{(d)}_z, H^{(d)}_z, G^{(d)}, H^{(d)}, A^{(d)}, B^{(d)}),\\ sk^{(d)}_{\mathrm{AHO}} = (\alpha^{(d)}_a, \alpha^{(d)}_b, \mu^{(d)}_z, \nu^{(d)}_z, \mu, \nu),$$
where $d \in \{0, 1\}$.
4. Generate a CRS for the GS NIWI proof: select $\boldsymbol{f} = (\boldsymbol{f}_1, \boldsymbol{f}_2, \boldsymbol{f}_3)$, where $\boldsymbol{f}_1 = (f_1, 1, g)$, $\boldsymbol{f}_2 = (1, f_2, g)$, $\boldsymbol{f}_3 = \boldsymbol{f}_1^{\xi_1} \cdot \boldsymbol{f}_2^{\xi_2}$ for $f_1, f_2 \in_R \mathcal{G}$ and $\xi_1, \xi_2 \in_R Z_p^*$.
5. For $\mathcal{C}$, define set $\Phi = \{u = \sum_{\ell=1}^{L} \delta_\ell c_\ell | 1 \le \delta_\ell \le \zeta_\ell \text{ for all } 1 \le \ell \le L\}$, where $|\Phi| = \prod_{1 \le \ell \le L} \zeta_\ell$. For every $u \in \Phi$, generate the AHO signature on g_1^u. The signature is denoted as $\tilde{\sigma}_u = (\tilde{\theta}_{u1}, \dots, \tilde{\theta}_{u7})$.
6. Output the issuer public key
$$ipk = (p, \mathcal{G}, \mathcal{T}, e, g, pk^{(0)}_{\mathrm{acc}}, pk^{(1)}_{\mathrm{acc}}, pk_{\mathrm{AHO}}, \boldsymbol{f}, \{\tilde{\sigma}_u\}_{u \in \Phi}),$$
and the issuer secret key $isk = (sk^{(0)}_{\mathrm{AHO}}, sk^{(1)}_{\mathrm{AHO}})$.

CertObtain. This is an interactive protocol between **CertObtain-$\mathcal{U}_k$** (user) and **CertObtain-$\mathcal{I}$** (issuer). The common inputs of this protocol consist of ipk, and U_k that is the indexes of attribute values of the user. The input of **CertObtain-$\mathcal{I}$** is isk. We introduce a special attribute value a_{SP}. Every user has a_{SP}.

1. **CertObtain-$\mathcal{I}$**: Generate $P_k = \prod_{i \in U_k} g_i$.
2. **CertObtain-$\mathcal{I}$**: Using $sk^{(1)}_{\mathrm{AHO}}$, generate an AHO signature $\sigma_k = (\theta_1, \dots, \theta_7)$ on message P_k. Return σ_k to **CertObtain-$\mathcal{U}_k$** as the certificate.
3. **CertObtain-$\mathcal{U}_k$**: Compute $P_k = \prod_{i \in U_k} g_i$, and verify the AHO signature σ_k on P_k. Output $cert_k = (P_k, \sigma_k)$.

ProofGen. The inputs are ipk, U_k, $cert_k$, and the CNF formula Ψ. For given formula $\Psi = (a_{11} \vee a_{12} \vee \cdots) \wedge (a_{21} \vee a_{22} \vee \cdots) \wedge \cdots (a_{L'1} \vee a_{L'2} \vee \cdots)$ with $a_{11}, a_{12}, \ldots, a_{21}, a_{22}, \ldots \in \{1, \ldots, n\}$, define $V_1 = \{a_{11}, a_{12}, \ldots\}, V_2 = \{a_{21}, a_{22}, \ldots\}, \ldots$. If $L' < L$, define $V_{L'+1} = \cdots = V_L = \{a_{\mathrm{SP}}\}$.

1. Compute the accumulator: $acc_{\mathcal{V}} = \prod_{1 \le \ell \le L}(\prod_{j \in V_\ell} g_{n+1-j})^{c_\ell}$.
2. Compute the witness $W_{\mathcal{V}} = \prod_{i \in U_k} \prod_{1 \le \ell \le L}(\prod_{j \in V_\ell}^{j \ne i} g_{n+1-j+i})^{c_\ell}$ that U_k satisfies $\mathcal{V}$ for $acc_{\mathcal{V}}$, and sets $u = \delta_1 c_1 + \ldots + \delta_L c_L$, where $\delta_\ell = |U_k \cap V_\ell|$ for all $1 \le \ell \le L$.
3. Set $\tau_u = g_1^u$. From ipk, select the AHO signature $\tilde{\sigma}_u = (\tilde{\theta}_{u1}, \ldots, \tilde{\theta}_{u7})$ on the g_1^u.
4. Compute GS commitments $com_{P_k}, com_{W_{\mathcal{V}}}, com_{\tau_u}$ to $P_k, W_{\mathcal{V}}, \tau_u$. Then, re-randomize the AHO signature σ_k to obtain $\sigma'_k = \{\theta'_1, \ldots, \theta'_7\}$, and compute GS commitments $\{com_{\theta'_i}\}_{i \in \{1,2,5\}}$ to $\{\theta'_i\}_{i \in \{1,2,5\}}$. Similarly, re-randomize the AHO signature $\tilde{\sigma}_u$ to obtain $\tilde{\sigma}'_u = \{\tilde{\theta'}_{u1}, \ldots, \tilde{\theta'}_{u7}\}$, and compute GS commitments $\{com_{\tilde{\theta'}_{ui}}\}_{i \in \{1,2,5\}}$ to $\{\tilde{\theta'}_{ui}\}_{i \in \{1,2,5\}}$.
5. Generate the GS proofs $\{\pi_i\}_{i=1}^5$ s.t.

$$1_{\mathcal{T}} = e(P_k, acc_{\mathcal{V}}) \cdot e(g, W_{\mathcal{V}})^{-1} \cdot e(\tau_u, g_n)^{-1}, \tag{1}$$
$$A \cdot e(\theta'_3, \theta'_4)^{-1} = e(G_z, \theta'_1) \cdot e(G_r, \theta'_2) \cdot e(G, P_k), \tag{2}$$
$$B \cdot e(\theta'_6, \theta'_7)^{-1} = e(H_z, \theta'_1) \cdot e(H_r, \theta'_5) \cdot e(H, P_k), \tag{3}$$
$$A \cdot e(\tilde{\theta'}_{u3}, \tilde{\theta'}_{u4})^{-1} = e(G_z, \tilde{\theta'}_{u1}) \cdot e(G_r, \tilde{\theta'}_{u2}) \cdot e(G, \tau_u), \tag{4}$$
$$B \cdot e(\tilde{\theta'}_{u6}, \tilde{\theta'}_{u7})^{-1} = e(H_z, \tilde{\theta'}_{u1}) \cdot e(H_r, \tilde{\theta'}_{u5}) \cdot e(H, \tau_u), \tag{5}$$

6. Output $\sigma = (\{\theta'_i\}_{i=3,4,6,7}, \{\tilde{\theta'}_{ui}\}_{i=3,4,6,7}, com_{P_k}, com_{W_{\mathcal{V}}}, com_{\tau_u}, \{com_{\theta'_i}\}_{i=1,2,5}, \{com_{\tilde{\theta'}_{ui}}\}_{i=1,2,5}, \{\pi_i\}_{i=1}^5)$.

The equation (1) shows one of verification relations of accumulator:

$$\frac{e(\prod_{i \in U_k} g_i, acc_{\mathcal{V}})}{e(g, W_{\mathcal{V}})} = e(g_1^u, g_n) = z^u,$$

where $P_k = \prod_{i \in U_k} g_i$ and $\tau_u = g_1^u$. The equations (2), (3) show the knowledge of the AHO signature of P_k, i.e., the certificate $cert_k$. The equations (4), (5) show the knowledge of the AHO signature of τ_u. This ensures that $1 \le \delta_\ell \le \zeta$ where $u = \delta_1 c_1 + \ldots + \delta_L c_L$. Thus, together with the equation (1), it ensures the verification of the accumulator. This is why the verifier is ensured that $U_k \cap V_\ell \ne \emptyset$, i.e, attributes in U_k satisfies the CNF formula Ψ.

Verify. The inputs are ipk, the proof σ, and the CNF formula Ψ.

1. Compute the accumulator $acc_{\mathcal{V}}$, as in **ProofGen**.
2. Accept σ, if the verifications of all GS proofs $\{\pi_i\}_{i=1}^5$ are successful.

5.3 Security

We can prove the following security of our construction.

Theorem 3. *The proposed system satisfies the misauthentication resistance under the security of the AHO signatures and the extended accumulators.*

Theorem 4. *The proposed system satisfies the anonymity under the DLIN assumption.*

The proofs of these theorems will be shown in the full paper.

5.4 Protection against Reply Attack

In the authentication, the re-use of the proof should be prevented. In our system, the proof depends on the predicate, as the signature depends on the message. Thus, to prevent the re-use, we can make the predicate include a random nonce, as follows. Consider T-bit nonce $b_1 \cdots b_T$ with $b_t \in \{0, 1\}$. Then, we introduce virtual attributes $\tilde{a}_{1,0}, \tilde{a}_{1,1}, \ldots, \tilde{a}_{T,0}, \tilde{a}_{T,1}$. To original CNF formula Ψ, we append the following OR clause, $(a_{\mathrm{SP}} \vee \tilde{a}_{1,b_1} \vee \cdots \vee \tilde{a}_{T,b_T})$. Since this clause includes a_{SP}, this clause is meaningless in the attribute proof (i.e,. this clause is satisfied anytime). On the other hand, an appended formula is different from other appended formulas, due to the random nonce. In this method, the public parameters of the accumulator for the virtual attributes are additionally required. The number of the parameters is only $2T$.

6 Comparisons

We compare our system with the system in [10] that allows us to prove inner product relations on attributes. Since both systems achieve constant-size proofs, we mainly concentrate on the computational costs for generating proofs.

As mentioned in [10], using the inner product relations and suitable attribute encoding, CNF formulas (also DNF formulas) can be expressed (the encoding is shown in [11]). As the encoding, a polynomial is used. Consider polynomial $f(x) = c_d x^d + \cdots + c_0 x^0$ and the coefficients vector $\boldsymbol{p} = (c_d, \ldots, c_0)$. In the system of [10], the user's attribute is expressed by $\omega \in Z_p$. Let $\boldsymbol{\omega} = (\omega^d \bmod p, \ldots, \omega^0 \bmod p)$. Using the inner product proof system, the user can prove $\boldsymbol{\omega} \cdot \boldsymbol{p} = 0$. This means that $f(\omega) = c_d \omega^d + \cdots + c_0 \omega^0 = 0$. In the computations of the system, each element of the vector is set as the exponent on some base parameter. This means that an exponentiation for each element is needed, and thus the computational cost depends on the size of the vector. In the encoding of OR relation, for example, $(x = a_1) \vee (x = a_2)$ can be encoded as the univariate polynomial $(x - a_1) \cdot (x - a_2)$. The case of more literals is similar. Let d be the number of attribute values that are used in the OR relation. Then, this encoding needs d coefficients and the vector size becomes d. Thus, the computational cost of the proof is d exponentiations. Consider a CNF formula such that the ℓ-th

OR clause has d_ℓ literals ($1 \le \ell \le L$), where L is the number of OR clauses. Then, by the encoding for AND relation in [11], the computational cost becomes $\sum_{1\le\ell\le L} d_\ell$ exponentiations.

In our system, the computations of $acc_{\mathcal{V}}$ and $W_{\mathcal{V}}$ need only $O(d)$ multiplications, while L exponentiations by c_ℓ are needed. In cases that some OR clauses have lots of literals, the cost of our system is much more efficient than the system [10].

On the other hand, our system has disadvantages against the system [10]: The inner product proofs can be converted to proofs of DNF formulas, while our system cannot support the DNF formulas directly. In some applications, DNF formulas may be better. Another disadvantage is the public key size. In our system, we need to publish signatures for set Φ, where the size $|\Phi|$ is $\prod_{1\le\ell\le L} \zeta_\ell$. The size may become large.

To show the effectiveness of our system, we will discuss a concrete application. Consider the eID application as mentioned in Introduction. In such an application, a user often proves the following CNF formula on user's attributes.

$$gender = male \wedge birth\_year \in \{1900, \ldots, 1992\}$$

$$\wedge profession \in \{student, teacher, professor, \ldots\} \wedge \cdots .$$

Namely, for each attribute type, the user's attribute value is included in a set of attribute values. This example considers that a service provider grasps user's profile that can be useful for marketing, while serious private data are concealed. By the OR relation of *birth_year*, the user proves that he is adult, but the concrete age is concealed. As in this example, for the proof including OR relations with lots of literals, the system [10] needs heavy computations of $\sum_{1\le\ell\le L} d_\ell$ exponentiations. On the other hand, as shown above, our system has the additional public key size. However, in this eID application, ζ_ℓ (i.e., the maximum of $|U \cap V_\ell|$) can be 1 for the attribute types such that the user owns a single attribute value such as gender and birth_year. For the attribute types such as the user owns multiple attribute values such as the professions, ζ_ℓ can be more than 1. Most attribute types are former, and for the latter type, a user does not own lots of attribute values. Thus, in this application, the public key size depending on $\prod_{1\le\ell\le L} \zeta_\ell$ is not so huge.

7 Conclusions

In this paper, we have proposed a pairing-based anonymous credential system with the constant-size proofs of CNF formulas. Using our extended accumulator, the proof generation cost is more efficient than the system [10], since only multiplications depending on the number of literals are needed. The compensation is the increase of public parameters. We have demonstrated that, for CNF formulas that can be often used in eID applications, the increase is not so huge.

Our future works include the evaluation based on the implementation, and the applications to the electronic ID card devises or in the mobile environments.

References

1. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 209–236. Springer, Heidelberg (2010)
2. Abe, M., Haralambiev, K., Ohkubo, M.: Signing on elements in bilinear groups for modular protocol design. Cryptology ePrint Archive, Report 2010/133 (2010), `http://eprint.iacr.org/`
3. Bichsel, P., Camenisch, J., Groß, T., Shoup, V.: Anonymous credentials on a standard java card. In: Proc. ACM Conference on Computer and Communications Security 2009 (ACM-CCS 2009), pp. 600–610 (2009)
4. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004)
5. Camenisch, J., Chaabouni, R., Shelat, A.: Efficient protocols for set membership and range proofs. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 234–252. Springer, Heidelberg (2008)
6. Camenisch, J., Groß, T.: Efficient attributes for anonymous credentials. In: Proc. ACM Conference on Computer and Communications Security 2008 (ACM-CCS 2008), pp. 345–356 (2008)
7. Camenisch, J., Kohlweiss, M., Soriente, C.: An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 481–500. Springer, Heidelberg (2009)
8. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer, Heidelberg (2002)
9. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (2008)
10. Izabachène, M., Libert, B., Vergnaud, D.: Block-wise p-signatures and non-interactive anonymous credentials with efficient attributes. In: Chen, L. (ed.) IMACC 2011. LNCS, vol. 7089, pp. 431–450. Springer, Heidelberg (2011)
11. Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 146–162. Springer, Heidelberg (2008)
12. Libert, B., Peters, T., Yung, M.: Scalable group signatures with revocation. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 609–627. Springer, Heidelberg (2012)
13. Sudarsono, A., Nakanishi, T., Funabiki, N.: Efficient proofs of attributes in pairing-based anonymous credential system. In: Fischer-Hübner, S., Hopper, N. (eds.) PETS 2011. LNCS, vol. 6794, pp. 246–263. Springer, Heidelberg (2011)

A Proof for Extended Accumulator

Theorem 1. *Assume that* **AccSetup**, **AccGen**, **AccWitGen** *correctly compute all parameters. Then,* **AccVerify** *accepts* $U, acc_{\mathcal{V}}, W, \{\delta_\ell\}_{1\le\ell\le L}$ *that they outputs.*

Proof. We have

$$acc_{\mathcal{V}} = \prod_{1\le\ell\le L}(\prod_{j\in V_\ell} g_{n+1-j})^{c_\ell}, \qquad W = \prod_{i\in U}\prod_{1\le\ell\le L}(\prod_{j\in V_\ell}^{j\ne i} g_{n+1-j+i})^{c_\ell}.$$

Thus, the left hand of the verification equation is as follows.

$$\begin{aligned}\frac{e(\prod_{i\in U} g_i, acc_{\mathcal{V}})}{e(g,W)} &= \frac{e(\prod_{i\in U} g_i, \prod_{1\le\ell\le L}(\prod_{j\in V_\ell} g_{n+1-j})^{c_\ell})}{e(g, \prod_{i\in U}\prod_{1\le\ell\le L}(\prod_{j\in V_\ell}^{j\ne i} g_{n+1-j+i})^{c_\ell})} \\ &= \frac{e(g, \prod_{i\in U}\prod_{1\le\ell\le L}(\prod_{j\in V_\ell} g_{n+1-j+i})^{c_\ell})}{e(g, \prod_{i\in U}\prod_{1\le\ell\le L}(\prod_{j\in V_\ell}^{j\ne i} g_{n+1-j+i})^{c_\ell})} \\ &= e(g, \prod_{i\in U}\prod_{1\le\ell\le L}(\prod_{j\in V_\ell}^{j=i} g_{n+1-j+i})^{c_\ell}).\end{aligned}$$

Set $\delta_\ell = |U \cap V_\ell|$ for $1 \le \ell \le L$. Then, the above expression is equal to $e(g, \prod_{1\le\ell\le L} g_{n+1}^{\delta_\ell c_\ell}) = e(g, g_{n+1})^u = z^u$ for $u = \delta_1 c_1 + \ldots + \delta_L c_L$. Due to $U \cap V_\ell \ne \emptyset$ and $\delta_\ell \le \zeta_\ell$, we obtain $1 \le \delta_\ell \le \zeta_\ell$, for all $1 \le \ell \le L$. □

B Security Model of Anonymous Credential System

B.1 Misauthentication Resistance

Consider the following misauthentication resistance game.

Misauthentication Resistance Game: The challenger runs **IssuerKeyGen**, and obtains ipk and isk. He provides $\mathcal{A}$ with ipk, and run $\mathcal{A}$. He sets CU with empty, where CU denotes the set of IDs of users corrupted by $\mathcal{A}$. In the run, $\mathcal{A}$ can query the challenger about the following issuing query:

C-Issuing: $\mathcal{A}$ can request the k-th user's certificate on U_k. Then, $\mathcal{A}$ as the user executes **CertObtain** protocol with the challenger as the issuer. The challenger adds k to CU.

Finally, $\mathcal{A}$ outputs a predicate Ψ^*, and a proof σ^*.

Then, $\mathcal{A}$ wins if

1. **Verify**$(ipk, \sigma^*, \Psi^*) =$ valid, and
2. for all $k \in CU$, U_k does not satisfy Ψ.

Misauthentication resistance requires that for all PPT $\mathcal{A}$, the probability that $\mathcal{A}$ wins the misauthentication resistance game is negligible.

B.2 Anonymity

Consider the following anonymity game.

Anonymity Game: The challenger runs **IssuerKeyGen**, and obtains ipk, isk. He provides $\mathcal{A}$ with ipk, isk, and run $\mathcal{A}$. He sets HU with empty. In the run, $\mathcal{A}$ can query the challenger, as follows.

H-Issuing: $\mathcal{A}$ can request the k-th user's certificate on U_k. Then, $\mathcal{A}$ as the issuer executes **CertObtain** protocol with the challenger as the user. The challenger adds k to HU.

Proving: $\mathcal{A}$ can request the k-th user's proof on predicate Ψ. Then, the challenger responds the proof on Ψ of user k, if $k \in HU$.

During the run, as the challenge, $\mathcal{A}$ outputs a predicate Ψ^*, and two users k_0 and k_1, such that both U_{k_0} and U_{k_1} satisfy Ψ^*. If $k_0 \in HU$ and $k_1 \in HU$, the challenger chooses $\phi \in_R \{0, 1\}$, and responds the proof on Ψ^* of user k_ϕ. After that, similarly, $\mathcal{A}$ can make the queries.
Finally, $\mathcal{A}$ outputs a bit ϕ' indicating its guess of ϕ.

If $\phi' = \phi$, $\mathcal{A}$ wins. We define the advantage of $\mathcal{A}$ as $|\Pr[\phi' = \phi] - 1/2|$.

Anonymity requires that for all PPT $\mathcal{A}$, the advantage of $\mathcal{A}$ on the anonymity game is negligible.

Erratum: An Enhanced Anonymous Authentication and Key Exchange Scheme Using Smartcard

Kyung-kug Kim and Myung-Hwan Kim

ISaC and Department of Mathematical Sciences,
Seoul National University, Seoul 151-747, Korea
gggim1231000@gmail.com, mhkim@math.snu.ac.kr

T. Kwon, M.-K. Lee, and D. Kwon (Eds.): ICISC 2012, LNCS 7839, pp. 487–494, 2013.

DOI 10.1007/978-3-642-37682-5_36

The paper "An Enhanced Anonymous Authentication and Key Exchange Scheme Using Smartcard" authored by Kyung-kug Kim and Myung-Hwan Kim, DOI 10.1007/978-3-642-37682-5_34, appearing on pages 487–494 of this publication has been retracted at the request of the authors and the volume editor due to duplicate publication. By mistake, the paper was simultaneously submitted to and published first in the Proceedings of the CUBE International Information Technology Conference, CUBE 2012, DOI 10.1145/2381716.2381857, ACM New York.

The original online version for this chapter can be found at
http://dx.doi.org/10.1007/978-3-642-37682-5_34

Author Index